Duet nie do pokonania!

JavaScript i jQuery

nieoficjalny podręcznik

Wydanie III

David Sawyer McFarland

O'REILLY®

Helion

Tytuł oryginału: JavaScript & jQuery: The Missing Manual, Third Edition

Tłumaczenie: Piotr Rajca

ISBN: 978-83-283-0550-2

Wydawnictwo HELION
ul. Kościuszki 1c, 44-100 GLIWICE
tel. 32 231 22 19, 32 230 98 63
e-mail: *helion@helion.pl*
WWW: *http://helion.pl* (księgarnia internetowa, katalog książek)

Pliki z przykładami omawianymi w książce można znaleźć pod adresem:
ftp://ftp.helion.pl/przyklady/jsjqn3.zip

Drogi Czytelniku!
Jeżeli chcesz ocenić tę książkę, zajrzyj pod adres
http://helion.pl/user/opinie/jsjqn3_ebook
Możesz tam wpisać swoje uwagi, spostrzeżenia, recenzję.

Spis treści

Część IV. Zaawansowane zastosowania jQuery i języka JavaScript 469

Nieoficjalna czołówka

O autorze

David Sawyer McFarland jest prezesem firmy Sawyer McFarland Media, Inc. z siedzibą w Portland w stanie Oregon. Firma ta świadczy usługi z zakresu programowania sieciowego i szkoleń. David tworzy strony WWW od 1995 roku, kiedy to zaprojektował swoją pierwszą witrynę — internetowy magazyn dla specjalistów z branży komunikacyjnej. Pracował też jako webmaster na University of California w Berkeley i w instytucie Berkeley Multimedia Research Center, a także sprawował pieczę nad przebudową witryny *Macworld.com* z wykorzystaniem stylów CSS.

Oprócz tworzenia witryn WWW David zajmuje się pisaniem, szkoleniami i prowadzeniem zajęć. Wykładał projektowanie stron WWW w licznych szkołach: Graduate School of Journalism w Berkeley, Center for Electronic Art, Academy of Art College, Ex'Pressions Center for New Media i Portland State University. Ponadto publikuje artykuły na temat sieci WWW w magazynach *Practical Web Design*, *MX Developer's Journal* i *Macworld* oraz w witrynie *CreativePro.com*.

David czeka na opinie na temat książki pod adresem *missing@sawmac.com*, jeśli jednak szukasz pomocy technicznej, zapoznaj się z listą materiałów podanych w dodatku A.

O zespole pracującym nad książką

Nan Barber (redaktorka) pracuje nad serią „Nieoficjalny podręcznik" jako zastępca redaktora. Mieszka w Massachusetts wraz z mężem i różnymi urządzeniami elektronicznymi. Jej adres e-mail to *nanbarber@gmail.com*.

Melanie Yarbrough (redaktorka wydania) pracuje i bawi się w Cambridge, Massachusetts, gdzie piecze, co tylko jest w stanie sobie wyobrazić, i jeździ po mieście na rowerze. Jej adres e-mail to *myarbrough@oreilly.com*.

Jennifer Davis (recenzentka techniczna) jest inżynierem mającym wieloletnie doświadczenia w zakresie poprawiania wydajności tworzenia platform programowych. Jako inżynier Chef Automation pomaga firmom odkrywać ich najlepsze praktyki i poprawiać organizację pracy, skracając średni czas pracy nad oprogramowaniem. Zajmuje się także organizacją różnego rodzaju imprez dla Reliability Engineering, w ramach grupy użytkowników oprogramowania Chef z rejonu Bay Area.

Alex Stangl (recenzent techniczny) zajmuje się profesjonalnym tworzeniem oprogramowania od ponad 25 lat, używa wielu różnych języków i technologii. Lubi problemy stanowiące duże wyzwanie, zagadki, poznawanie nowych języków (takich jak Clojure), przygotowywanie recenzji technicznych. Realizuje się jako dobry mąż i ojciec. Jego adres e-mail to *alex@stangl.us*.

Jasmine Kwityn (korektorka) jest niezależną adjustatorką i indeksatorką. Mieszka w New Jersey wraz ze swym mężem Edem oraz trzema kotami: Mushki, Axle oraz Punky. Jej adres e-mail to *jasminekwityn@gmail.com*.

Bob Pfahler (indekser) jest niezależnym indekserem, który pracował nad tą książką z ramienia firmy Potomac Indexing, LLC, międzynarodowej grupy indeksatorskiej (*http://www.potomacindexing.com*). Oprócz technologii komputerowych specjalizuje się w zagadnieniach związanych z biznesem, zarządzaniem, biografiami i historią. Jego adres e-mail to *bobpfahler@hotmail.com*.

Podziękowania

Gorąco dziękuję wszystkim, którzy pomagali mi w czasie prac nad tą książką i ustrzegli mnie od popełnienia kłopotliwych błędów, w tym Jennifer Davis oraz Aleksowi Stanglowi. Dziękuję także studentom z Portland State University, którzy przetrwali moje wykłady z języka JavaScript i walczyli z zadawanymi im zadaniami — a szczególnie członkom Team Futzbit (kombinacji Pizza Hut i Taco Bell) za testowanie przykładów; byli to Julia Hall, Amber Brucker, Kevin Brown, Josh Elliot, Tracy O'Connor oraz Blake Womack. Ponadto wszyscy powinniśmy być wdzięczni Johnowi Resigowi i zespołowi pracującemu nad biblioteką jQuery za utworzenie najlepszego dotąd narzędzia, które sprawia, że praca z językiem JavaScript to świetna zabawa.

Na zakończenie dziękuję Davidowi Pogue'owi za pomoc w rozpoczęciu pracy, Nan Barber za poprawę stylu książki, mojej żonie Scholle za znoszenie mych humorów oraz moim dzieciom Grahamowi i Kate za to, że są wspaniałe.

Seria Nieoficjalny podręcznik

Książki z serii „Nieoficjalny podręcznik" (ang. *Missing Manual*) to mądre, świetnie napisane poradniki dotyczące produktów komputerowych, do których nie dołączono drukowanych podręczników (co dotyczy większości narzędzi z tej branży). Każda książka ma ręcznie opracowany indeks i odwołania do konkretnych stron (nie tylko rozdziałów).

Poniższa lista zawiera wydane i przygotowywane tytuły z tej serii:

Access 2007 PL. Nieoficjalny podręcznik, Matthew MacDonald

Access 2010: The Missing Manual, Matthew MacDonald

Access 2013: The Missing Manual, Matthew MacDonald

Adobe Edge Animate: The Missing Manual, Chris Grover

Buying a Home: The Missing Manual, Nancy Conner

CSS3. Nieoficjalny podręcznik. Wydanie III, David Sawyer McFarland

Dreamweaver CC: The Missing Manual, David Sawyer McFarland i Chris Grover

Dreamweaver CS6: The Missing Manual, David Sawyer McFarland

Excel 2007 PL. Nieoficjalny podręcznik, Matthew MacDonald

Excel 2010: The Missing Manual, Matthew MacDonald

Excel 2013: The Missing Manual, Matthew MacDonald

Facebook: The Missing Manual, Third Edition, E.A. Vander Veer

FileMaker Pro 13: The Missing Manual, Susan Prosser i Stuart Gripman

Flash CS6: The Missing Manual, Chris Brover

Fotografia cyfrowa według Davida Pogue'a, David Pogue

Fotografia cyfrowa. Nieoficjalny podręcznik, Chris Grover i Barbara Brundage

Galaxy S4: The Missing Manual, Preston Gralla

Galaxy S5: The Missing Manual, Preston Gralla

Galaxy Tab: The Missing Manual, Preston Gralla

Google+: The Missing Manual, Kevin Purdy

iMovie '11 & iDVD: The Missing Manual, David Pogue i Aaron Miller

Internet. Nieoficjalny podręcznik, David Pogue i J.D. Biersdorfer

iPad: The Missing Manual, Sixth Edition, J.D. Biersdorfer

iPhone App Development: The Missing Manual, Craig Hockenberry

iPhone: The Missing Manual, Seventh Edition, David Pogue

iPhoto '11: The Missing Manual, David Pogue i Lesa Snider

iPod: The Missing Manual, Eleventh Edition, J.D. Biersdorfer i David Pogue

Kindle Fire HD: The Missing Manual, Peter Meyers

Komputery PC. Nieoficjalny podręcznik, Andy Rathbone

Living Green: The Missing Manual, Nancy Conner

Mac OS X Lion: The Missing Manual, David Pogue

Mac OS X Snow Leopard: The Missing Manual, David Pogue

Microsoft Project 2007 PL. Nieoficjalny podręcznik, Bonnie Biafore

Microsoft Project 2010: The Missing Manual, Bonnie Biafore

Microsoft Project 2013: The Missing Manual, Bonnie Biafore
Motorola Xoom: The Missing Manual, Preston Gralla
Mózg. Nieoficjalny podręcznik, Matthew MacDonald
NOOK HD: The Missing Manual, Preston Gralla
Office 2010 PL. Nieoficjalny podręcznik, Nancy Connor i Matthew MacDonald
Office 2011 for Macintosh: The Missing Manual, Chris Grover
Office 2013: The Missing Manual, Nancy Conner i Matthew MacDonald
OS X Mavericks: The Missing Manual, David Pogue
OS X Mountain Lion: The Missing Manual, David Pogue
OS X Yosemite: The Missing Manual, David Pogue
Personal Investing: The Missing Manual, Bonnie Biafore
Photoshop CC: The Missing Manual, Lesa Snider
Photoshop CS6: The Missing Manual, Lesa Snider
Photoshop Elements 12: The Missing Manual, Barbara Brundage
PHP & MySQL: The Missing Manual, Second Edition, Brett McLaughlin
PowerPoint 2007 PL. Nieoficjalny podręcznik, E.A. Vander Veer
QuickBooks 2014: The Missing Manual, Bonnie Biafore
QuickBooks 2015: The Missing Manual, Bonnie Biafore
Switching to the Mac: The Missing Manual, Mavericks Edition, David Pogue
Switching to the Mac: The Missing Manual, Yosemite Edition, David Pogue
Tworzenie stron WWW. Nieoficjalny podręcznik. Wydanie II, Matthew MacDonald
Windows 7: The Missing Manual, David Pogue
Windows 8: The Missing Manual, David Pogue
Windows Vista PL. Nieoficjalny podręcznik, David Pogue
Word 2007 PL. Nieoficjalny podręcznik, Chris Grover
WordPress: The Missing Manual, Second Edition, Matthew MacDonald
Your Body: The Missing Manual, Matthew MacDonald
Your Money: The Missing Manual, J.D. Roth

Wprowadzenie

Na początku swego istnienia sieć WWW była dość nudnym miejscem. Strony WWW oparte na zwykłym HTML-u służyły tylko do wyświetlania informacji. Interakcja ograniczała się do kliknięcia odnośnika i oczekiwania na wczytanie nowej strony.

Dziś większość witryn WWW działa niemal tak szybko jak tradycyjne programy i natychmiast reaguje na każde kliknięcie myszą. Jest to możliwe dzięki narzędziom, którym poświęcona jest ta książka, czyli językowi JavaScript oraz wspomagającej go bibliotece jQuery.

Czym jest JavaScript?

JavaScript to język programowania, który umożliwia wzbogacanie kodu HTML o animacje, interaktywność i dynamiczne efekty wizualne.

JavaScript pozwala zwiększyć użyteczność stron WWW przez udostępnianie natychmiastowych informacji zwrotnych. Przykładowo koszyk zakupów oparty na tym języku może wyświetlać łączną cenę zakupów (z uwzględnieniem podatków i kosztów wysyłki) natychmiast po wybraniu produktów. Przy użyciu języka JavaScript można też wyświetlić komunikat o błędzie bezpośrednio po próbie przesłania niekompletnego formularza.

JavaScript umożliwia także tworzenie zabawnych, dynamicznych i interaktywnych interfejsów. Za jego pomocą można przekształcić statyczną stronę zawierającą miniaturki zdjęć w animowany pokaz slajdów. Można też zrobić coś bardziej wyrafinowanego, na przykład pokazać na stronie znacznie więcej danych bez jednoczesnego natłoku informacyjnego, umieszczając je w niewielkich panelach, które użytkownik może wyświetlić po kliknięciu myszą (patrz strona 351). Można także utworzyć coś zabawnego i atrakcyjnego, takiego jak etykiety ekranowe pokazujące uzupełniające informacje na temat elementów prezentowanych na stronie (patrz strona 345).

Kolejną zaletą języka JavaScript jest błyskawiczność działania. Ta cecha umożliwia natychmiastowe reagowanie na działania użytkowników: kliknięcie odnośnika, wypełnienie formularza lub przesunięcie kursora myszy. JavaScript nie powoduje uciążliwych opóźnień specyficznych dla języków używanych po stronie serwera (na przykład języka PHP), które wymagają przesyłania danych między przeglądarką a serwerem. Ponieważ JavaScript nie wymaga ciągłego odświeżania stron, umożliwia tworzenie witryn, które działają bardziej jak tradycyjne programy niż strony WWW.

Jeśli odwiedziłeś kiedyś witrynę Google Maps (*http://maps.google.com/*), widziałeś już JavaScript w akcji. W tej witrynie można wyświetlić mapę swojej (a właściwie zupełnie dowolnej) miejscowości, a następnie zwiększyć przybliżenie, aby zobaczyć ulice i przystanki autobusowe, lub oddalić obraz w celu uzyskania ogólnego obrazu miasta, województwa albo kraju. Choć już wcześniej istniały liczne witryny z mapami, pobranie potrzebnych informacji wymagało w nich długiego odświeżania wielu stron. Google Maps nie wymaga wczytywania nowych stron i natychmiast reaguje na działania użytkownika.

Za pomocą języka JavaScript można rozwijać zarówno bardzo proste programy (na przykład skrypty wyświetlające stronę WWW w nowym oknie przeglądarki), jak i kompletne aplikacje sieciowe. Do tej drugiej grupy należą na przykład narzędzia z rodziny Google Docs (*http://docs.google.com/*), które umożliwiają przygotowywanie prezentacji, edycję dokumentów i tworzenie arkuszy kalkulacyjnych w przeglądarce w podobny sposób, jak robi się to przy użyciu tradycyjnych programów.

Trochę historii

Język JavaScript został opracowany w 1995 roku przez pracownika firmy Netscape Brendana Eicha, który pracował nad nim 10 dni. Oznacza to, że ma on niemal tyle samo lat co sieć WWW. Choć JavaScript jest dziś traktowany jak pełnowartościowe narzędzie, nie zawsze tak było. W przeszłości uznawano go za język programowania dla amatorów, służący do dodawania bezużytecznych komunikatów w pasku statusu przeglądarki lub animowanych motylków podążających za kursorem myszy. W sieci można było łatwo znaleźć tysiące bezpłatnych programów w języku JavaScript (nazywanych *skryptami*), jednak wiele z nich działało tylko w wybranych przeglądarkach lub powodowało ich awarie.

Uwaga: JavaScript nie ma nic wspólnego z językiem Java. Pierwotna nazwa języka JavaScript to LiveScript, jednak dział marketingu firmy Netscape uznał, że powiązanie nowego narzędzia z bardzo popularnym wówczas językiem Java zwiększy zainteresowanie użytkowników. Nie popełnij błędu i nie pomyl obu tych języków... zwłaszcza na rozmowie kwalifikacyjnej!

Początkowo negatywny wpływ na rozwój języka JavaScript miał brak zgodności między dwiema najpopularniejszymi przeglądarkami: Netscape Navigatorem i Internet Explorerem. Ponieważ firmy Netscape i Microsoft starały się udostępnić produkt lepszy od konkurencji, oferując nowsze i (pozornie) lepsze funkcje, obie przeglądarki działały w odmienny sposób. Utrudniało to tworzenie programów JavaScript, które funkcjonowałyby prawidłowo w obu aplikacjach.

Uwaga: Po udostępnieniu przez Netscape języka JavaScript Microsoft wprowadził jScript — własną wersję języka JavaScript obsługiwaną przez przeglądarkę Internet Explorer.

Na szczęście, te straszliwe dni już dawno minęły i nowoczesne przeglądarki, takie jak Firefox, Safari, Chrome, Opera czy też Internet Explorer 11, korzystają ze standardowego sposobu obsługi JavaScriptu, co znacznie ułatwia pisanie w tym języku programów, które mogą działać w niemal wszystkich przeglądarkach. (Pomiędzy aktualnie używanymi przeglądarkami wciąż można znaleźć trochę niezgodności, dlatego też będziesz musiał poznać kilka sztuczek, by pisać programy, które naprawdę będą mogły działać we wszystkich przeglądarkach. W tej książce dowiesz się, jak poradzić sobie z problemami związanymi z niezgodnością przeglądarek).

W ciągu kilku ostatnich lat nastąpiło odrodzenie języka JavaScript, napędzane przez popularne witryny, na przykład Google, Yahoo i Flickr, w których język ten posłużył do utworzenia interaktywnych aplikacji sieciowych. Nigdy wcześniej nie było lepszego czasu na naukę języka JavaScript. Dzięki bogatej wiedzy i wysokiej jakości skryptom nawet początkujący programiści mogą wzbogacić witryny o zaawansowane, interaktywne funkcje.

Uwaga: Język JavaScript jest także znany pod nazwą ECMAScript. ECAMScript jest „oficjalną" specyfikacją języka, opracowaną i utrzymywaną przez międzynarodową organizację standaryzacyjną o nazwie Ecma International (*http://www.ecmascript.org*).

JavaScript jest wszędzie

JavaScript działa nie tylko na stronach WWW. Język ten jest tak użyteczny, że kiedy już go poznasz, będziesz mógł tworzyć widżety Yahoo i Google Apps, pisać programy na iPhone'y i dodawać nowe, skryptowe możliwości do programów firmy Adobe, takich jak Acrobat, Photoshop, Illustrator oraz Dreamweaver. Swoją drogą, ta ostatnia aplikacja zawsze umożliwiała dodawanie nowych poleceń zaawansowanym programistom używającym języka JavaScript.

W swoim systemie operacyjnym Mac OS X Yosemite firma Apple zapewniła użytkownikom możliwość automatyzowania ich komputerów przy użyciu języka JavaScript. Co więcej, JavaScript jest używany w wielu pomocnych narzędziach programistycznych, takich jak Gulp.js (może automatycznie kompresować obrazy, pliki CSS i JavaScript) lub Bower (ułatwia pobieranie często używanych bibliotek JavaScript, na przykład jQuery, jQuery UI czy też AngularJS).

JavaScript jest także coraz częściej stosowany do pisania kodu wykonywanego po stronie serwera. Platforma Node.js (wersja silnika JavaScript V8, opracowanego przez Google, która pozwala na wykonywanie skryptów JavaScript na serwerze) jest chętnie wykorzystywana przez takie firmy jak Walmart, PayPal oraz eBay. A zatem poznanie języka JavaScript może nawet stanowić początek kariery związanej z tworzeniem złożonych aplikacji serwerowych. W rzeczywistości połączenie języka JavaScipt używanego do tworzenia *interfejsu użytkownika aplikacji* (czyli fragmentów działających w przeglądarce) oraz części serwerowej jest określane jako *stosowanie JavaScriptu w pełnym zakresie*.

Innymi słowy, nigdy nie było lepszego momentu do nauki JavaScript niż ten czas!

Czym jest jQuery?

JavaScript ma jeden mały, krępujący sekret: pisanie w tym języku jest dosyć trudne. Choć pisanie w JavaScripcie i tak jest prostsze niż w wielu innych językach, wciąż jest to język programowania. A dla wielu osób, zaliczają się do nich także projektanci stron WWW, programowanie jest trudne. Aby dodatkowo skomplikować cały problem, różne przeglądarki rozumieją JavaScript nieco inaczej, przez co program, który na przykład w przeglądarce Chrome działa prawidłowo, w Internet Explorerze 9 może nie działać w ogóle. Ta często występująca sytuacja może kosztować wiele godzin żmudnego testowania programu na wielu różnych komputerach i w wielu przeglądarkach, zanim upewnimy się, że program będzie działał prawidłowo u wszystkich użytkowników witryny.

I właśnie w tym miejscu pojawia się jQuery. Jest to biblioteka języka JavaScript zbudowana w celu ułatwienia pisania programów w tym języku. Biblioteka jQuery jest złożonym programem napisanym w JavaScripcie, który zarówno ułatwia opracowanie skomplikowanych zadań, jak i rozwiązuje wiele problemów związanych ze zgodnością przeglądarek. Innymi słowy, jQuery uwalnia od dwóch największych problemów języka JavaScript — złożoności oraz drobiazgowej natury różnych przeglądarek.

Biblioteka jQuery jest tajemną bronią projektantów strony w walce z programowaniem w języku JavaScript. Dzięki zastosowaniu jQuery w jednym wierszu kodu można wykonać operacje, które w innym przypadku wymagałyby napisania setek wierszy własnego kodu i poświęcenia długich godzin na ich testowanie. W rzeczywistości szczegółowa książka poświęcona wyłącznie językowi JavaScript byłaby przynajmniej dwukrotnie grubsza od tej, a po jej przeczytaniu (gdybyś w ogóle dotrwał do końca) byłbyś w stanie zrobić dwukrotnie mniej niż przy wykorzystaniu choćby podstawowej znajomości biblioteki jQuery.

To właśnie z tego powodu znaczna część tej książki jest poświęcona bibliotece jQuery. Za jej pomocą można zrobić tak wiele w tak prosty sposób. Jej kolejną wspaniałą cechą jest to, że dzięki tysiącom tak zwanych „wtyczek" pozwala w bardzo prosty sposób dodawać do tworzonych witryn zaawansowane możliwości. Przykładowo wtyczka jQuery UI (którą poznasz na stronie 323) pozwala tworzyć złożone elementy interfejsu użytkownika, takie jak zestawy kart, rozwijane menu czy kalendarze do wybierania dat, przy użyciu jednego wiersza kodu!

Nic zatem dziwnego, że jQuery jest używana na milionach witryn (*http://trends.builtwith.com/javascript/jQuery*). Została wbudowana w popularne systemy zarządzania treścią, takie jak Drupal lub WordPress. Nawet w ogłoszeniach o pracę można znaleźć firmy poszukujące „programistów jQuery", bez wspominania o znajomości języka JavaScript. Poznając jQuery, dołączasz do ogromnej społeczności programistów i projektantów, korzystających z prostszego i dającego większe możliwości sposobu tworzenia interaktywnych witryn WWW.

HTML: podstawowa struktura

JavaScript nie jest przydatny bez dwóch innych podstawowych narzędzi do tworzenia stron WWW — języków HTML i CSS. Wielu programistów łączy te trzy języki z „warstwami" stron. HTML służy do tworzenia warstwy *strukturalnej*, która umożliwia uporządkowanie grafiki i tekstu w sensowny sposób. CSS (kaskadowe arkusze stylów) zapewniają warstwę *prezentacji* i umożliwiają atrakcyjne przedstawianie treści zapisanej w kodzie HTML. Język JavaScript tworzy warstwę *operacyjną* i wprowadza życie w strony WWW, umożliwiając interakcję z użytkownikami.

Oznacza to, że do opanowania języka JavaScript potrzebna jest znajomość języków HTML i CSS.

Wskazówka: Kompletne wprowadzenie do języka HTML5 znajdziesz w książce *HTML5. Rusz głową!* Erica T Freemana i Elisabeth Robson. Szczegółowe omówienie kaskadowych arkuszy stylów przedstawiono w książce *CSS3. Nieoficjalny podręcznik* Davida Sawyera McFarlanda (obie wydane przez wydawnictwo Helion).

HTML (ang. *Hypertext Markup Language*, czyli hipertekstowy język znaczników) zawiera proste polecenia nazywane *znacznikami*, które określają różne części stron WWW. Poniższy kod HTML tworzy prostą stronę:

```
<!DOCTYPE html>
<html>
<head>
<meta charset=utf-8>
<title>To tytuł strony.</title>
</head>
<body>
A to tekst w ciele strony.
</body>
</html>
```

Nie jest to ciekawy kod, ale przedstawia wszystkie podstawowe elementy stron WWW. Ta strona rozpoczyna się od wiersza określającego, z jakiego typu dokumentem mamy do czynienia i z jakimi standardami jest on zgodny. Wiersz ten nazywany jest deklaracją typu dokumentu, w skrócie — `doctype`. Język HTML jest dostępny w różnych wersjach, a w każdej z nich można używać innej deklaracji typu. W tym przypadku zastosowana została deklaracja typu dokumentu dla języka HTML5. Analogiczne deklaracje dla dokumentów HTML 4.01 oraz XHTML są znacznie dłuższe, a dodatkowo zawierają adres URL wskazujący przeglądarce położenie specjalnego pliku definicji danego języka.

Deklaracja typu informuje przeglądarkę o sposobie wyświetlania strony. Może wpływać nawet na działanie kodu CSS i JavaScript. Jeśli programista poda błędną deklarację lub w ogóle ją pominie, może długo szukać przyczyny niezgodności w funkcjonowaniu skryptów w różnych przeglądarkach. Dlatego zawsze należy pamiętać o podaniu typu dokumentu.

Niegdyś używano wielu typów dokumentów; były to na przykład HTML 4.01 Transitional, HTML 4.01 Strict, XHTML 1.0 Transitional, XHTML 1.0 Strict — wszystkie miały postać długiego wiersza trudnego kodu, w którym bardzo łatwo można było zrobić błąd. Deklaracja typu dokumentu języka HTML5 — `<!DOCTYPE html>` — jest krótka, prosta i to właśnie jej powinieneś używać.

Działanie znaczników HTML

W przykładowym kodzie zamieszczonym na stronie 21, podobnie jak w kodzie HTML każdej strony WWW, większość poleceń występuje w parach zawierających bloki tekstu i inne instrukcje. **Znaczniki** wewnątrz nawiasów ostrych to polecenia, które informują przeglądarkę o tym, jak ma wyświetlić stronę. Są to znaczniki z nazwy „hipertekstowy język znaczników".

Znacznik początkowy (*otwierający*) wskazuje przeglądarce początek polecenia, a znacznik końcowy określa koniec instrukcji. Znacznik końcowy (*zamykający*) zawsze zawiera ukośnik (/) po pierwszym nawiasie ostrym (<). Przykładowo znacznik `<p>` oznacza początek akapitu, a znacznik `</p>` — jego koniec. Niektóre znaczniki, takie jak `<img>`, `<input>` czy też `<br>`, nie występują w parach — w ich przypadku używany jest tylko znacznik otwierający.

Aby strona WWW działała poprawnie, musi obejmować przynajmniej trzy poniższe znaczniki:

- Znacznik `<html>`, który pojawia się raz na początku strony (po deklaracji typu), a następnie — z ukośnikiem — na jej końcu. Informuje on przeglądarkę o tym, że dokument jest napisany w języku HTML. Cała zawartość strony, w tym inne znaczniki, znajduje się między otwierającym a zamykającym znacznikiem `<html>`.

  Jeśli stronę WWW potraktować jak drzewo, znacznik `<html>` to pień. Wychodzą z niego dwie gałęzie, które reprezentują dwie podstawowe części każdej strony — *sekcję nagłówkową* i *ciało*.

- *Sekcja nagłówkowa* strony WWW znajduje się wewnątrz znaczników `<head>` i zawiera między innymi tytuł strony. Można tu umieścić także inne, niewidoczne informacje (na przykład słowa kluczowe używane przy wyszukiwaniu), które są przydatne dla przeglądarek i wyszukiwarek.

  Sekcja nagłówkowa może też zawierać informacje używane przez przeglądarkę do wyświetlania stron i zwiększania interaktywności. Mogą to być na przykład kaskadowe arkusze stylów. Ponadto programiści często umieszczają w tej sekcji kod JavaScript i odnośniki do plików z takim kodem.

- *Ciało* strony znajduje się w znacznikach `<body>` i obejmuje wszystkie informacje widoczne w oknie przeglądarki: nagłówki, tekst, obrazki i tak dalej.

Znacznik `<body>` zawiera zwykle następujące elementy:

- Znaczniki `<p>` (otwierający) i `</p>` (zamykający), które informują przeglądarkę o początku i końcu akapitu.
- Znacznik `<strong>`, który służy do wyróżniania tekstu. Umieszczenie słów między tym znacznikiem a tagiem `</strong>` powoduje pogrubienie czcionki. Fragment kodu HTML `<strong>Uwaga!</strong>` to polecenie wyświetlenia słowa „Uwaga!" pogrubioną czcionką.
- Znacznik `<a>` (znacznik kotwicy) tworzy *odnośnik* na stronie WWW. Odnośnik (inaczej *odsyłacz*, *łącze*, *hiperłącze* lub *link*) może prowadzić do dowolnego miejsca w sieci WWW. Aby poinformować przeglądarkę o docelowej lokalizacji, należy w znaczniku `<a>` podać adres, na przykład `<a href= http://www.missingmanuals.com/ >Kliknij tutaj!</a>`.

WIEDZA W PIGUŁCE

Walidacja stron WWW

Na stronie 21 wspomniano, że deklaracja typu określa wersję języka HTML lub XHTML użytą do utworzenia strony. Między typami stron występują drobne różnice. Na przykład XHTML, w odróżnieniu od HTML 4.01, wymaga zamknięcia znacznika `<p>` oraz pisania nazw znaczników i atrybutów małymi literami (`<a>` *zamiast* `<A>`). HTML5 zawiera nowe znaczniki i pozwala na korzystanie ze składni HTML lub XHTML. Z uwagi na różne reguły obowiązujące w poszczególnych wersjach należy zawsze przeprowadzić *walidację* strony.

Walidator kodu HTML to program, który sprawdza, czy strona jest prawidłowo napisana. Takie narzędzie określa typ dokumentu, a następnie analizuje kod, aby sprawdzić, czy jest zgodny z podaną deklaracją. Walidator wskazuje między innymi błędnie napisane nazwy znaczników i niezamknięte znaczniki. Organizacja W3C (ang. *World Wide Web Consortium*), odpowiedzialna za zarządzanie licznymi technologiami sieciowymi, udostępnia bezpłatny walidator pod adresem *http://validator.w3.org*. Wystarczy skopiować kod HTML i wkleić go w formularzu, przesłać stronę lub wskazać walidatorowi istniejącą witrynę. Narzędzie sprawdzi wtedy kod HTML i poinformuje, czy strona jest prawidłowa. Jeśli wykryje błędy, opisze je i wskaże wiersz ich wystąpienia w pliku HTML.

Poprawny kod HTML to nie tylko prawidłowa forma dokumentu. Strona musi przejść walidację, aby programy w języku JavaScript działały poprawnie. Wiele skryptów manipuluje kodem HTML — na przykład sprawdza wartość pól formularza lub umieszcza nowy fragment kodu (taki jak komunikat o błędzie) w określonym miejscu. Aby kod JavaScript mógł uzyskać dostęp do strony i manipulować nią, kod HTML musi być odpowiednio uporządkowany. Pominięcie znacznika zamykającego, dwukrotne użycie tego samego identyfikatora lub błędne zagnieżdżenie znaczników może spowodować, że skrypt będzie zachowywał się w nieoczekiwany sposób lub w ogóle przestanie działać.

Przeglądarka wykrywa, że po kliknięciu słów „Kliknij tutaj!" ma przejść do witryny poświęconej serii „Missing Manual". Część `href` znacznika to *atrybut*, a sam adres URL to *wartość* tego atrybutu. W przykładowym kodzie `http://www.missingmanuals.com` to *wartość* atrybutu `href`.

CSS: dodawanie stylu do stron

Dawniej HTML był jedynym językiem, który trzeba było poznać. Programista mógł tworzyć strony z kolorowym tekstem i grafiką oraz wyróżniać słowa za pomocą czcionek o różnych rozmiarach, krojach i kolorach. Jednak obecnie użytkownicy witryn mają większe oczekiwania, dlatego trzeba zastosować nowszą, bardziej elastyczną technologię, o nazwie kaskadowe arkusze stylów (ang. *Cascading Style Sheets* — CSS), która umożliwia tworzenie bardziej zaawansowanych wizualnie stron. CSS to język do obsługi formatowania, który pozwala zwiększyć atrakcyjność tekstu, budować strony o złożonym układzie i nadawać styl witrynie.

HTML służy do określania struktury strony. Pomaga wskazać elementy, które programista chce zaprezentować światu. Znaczniki `<h1>` i `<h2>` określają nagłówki i ich względne znaczenie: *nagłówek z poziomu 1* jest ważniejszy od *nagłówka z poziomu 2*. Znacznik `<p>` określa podstawowy akapit z informacjami. Inne znaczniki zapewniają dodatkowe wskazówki strukturalne. Przykładowo element `<ul>` określa listę wypunktowaną, która pozwala poprawić czytelność listy produktów wymienionych w przepisie.

CSS natomiast pozwala zaprojektować wygląd dobrze uporządkowanej zawartości dokumentu HTML i sprawić, że będzie ona bardziej atrakcyjna i czytelna. *Styl* CSS to reguła, która informuje przeglądarkę o tym, jak ma wyświetlać dany element na stronie. Na przykład można utworzyć regułę CSS, zgodnie z którą tekst wszystkich znaczników `<h1>` ma mieć 36 pikseli wysokości, czcionkę Verdana i pomarańczowy kolor. Język CSS umożliwia też wykonywanie bardziej zaawansowanych zadań, takich jak dodawanie obramowań, określanie szerokości marginesów, a nawet precyzyjne wskazywanie położenia elementów strony.

Jedne z najciekawszych zmian, jakie można wprowadzić na stronie za pomocą języka JavaScript, dotyczą właśnie stylów CSS. JavaScript umożliwia dodawanie stylów do znaczników HTML, usuwanie reguł i dynamiczne zmienianie właściwości stylów CSS na podstawie danych wprowadzonych przez użytkownika lub w wyniku kliknięcia myszą. Można nawet animować elementy, zmieniając właściwości jednego stylu na właściwości innego (na przykład animować kolor tła, by zmienił się z żółtego na czerwony). Można też wyświetlić lub ukryć element przez zmianę wartości właściwości `display` albo uruchomić animację w postaci przesuwania się elementu po stronie, co wymaga dynamicznego zmieniania wartości właściwości `position`.

Anatomia stylu

Pojedynczy styl, który określa wygląd jednego elementu, jest dość prosty. Jest to reguła, która informuje przeglądarkę, jak ma sformatować element (na przykład wyświetlić nagłówek na niebiesko, dodać czerwone obramowanie wokół rysunku lub utworzyć 150-pikselową ramkę z listą odnośników). Styl to informacja: „Przeglądarko, spraw, aby *to* wyglądało *tak*". Style składają się z dwóch elementów: formatowanego elementu strony (*selektora*) i instrukcji formatujących (*bloku deklaracji*). Selektorem może być nagłówek, akapit, rysunek i tak dalej. Bloki deklaracji pozwalają zmienić kolor tekstu na niebieski, dodać czerwone obramowanie wokół akapitu, umieścić zdjęcie na środku strony — możliwości są niemal nieskończone.

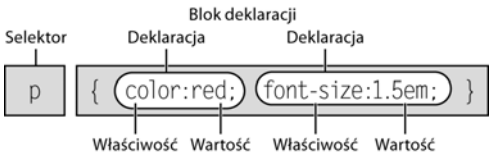
Uwaga: Autorzy tekstów technicznych często używają słownictwa organizacji W3C i nazywają style CSS *regułami*. Tu oba pojęcia występują wymiennie.

Oczywiście style CSS nie są zapisane w języku naturalnym, dlatego używają swojego własnego. Aby na przykład zmienić kolor i rozmiar czcionki wszystkich akapitów strony, należy użyć następującego kodu:

```
p { color: red; font-size: 1.5em; }
```

Ten styl to informacja: „Użyj do wyświetlania tekstu wszystkich akapitów (tekstu w znacznikach `<p>`) czerwonej czcionki o wysokości 1,5 em". Jednostka `em` jest oparta na standardowym rozmiarze tekstu w przeglądarce. Rysunek W.1 pokazuje, że nawet tak prosty styl składa się z kilku elementów. Oto one.

- **Selektor**, który wskazuje przeglądarce formatowane elementy strony. Mogą to być na przykład nagłówki, akapity, rysunki lub odnośniki. Na rysunku W.1 selektor (`p`) wskazuje znacznik `<p>`, dlatego przeglądarka sformatuje tekst w tych



Rysunek W.1. *Styl (reguła) składa się z dwóch głównych części: selektora, który wskazuje przeglądarce formatowany element, i bloku deklaracji, zawierającego instrukcje formatujące*

znacznikach za pomocą instrukcji podanych w stylu. Przy użyciu wielu dostępnych selektorów i szczypty wyobraźni można uzyskać pełną kontrolę nad wyglądem stron. (Selektory odgrywają także kluczowe znaczenie podczas korzystania z biblioteki jQuery, dlatego też w tej książce, zaczynając od strony 147, znajdziesz szczegółowe informacje na ich temat).

- **Blok deklaracji**, który obejmuje opcje formatowania stosowane do elementów określonych za pomocą selektora. Blok ten rozpoczyna się od otwierającego nawiasu klamrowego ({) i kończy nawiasem zamykającym tego typu (}).
- **Deklaracja**. Pomiędzy otwierającym i zamykającym nawiasem klamrowym umieszczana jest jedna lub kilka deklaracji czy też instrukcji formatujących. Każda deklaracja składa się z dwóch części — *właściwości* i *wartości* — a kończy średnikiem. Właściwość oraz wartość są od siebie oddzielone dwukropkiem, na przykład `color: red`.
- **Właściwości**. CSS udostępnia szeroki zakres opcji formatowania, nazywanych *właściwościami*. Właściwość to słowo (lub kilka słów połączonych dywizami) określające efekt działania stylu. Większość właściwości ma proste nazwy, na przykład `font-size`, `margin-top` i `background-color`. Ta ostatnia — co łatwo zgadnąć — określa kolor tła.

Uwaga: Jeśli chcesz lepiej poznać język CSS, zapoznaj się z książką *CSS3. Nieoficjalny podręcznik.*

- **Wartości.** Programista może zrealizować swój twórczy potencjał przez przypisanie *wartości* do właściwości CSS, zmieniając na przykład kolor tła na niebieski, czerwony, fioletowy lub jaskrawojasnozielony. Różne właściwości CSS wymagają wartości określonego typu — koloru (na przykład `red` lub `#FF0000`), długości (na przykład `18px`, `2in` lub `5em`), adresu URL (na przykład *images/background.gif*) lub słowa kluczowego (na przykład `top`, `center` lub `bottom`).

Na rysunku W.1 styl zapisany jest w jednym wierszu, jednak nie jest to wymagana forma. Wiele stylów obejmuje liczne właściwości formatujące, dlatego łatwiej odczytać regułę po podzieleniu jej na wiersze. Na przykład selektor i otwierający nawias klamrowy można umieścić w pierwszym wierszu, poszczególne deklaracje w dalszych wierszach, a na końcu dodać zamykający nawias klamrowy:

```
p {
  color: red;
  font-size: 1.5em;
}
```

Pomocne jest dodawanie za pomocą tabulacji lub kilku odstępów wcięć przy właściwościach. Pozwala to oddzielić wizualnie selektor od deklaracji. Odstęp między dwukropkiem a wartością właściwości jest opcjonalny, jednak poprawia czytelność stylu. Odstępy można dodawać w dowolny sposób. Poprawne są wszystkie wersje: `color:red`, `color: red` i `color : red`.

Narzędzia do programowania w języku JavaScript

Do tworzenia stron zawierających kody HTML, CSS i JavaScript wystarczy prosty edytor tekstu, na przykład Notatnik (Windows) lub Text Edit (Mac). Jednak po wpisaniu kilkuset wierszy kodu JavaScript warto wypróbować program lepiej dostosowany do tworzenia stron WWW. W tym punkcie znajdziesz listę popularnych programów tego typu — zarówno bezpłatnych, jak i komercyjnych.

Uwaga: Dostępne są dosłownie setki narzędzi, które pomagają tworzyć strony WWW i pisać programy w języku JavaScript, dlatego podana lista nie jest kompletna. Znalazły się tu tylko „największe hity" używane obecnie przez fanów tego języka.

Programy bezpłatne

Dostępnych jest wiele bezpłatnych programów do edycji stron WWW i arkuszy stylów. Jeśli wciąż używasz Notatnika lub edytora Text Edit, wypróbuj jedno z poniższych narzędzi.

- **Brackets** (Windows, Max, Linux, *http://brackets.io/*) jest edytorem o otwartym kodzie źródłowym, opracowanym przez firmę Adobe. Jest to program darmowy, w którym zaoferowano wiele wspaniałych możliwości, takich jak podgląd edytowanej strony na bieżąco w przeglądarce. Co ciekawe, Brackets został napisany w języku JavaScript.
- **Notepad++** (Windows, *http://notepad-plus-plus.org*) to przyjaciel programisty. Koloruje składnię kodów JavaScript i HTML oraz umożliwia zapisywanie makr i przypisywanie do nich skrótów klawiaturowych, co pozwala zautomatyzować proces wstawiania najczęściej używanych fragmentów kodu.
- **HTML-Kit** (Windows, *http://www.chami.com/html-kit*) to rozbudowany edytor HTML/XHTML, który zawiera wiele przydatnych funkcji, na przykład możliwość podglądu strony WWW bezpośrednio w programie (dzięki temu nie trzeba nieustannie przełączać się między przeglądarką i edytorem), kombinacje klawiszy do dodawania znaczników HTML i tak dalej.
- **CoffeeCup Free HTML Editor** (Windows, *http://www.coffeecup.com/FREE-EDITOR*) to bezpłatna wersja komercyjnego edytora CoffeeCup HTML (69 dolarów).
- **TextWrangler** (Mac, *http://www.barebones.com/products/textwrangler/*) to bezpłatne narzędzie, które jest zubożoną wersją BBEdit — rozbudowanego, popularnego edytora tekstu dla komputerów Mac. TextWrangler nie ma wszystkich wbudowanych narzędzi do tworzenia kodu znanych z BBEdit, ale udostępnia kolorowanie składni (wyróżnianie znaczników i właściwości odmiennymi kolorami, co ułatwia przeglądanie strony i wyszukiwanie jej obszarów), obsługę FTP (co umożliwia przesyłanie plików na serwer sieciowy) i inne funkcje.

- **Eclipse** (Windows, Linux, Mac; *http://www.eclipse.org/*) jest bezpłatnym środowiskiem programistycznym, bardzo popularnym wśród programistów używających języka Java, lecz posiada także narzędzia umożliwiające pracę z kodem HTML, CSS i JavaScript. Jego wersję przeznaczoną dla programistów JavaScript można znaleźć na stronie *http://www.eclipse.org/downloads/packages/eclipse-ide-javascript-web-developers/indigor*; dostępna jest także wtyczka ułatwiająca dodawanie mechanizmu automatycznego uzupełniania dla jQuery (*http://marketplace.eclipse.org/category/free-tagging/jquery*).
- **Aptana Studio** (Windows, Linux, Mac; (*http://www.aptana.com*) jest potężnym, bezpłatnym środowiskiem przeznaczonym do tworzenia kodów HTML, CSS, JavaScript, PHP oraz Ruby on Rails.
- **Vim** oraz **Emacs** są znanymi od dawna, doskonałymi edytorami używanymi w systemie Unix. Są dostępne także w systemach Mac OS X oraz Linux; można także pobrać ich wersje przeznaczone dla systemu Windows. Lubią je poważni programiści, jednak dla większości osób nauka korzystania z nich może być trudnym zadaniem.
- **Atom** (Windows i Mac, *https://atom.io/*). To zupełnie nowy gracz w grupie edytorów dla programistów. Jego kody źródłowe są dostępne w serwisie GitHub (służącym do dzielenia się i wspólnej, grupowej pracy nad projektami), a sam program zawiera wiele możliwości stworzonych specjalnie z myślą o potrzebach nowoczesnych programistów. Cechuje go budowa modularna, pozwalająca na tworzenie wielu wtyczek rozszerzających jego wbudowane możliwości funkcjonalne.

Oprogramowanie komercyjne

Komercyjne programy do tworzenia witryn są bardzo zróżnicowane — od niedrogich edytorów tekstu po kompletne, pełne dodatkowych funkcji narzędzia do budowy witryn.

- **SublimeText** (Windows, Max, Linux, *https://www.sublimetext.com/*, 70 dolarów) jest ukochanym narzędziem wielu programistów. Edytor pozwala zaoszczędzić sporo czasu osobom piszącym w języku JavaScript. Przykładem jego możliwości może być „automatyczne tworzenie par znaków" — mechanizm, który automatycznie dodaje drugi znak z pary znaków o specjalnym znaczeniu (na przykład program automatycznie doda nawias zamykający, kiedy programista wpisze znak nawiasu otwierającego).
- **EditPlus** (Windows, *https://www.editplus.com/*, 35 dolarów) to niedrogi edytor tekstu z obsługą kolorowania składni, protokołu FTP, automatycznego uzupełniania tekstu i innymi funkcjami przyspieszającymi pracę.
- **BBEdit** (Mac, *http://www.barebones.com/products/bbedit/*, 49,99 dolarów). Ten bardzo popularny edytor tekstu dla komputerów Mac udostępnia wiele narzędzi do pisania kodu w językach HTML, XHTML, CSS, JavaScript i innych, a także liczne mechanizmy i skróty przydatne przy budowaniu stron WWW.

- **Dreamweaver** (Mac i Windows, *http://www.adobe.com/products/dreamweaver.html*, roczny abonament 294,97 euro) to graficzny edytor stron WWW. Narzędzie to pozwala zobaczyć, jak strona będzie wyglądać w przeglądarce. Dreamweaver obejmuje rozbudowany edytor tekstu do tworzenia programów w języku JavaScript oraz doskonałe narzędzia do budowania stylów CSS i zarządzania nimi. Dokładne informacje o sposobach korzystania z tego użytecznego programu można znaleźć w książce *Dreamweaver CC: The Missing Manual*.

O książce

JavaScript, w odróżnieniu od aplikacji Microsoft Word, Dreamweaver i innych, nie jest pojedynczym produktem rozwijanym przez jedną firmę. Nie istnieje dział pomocy technicznej, który opracowuje zrozumiałe podręczniki dla przeciętnych programistów stron WWW. Choć można znaleźć wiele informacji w witrynach *Mozilla.org* (*https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference*), *Ecmascript.org* (*http://www.ecmascript.org/docs.php*) i innych, nie ma jednego, centralnego źródła wiedzy na temat języka JavaScript.

Ponieważ nie istnieje oficjalny podręcznik języka JavaScript, programiści poznający ten język często nie wiedzą, od czego zacząć. Bardziej zaawansowane techniki tego języka mogą sprawiać trudności nawet doświadczonym programistom. Książka ta ma pełnić funkcję nieoficjalnego podręcznika, który powinien być udostępniany z samym językiem. Znajdziesz tu podane krok po kroku instrukcje pokazujące, jak używać języka JavaScript do tworzenia wysoce interaktywnych stron WWW.

Dobrą dokumentację jQuery można znaleźć na stronie *http://api.jquery.com/*. Jednak została ona napisana przez programistów i jest przeznaczona dla programistów; dlatego też zamieszczone w niej opisy są krótkie i mają charakter techniczny. I choć korzystanie z jQuery jest zazwyczaj łatwiejsze niż tworzenie standardowego kodu w języku JavaScript, ta książka i tak nauczy Cię podstawowych zasad i technik stosowania biblioteki jQuery, abyś, używając jej na własnych stronach, podążał właściwą drogą.

Książka *JavaScript i jQuery. Nieoficjalny podręcznik* jest przeznaczona dla osób, które mają już pewne doświadczenie w tworzeniu stron WWW. Aby w pełni wykorzystać przedstawione tu informacje, powinieneś znać HTML i CSS, ponieważ działanie języka JavaScript jest często zależne od tych technologii. Rozdziały wprowadzające są napisane dla zaawansowanych początkujących i średnio zaawansowanych użytkowników komputerów. Jeśli nie masz doświadczenia w tworzeniu stron WWW, w specjalnych ramkach zatytułowanych „Wiedza w pigułce" znajdziesz informacje potrzebne do zrozumienia omawianych zagadnień. Z kolei doświadczeni programiści stron WWW powinni zwrócić szczególną uwagę na ramki z serii „Poradnia dla zaawansowanych", które zawierają dodatkowe techniczne wskazówki, sztuczki i skróty przeznaczone właśnie dla nich.

Uwaga: W tej książce znajdują się odwołania do *innych* pozycji, poświęconych zagadnieniom, które są zbyt zaawansowane lub niezwiązane bezpośrednio z tematem, aby umieszczać je w podręczniku do języka JavaScript. Czasem polecane tytuły to publikacje wydawnictwa O'Reilly Media, które odpowiada też za serię „Missing Manual" (w Polsce książki z serii „Nieoficjalny podręcznik" wydaje wydawnictwo Helion), jednak nie zawsze tak jest. Jeśli inne wydawnictwo opublikowało doskonałą pozycję, polecam właśnie ją.

Podejście do języka JavaScript stosowane w tej książce

JavaScript to prawdziwy język programowania. Działa inaczej niż języki HTML i CSS oraz ma własny zestaw często skomplikowanych reguł. Projektanci stron WWW mają czasem problemy z przestawieniem się na sposób myślenia specyficzny dla programistów, a żadna *pojedyncza* książka nie zawiera wszystkich informacji na temat języka JavaScript.

Książka *JavaScript i jQuery. Nieoficjalny podręcznik* nie ma zmienić Cię w następnego doskonałego programistę. Jej zadaniem jest zapoznanie projektantów stron WWW ze szczegółami języka JavaScript, a następnie przejście do prezentacji biblioteki jQuery, która umożliwia wbudowanie przydatnych interaktywnych funkcji w witrynę WWW w jak najszybszy i najłatwiejszy sposób.

W tej książce poznasz podstawy języka JavaScript i programowania, jednak nie umożliwiają one tworzenia fascynujących stron WWW. Na 650 stronach nie sposób zmieścić wszystkich informacji o języku JavaScript, które są potrzebne do budowania zaawansowanych, interaktywnych stron. Zamiast tego znaczna część tej książki koncentruje się na przedstawieniu bardzo popularnej biblioteki języka JavaScript — jQuery — która, jak się sam niebawem przekonasz, uwolni Cię od żmudnych i czasochłonnych szczegółów tworzenia programów JavaScript działających w wielu różnych przeglądarkach.

Najpierw poznasz podstawy języka JavaScript, a następnie przejdziesz bezpośrednio do zaawansowanych mechanizmów interaktywnych z pewną — w porządku, *znaczną* — pomocą w postaci biblioteki jQuery. To prawda, można zbudować dom, samodzielnie ścinając i obrabiając drzewo, tworząc okna, drzwi i framugi, produkując kafelki i tak dalej. Podejście „zrób to sam" jest powszechnie stosowane w wielu książkach na temat języka JavaScript. Jednak kto ma czas na pracę w takim stylu? Podejście stosowane w tej książce bardziej przypomina budowanie domu z gotowych elementów i składanie ich z wykorzystaniem podstawowej wiedzy. Efekt końcowy to piękny i funkcjonalny budynek postawiony dużo szybciej niż przy zastosowaniu metody wymagającej opanowania wszystkich etapów budowy.

Struktura książki

Książka *JavaScript i jQuery. Nieoficjalny podręcznik* składa się z pięciu części, a każda z nich zawiera kilka rozdziałów.

- Część I, „Wprowadzenie do języka JavaScript", obejmuje podstawy. Poznasz tu „cegiełki" języka JavaScript, a także znajdziesz ogólne wskazówki na temat programowania. Dowiesz się, jak dodawać skrypty do stron WWW, jak przechowywać informacje i manipulować nimi, a także jak wzbogacić program o możliwość reagowania na różne sytuacje. Zobaczysz, jak komunikować się z oknem przeglądarki, zapisywać i wczytywać pliki cookie, reagować na różne zdarzenia (na przykład kliknięcie myszą lub przesłanie formularza) i modyfikować kod HTML stron WWW.

- Część II, „Wprowadzenie do biblioteki jQuery", zawiera podstawowe informacje o jQuery — najpopularniejszej bibliotece języka JavaScript. Zdobędziesz tu podstawowe informacje na temat tego zadziwiającego narzędzia programistycznego, które sprawią, że staniesz się bardziej efektywnym i zdolnym programistą. Dowiesz się, jak pobierać elementy stron i operować na nich, dodawać do nich elementy interaktywności poprzez zapewnienie możliwości reakcji na poczynania użytkownika oraz jak dodawać atrakcyjne efekty wizualne i animacje.
- Część III, „Wprowadzenie do biblioteki jQuery UI", zawiera informacje o siostrzanym projekcie jQuery — bibliotece jQuery UI. Jest to biblioteka języka JavaScript, która udostępnia wiele widżetów i efektów. Ułatwia ona tworzenie i dodawanie do stron popularnych elementów interfejsu użytkownika, takich jak zestawy kart, okna dialogowe, akordeony, rozwijane menu i tak dalej. Biblioteka jQuery UI pomoże Ci opracować w swojej następnej aplikacji sieciowej stylowy i spójny interfejs użytkownika.
- Część IV, „Zaawansowane zastosowania jQuery i języka JavaScript", jest poświęcona zaawansowanym zastosowaniom jQuery i języka JavaScript. Szczególnie rozdział 13. jest poświęcony technologii, która już sama sprawia, że JavaScript jest najbardziej olśniewającym językiem do tworzenia aplikacji sieciowych. W tym rozdziale dowiesz się, jak używać JavaScriptu do komunikowania się z serwerem WWW, tak aby strony mogły pobierać informacje z serwera i na ich podstawie aktualizować swoją treść — a wszystko bez konieczności wczytywania następnej strony. Z kolei w rozdziale 14. prześledzisz krok po kroku proces tworzenia aplikacji do zarządzania listą zadań, w której skorzystano z bibliotek jQuery i jQuery UI.
- Część V, „Wskazówki, sztuczki i rozwiązywanie problemów", to zakończenie zagadnień podstawowych i przejście do bardziej zaawansowanych. W tej części książki dowiesz się, jak efektywnie korzystać z biblioteki jQuery oraz jak posługiwać się jej bardziej zaawansowanymi funkcjami. Dowiesz się tu także, co można zrobić, gdy program nie będzie działał zgodnie z oczekiwaniami lub — co gorsza — w ogóle nie będzie działał. Poznasz błędy często popełniane przez początkujących programistów, a także techniki wykrywania i naprawiania usterek w programach.

Dodatek znajdujący się na końcu książki zawiera szczegółową listę materiałów, które pomogą Ci w dalszym poznawaniu języka JavaScript.

Podstawy

Aby korzystać z tej książki, a nawet z samego komputera, trzeba znać pewne podstawy. Zakładam, że dobrze rozumiesz poniższe pojęcia i zagadnienia.

- **Klikanie**. W książce pojawiają się trzy rodzaje instrukcji wymagające użycia myszy lub touchpada. *Kliknięcie* oznacza najechanie wskaźnikiem myszy na element widoczny na ekranie i wciśnięcie oraz zwolnienie przycisku myszy (lub

touchpada w laptopie) bez poruszania wskaźnikiem. *Kliknięcie prawym przyciskiem myszy* oznacza wykonanie tej operacji przy użyciu prawego przycisku. *Kliknięcie dwukrotne* polega oczywiście na szybkim dwukrotnym kliknięciu myszą bez poruszania jej wskaźnika. *Przeciąganie* wymaga przesunięcia kursora **przy** wciśniętym przycisku myszy.

Wskazówka: Jeśli używasz komputera Mac i nie masz myszy z prawym przyciskiem, możesz uzyskać taki sam efekt przez przytrzymanie klawisza *Control* w czasie kliknięcia.

Polecenie *kliknięcia z przyciskiem* ⌘ (komputery Mac) lub *z przyciskiem Ctrl* (komputery PC) wymaga kliknięcia myszą przy wciśniętym odpowiednim klawiszu, znajdującym się obok klawisza spacji.

- **Menu**. Na *menu* składają się słowa w górnej części ekranu lub okna: *Plik*, *Edycja* i tak dalej. Kliknięcie jednego z nich powoduje wyświetlenie listy poleceń w rozwiniętym okienku.
- **Skróty klawiaturowe**. Jeśli szybko wpisujesz kod w przypływie twórczej energii, odrywanie dłoni od klawiatury i używanie myszy do wyboru polecenia z menu (na przykład w celu włączenia pogrubienia) bywa rozpraszające. Dlatego wielu doświadczonych programistów woli uruchamiać instrukcje za pomocą kombinacji klawiszy klawiatury. Na przykład w przeglądarce Firefox można wcisnąć kombinację *Ctrl*-+ (Windows) lub ⌘-+ (Mac), aby powiększyć tekst strony i poprawić jego czytelność. Jeśli natrafisz na polecenia typu „wciśnij kombinację ⌘-*B*", najpierw wciśnij klawisz ⌘ i — przytrzymując go — wybierz literę *B*. Następnie możesz zwolnić oba klawisze.
- **Podstawowe funkcje systemu operacyjnego**. Zakładam też, że wiesz, jak uruchamiać programy, poruszać się w sieci WWW i pobierać pliki. Powinieneś umieć korzystać z menu *Start* (Windows) i *Dock* lub (Macintosh), a także z menu *Panel sterowania* (Windows) i *Preferencje systemowe* (Mac OS X).

Jeśli opanowałeś już te umiejętności, możesz rozpocząć przygodę z książką *JavaScript i jQuery. Nieoficjalny podręcznik*.

O/tych/ukośnikach

W tej książce, a także w innych pozycjach z serii „Nieoficjalny podręcznik", znajdziesz zdania typu: „Otwórz katalog *System/Biblioteka/Czcionki*". Jest to skrótowy zapis znacznie dłuższego polecenia, które nakazuje otworzyć kolejno trzy katalogi: „Znajdź na dysku twardym katalog *System* i otwórz go. Następnie znajdź w tym katalogu folder *Biblioteka*. Kliknij go dwukrotnie, aby go otworzyć. W **tym** katalogu znajduje się folder o nazwie *Czcionki*. Kliknij go dwukrotnie, aby go otworzyć".

Podobny skrótowy zapis pomaga uprościć opis wyboru poleceń menu, co ilustruje rysunek W.2.

Rysunek W.2. *Zapis skrótowy pomaga uprościć wskazywanie poleceń menu. Na przykład Widok/Powiększenie/Powiększ to zwięzła postać instrukcji: „Wybierz opcję Powiększenie z menu Widok. Pojawi się menu podrzędne, w którym należy kliknąć opcję Powiększ"*

Zasoby internetowe

Książka ma Ci pomóc w szybszym i bardziej profesjonalnym tworzeniu stron WWW. Naturalne jest więc, że połowa wartości tej pozycji związana jest z siecią WWW. W internecie możesz znaleźć kody przykładów do książki, które ułatwią Ci zdobywanie doświadczenia. Możesz także napisać do zespołu zajmującego się tworzeniem serii książek „Nieoficjalny podręcznik" i przekazać informacje o tym, co Ci się w tej książce podoba (lub co doprowadza Cię do szału).

Przykłady

W czasie czytania książki natrafisz na liczne *przykłady* — szczegółowo opisane strony, które możesz przygotować samodzielnie z wykorzystaniem dostępnych materiałów (grafiki i częściowo przygotowanych stron). Materiały te możesz pobrać ze strony poświęconej książce w witrynie wydawnictwa Helion (*helion.pl*, bądź bezpośrednio spod adresu *ftp://ftp.helion.pl/przyklady/jsjqn3.zip*). Prawdopodobnie nie nauczysz się zbyt wiele, jeśli tylko przeczytasz tekst, leżąc wygodnie w hamaku. Jeśli jednak prześledzisz przykłady przy komputerze, odkryjesz, że są doskonałym sposobem na zrozumienie, jak profesjonalni projektanci tworzą strony WWW.

W poszczególnych przykładach znajdziesz też adresy URL prowadzące do angielskich wersji gotowych stron, z którymi będziesz mógł porównać efekty swej pracy. Oznacza to, że będziesz mógł zobaczyć nie tylko efekt działania kodu JavaScript na kartach książki, ale też działające strony WWW w internecie.

Opinie i uwagi

Masz jakieś pytania? Potrzebujesz więcej informacji? Chciałbyś napisać recenzję książki? Na stronie autorów możesz znaleźć eksperckie odpowiedzi na pytania, które przyjdą Ci do głowy podczas lektury książki, podzielić się swoimi uwagami na jej temat oraz znaleźć społeczność osób, które, podobnie jak Ty, interesują się językiem JavaScript i biblioteką jQuery. Abyś mógł wyrazić swoją opinię, wejdź na stronę *http://helion.pl/user/opinie/jsjqn3*.

Errata

Staramy się, by książka była możliwie najbardziej aktualna i dokładna, dlatego też podczas dodrukowywania kolejnych egzemplarzy będziemy uwzględniali w jej tekście wszystkie potwierdzone błędy i sugestie. Informacje o wszelkich takich zmianach umieszczamy także na stronie poświęconej tej książce, abyś, jeśli tylko zechcesz, mógł je zanotować we własnym egzemplarzu. Aby zgłosić poprawkę lub przejrzeć listę już zgłoszonych błędów, zajrzyj na stronę książki.

CZĘŚĆ

I

Wprowadzenie do języka JavaScript

1

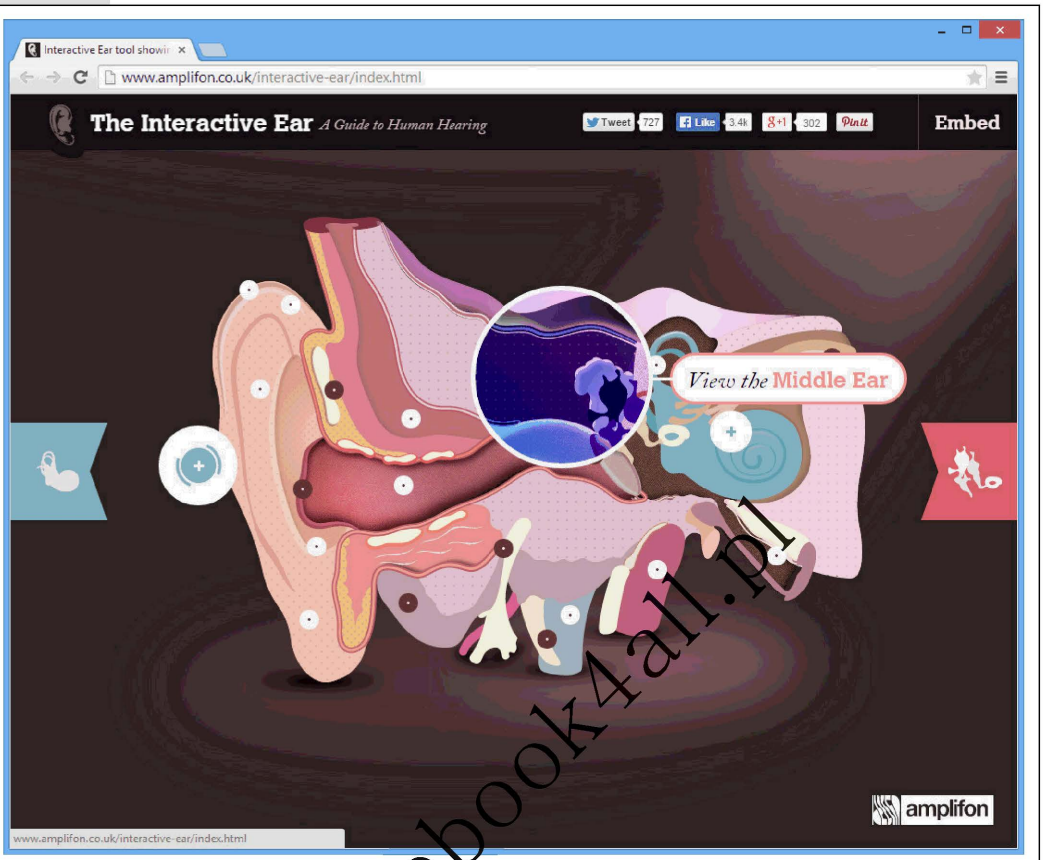
ROZDZIAŁ

Pierwszy program w języku JavaScript

Sam język HTML nie ma dużych możliwości. Nie obsługuje operacji matematycznych, nie wykrywa, czy użytkownik prawidłowo wypełnił formularz, i nie potrafi podejmować decyzji na podstawie działań internautów. HTML umożliwia czytanie artykułów, oglądanie obrazków i klikanie odnośników do innych stron WWW z tekstem i grafiką. Aby wzbogacić możliwości stron WWW o uwzględnianie zachowań użytkowników, należy użyć języka JavaScript.

JavaScript umożliwia reagowanie stron WWW na działania internautów. Przy jego użyciu można budować „inteligentne" formularze, które informują użytkowników o braku wymaganych informacji. Można też sprawić, że elementy będą się pojawiać, znikać i poruszać na stronie (patrz rysunek 1.1). Można nawet zaktualizować zawartość strony za pomocą informacji pobranych z serwera sieciowego bez konieczności wczytywania całego nowego dokumentu. JavaScript umożliwia tworzenie bardziej angażujących, efektywnych i użytecznych witryn.

Uwaga: W wersji 5. do języka HTML dodano pewne „inteligentne" rozwiązania, takie jak podstawowe mechanizmy weryfikacji danych wpisywanych w formularzach. Ponieważ jednak nie wszystkie przeglądarki obsługują te możliwości (jak również dlatego, że przy użyciu zwyczajnych formularzy i języka JavaScript można zrobić znacznie więcej), konieczne jest korzystanie z języka JavaScript podczas tworzenia najlepszych, najbardziej przyjaznych dla użytkownika i w pełni interaktywnych formularzy. Więcej informacji na temat języka HTML5 oraz formularzy internetowych można znaleźć w książce Bena Henicka *HTML5 Forms* (O'Reilly) oraz Gaurava Gupty *Mastering HTML5 Forms* (Packt Piublishing).



Rysunek 1.1. *Witryna The Interactive Ear[1] (http://www.amplifon.co.uk/interactive-ear/) jest interaktywnym przewodnikiem po ludzkim uchu; umożliwia poznanie jego działania i elementów. Nowe informacje są prezentowane w odpowiedzi na ruchy wskaźnika myszy oraz kliknięcia. Zastosowanie JavaScriptu pozwala na tworzenie własnych efektów interaktywnych*

Wprowadzenie do programowania

Wielu osobom pojęcie „programowanie komputerowe" przywodzi na myśl obraz niezwykle inteligentnych „mózgowców", pochylonych nad klawiaturami i przez godziny wpisujących niezrozumiałe znaczki. To prawda, czasem tak to wygląda. Programowanie może sprawiać wrażenie zaawansowanej dziedziny, której opanowanie wykracza poza możliwości przeciętnego śmiertelnika. Jednak wiele zagadnień z tego obszaru jest stosunkowo łatwych do zrozumienia, a wśród języków programowania JavaScript jest jednym z bardziej przyjaznych dla osób, które nie mają jeszcze żadnych doświadczeń z programowaniem.

[1] Interaktywne ucho — *przyp. tłum.*

JavaScript jest jednak bardziej skomplikowany od języków HTML i CSS, a programowanie to zagadnienie często obce projektantom stron WWW. Dlatego jednym z zadań tej książki jest pomóc Ci nauczyć się myśleć tak, jak robią to programiści. Znajdziesz tu omówienie podstawowych technik, przydatnych przy pisaniu kodu w języku JavaScript i ActionScript, a nawet przy tworzeniu tradycyjnych programów w języku C++. Co jednak ważniejsze, zobaczysz, jak podchodzić do zadań programistycznych, dzięki czemu jeszcze przed dodaniem kodu JavaScript do strony będziesz dokładnie wiedział, jakich efektów oczekiwać.

Wielu projektantów stron WWW zniechęca się na widok dziwnych symboli i słów używanych w języku JavaScript. Standardowy skrypt w tym języku jest pełen symboli (`{}[];,()!=`) i nieznanych słów (`var`, `null`, `else if`). Taki kod przypomina tekst w języku obcym, a pod wieloma względami poznawanie nowych języków programowania jest podobne do nauki języków naturalnych. Aby skutecznie się komunikować, trzeba opanować zbiór nowych słów i znaków przestankowych oraz zrozumieć, jak je łączyć.

Każdy język programowania ma własny zestaw słów kluczowych i znaków oraz specyficzne zasady ich łączenia. Te elementy to *składnia* języka. Trzeba zapamiętać pewne zwroty i reguły (lub przynajmniej mieć pod ręką tę książkę). Przy posługiwaniu się językiem obcym często położenie akcentu na złej sylabie powoduje, że słowo staje się niezrozumiałe. Podobnie prosta literówka, a nawet brak znaku przestankowego mogą uniemożliwić działanie programu lub spowodować błąd w przeglądarce. Będziesz prawdopodobnie popełniał wiele pomyłek tego rodzaju, co jest normalne przy nauce programowania.

Na początku programowanie w języku JavaScript może Ci się wydać frustrujące. Dużo czasu zajmie wykrywanie błędów popełnionych w czasie wpisywania skryptów. Ponadto niektóre zagadnienia są początkowo trudne do zrozumienia. Nie martw się jednak. Jeśli próbowałeś już nauczyć się JavaScriptu i zrezygnowałeś, ponieważ

CZĘSTO ZADAWANE PYTANIA

Języki kompilowane i skryptowe

JavaScript jest nazywany językiem skryptowym, podobnie jak języki PHP i ColdFusion. Czym są języki skryptowe?

Większość programów działających w komputerach jest napisana w językach *kompilowanych*. Kompilacja to proces polegający na generowaniu pliku, który będzie mógł być wykonywany na komputerze. Proces ten to przekształcenie kodu napisanego przez programistę na instrukcje zrozumiałe dla komputera. Skompilowany program można uruchomić na komputerze, a ponieważ kod przekształcono na postać zrozumiałą dla maszyny, taka aplikacja działa szybciej od programów napisanych w języku skryptowym. Niestety, kompilacja kodu to czasochłonny proces. Należy napisać program, skompilować go, a następnie przetestować. Jeśli wystąpią problemy, cały proces trzeba zacząć od początku.

Języki skryptowe są kompilowane dopiero przy ich odczycie przez *interpreter* (program, który przekształca skrypt na formę zrozumiałą dla komputera). Interpreter języka JavaScript jest wbudowany w przeglądarki. Dlatego kiedy przeglądarka wczytuje stronę z programem w języku JavaScript, przekształca go na postać odpowiednią dla maszyny. Ponieważ programy w językach skryptowych wymagają przekształcania przy każdym ich uruchomieniu, działają wolniej od kodu napisanego w językach kompilowanych. Języki skryptowe to doskonałe rozwiązanie dla projektantów stron WWW. Skrypty są zwykle dużo mniejsze i prostsze od tradycyjnych programów, dlatego niższa szybkość nie jest bardzo istotna. Ponadto z uwagi na brak etapu kompilacji tworzenie i testowanie programów w językach skryptowych trwa dużo krócej.

wydał Ci się zbyt trudny, dzięki tej książce pokonasz przeszkody, które często zniechęcają początkujących programistów. (Jeśli natomiast umiesz już programować, poznasz wyjątkowe cechy JavaScriptu i specyficzne zagadnienia związane z tworzeniem programów działających w przeglądarkach).

Co więcej, książka ta nie jest poświęcona wyłącznie językowi JavaScript — zajmujemy się w niej także biblioteką jQuery — najpopularniejszą na świecie biblioteką napisaną w JavaScripcie. Sprawia ona, że pisanie złożonych programów w tym języku staje się prostsze… *znacznie* prostsze. A zatem, dysponując pewną znajomością języka JavaScript i korzystając z biblioteki jQuery, będziesz mógł bardzo szybko tworzyć wyrafinowane, interaktywne witryny WWW.

Czym jest program komputerowy?

Kiedy dodajesz kod JavaScript do strony, tworzysz program komputerowy. To prawda, większość programów w tym języku jest dużo prostsza od aplikacji używanych do czytania e-maili, retuszowania zdjęć i tworzenia stron WWW. Jednak choć programy w języku JavaScript (nazywane też *skryptami*) są prostsze i krótsze, mają wiele cech ich bardziej złożonych odpowiedników.

Każdy program komputerowy to zestaw instrukcji wykonywanych w określonej kolejności. Jeśli chcesz wyświetlić komunikat powitalny z imieniem internauty, na przykład „Witaj, Janie!", musisz wykonać kilka operacji:

1. **Poprosić użytkownika o podanie imienia.**
2. **Pobrać odpowiedź.**
3. **Wyświetlić komunikat na stronie.**

Możliwe, że nie chcesz wyświetlać komunikatów powitalnych, jednak punkty te ilustrują podstawowy proces programowania: określanie oczekiwanych efektów i podział zadania na etapy. Przy tworzeniu każdego programu w języku JavaScript trzeba określić kroki potrzebne do wykonania zadania. Następnie można przystąpić do pisania programu, czyli przekształcania pomysłów na *kod* — słowa i znaki, które sprawią, że przeglądarka wykona pożądane operacje.

Jak dodać kod JavaScript do strony?

Przeglądarki rozumieją kod HTML i CSS oraz przekształcają go na strony widoczne na ekranie. Część przeglądarki obsługująca języki HTML i CSS to *silnik zarządzania układem* (*renderujący*). Większość przeglądarek ma też *interpreter języka JavaScript*. Ten mechanizm przetwarza kod JavaScript i wykonuje operacje opisane w programie. Przeglądarki obsługują domyślnie kod HTML, dlatego też trzeba poinformować je o wystąpieniu kodu JavaScript za pomocą znacznika `<script>`.

Znacznik `<script>` to zwykły kod HTML. Działa jak przełącznik, który informuje: „Przeglądarko, oto nadchodzi kod JavaScript. Nie wiesz, co z nim zrobić, dlatego przekaż go interpreterowi języka JavaScript". Kiedy przeglądarka natrafi na znacznik zamykający `</script>`, zakończy przetwarzanie programu JavaScript i wróci do standardowego trybu działania.

Często znaczniki <script> umieszczane są w sekcji <head> strony:

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN" http://www.w3.org/TR/
html4/strict.dtd">
<html>
<head>
<title>Strona WWW</title>
<script type="text/javascript">
</script>
</head>
```

WIEDZA W PIGUŁCE

Skrypty po stronie klienta i serwera

JavaScript początkowo powstał jako język działający po stronie klienta. Pisany w nim kod jest przekazywany do przeglądarki przez serwer WWW. Osoby przeglądające naszą witrynę pobierają tworzące ją strony WWW wraz z używanym na nich kodem JavaScript, a następnie ich przeglądarki — czyli klienty — przetwarzają pobrany kod i realizują wszystkie magiczne sztuczki.

JavaScript to język działający *po stronie klienta*, co po polsku oznacza, że skrypty funkcjonują w przeglądarce. Drugi typ języków programowania używanych w sieci działa *po stronie serwera*. Do tej grupy należą: PHP, .NET, ASP, ColdFusion, Ruby on Rails i inne technologie serwerowe. Programy działające na serwerze mogą wykonywać zaawansowane operacje, na przykład korzystać z baz danych, przetwarzać operacje z użyciem kart kredytowych i rozsyłać e-maile po całym świecie. Problemem jest to, że przeglądarka musi najpierw przesłać żądanie na serwer, co zmusza odwiedzających do oczekiwania na pobranie strony z nowymi informacjami.

Języki działające po stronie klienta mogą reagować natychmiast i zmieniać wygląd dokumentu bez konieczności pobierania nowej strony. Treść można wyświetlać i ukrywać, przenosić po ekranie i aktualizować w odpowiedzi na działania użytkownika. Pozwala to tworzyć witryny przypominające bardziej tradycyjne programy niż statyczne strony WWW. JavaScript to nie jedyna technologia działająca w ten sposób. Do zwiększania możliwości stron można użyć także wtyczek. Aplety języka Java to jedno z takich rozwiązań. Są to małe, napisane w Javie programy, które działają w przeglądarce. Zwykle ich uruchamianie trwa długo i często powoduje awarie przeglądarek.

Flash to następna technologia oparta na wtyczkach. Pozwala dodawać złożone animacje, filmy, dźwięki oraz interakcję. Czasem trudno ocenić, czy interaktywna strona jest napisana w języku JavaScript, czy we Flashu. Na przykład witrynę Google Maps można utworzyć także we Flashu, a serwis Yahoo Maps był pierwotnie aplikacją flashową (dopiero później zastąpiono go wersją opartą na języku JavaScript). Jest jednak szybki sposób na sprawdzenie użytej technologii. Należy kliknąć stronę (samą mapę w witrynie Google Maps) prawym przyciskiem myszy. Jeśli użyto Flasha, pojawi się menu wyskakujące z opcją *About Adobe Flash Player...*

W części IV znajdziesz omówienie Ajaksa, który łączy stronę kliencką z serwerową. W technologii tej wykorzystuje się język JavaScript do komunikacji z serwerem, pobierania z niego informacji i aktualizowania stron bez konieczności ich odświeżania. W witrynie Google Maps metoda ta umożliwia przesuwanie mapy bez wczytywania nowych stron.

Obecnie JavaScript znajduje także wiele zastosowań poza przeglądarkami WWW. Przykładowo w serwerze WWW node.js (*http://nodejs.org/*) użyto właśnie języka JavaScript do nawiązywania połączeń z bazami danych, dostępu do systemu plików na serwerze oraz wykonywania wielu innych operacji. W tej książce nie będziemy zajmowali się tymi aspektami wykorzystania języka JavaScript, jednak osoby zainteresowane Node.js mogą obejrzeć jego krótką prezentację, dostępną w serwisie YouTube na stronie *https://www.youtube.com/watch?v=hKQ--r2DGJjUQ*.

Co więcej, w kilku stosunkowo nowych bazach danych użyto JavaScriptu jako języka służącego do tworzenia, pobierania i aktualizowania rekordów. Popularnymi przykładami takich baz danych są MongoDB oraz CouchDB. Można nawet spotkać się z terminem „pełen pakiet JavaScriptu" oznaczającym wykorzystanie tego języka w przeglądarce, na serwerze WWW oraz do obsługi baz danych. Jeden język, by zarządzać wszystkim!

Atrybut `type` znacznika `<script>` określa format i rodzaj danego skryptu. Kod `type= text/javascript` oznacza, że skrypt jest napisany jako zwykły tekst (podobnie jak kod HTML) w języku JavaScript.

Jeśli użyjesz języka HTML5, Twoje zadanie będzie jeszcze łatwiejsze. Możesz całkowicie pominąć atrybut `type`:

```
<!doctype html>
<html>
<head>
<meta charset="UTF-8">
<title>Strona WWW</title>
<script>

</script>
</head>
```

Przeglądarki pozwalają na pominięcie atrybutu `type` także wtedy, kiedy strony WWW tworzone są w językach HTML 4.01 oraz XHTML 1.0 — skrypty będą działały prawidłowo, choć gdy zabraknie atrybutu, sama strona nie przejdzie procesu walidacji (patrz ramka na stronie 23). W książce używamy deklaracji typu dokumentu charakterystycznej dla języka HTML5, jednak przedstawiane w niej skrypty JavaScript będą działać także na stronach WWW pisanych w języku HTML 4.01 oraz XHTML 1.0.

Następnie między znacznikami `<script>` otwierającym a zamykającym wpisz kod JavaScript:

```
<!doctype html>
<html>
<head>
<meta charset="UTF-8">
<title>Strona WWW</title>
<script>
  alert('Witaj, świecie!');
</script>
</head>
```

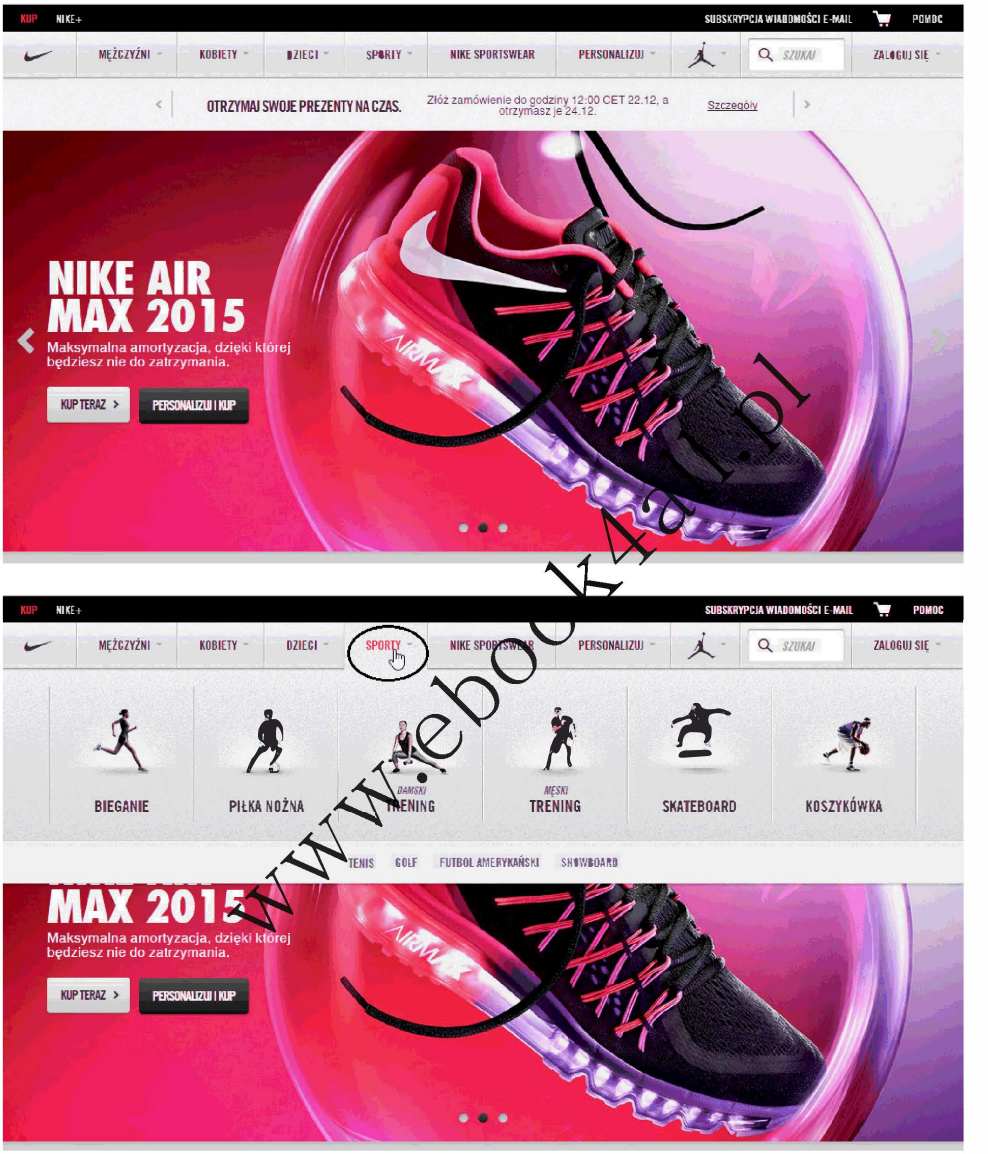
Wkrótce dowiesz się, jak działa ten kod. Na razie przyjrzyj się znacznikom `<script>`. Aby dodać skrypt do strony, najpierw wstaw te znaczniki. Zwykle warto je zapisać w sekcji `<head>`, co pozwala uporządkować kod JavaScript w jednym obszarze strony.

Jednak w pełni dopuszczalne jest umieszczanie znaczników `<script>` w dowolnym miejscu strony. W dalszej części rozdziału poznasz polecenie języka JavaScript, które umożliwia zapisywanie informacji bezpośrednio na stronie WWW. Przy stosowaniu tego polecenia należy wstawić znaczniki `<script>` w ciele strony, tam gdzie skrypt ma wyświetlić komunikat. Często zdarza się nawet, że kod JavaScript jest umieszczany za zamykającym znacznikiem `</body>` — takie rozwiązanie daje pewność, że przed wykonaniem skryptu strona zostanie w całości wczytana i wyświetlona.

Zewnętrzne pliki JavaScript

Znacznik `<script>` w postaci użytej w poprzednim punkcie pozwala dodać kod JavaScript do jednej strony. Jednak wiele skryptów działa na wszystkich stronach witryny. Może to być na przykład panel zawierający dodatkowe opcje nawigacyjne, który pojawia się na stronie po umieszczeniu wskaźnika myszy na wybranym

elemencie (patrz rysunek 1.2). Ten sam wymyślny pasek nawigacyjny powinien znaleźć się na każdej stronie witryny, jednak rozwiązanie polegające na kopiowaniu i wklejaniu kodu JavaScript we wszystkich plikach ma wiele wad.



Rysunek 1.2. *W witrynie Nike.com użyto języka JavaScript w celu utworzenia atrakcyjnych prezentacji swoich produktów. Na stronie głównej (pokazanej u góry) widoczny jest pasek przycisków nawigacyjnych, umieszczonych na samej górze strony — Mężczyźni, Kobiety, Dzieci i tak dalej — które po wskazaniu myszą powodują wyświetlenie panelu z dodatkowymi opcjami nawigacyjnymi. Wskazanie na przykład przycisku Sporty (zakreślonego u dołu) powoduje wyświetlenie panelu z listą różnych sportów, do uprawiania których są przeznaczone produkty firmy Nike*

Po pierwsze, kopiowanie i wklejanie tego samego kodu wymaga dużo pracy, zwłaszcza jeśli witryna składa się z setek stron. Po drugie, jeśli zechcesz zmienić lub wzbogacić skrypt, będziesz musiał znaleźć każdą stronę, na której go użyłeś, i poprawić kod. Po trzecie, ponieważ cały kod programu JavaScript musi znaleźć się na wszystkich stronach, każda z nich będzie dużo większa, a jej wczytywanie potrwa dłużej.

Lepsze podejście polega na użyciu zewnętrznego pliku JavaScript. Jeśli na stronach korzystasz z zewnętrznych arkuszy CSS, technika ta nie będzie dla Ciebie niczym nowym. Zewnętrzny plik JavaScript to po prostu plik tekstowy o rozszerzeniu *.js*, na przykład *navigation.js*. Powinien on zawierać tylko kod JavaScript, a na stronie można go wskazać za pomocą znacznika `<script>`. Aby na przykład dodać do strony głównej plik JavaScript *navigation.js*, można użyć następującego kodu:

```
<!doctype html>
<html>
<head>
<meta charset="UTF-8">
<title>Strona WWW</title>
<script src="navigation.js"></script>
</head>
```

Atrybut `src` znacznika `<script>` działa podobnie jak atrybut `src` znacznika `<img>` lub atrybut `href` znacznika `<a>`. Wszystkie te atrybuty wskazują plik w tej samej lub innej witrynie (patrz ramka na stronie 45).

Uwaga: Jeśli stosujesz atrybut `src` do wskazania zewnętrznego pliku JavaScript, nie umieszczaj kodu JavaScript w użytych do tego znacznikach `<script>`. Jeżeli chcesz dołączyć zewnętrzny plik, a ponadto dodać do strony niestandardowy kod JavaScript, użyj drugiej pary znaczników `<script>`. Oto przykład:

```
<script src="navigation.js"></script>
<script>
  alert('Witaj, świecie!');
</script>
```

Programiści często dołączają kilka zewnętrznych plików JavaScript do jednej strony. Jeden plik może kontrolować rozwijany pasek nawigacji, a inny — umożliwiać dodanie eleganckiego pokazu slajdów do strony ze zdjęciami. Na stronie z galerią potrzebne są oba programy JavaScript, dlatego należy dołączyć oba pliki.

Ponadto na tej samej stronie można umieścić kilka zewnętrznych plików JavaScript i dodatkowo program w tym języku:

```
<!doctype html>
<html>
<head>
<meta charset="UTF-8">
<title>Strona WWW</title>
<script src="navigation.js"></script>
<script src="slideshow.js"></script>
<script>
  alert('Witaj, świecie!');
</script>
</head>
```

Należy pamiętać o użyciu otwierającego i zamykającego znacznika `<script>` przy każdym zewnętrznym pliku JavaScript. Plik tego typu utworzysz w przykładzie rozpoczynającym się na stronie 49.

WIEDZA W PIGUŁCE

Rodzaje adresów URL

Przy dołączaniu zewnętrznego pliku JavaScript w atrybucie `src` znacznika `<script>` trzeba podać *adres URL* (ang. *Uniform Resource Locator*). Jest to ścieżka do pliku w sieci WWW. Są trzy rodzaje takich ścieżek: *bezwzględne*, podane *względem katalogu głównego* i podane *względem dokumentu*. Wszystkie trzy informują przeglądarkę o lokalizacji danego pliku.

Ścieżka bezwzględna przypomina adres pocztowy — zawiera wszystkie informacje potrzebne do znalezienia pliku przeglądarce uruchomionej w dowolnym komputerze. Taka ścieżka zawiera człon *http://* oraz nazwę domeny, katalogu i pliku, na przykład *http://www.uptospeedguides.com/scripts/site.js*.

Ścieżka podana *względem katalogu głównego* określa lokalizację pliku względem folderu z najwyższego poziomu. Nie obejmuje ona członu *http://* ani nazwy domeny. Rozpoczyna się od ukośnika (*/*), który reprezentuje katalog główny witryny (to w nim znajduje się strona główna). Na przykład ścieżka */scripts/site.js* prowadzi do pliku *site.js*, który znajduje się w katalogu *scripts* w katalogu głównym. Łatwy sposób na utworzenie takiej ścieżki polega na usunięciu członu *http://* i nazwy domeny ze ścieżki bezwzględnej. Adres *http://www.uptospeedguides.com/index.html* w formie ścieżki podanej względem katalogu głównego to */index.html*.

Ścieżka podana *względem dokumentu* prowadzi od danej strony do pliku JavaScript. Jeśli witryna obejmuje kilka poziomów katalogów, do tego samego pliku mogą prowadzić różne ścieżki. Załóżmy, że plik *site.js* znajduje się w katalogu *scripts* w katalogu głównym witryny. Ścieżka podana względem dokumentu na stronie głównej to *scripts/site.js*, jeśli jednak strona znajduje się w katalogu *about*, trzeba użyć ścieżki *../scripts/site.js*. Sekwencja *../* oznacza *wyjście* z katalogu *about*, a fragment *scripts/site.js* powoduje przejście do katalogu *scripts* i pobranie pliku *site.js*.

Oto kilka wskazówek pomocnych przy wyborze typu adresu.

- Jeśli podajesz adres pliku na innym serwerze, *musisz* użyć ścieżki bezwzględnej. Tylko w ten sposób można wskazać inną witrynę.
- Ścieżki podawane względem katalogu głównego są dobre do wskazywania plików JavaScript zapisanych w danej witrynie. Ponieważ adres URL zawsze zaczyna się od katalogu głównego, będzie taki sam dla wszystkich stron, nawet jeśli znajdują się w odmiennych katalogach i podkatalogach witryny. Jednak rozwiązanie to nie działa, jeżeli nie przeglądasz stron za pośrednictwem serwera (albo serwera działającego w internecie, albo serwera testowego na własnym komputerze). Jeśli otwierasz strony bezpośrednio na komputerze za pomocą polecenia *Plik/Otwórz*, przeglądarka nie zdoła zlokalizować, wczytać i uruchomić plików JavaScript, które wskazano za pomocą takiej ścieżki.
- Ścieżki podawane względem dokumentu są najlepsze, kiedy projektujesz witrynę na własnym komputerze bez wykorzystania serwera. Możesz przygotować zewnętrzny plik JavaScript, dołączyć go do strony, a następnie sprawdzić jego działanie w przeglądarce przez otwarcie strony z dysku twardego. Ścieżki tego typu działają poprawnie po przeniesieniu działającej witryny do internetu, jednak przy zmianie lokalizacji strony na serwerze trzeba zmienić również adres pliku JavaScript. W tej książce używane są właśnie takie ścieżki, ponieważ pozwalają uruchamiać i testować przykłady przy użyciu komputerów bez zainstalowanego serwera sieciowego.

Zewnętrzne pliki JavaScript można przechowywać w dowolnej lokalizacji w katalogu głównym witryny lub w jednym z jego podkatalogów. Wielu programistów tworzy w katalogu głównym specjalny folder na takie pliki. Jego standardowe nazwy to *js* (od „JavaScript") i *libs* (od ang. „libraries", czyli biblioteki).

Uwaga: Czasem ważna jest kolejność dołączania zewnętrznych plików JavaScript. W dalszej części książki zobaczysz, że niektóre skrypty korzystają z kodu umieszczonego w plikach zewnętrznych. Zdarza się to często przy stosowaniu bibliotek JavaScript (zawierają one kod JavaScript upraszczający wykonywanie złożonych zadań programistycznych). W przykładzie rozpoczynającym się na stronie 49 zobaczysz, jak używać bibliotek JavaScript.

Pierwszy program w języku JavaScript

Najlepszy sposób na naukę programowania w języku JavaScript polega na… programowaniu. W dalszej części książki znajdziesz praktyczne przykłady, które przeprowadzą Cię krok po kroku przez proces tworzenia programów JavaScript. Będziesz potrzebował edytora tekstu (na stronie 26 znajdziesz listę zalecanych aplikacji), przeglądarki i plików dostępnych na stronie książki w witrynie *helion.pl* (dokładne instrukcje znajdziesz w poniższej uwadze).

Uwaga: Przykłady z tego rozdziału wymagają pobrania plików ze strony poświęconej książce w witrynie *helion.pl*. Kliknij odnośnik *Przykłady na ftp*, aby pobrać potrzebny kod. Jest on zapisany w jednym pliku ZIP.

Użytkownicy systemu Windows powinni pobrać plik ZIP i dwukrotnie go kliknąć, aby otworzyć archiwum. Następnie należy wybrać opcję *Wyodrębnij pliki*, wykonać instrukcje kreatora wyodrębniania i wybrać miejsce na zapisanie plików w komputerze. Użytkownicy systemu Mac muszą tylko kliknąć archiwum dwukrotnie, aby je rozpakować. Po pobraniu i wyodrębnieniu plików w komputerze powinien znaleźć się katalog *Przykłady* z plikami z wszystkich przykładów omówionych w książce.

Pierwszy program jest bardzo prosty, co pozwala łagodnie zapoznać się z językiem JavaScript.

1. **Otwórz w ulubionym edytorze tekstu plik *hello.html*.**

   Plik ten znajduje się w katalogu *R01* w katalogu *Przykłady*, pobranym ze strony poświęconej książce w witrynie *helion.pl*. Jest to bardzo prosta strona HTML z zewnętrznym arkuszem CSS, który zwiększa atrakcyjność wizualną dokumentu.

2. **Kliknij pusty wiersz tuż *przed* zamykającym znacznikiem `</head>` i dodaj poniższy kod:**

   ```
   <script>
   ```

   Jest to kod w języku HTML, a nie JavaScript. Ten wiersz informuje przeglądarkę o tym, że następny fragment to kod JavaScript.

3. **Wciśnij klawisz *Enter*, aby utworzyć nowy pusty wiersz. Wpisz w nim następujący kod:**

   ```
   alert('Witaj, świecie!');
   ```

   Właśnie wpisałeś pierwszy wiersz kodu JavaScript. Funkcja `alert()` wyświetla okno dialogowe z komunikatem podanym w nawiasach. Tu jest to tekst `Witaj, świecie!`. Na razie nie przejmuj się znakami przestankowymi (nawiasami, apostrofami i średnikiem). W następnym rozdziale dowiesz się, do czego służą.

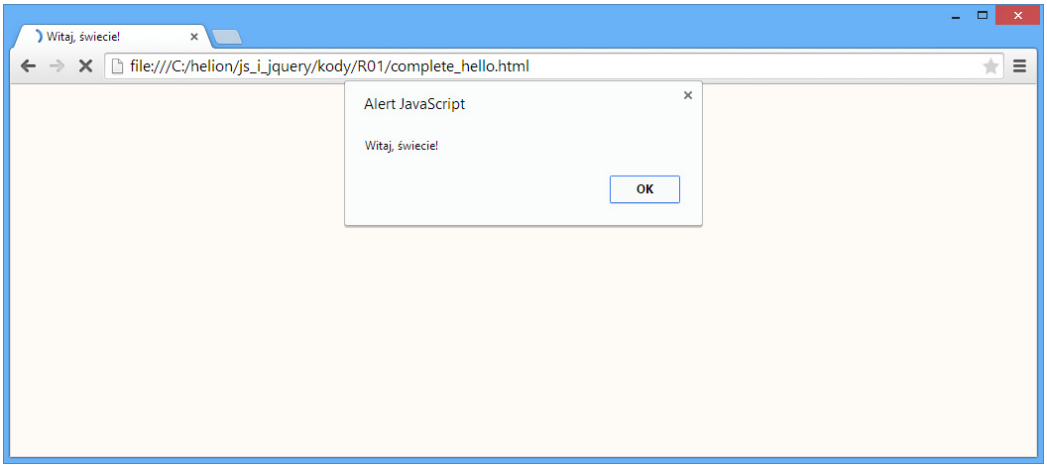
4. **Wciśnij ponownie klawisz *Enter* i dodaj znacznik `</script>`. Kod powinien wyglądać następująco:**

   ```
   <link href="../_css/site.css" rel="stylesheet">
   <script>
   alert('Witaj, świecie!');
   </script>
   </head>
   ```

Tekst, który wpisałeś samodzielnie, jest wyróżniony pogrubieniem. Dwa znaczniki HTML znajdowały się już w pliku. Upewnij się, że kod wygląda dokładnie tak, jak ten w tekście.

5. **Uruchom przeglądarkę i otwórz plik *hello.html*, aby zobaczyć jego podgląd.**

   Pojawi się okno dialogowe języka JavaScript (patrz rysunek 1.3). Zauważ, że w momencie pojawienia się okna strona jest pusta. Jeśli nie widzisz okienka przedstawionego na rysunku 1.3, możliwe, że w kodzie pojawił się błąd. Sprawdź dokładnie kod i zapoznaj się z następną wskazówką.



Rysunek 1.3. *Okno dialogowe języka JavaScript pozwala szybko przykuć uwagę internauty. Wyświetlające je polecenie jest jednym z najprostszych w nauce i użytkowaniu*

Wskazówka: W czasie nauki programowania będziesz zaskoczony, jak często programy JavaScript w ogóle nie działają. Wśród początkujących programistów najczęstszą przyczyną błędów są zwykłe literówki. Zawsze dokładnie sprawdź kod, aby się upewnić, że polecenia (takie jak `alert` w pierwszym skrypcie) są zapisane poprawnie. Ponadto pamiętaj, że znaki przestankowe często pojawiają się w parach (na przykład nawias otwierający i zamykający oraz apostrofy w przykładowym kodzie). Upewnij się, że kod zawiera wszystkie potrzebne symbole początkowe i końcowe.

6. **Kliknij przycisk *OK* okna dialogowego, aby je zamknąć.**

Kiedy okienko zniknie, w oknie przeglądarki pojawi się zawartość strony.

Choć pierwszy program nie jest zbyt złożony (a nawet interesujący), ilustruje ważne zagadnienie: przeglądarka uruchamia programy JavaScript w momencie wczytywania kodu. W przykładzie polecenie `alert()` zostało wykonane *przed* wyświetleniem strony w przeglądarce, ponieważ kod JavaScript znajdował się *przed* kodem HTML zapisanym w znaczniku `<body>`. Będzie to istotne, kiedy zaczniesz pisać programy manipulujące kodem HTML strony (nauczysz się tego w rozdziale 3.).

Uwaga: Niektóre wersje przeglądarki Internet Explorer nie lubią wykonywać programów JavaScript na stronach otwieranych bezpośrednio z dysku twardego; istnieje obawa, że mogą być niebezpieczne. Jeśli zatem będziesz próbował wyświetlać w tej przeglądarce przykładowe pliki dołączone do tej książki, zapewne zobaczysz komunikat informujący o tym, że Internet Explorer zablokował umieszczone w nich skrypty. Aby je wykonać, należy kliknąć przycisk „Zezwól na zablokowaną zawartość".

To dosyć denerwujące zachowanie odnosi się wyłącznie do stron WWW otwieranych z dysku twardego komputera, a nie z serwera WWW. Aby uniknąć konieczności ciągłego klikania powyższego przycisku, warto przeglądać strony przy użyciu dowolnej innej przeglądarki, takiej jak Chrome lub Firefox.

Dodawanie tekstu do stron

Poprzedni skrypt wyświetlał okno dialogowe na środku monitora. Jak jednak za pomocą języka JavaScript wyświetlić komunikat bezpośrednio na stronie? Istnieje na to wiele sposobów, a w dalszej części książki poznasz kilka zaawansowanych rozwiązań. Jednak ten prosty cel można zrealizować także dzięki wbudowanej instrukcji języka JavaScript, której użyjesz w drugim skrypcie.

1. **Otwórz w edytorze plik *hello2.html*.**

   Choć znaczniki `<script>` znajdują się zwykle w sekcji `<head>` strony, można umieszczać je i same skrypty także bezpośrednio w ciele strony.

2. **Bezpośrednio pod kodem `<h1>Zapis w oknie dokumentu </h1>` wpisz następujący fragment:**

   ```
   <script>
   document.write('<p>Witaj, świecie!</p>');
   </script>
   ```

   Funkcja `document.write()`, podobnie jak `alert()`, to polecenie języka JavaScript. Ta instrukcja powoduje dodanie do strony tekstu wpisanego w nawiasach. Tu jest to kod HTML `<p>Witaj, świecie!</p>`, który obejmuje znaczniki akapitu i dwa słowa.

3. **Zapisz stronę i otwórz ją w przeglądarce.**

   Strona otworzy się, a pod niebieskim nagłówkiem pojawi się napis „Witaj, świecie!" (patrz rysunek 1.4).

Uwaga: Wśród pobranych plików znajdziesz też gotową wersję każdego przykładu. Jeśli wpisany kod JavaScript nie działa, porównaj własne rozwiązanie z plikiem ze słowem *kompletny_* w nazwie, zapisanym w tym samym katalogu, w którym znajduje się kod przykładów. Plik *complete_hello2.html* zawiera działającą wersję skryptu dodanego do pliku *hello2.html* i tak dalej.

Po utworzeniu dwóch pierwszych skryptów możesz czuć się nieco zawiedziony możliwościami języka JavaScript… lub tą książką. Nie zniechęcaj się — to jest dopiero sam początek. Ważne, aby najpierw dobrze opanować podstawy. W kolejnych rozdziałach nauczysz się wykonywać za pomocą języka JavaScript bardzo przydatne i skomplikowane zadania. Kilka następnych punktów tego rozdziału daje przedsmak zaawansowanych funkcji, które będziesz umiał dodać do stron po zapoznaniu się z dwiema pierwszymi częściami książki.

Rysunek 1.4. *Świetnie. Ten skrypt nie jest imponujący, ale pokazuje, jak używać języka JavaScript do dodawania treści do stron. Jest to przydatne, kiedy chcesz wyświetlić komunikat (na przykład: „Witaj ponownie w witrynie, Jacku") po wczytaniu strony*

Dołączanie zewnętrznych plików JavaScript

Na stronie 42 dowiedziałeś się, że jeśli ten sam skrypt ma działać na kilku stronach, kod JavaScript zwykle warto umieścić w odrębnym pliku. Następnie można nakazać stronie wczytanie tego pliku i użycie zapisanego w nim kodu. Zewnętrzne pliki JavaScript są przydatne także przy używaniu kodu napisanego przez innych programistów. Dotyczy to przede wszystkim zbiorów kodu nazywanych *bibliotekami*. Zwykle ułatwiają one wykonywanie trudnych zadań. Więcej informacji o bibliotekach JavaScript znajdziesz na stronie 133. Szczególnie dobrze poznasz bibliotekę używaną w tej książce (oraz na bardzo wielu witrynach WWW), czyli jQuery.

Jednak na razie poszerzysz swą wiedzę przez dołączenie do strony zewnętrznego pliku JavaScript i napisanie krótkiego programu, który będzie robił coś niesamowitego.

1. **Otwórz w edytorze plik *slide.html*.**

   Strona ta zawiera prosty kod HTML — kilka znaczników `<div>`, nagłówek oraz parę akapitów tekstu. Już zaraz dodasz do niej prosty efekt wizualny, który sprawi, że cała zawartość powoli stanie się widoczna.

2. **Kliknij pusty wiersz między znacznikiem `<link>` a zamykającym znacznikiem `</head>` w górnej części strony i wpisz następujący kod:**

   ```
   <script src="../_js/jquery.min.js"></script>
   ```

   Ten kod dołącza do dokumentu plik o nazwie *jquery.min.js*, znajdujący się w katalogu *js*. Kiedy przeglądarka wczyta stronę, pobierze także plik *jquery.min.js* i uruchomi zapisany w nim kod.

Następnie trzeba dodać do strony własny kod JavaScript.

Uwaga: Słowo min oznacza, że plik został *zminimalizowany*, czyli specjalnie zmniejszony — poprzez usunięcie wszelkich niepotrzebnych odstępów i zapisanie kodu w bardziej zwartej postaci — by jego pobieranie zajmowało mniej czasu.

3. **Wciśnij klawisz *Enter*, aby dodać nowy wiersz, i wpisz w nim poniższy kod:**

   ```
   <script>
   ```

   Znaczniki HTML zwykle występują w parach. Aby nie zapomnieć o dodaniu znacznika zamykającego, warto dołączyć go bezpośrednio po wpisaniu znacznika otwierającego, a następnie wpisać kod między tymi znacznikami.

4. **Wciśnij dwukrotnie klawisz *Enter*, aby dodać dwa puste wiersze, i wpisz następujący kod:**

   ```
   </script>
   ```

   To kończy blok kodu JavaScript. Teraz dodasz sam kod.

5. **Kliknij pusty wiersz między otwierającym a zamykającym znacznikiem skryptu i dodaj poniższy kod:**

   ```
   $(document).ready(function() {
   ```

   Prawdopodobnie zastanawiasz się, co to oznacza. Szczegółowy opis tego kodu znajdziesz na stronie 141, a na razie zapamiętaj, że fragment ten wykorzystuje kod z pliku *jquery.min.js*. Kod ten gwarantuje, że przeglądarka uruchomi następny wiersz w odpowiednim czasie.

6. **Wciśnij klawisz *Enter*, aby dodać nowy wiersz, i wpisz w nim poniższy kod:**

   ```
   $('header').hide().slideDown(3000);
   ```

   Wyniki wykonania tej instrukcji są magiczne. Sprawia ona, że zawartość strony najpierw znika, a następnie, powoli, wsunie się do góry na stronę (co zajmuje 3 sekundy, czyli 3000 milisekund). W jaki sposób to się dzieje? Cóż, to właśnie próbka magicznych możliwości biblioteki jQuery, która pozwala tworzyć złożone efekty przy użyciu jednego wiersza kodu.

7. **Wciśnij klawisz *Enter* po raz ostatni, a następnie wpisz następujące znaki:**

   ```
   });
   ```

   Ta sekwencja kończy kod JavaScript, podobnie jak zamykający znacznik `</script>` oznacza koniec programu JavaScript. Nie przejmuj się na razie dziwnymi znakami przestankowymi. W dalszej części książki dowiesz się, jak działają. Na razie upewnij się, że dokładnie przepisałeś kod. Jedna literówka może sprawić, że skrypt nie będzie działał.

   Kod, który należy dodać do strony, jest wyróżniony pogrubieniem:

```
<link href="../css/global.css" rel="stylesheet">
<script src="../_js/jquery.min.js"></script>
<script>
$(document).ready(function() {
  $('header').hide().slideDown(3000);
});
</script>
</head>
```

Wskazówka: Aby poprawić czytelność kodu, warto stosować wcięcia. Wiersze kodu JavaScript umieszczone wewnątrz bloku można wcinać, tak samo jak wcinamy znaczniki HTML, by pokazać które z nich znajdują się wewnątrz innych. Przykładowo wiersz kodu dodany w kroku 6. jest umieszczony wewnątrz kodu dodanego w krokach 5. i 7., zatem naciśnięcie klawisza tabulacji lub kilkakrotne naciśnięcie klawisza spacji przed dodaniem kodu poprawią jego czytelność i ułatwią zrozumienie (co widać na przykładzie kodu przedstawionego w kroku 7.).

8. **Zapisz plik HTML i otwórz go w przeglądarce.**

   Powinieneś zobaczyć nagłówek — zsuwający się nagłówek — i akapit tekstu, a u dołu strony stopkę oraz powoli wsuwający się od góry na stronę nagłówek „JavaScript i jQuery. Nieoficjalny podręcznik". Spróbuj wpisać inną liczbę zamiast 3000 (na przykład 250 lub 10000), by przekonać się, jaki to będzie miało wpływ na sposób działania strony.

Uwaga: Jeśli spróbujesz wyświetlić tę stronę w Internet Explorerze i okaże się, że nic się nie dzieje, będziesz musiał kliknąć napis *Zezwalaj na zablokowaną zawartość*, wyświetlony u dołu okna programu (przeczytaj także uwagę zamieszczoną na stronie 48).

Jak widać, krótki fragment kodu JavaScript pozwala uzyskać niezwykłe efekty na stronach WWW. Dzięki bibliotekom języka JavaScript, takim jak jQuery, możesz tworzyć złożone, interaktywne witryny bez zaawansowanych umiejętności programistycznych. Pomocna jest jednak podstawowa wiedza z obszaru języka JavaScript i programowania. W rozdziałach 2. i 3. poznasz podstawy JavaScriptu oraz opanujesz główne zagadnienia i składnię tego języka.

Wykrywanie błędów

Najbardziej frustrujący przy programowaniu w języku JavaScript jest moment, kiedy programista próbuje uruchomić w przeglądarce stronę z kodem JavaScript, ale nic się nie dzieje. Programiści bardzo często spotykają się z taką sytuacją. Nawet doświadczeni profesjonaliści nie zawsze od razu piszą poprawny kod, dlatego wykrywanie usterek jest nieodłącznym elementem programowania.

Większość przeglądarek ignoruje błędy w kodzie JavaScript, dlatego zwykle nie zobaczysz nawet okienka z informacją: „Ten program nie działa!". Przeważnie jest to korzystne, ponieważ błędy w kodzie JavaScript nie powinny zakłócać przeglądania stron.

Jak więc można wykryć, gdzie wystąpił błąd? Jest wiele sposobów wyszukiwania usterek w programach JavaScript. W rozdziale 13. poznasz zaawansowane techniki *diagnostyczne*, jednak najprostszą metodą jest użycie przeglądarki. Większość przeglądarek śledzi błędy w kodzie JavaScript i zapisuje je w odrębnym oknie — w *konsoli JavaScript*. Kiedy wczytasz stronę, która zawiera błąd, możesz wyświetlić konsolę, aby uzyskać pomocne informacje na temat usterki, na przykład numer wiersza strony, w którym wystąpił błąd, i opis problemu.

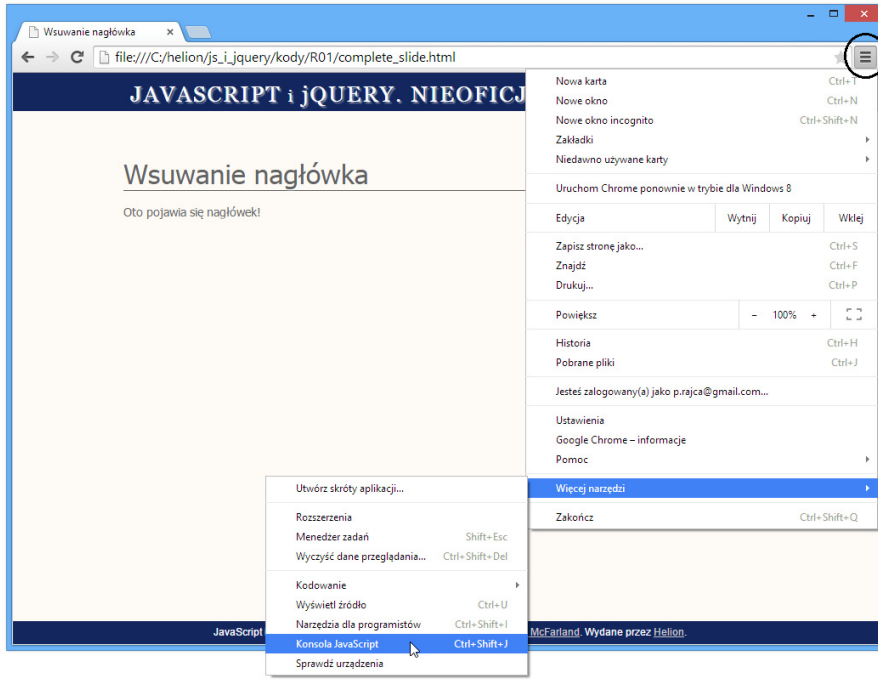
Często, korzystając z konsoli JavaScript, uda Ci się znaleźć przyczynę problemu, naprawić kod JavaScript i strona zacznie działać. Konsola ta pomaga dostrzec proste literówki typowe dla początkujących, na przykład brakujące końcowe znaki

przestankowe lub błędne nazwy poleceń języka JavaScript. Możesz używać konsoli w swoim ulubionym programie. Jednak czasem skrypty działają tylko w niektórych przeglądarkach, zatem w tym podrozdziale dowiesz się, jak uruchomić konsolę JavaScript we wszystkich popularnych przeglądarkach, aby wykryć błędy w każdej z nich.

Konsola JavaScript w przeglądarce Chrome

Przeglądarka Chrome napisana w firmie Google jest ulubioną przeglądarką wielu twórców stron WWW. Dostępne w niej narzędzia dla programistów zapewniają wiele możliwości rozwiązywania problemów z kodami HTML, CSS oraz JavaScript. Dodatkowo dostępna w niej konsola JavaScript jest miejscem, które idealnie nadaje się do rozpoczęcia odnajdywania błędów w kodzie — nie tylko opisuje błędy, które wystąpiły, lecz także podaj plik i numer wiersza, w którym błąd się pojawił.

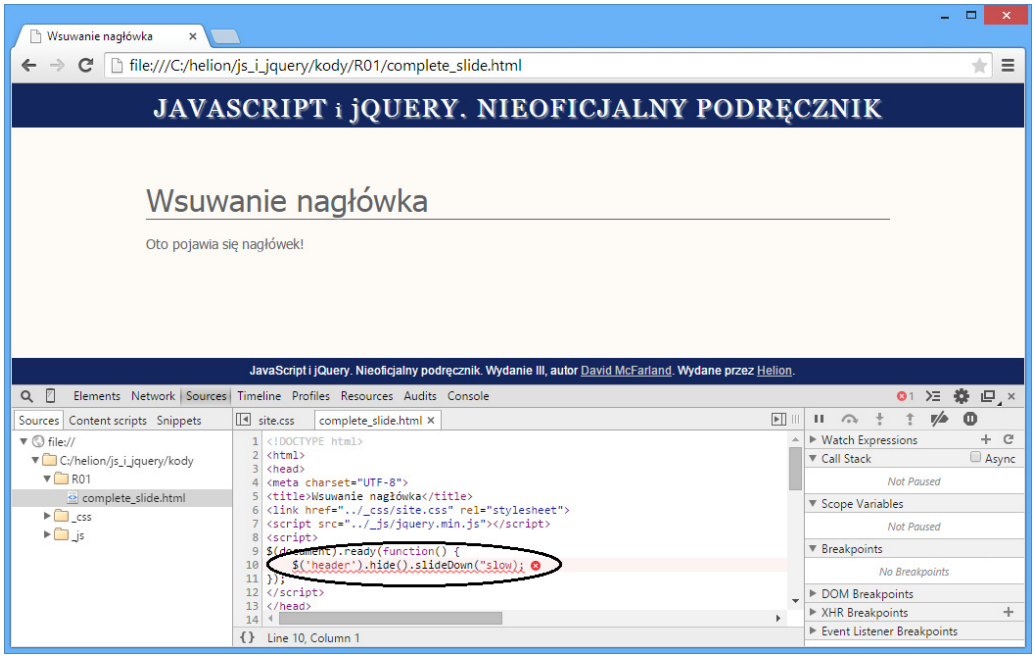
Aby wyświetlić konsolę, należy kliknąć przycisk *Dostosowywanie i kontrolowanie Google Chrome* (zakreślony na rysunku 1.5), a następnie wybrać opcję *Więcej narzędzi/Konsola JavaScript*. Można także użyć kombinacji klawiszy *Ctrl+Shift+J* (w systemie Windows) lub *⌘+Option+J* (w systemie Mac OS).



Rysunek 1.5. *Kliknij menu Dostosowywanie (zakreślone), aby uzyskać dostęp do konsoli JavaScript oraz wielu innych, przydatnych narzędzi. Innym sposobem uzyskania dostępu do konsoli jest wybranie opcji Narzędzia dla programistów, gdyż stanowi ona element większego zestawu narzędzi o tej właśnie nazwie. Narzędzia te poznasz dokładniej w rozdziale 17.*

Po wyświetleniu konsoli można obejrzeć wszystkie błędy, które wystąpiły na aktualnie wyświetlonej stronie. Przykładowo w konsoli przedstawionej na rysunku 1.6 określono napotkany błąd jako „Uncaught SyntaxError: Unexpected token ILLEGAL". No dobrze, być może znaczenie tego błędu nie jest od razu oczywiste, jednak wraz z odnajdywaniem (i poprawianiem) kolejnych błędów na pewno przyzwyczaisz się do tych lakonicznych opisów. Najprościej rzecz ujmując, błąd syntaktyczny informuje o jakimś problemie typograficznym, związanym ze składnią lub językiem programu. Fragment „Unexpected token ILLEGAL" oznacza, że przeglądarka odnalazła niedozwolony znak bądź też (i to już jest nieco trudniejszy problem) czegoś w kodzie brakuje. W naszym przypadku po dokładniejszym przeanalizowaniu kodu przykładu można zauważyć, że przed słowem `slow` jest umieszczony cudzysłów, lecz za nim cudzysłowu nie ma.

Konsola wyświetla także nazwę pliku, w którym wystąpił błąd (w tym przypadku jest to *complete_slide.html*) oraz numer wiersza (wiersz 10.). Wystarczy kliknąć nazwę pliku, by Chrome wyświetliła plik powyżej konsoli i przez chwilę podświetliła odpowiedni wiersz kodu (patrz rysunek 1.6).



Rysunek 1.6. *Konsola JavaScript przeglądarki Chrome wyświetla błędy, które wystąpiły w kodzie. Można kliknąć nazwę pliku wyświetloną z prawej strony opisu błędu, a przeglądarka pokaże kod strony i na chwilę podświetli wiersz kodu, w którym wystąpił błąd*

Wskazówka: Ponieważ konsola wyświetla numery wierszy, w których występują błędy, zatem warto używać także edytora numerującego wiersze kodu. Wtedy bez trudu będzie można przejść z konsoli JavaScript do edytora i odszukać w nim problematyczny wiersz, który należy pobrać.

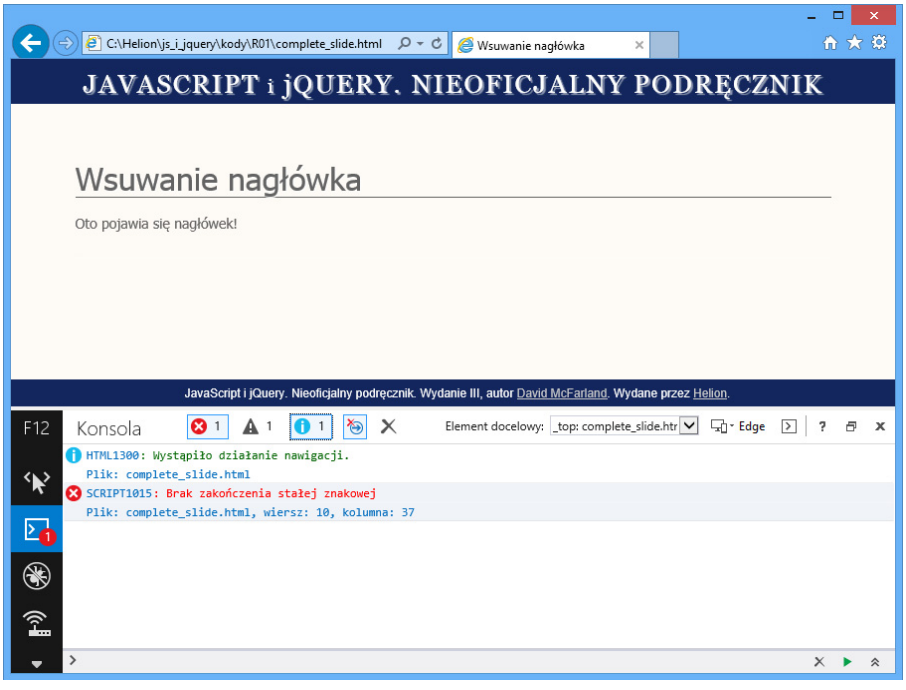
Niestety, w skrypcie może wystąpić wiele usterek — od prostych literówek po złożone błędy w logice kodu. Osoby poznające dopiero język JavaScript popełniają wiele błędów przy wpisywaniu kodu: zapominają o średnikach, cudzysłowach i nawiasach lub błędnie wpisują polecenia języka. Literówki pojawiają się szczególnie często przy przepisywaniu przykładów z książek. Poniżej znajduje się lista często popełnianych błędów oraz (niezbyt oczywistych) komunikatów, które o nich informują.

- **Brakujące znaki interpunkcyjne.** W kodzie JavaScript występuje bardzo wiele symboli, takich jak otwierające i zamykające nawiasy oraz nawiasy klamrowe. Przykładowo w kodzie `alert('Witaj';` brakuje nawiasu zamykającego po słowie `Witaj`, przez co podczas próby obejrzenia takiej strony Chrome wyświetli komunikat „Unexpected token ;", informując, że oczekiwała innego znaku niż wyświetlony w komunikacie. W tym przypadku zamiast nawiasu zamykającego został odnaleziony średnik.
- **Brakujący znak cudzysłowu lub apostrofu.** *Łańcuch znaków* to zbiór znaków umieszczony w cudzysłowach lub apostrofach (więcej informacji na temat znajdziesz na stronie 61), na przykład słowo `Witaj` w kodzie `alert('Witaj');`. Łatwo zapomnieć o dodaniu otwierającego lub zamykającego cudzysłowu. Można też pomieszać cudzysłowy i apostrofy; na przykład przez nieuwagę można napisać `alert( Witaj');`. W obu tych przypadkach zostanie zapewne wyświetlony błąd „Uncaught SyntaxError: Unexpected token ILLEGAL".
- **Nieprawidłowo zapisane polecenia.** Jeśli nieprawidłowo zapiszemy nazwę polecenia JavaScript, na przykład: `aler( Witaj );`, zostanie wyświetlony błąd informujący o tym, że podane polecenie nie zostało zdefiniowane. Przykładowo w przedstawionym przykładzie nieprawidłowo zapisanego polecenia `alert` komunikat może mieć postać: „Uncaught ReferenceError: aler is not defined". Błędy będą także zgłaszane w przypadku nieprawidłowego zapisania nazw funkcji biblioteki jQuery (takich jak `.hide()` lub `.slideDown()` używanych w przedstawionym wcześniej programie przykładowym). W tym przypadku będzie im jednak towarzyszył inny komunikat. Jeśli w poprzednim przykładzie, w kroku 6. na stronie 50 zamiast `hide` napiszemy `hid`, Chrome wyświetli komunikat o treści: „Uncaught TypeError: Object [object Object] has no method 'hid'".
- **Błąd składni.** Czasem Chrome nie potrafi zrozumieć znaczenia kodu i wyświetla ogólny komunikat o błędzie. Błąd składni wskazuje na usterkę w kodzie. Może to być na przykład połączenie poleceń JavaScript w niedozwolony sposób, a nie prosta literówka. Należy wtedy dokładnie przyjrzeć się wierszowi z błędem i spróbować ustalić przyczynę problemu. Niestety, naprawa usterek tego typu często wymaga doświadczenia i zrozumienia kodu JavaScript.

Powyższa lista pokazuje, że wiele błędów wynika po prostu z pominięcia znaku przestankowego, na przykład cudzysłowu lub nawiasu. Na szczęście takie usterki można łatwo naprawić, a wraz z nabywaniem doświadczenia prawie przestaniesz popełniać podobne pomyłki (żaden programista nie jest całkowicie bezbłędny).

Konsola przeglądarki Internet Explorer

Przeglądarka Internet Explorer udostępnia zbiór wyszukanych narzędzi programistycznych, służących nie tylko do odnajdywania błędów w kodzie JavaScript, lecz także do analizowania kodów CSS, HTML oraz przesyłu danych siecią. Po otworzeniu okno narzędzi programistycznych jest widoczne w dolnej części okna przeglądarki. Okno narzędzi programistycznych można wyświetlić, naciskając klawisz *F12*; ponowne naciśnięcie tego samego klawisza spowoduje jego zamknięcie. Błędy odnalezione w kodzie JavaScript są wyświetlane na karcie *Konsola* (patrz rysunek 1.7).



Rysunek 1.7. *Okno narzędzie deweloperskich w przeglądarce Internet Explorer zapewnia dostęp do błędów w kodzie JavaScript, a także do bardzo wielu innych informacji*

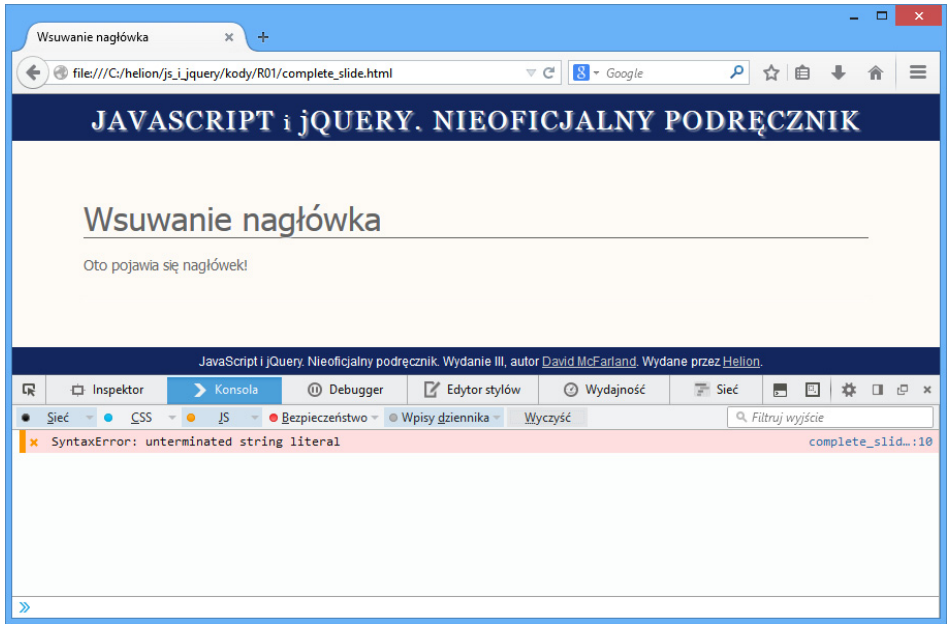
Uwaga: Jeśli najpierw otworzymy stronę WWW, a dopiero potem konsolę JavaScript, Internet Explorer nie wyświetli w niej żadnych błędów (nawet jeśli takie na stronie występują). Aby je pokazać, konieczne jest ponowne wyświetlenie strony. Kiedy konsola będzie już wyświetlona, błędy będą w niej widoczne podczas oglądania strony.

Konsola przeglądarki IE wyświetla komunikaty błędów, podobne do prezentowanych w przedstawionej wcześniej przeglądarce Chrome. Jednak czasami wyświetlane komunikaty są nieco inne. Przykładowo błąd „Unexpected token ILLEGAL" w przeglądarce Chrome w Internet Explorerze jest prezentowany jako „Brak zakończenia stałej znakowej". Podobnie jak Chrome, także Internet Explorer wyświetla informacje o miejscu wystąpienia błędu w kodzie dokumentu HTML i pozwala kliknąć, by go wyświetlić.

Konsola JavaScript w przeglądarce Firefox

Także przeglądarka Firefox fundacji Mozilla udostępnia konsolę pozwalającą na przeglądanie błędów JavaScript. Aby ją wyświetlić, należy kliknąć przycisk *Otwórz menu* widoczny w prawym górnym rogu okna przeglądarki, a następnie wybrać opcję *Narzędzia* i *Konsola WWW*.Na komputerach Mac trzeba wybrać opcję *Narzędzia/Dla twórców witryn/Konsola przeglądarki*. Można także skorzystać z kombinacji klawiszy *Ctrl+Shift+I* (w systemie Windows) oraz ⌘+*Option*+*K* (Mac).

Po wyświetleniu konsoli widoczna w niej będzie strona z błędami JavaScript. Niestety, w przypadku przeglądarki Firefox konsola WWW to prawdziwa skarbnica danych, a nie zwyczajne narzędzie do wyświetlania błędów JavaScriptu (patrz rysunek 1.8). Dzieje się tak dlatego, że udostępnia ona informacje o wielu różnych zdarzeniach: pobranych plikach, napotkanych błędach CSS, HTML i tym podobnych.



Rysunek 1.8. *Jeśli nie chcemy oglądać wszystkich komunikatów widocznych w konsoli WWW przeglądarki Firefox, wystarczy kliknąć przycisk odpowiadający typowi komunikatów, które chcemy ukryć. I tak kliknięcie przycisku CSS spowoduje ukrycie komunikatów o błędach CSS, a przycisku Bezpieczeństwo — o błędach bezpieczeństwa. Wyłączone przyciski można poznać po tym, że są jaśniejsze, tak jak przedstawione na tym rysunku przyciski CSS i Bezpieczeństwo. Przycisk jest włączony, jeśli jest ciemniejszy i wygląda jakby był „wciśnięty", jak widoczne przyciski Sieć, JS (skrót od JavaScript) oraz Wpisy dziennika*

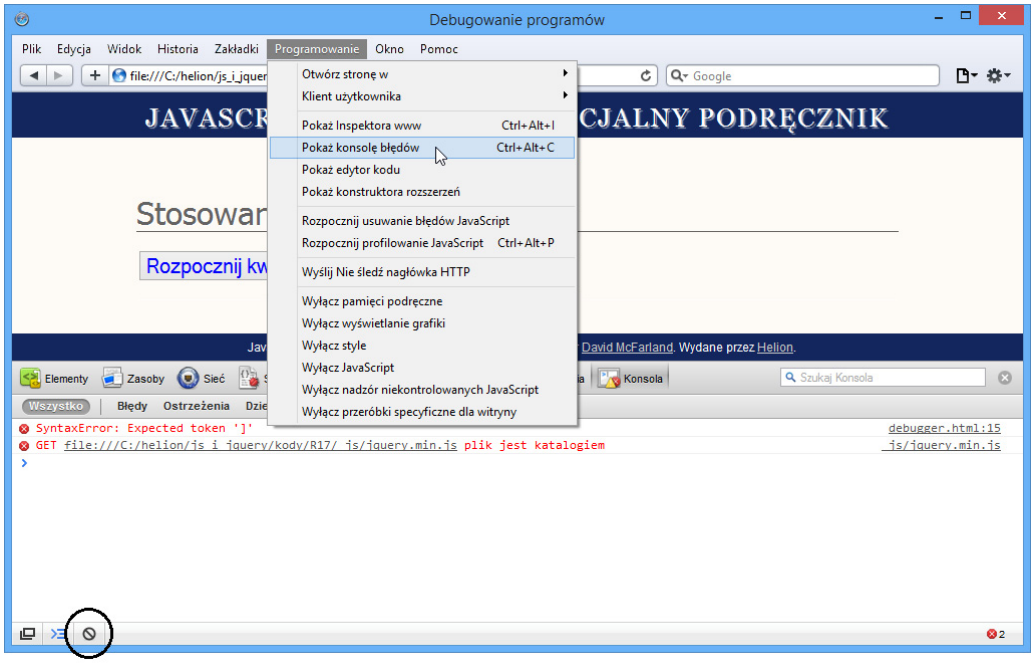
Uwaga: Wtyczka Firebug (*http://getfirebug.com*) znacznie rozszerza możliwości standardowej konsoli WWW przeglądarki Firefox. W rzeczywistości wtyczka ta stanowiła wzór dla narzędzi programistycznych dostępnych obecnie w przeglądarkach Internet Explorer, Chrome oraz Safari (przedstawionej w następnym punkcie rozdziału).

Konsola błędów w przeglądarce Safari

Konsolę błędów w przeglądarce Safari można otworzyć przy użyciu opcji *Programowanie/Pokaż konsolę błędów* (na komputerach Mac można użyć skrótu *Option+⌘+C*, a na komputerach z systemem Windows skrótu *Ctrl+Alt+C*). Jednak po zainstalowaniu Safari menu *Programowanie* jest zwykle wyłączone, dlatego aby uruchomić konsolę JavaScript, trzeba zwykle wykonać kilka operacji.

Aby wyświetlić menu *Programowanie* na komputerach Mac, wybierz opcję *Safari/Preferencje*. Po wyświetleniu okna dialogowego kliknij przycisk *Zaawansowane*, a następnie zaznacz pole wyboru *Pokazuj menu Programowanie w pasku menu* i zamknij okno dialogowe.

Kiedy ponownie uruchomisz Safari, menu *Programowanie* pojawi się na pasku menu między opcjami *Zakładki* a *Okno*. Aby uruchomić konsolę, wybierz opcję *Programowanie/Pokaż konsolę błędów* (patrz rysunek 1.9).



Rysunek 1.9. *Konsola błędów w Safari wyświetla nazwę błędu JavaScript, nazwę i lokalizację pliku oraz wiersz, w którym przeglądarka natrafiła na problem. Każda zakładka i okno przeglądarki ma odrębną konsolę, dlatego jeśli już otworzyłeś okno błędów, a chcesz zobaczyć informacje dotyczące innej strony, musisz ponownie wybrać opcję Programowanie/Pokaż konsolę błędów. Co więcej, jeśli odświeżysz stronę, starsze wersje Safari nie wyczyszczą błędów wyświetlonych dla poprzedniej strony, dlatego może się okazać, że lista zawiera komunikaty dotyczące już poprawionych błędów. Rozwiązaniem tego problemu jest kliknięcie przycisku zakazu wjazdu (zakreślonego), który czyści listę, oraz ponowne wyświetlenie strony*

Uwaga: Użytkownicy systemu Windows mogą aktualnie dysponować starą wersją przeglądarki Safari. Firma Apple zaniechała rozwoju wersji programu dla systemu Windows, dlatego zamieszczone tu informacje mogą być nieaktualne.

2

ROZDZIAŁ

Gramatyka języka JavaScript

Poznawanie języka programowania przypomina naukę nowego języka naturalnego. Trzeba nauczyć się słówek, zrozumieć zasady dodawania znaków przestankowych i opanować nowy zestaw reguł. Podobnie jak trzeba poznać gramatykę języka francuskiego, aby się nim posługiwać, należy nauczyć się gramatyki języka JavaScript, aby w nim programować. W tym rozdziale opisano podstawy, na których oparte są wszystkie programy w języku JavaScript.

Jeśli programowałeś już w języku JavaScript, prawdopodobnie znasz wiele omawianych tu zagadnień, dlatego możesz pobieżnie przejrzeć ten rozdział. Jeśli jednak dopiero poznajesz ten język lub wciąż nie znasz wszystkich jego podstaw, znajdziesz tu wprowadzenie do kluczowych zagadnień.

Instrukcje

Instrukcja języka JavaScript to podstawowa jednostka programowania, zwykle reprezentująca jeden krok programu. Możesz traktować instrukcje jak zdania. Podobnie jak łańcuch zdań tworzy akapit (lub rozdział albo książkę), tak instrukcje składają się na program w języku JavaScript. W poprzednim rozdziale zobaczyłeś kilka instrukcji, na przykład:

```
alert('Witaj, świecie!');
```

Ta pojedyncza instrukcja otwiera okno dialogowe z wiadomością „Witaj, świecie!". Instrukcje to często pojedyncze wiersze kodu. Każda instrukcja kończy się średnikiem, który działa podobnie jak kropka w zdaniu. Średnik jednoznacznie określa, że dany etap jest zakończony, a interpreter JavaScript powinien przejść do następnego wiersza. Proces tworzenia programów w języku JavaScript polega — ogólnie rzecz biorąc — na wpisaniu instrukcji, dodaniu średnika, wciśnięciu klawisza *Enter* w celu utworzenia nowego, pustego wiersza, wpisaniu następnej instrukcji ze średnikiem i tak dalej do momentu ukończenia skryptu.

Uwaga: Przejrzenie bardziej zaawansowanych przykładów kodu JavaScript (takich jak zamieszczone dalej w tej książce) pozwoli zauważyć, że średniki nie są umieszczane na końcu *każdego* z wierszy. Czasami może się zdarzyć, że średniki będą się pojawiać co kilka, kilkanaście lub nawet wiele wierszy. Jednak nawet tak wiele wierszy może tworzyć jedną *instrukcję* — należy je sobie wyobrazić jako jedną, naprawdę długą instrukcję zawierającą wiele znaków przestankowych (czyli trochę tak, jak w tym zdaniu).

Oficjalnie umieszczanie średników po *instrukcjach* jest opcjonalne, a niektórzy programiści pomijają je, aby skrócić kod, jednak nie należy ich naśladować. Pomijanie średników utrudnia czytanie kodu, a w pewnych warunkach powoduje błędy w programach. Jeśli chcesz skrócić kod JavaScript, aby przyspieszyć jego pobieranie, zapoznaj się ze wskazówkami ze strony 609.

Wbudowane funkcje

JavaScript i przeglądarki umożliwiają używanie różnych poleceń do wykonywania zadań w programach i na stronach. Polecenia te, nazywane *funkcjami*, można porównać z czasownikami w zdaniach. Na przykład przedstawione już polecenie `alert()` powoduje otwarcie okna dialogowego w przeglądarce i wyświetlenie komunikatu.

Niektóre polecenia, na przykład `alert()` i `document.write()` użyte na stronie 48, są specyficzne dla przeglądarek. Oznacza to, że działają tylko na stronach WWW i nie są dostępne w innych środowiskach, w których można używać języka JavaScript (na przykład w skryptach dla Node.js lub aplikacjach firmy Adobe, takich jak Photoshop, oraz w używanym we Flashu języku ActionScript, którego podstawą jest JavaScript).

Inne polecenia są uniwersalne i działają wszędzie tam, gdzie można korzystać z języka JavaScript. Na przykład polecenie `isNaN()` sprawdza, czy dana wartość jest liczbą, czy nie. Jest ono przydatne, kiedy trzeba sprawdzić, czy użytkownik wpisał wartość odpowiedniego typu w polu na dane liczbowe (na przykład przy pytaniu „Ilu kontrolek chcesz użyć?"). Więcej o poleceniu `isNaN()` i sposobach jego używania dowiesz się na stronie 589.

JavaScript udostępnia wiele różnych poleceń, które poznasz na kartach tej książki. Szybkim sposobem na znalezienie poleceń w programach jest zwrócenie uwagi na nawiasy. Można łatwo stwierdzić, że `isNaN()` to polecenie, ponieważ po nazwie `isNaN` występują nawiasy.

Ponadto JavaScript umożliwia tworzenie własnych funkcji, dlatego skrypty mogą wykonywać operacje wykraczające poza możliwości standardowych poleceń języka JavaScript. Szczegółowy opis funkcji znajdziesz w rozdziale 3., na stronie 115.

Uwaga: Niektórzy funkcje języka JavaScript nazywają *metodami*.

Typy danych

Każdego dnia stykasz się z informacjami różnego typu. Nazwisko, cena posiłku, adres gabinetu doktora i data urodzin — to wszystko istotne dane. Na ich podstawie podejmujesz decyzje związane z tym, co robić. Programy komputerowe funkcjonują podobnie — także przy wykonywaniu zadań polegają na informacjach. Aby na

przykład obliczyć łączną cenę zakupów z koszyka, potrzebne są cena i liczba wszystkich produktów. Aby wyświetlić na stronie imię użytkownika („Witaj ponownie, *Aniu*"), trzeba je ustalić.

W językach programowania informacje są zwykle pokategoryzowane według typu, a przetwarzanie danych każdego rodzaju przebiega w odmienny sposób. W języku JavaScript są trzy podstawowe typy danych: *liczbowe*, *łańcuchowe* i *logiczne*.

Liczby

Liczby służą do odliczania i obliczeń. Można ich użyć do odliczania liczby dni do letnich wakacji lub obliczenia ceny zakupu dwóch biletów do kina. Liczby są bardzo ważne w programach w języku JavaScript. Można ich używać do przechowywania liczby odwiedzin użytkownika na stronie, określania w pikselach pozycji elementu lub ustalania liczby produktów zamawianych przez klienta.

W języku JavaScript liczby są reprezentowane przez cyfry. Na przykład `5` odpowiada liczbie pięć. Można też przedstawiać liczby dziesiętne: `5.25` lub `10.3333333`. JavaScript pozwala również stosować liczby ujemne, na przykład `-130` lub `-459.67`.

Ponieważ liczby są często używane w obliczeniach, w programach pojawia się wiele operacji matematycznych. Szczegółowy opis *operatorów* znajdziesz na stronie 67, natomiast poniższy wiersz wyświetla sumę liczb 5 i 15 oraz ilustruje, jak używać liczb w kodzie JavaScript:

```
document.write(5 + 15);
```

Ten wiersz dodaje dwie liczby i wyświetla ich sumę (20) na stronie WWW. Liczb można używać na wiele różnych sposobów, których omówienie rozpoczyna się na stronie 67.

Łańcuchy znaków

Do wyświetlania imion, zdań i serii znaków służą łańcuchy znaków. *Łańcuch znaków* to po prostu ciąg znaków (liter i innych symboli) umieszczonych w cudzysłowach lub apostrofach. Na przykład `'Witaj, Hal'` i `Jesteś tutaj` to łańcuchy znaków. Danych tego typu użyłeś w poprzednim rozdziale w poleceniu `alert` — `alert('Witaj, świecie!');`.

Cudzysłów otwierający informuje interpreter języka JavaScript o tym, że zaraz natrafi na łańcuch znaków, czyli sekwencję symboli. Interpreter traktuje te znaki dosłownie i nie próbuje interpretować ich jako słów specyficznych dla języka JavaScript (na przykład poleceń). Kiedy interpreter natrafi na cudzysłów zamykający, wykrywa koniec łańcucha znaków i przechodzi do analizy dalszej części programu.

Łańcuchy znaków można umieszczać zarówno w cudzysłowach (`Witaj, świecie`), jak i w apostrofach (`'Witaj, świecie'`), przy czym symbol otwierający i zamykający musi być *taki sam*. Na przykład kod `to nie jest poprawny zapis'` nie jest prawidłowym łańcuchem znaków, ponieważ rozpoczyna się od cudzysłowu, a kończy apostrofem.

Dlatego aby wyświetlić komunikat „Uwaga, uwaga!”, można użyć zapisu:

```
alert('Uwaga, uwaga!');
```

lub:

```
alert("Uwaga, uwaga!");
```

Łańcuchy znaków są często używane w programach. Służą między innymi do wyświetlania komunikatów w oknach dialogowych, pobierania danych w formularzach i manipulowania zawartością stron WWW. Łańcuchy znaków są tak ważne, że na stronie 69 znajduje się ich szczegółowe omówienie.

Wartości logiczne

Podczas gdy liczby i łańcuchy znaków dają niemal nieskończone możliwości, typ logiczny jest prosty. Ma tylko dwie wartości: `true` (prawda) i `false` (fałsz). Typy logiczne są potrzebne przy tworzeniu w języku JavaScript programów reagujących na działania użytkowników i wprowadzone przez nich dane. Jeśli przed przesłaniem formularza chcesz się upewnić, że użytkownik podał adres e-mail, możesz dodać do strony kod sprawdzający odpowiedź na proste pytanie: „Czy użytkownik wprowadził prawidłowy adres e-mail?”. Odpowiedź na to pytanie to wartość logiczna. Adres

CZĘSTO ZADAWANE PYTANIA

Umieszczanie cudzysłowów w łańcuchach znaków

Kiedy próbuję utworzyć łańcuch znaków zawierający cudzysłów, program nie działa. Dlaczego tak się dzieje?

W języku JavaScript cudzysłowy oznaczają początek i koniec łańcucha znaków, choć nie zawsze jest to pożądane. Kiedy interpreter natrafi na pierwszy cudzysłów, wie, że jest to początek łańcucha znaków. Gdy dojdzie do drugiego cudzysłowu, uznaje, że oznacza on koniec łańcucha. Dlatego nie można utworzyć łańcucha o treści `"Jan powiedział: "Witaj""`. Pierwszy cudzysłów (przed słowem "Jan") oznacza początek łańcucha, a kiedy interpreter natrafi na drugi cudzysłów (przed słowem "Witaj"), uzna, że łańcuch się skończył. Dlatego w programie znajdzie się łańcuch `"Jan powiedział: "` i słowo `Witaj`, które spowoduje błąd i przerwanie działania programu.

Ten problem można rozwiązać na kilka sposobów. Najłatwiejsza metoda polega na użyciu apostrofów do wydzielenia łańcucha znaków, który ma zawierać cudzysłowy. Na przykład `'Jan powiedział: "Witaj"'` to prawidłowy łańcuch. Apostrofy ograniczają łańcuch znaków, a cudzysłowy wewnątrz niego są *częścią* tego łańcucha. Można też użyć cudzysłowów do wyodrębnienia łańcucha zawierającego apostrofy, na przykład `"pana Moore'a"`.

Inne rozwiązanie polega na nakazaniu interpreterowi dosłownego traktowania cudzysłowów w łańcuchu, czyli interpretowania ich jako części łańcucha, a nie jego końca. Służy do tego *sekwencja ucieczki*. Jeśli umieścisz przed cudzysłowem ukośnik odwrotny (`\`), cudzysłów zostanie potraktowany jak zwykły znak. Wcześniejszy fragment można zapisać również w następujący sposób: `"Jan powiedział: \"Witaj\""`. Czasem użycie sekwencji ucieczki to jedyna możliwość, przykładem jest kod: `'Jan powiedział: "Witam pana Moore\'a"'`. Ponieważ łańcuch jest ograniczony apostrofami, apostrof w zwrocie „pana Moore'a” trzeba poprzedzić ukośnikiem odwrotnym: `pana Moore\'a`.

Sekwencji ucieczki można użyć nawet wtedy, kiedy nie jest konieczna. Pozwala to podkreślić, że cudzysłów należy traktować dosłownie. Choć w kodzie `'Jan powiedział: "Witaj"'` nie trzeba używać sekwencji ucieczki, ponieważ łańcuch jest ograniczony przez apostrofy, niektórzy programiści dodadzą ukośniki, aby zaznaczyć, że cudzysłowy są częścią łańcucha.

e-mail jest albo prawidłowy (`true`), albo nieprawidłowy (`false`). Strona może następnie odpowiednio zareagować na uzyskaną odpowiedź. Jeśli na przykład adres jest prawidłowy (`true`), formularz zostanie przesłany. Jeżeli adres nie jest prawidłowy (`false`), strona wyświetli komunikat o błędzie i zablokuje wysyłanie formularza.

Okazuje się, że wartości logiczne są tak ważne dla języka JavaScript, iż dodano do niego dwa specjalne, reprezentujące je słowa kluczowe:

```
true
```

oraz

```
false.
```

Stosowania wartości logicznych nauczysz się przy dodawaniu logiki do programów, co opisano w ramce na stronie 99.

Zmienne

Liczby, łańcuchy znaków i wartości logiczne można umieszczać bezpośrednio w programie JavaScript, jednak tylko wtedy, kiedy informacje są już dostępne. Można na przykład wyświetlić łańcuch znaków `Witaj, Robercie` w oknie dialogowym za pomocą następującego kodu:

```
alert("Witaj, Robercie");
```

Jednak ta instrukcja ma sens tylko wtedy, jeśli wszyscy użytkownicy strony mają na imię Robert. Jeżeli strona ma wyświetlać komunikat dostosowany do internautów, imię powinno się zmieniać w zależności od tego, kto odwiedził stronę: „Witaj, Mario", „Witaj, Józefie", „Witaj, Kasiu". Na szczęście wszystkie języki programowania udostępniają narzędzie pomocne w takich sytuacjach; są to *zmienne*.

Zmienna pozwala zapisać informacje, które można następnie wykorzystać lub zmienić. Wyobraź sobie napisaną w języku JavaScript grę w pinball, w której należy uzyskać jak największą liczbę punktów. Kiedy gracz rozpoczyna pierwszą rozgrywkę, jego wynik to 0, jednak później trafia kulką w elementy planszy, a liczba punktów rośnie. `Wynik` to w tym przypadku zmienna. Początkowo ma ona wartość 0, jednak rośnie wraz z przebiegiem gry. Zmienne przechowują więc informacje, które mogą się *zmieniać* zależnie od okoliczności. Na rysunku 2.1 widoczna jest inna gra, w której wykorzystano zmienne.

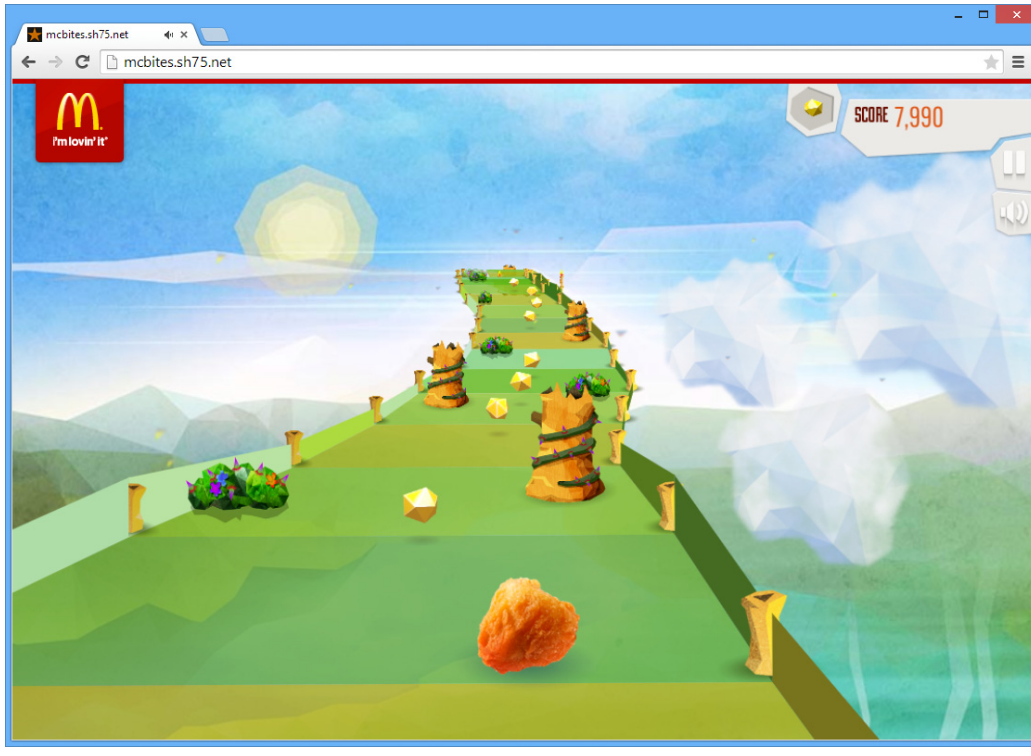
Możesz traktować zmienne jak specyficzny koszyk. Możesz umieszczać przedmioty w koszyku, zajrzeć do niego, opróżnić go, a także zmienić jego zawartość. Jednak choć koszyk przechowuje różne elementy, wciąż pozostaje taki sam.

Tworzenie zmiennych

W języku JavaScript zmienną o nazwie `score` można utworzyć w następujący sposób:

```
var score;
```

Pierwsza część, `var`, to słowo kluczowe języka JavaScript, które tworzy lub — jak można powiedzieć w języku technicznym — *deklaruje* zmienną. Druga część instrukcji, `score`, to nazwa zmiennej.



Rysunek 2.1. *W witrynie Supe Spice Dash (http://mcbites sh75.net) używa się języka JavaScript do utworzenia gry przypominającej nieco Super Monkey Ball, promującej oferowane w restauracjach McDonalds superkąski. Sterujemy (to nie żart) superkąskiem, prowadząc go po trasie, gromadząc okruchy złota i papryczki chili, a jednocześnie unikając innych przeszkód i przeskakując nad szczelinami. Podczas gry aktualny wynik jest prezentowany w prawym, górnym rogu okna przeglądarki. W tej grze do przechowywania wyniku jest używana zmienna, gdyż wartość wyniku nieustannie się zmienia*

Zmienne można nazwać w dowolny sposób, jednak trzeba przestrzegać kilku reguł.

- **Nazwy zmiennych muszą zaczynać się od litery, znaku $ lub _**. Nie można umieścić na początku nazwy zmiennej liczby ani znaku przestankowego. Dlatego nazwy `1thing` i `&thing` są nieprawidłowe, natomiast `score`, `$score` i `_score` — poprawne.
- **Nazwy zmiennych mogą zawierać wyłącznie litery, cyfry oraz znaki $ i _**. Nie można używać odstępów ani innych znaków specjalnych. Nazwy `fish&chips` i `fish and chips` są niedozwolone, natomiast `fish_n_ships` i `plan9` — prawidłowe.
- **W nazwach zmiennych istotna jest wielkość znaków**. Interpreter języka JavaScript traktuje duże i małe litery jako różne od siebie, dlatego zmienna `SCORE` nie jest zmienną `score`, a nazwy `sCoRE` i `Score` wskazują na jeszcze inne zmienne.
- **Należy unikać słów kluczowych**. Niektóre słowa są specyficzne dla samego języka. Na przykład słowo `var` służy do tworzenia zmiennych, dlatego nie można nadać zmiennej nazwy `var`. Ponadto niektóre słowa, na przykład `alert`, `document`

i `window`, mają specjalne znaczenie w przeglądarkach. Próba użycia tych słów jako nazw zmiennych wywoła błąd w kodzie JavaScript. Listę słów zarezerwowanych przedstawiono w tabeli 2.1. Nie wszystkie te słowa powodują problemy w każdej przeglądarce, jednak najlepiej unikać ich przy nazywaniu zmiennych.

Tabela 2.1. *Niektóre słowa są zarezerwowane do użytku w języku JavaScript i przeglądarkach. Należy unikać ich przy wymyślaniu nazw zmiennych*

Słowa kluczowe języka JavaScript	Słowa zarezerwowane do przyszłego użytku	Słowa zarezerwowane na potrzeby przeglądarek
break	abstract	alert
case	boolean	blur
catch	byte	closed
continue	char	document
debugger	class	focus
default	const	frames
delete	double	history
do	enum	innerHeight
else	export	innerWidth
false	extends	length
finally	final	location
for	float	navigator
function	goto	open
if	implements	outerHeight
in	import	outerWidth
instanceof	int	parent
new	interface	screen
null	let	screenX
return	long	screenY
switch	native	statusbar
this	package	window
throw	private	
true	protected	
try	public	
typeof	short	
var	static	
void	super	
while	synchronized	
with	throws	
	transient	
	volatile	
	yield	

Oprócz przestrzegania tych reguł należy pamiętać o tym, aby nazwy zmiennych były jasne i znaczące. Nazywanie zmiennych zgodnie z rodzajem przechowywanych danych znacznie ułatwia analizę kodu. Na przykład `wynik` to doskonała nazwa

zmiennej przechowującej liczbę punktów gracza. Można użyć także nazwy `w`, jednak pojedyncza litera „w" nie pozwala domyślić się, jakie dane przechowuje zmienna.

Nazwy zmiennych powinny być też czytelne. Jeśli nazwa składa się z kilku członów, należy albo rozdzielić wyrazy podkreśleniem, albo rozpoczynać słowa dużymi literami. Na przykład bardziej czytelna jest postać `sciezka_do_rysunku` lub `sciezkaDoRysunku` niż `sciezkadorysunku`.

Używanie zmiennych

Po utworzeniu zmiennej można zapisać w niej dane dowolnego typu. Do przypisywania wartości służy znak =. Aby zapisać liczbę `0` w zmiennej o nazwie `score`, można użyć następującego kodu:

```
var score;
score = 0;
```

Pierwszy wiersz tego kodu tworzy zmienną, a drugi zapisuje w niej liczbę `0`. Znak równości to *operator przypisania*, ponieważ służy do przypisywania wartości do zmiennych. Można też utworzyć zmienną i zapisać w niej wartość w jednej instrukcji języka JavaScript:

```
var score = 0;
```

W zmiennych można zapisywać łańcuchy znaków, liczby i wartości logiczne:

```
var firstName = 'Piotr';
var lastName = 'Nowak';
var age = 22;
var isSuperHero = true;
```

Wskazówka: Aby skrócić kod, można zadeklarować kilka zmiennych przy użyciu jednego słowa kluczowego `var`:

```
var x, y, z;
```

W jednej instrukcji języka JavaScript można nawet zadeklarować kilka zmiennych i przypisać im wartości:

```
var isSuperHero=true, isAfraidOfHeights=false;
```

Wartość zapisaną w zmiennej można pobrać za pomocą nazwy tej zmiennej. Aby na przykład wyświetlić w oknie dialogowym wartość umieszczoną w zmiennej `score`, należy użyć następującego kodu:

```
alert(score);
```

Warto zauważyć, że nazw zmiennych nie należy podawać w cudzysłowach. Cudzysłowy są potrzebne przy łańcuchach znaków, dlatego kod `alert('score')` wyświetli słowo „score", a nie wartość zapisaną w zmiennej `score`. Pomaga to zrozumieć, dlaczego łańcuchy znaków należy umieszczać w cudzysłowach — interpreter języka JavaScript traktuje słowa bez cudzysłowów jako specjalne obiekty języka (takie jak funkcja `alert()`) lub nazwy zmiennych.

Uwaga: Słowo kluczowe `var` jest potrzebne tylko raz — przy tworzeniu zmiennej. Następnie można przypisywać do zmiennej nowe wartości bez używania tego słowa.

CZĘSTO ZADAWANE PYTANIA

Odstępy, znaki tabulacji i znaki karetki w języku JavaScript

Język JavaScript jest najwyraźniej wrażliwy na literówki. Kiedy mogę używać w kodzie odstępów, a kiedy jest to niedozwolone?

Ogólnie rzecz biorąc, język JavaScript traktuje znaki odstępu, nowego wiersza i tabulacji w sposób dosyć pobłażliwy. Bardzo często można pozostawiać znaki odstępu i nowego wiersza, a nawet je dodawać, i nie będzie to powodować żadnych problemów. Interpreter JavaScript ignoruje wszystkie nadmiarowe znaki, można zatem dodawać odstępy, znaki nowego wiersza i tabulacji, by odpowiednio formatować kod. Na przykład nie trzeba dodać odstępów po obu stronach operatora przypisania, ale może to poprawić czytelność kodu. Oba poniższe wiersze są prawidłowe:

```
var formName='signup';
var formRegistration = 'newsletter';
```

Można dodać dowolną liczbę odstępów, a nawet wstawić w instrukcji znak karetki. Dlatego poprawne są też następujące wersje:

```
var formName     =     'signup';
var formRegistration
             =
        'newsletter';
```

Oczywiście, możliwość wstawiania dodatkowych odstępów nie oznacza, że należy to robić. Dwie ostatnie instrukcje są mało czytelne i zrozumiałe. Zgodnie z ogólną zasadą należy wstawiać dodatkowe odstępy, jeśli poprawia to czytelność kodu. Przykładowo zastosowanie znaków nowego wiersza pozwala poprawić czytelność kodu w przypadku deklarowania wielu zmiennych w jednej instrukcji. Poniższy kod został zapisany w jednym wierszu:

```
var score=0, highScore=0, player='';
```

Niektórzy programiści uważają jednak, że taki kod będzie bardziej przejrzysty, jeśli poszczególne zmienne zostaną zapisane w osobnych wierszach:

```
var score=0,
    highScore=0,
    player='';
```

Samodzielnie należy ocenić, czy taki sposób zapisu będzie bardziej przejrzysty, bo interpreter JavaScriptu i tak zignoruje znaki nowego wiersza. Przykłady pokazujące, jak korzystanie ze znaków odstępu może poprawić przejrzystość kodu, znajdziesz w podrozdziałach poświęconych literałom obiektowym (patrz strona 165) oraz tablicom (patrz strona 77).

Istnieje kilka ważnych odstępstw od omówionych reguł — nie można używać znaku nowego wiersza w łańcuchach znaków. Oznacza to, że nie należy dzielić łańcuchów znaków na dwa wiersze:

```
var name = 'Jan
           Kowalski';
```

Wstawienie w takim miejscu znaku nowego wiersza przez wciśnięcie klawisza *Enter* wywoła błąd języka JavaScript i awarię programu.

Co więcej, konieczne jest dodawanie znaków odstępu pomiędzy słowami kluczowymi. Przykładowo *varscore=0* to nie to samo co var *score=0*. Ten drugi przykład tworzy nową zmienną o nazwie score, natomiast pierwszy przypisuje wartość 0 zmiennej o nazwie varscore. Interpreter JavaScript potrzebuje znaku odstępu pomiędzy *var* i *score*, aby rozpoznać *var* jako słowo kluczowe: *var score=0*. Niemniej jednak nie trzeba umieszczać odstępów pomiędzy słowami kluczowymi i symbolami, takimi jak operator przypisania (=) lub średnik kończący instrukcję.

Używanie typów danych i zmiennych

Zapisanie w zmiennej informacji, na przykład liczby lub łańcucha znaków, to zwykle tylko pierwsza operacja w programie. Większość skryptów manipuluje danymi, aby uzyskać nowe wyniki. Programy dodają wartość punktową do wyniku, mnożą liczbę produktów przez ich cenę, aby obliczyć koszt zakupów, lub personalizują ogólne komunikaty przez dodanie imienia na końcu łańcucha znaków: „Miło znów Cię widzieć, Igorze". JavaScript udostępnia zestaw *operatorów* do manipulowania danymi. Operatory to symbole i słowa, które umożliwiają modyfikację wartości zmiennej. Na przykład symbol +, operator dodawania, dodaje liczby do siebie. Dostępne są różne operatory przeznaczone dla różnych typów danych.

Podstawowe operacje matematyczne

JavaScript obsługuje podstawowe operacje matematyczne, na przykład dodawanie, dzielenie, odejmowanie i tak dalej. W tabeli 2.2 przedstawiono najprostsze operatory matematyczne i sposób ich używania.

Tabela 2.2. *Podstawowe operacje matematyczne w języku JavaScript*

Operator	Działanie	Sposób używania
+	Dodaje dwie liczby.	`5 + 25`
-	Odejmuje jedną liczbę od drugiej.	`25 - 5`
*	Mnoży dwie liczby.	`5 * 10`
/	Dzieli jedną liczbę przez drugą.	`15/5`

Wiele osób jest przyzwyczajonych do używania przy mnożeniu symbolu × (na przykład 4×5), jednak w języku JavaScript służy do tego znak *.

W operacjach matematycznych można używać zmiennych. Ponieważ zmienna to tylko kontener na inne wartości, na przykład liczby i łańcuchy znaków, użycie zmiennej odpowiada użyciu jej zawartości:

```
var price = 10;
var itemsOrdered = 15;
var totalCost = price * itemsOrdered;
```

Dwa pierwsze wiersze tworzą dwie zmienne (`price` i `itemsOrdered`) i zapisują w nich liczby. Trzeci wiersz tworzy następną zmienną (`totalCost`) i zapisuje w niej wynik pomnożenia wartości zmiennej `price` (`10`) przez wartość zmiennej `itemsOrdered`. Powoduje to zapisanie w zmiennej `totalCost` liczby `150`.

Ten przykładowy kod ilustruje przydatność zmiennych. Wyobraź sobie, że masz napisać program do obsługi koszyka zakupów w sklepie internetowym. W programie do wielu obliczeń potrzebna będzie cena określonego produktu. Można ją zapisać w kilku miejscach (na przykład jeśli przedmiot kosztuje 10 złotych, należy wpisać tę wartość wszędzie tam, gdzie potrzebna jest cena). Jednak jeśli cena się zmieni, trzeba będzie znaleźć i zmodyfikować każdy wiersz, w którym jej użyto. Przy korzystaniu ze zmiennej wystarczy określić cenę jeden raz na początku programu. Następnie trzeba zmodyfikować tylko jeden wiersz kodu, aby zaktualizować wartość w całym programie:

```
var price = 20;
var itemsOrdered = 15;
var totalCost = price * itemsOrdered;
```

Z liczb można korzystać także na wiele innych sposobów (liczne z nich poznasz w omówieniu rozpoczynającym się na stronie 587), jednak najczęściej używane są podstawowe operatory wymienione w tabeli 2.2.

Kolejność wykonywania operacji

Przy przeprowadzaniu kilku operacji matematycznych — na przykład sumowaniu kilku liczb i mnożeniu ich wszystkich przez 10 — trzeba pamiętać o kolejności, w jakiej interpreter wykonuje działania. Niektóre operatory mają pierwszeństwo przed innymi, dlatego brak staranności może prowadzić do powstawania niepożądanych wyników, na przykład:

```
4 + 5 * 10
```

Może się wydawać, że program wykona operacje od lewej do prawej: 4+5 to 9, a 9*10 to 90. Jednak to nieprawda. Najpierw wykonywane jest mnożenie, dlatego wynik instrukcji to 54 (5*10, czyli 50, plus 4). Mnożenie (znak *) i dzielenie (znak /) mają pierwszeństwo przed dodawaniem (+) i odejmowaniem (-).

Aby mieć pewność, że program wykona działania zgodnie z oczekiwaniami, należy pogrupować operacje za pomocą nawiasów. Poprzednią instrukcję można zapisać w następujący sposób:

```
(4 + 5) * 10
```

Działania w nawiasach są wykonywane jako pierwsze, dlatego program doda liczby 4 i 5, a wynik (9) pomnoży przez 10. Jeśli jednak programista chce wykonać mnożenie jako pierwsze, bardziej jednoznaczny będzie następujący zapis:

```
4 + (5*10);
```

Łączenie łańcuchów znaków

Łączenie łańcuchów znaków w jeden ciąg to standardowa operacja programistyczna. Jeśli na przykład strona zawiera formularz, który w jednym polu pobiera imię użytkownika, a w innym — jego nazwisko, można połączyć zawartość obu pól. Ponadto jeżeli program ma wyświetlać wiadomość informującą o przesłaniu danych, należy połączyć imię i nazwisko z ogólnym komunikatem: „Użytkownik Jan Kowalski przesłał dane".

Łączenie łańcuchów znaków to tak zwana *konkatenacja*, a służy do niej operator +. Jest to ten sam operator, który pozwala dodawać liczby, jednak w przypadku łańcuchów znaków działa nieco inaczej. Oto przykład:

```
var firstName = 'Jan';
var lastName = 'Kowalski';
var fullName = firstName + lastName;
```

W ostatnim wierszu powyższego kodu zawartość zmiennej `firstName` jest łączona z wartością zmiennej `lastName`. Program dosłownie łączy je ze sobą i umieszcza wynik w zmiennej `fullName`. Tu nowy łańcuch ma wartość `JanKowalski`. Brakuje w niej odstępu między imieniem a nazwiskiem, ponieważ konkatenacja polega tylko na łączeniu łańcuchów. W wielu programach (na przykład w tym) trzeba dodać odstęp między łączonymi łańcuchami:

```
var firstName = 'Jan';
var lastName = 'Kowalski';
var fullName = firstName + ' ' + lastName;
```

Fragment `' '` w ostatnim wierszu to apostrof, odstęp i apostrof. Jest to po prostu łańcuch znaków zawierający odstęp. Umieszczenie go między zmiennymi użytymi w przykładzie powoduje utworzenie łańcucha `Jan Kowalski`. Ten kod ilustruje też, że jednocześnie można połączyć więcej niż dwa łańcuchy znaków, tak jak w przedstawionym przypadku, gdzie były trzy.

Uwaga: Trzeba pamiętać, że zmienna jest jedynie pojemnikiem, który może zawierać dane dowolnego typu, na przykład łańcuch znaków lub liczbę. Jeśli zatem łączymy dwie zmienne zawierające łańcuchy znaków (na przykład firstName + lastName), w rzeczywistości odpowiada to połączeniu dwóch łańcuchów ('Jan' + 'Kowalski').

Łączenie liczb i łańcuchów znaków

Większość operatorów matematycznych działa tylko na liczbach. Na przykład nie ma sensu mnożenie liczby `2` przez słowo `'pies'`. Próba wykonania takiej operacji spowoduje zwrócenie specjalnej wartości języka JavaScript, `NaN` (ang. *not a number*, czyli nie liczba). Jednak czasem programista chce połączyć łańcuch znaków z liczbą, na przykład aby wyświetlić, ile razy użytkownik odwiedził witrynę. Ta wartość to *liczba*, a treść wiadomości to *łańcuch znaków*. Operator + wykonuje w takim działaniu dwie operacje: przekształca liczbę na łańcuch znaków i łączy ją z drugim łańcuchem:

```
var numOfVisits = 101;
var message = 'Liczba odwiedzin: ' + numOfVisits + ' razy.';
```

Zmienna `message` przyjmuje wartość „Liczba odwiedzin: 101 razy.". Interpreter wykrywa obecność łańcucha znaków i na tej podstawie ustala, że nie są potrzebne działania matematyczne (dodawanie). W zamian traktuje znak + jak operator konkatenacji i przekształca liczbę na łańcuch znaków.

Na pozór kod ten ilustruje wygodny sposób wyświetlania słów i liczb w jednym komunikacie. W tym przykładzie oczywiste jest, że liczba to część łańcucha znaków i wiadomości, dlatego interpreter przekształca liczbę na łańcuch.

Ten mechanizm, nazywany *automatyczną konwersją typów*, może jednak prowadzić do problemów. Jeśli użytkownik wpisze w formularzu w odpowiedzi na pytanie „Ile par butów chcesz zamówić?" liczbę (na przykład 2), program potraktuje dane jak łańcuch znaków. Przyjrzyj się następnemu fragmentowi kodu:

```
var numOfShoes = '2';
var numOfSocks = 4;
var totalItems = numOfShoes + numOfSocks;
```

Można się spodziewać, że zmienna `totalItems` przyjmie wartość 6 (2 pary butów+ 4 pary skarpet). Jednak ponieważ zmienna `numOfShoes` zawiera łańcuch znaków, interpreter przekształci na łańcuch także wartość zmiennej `numOfSocks` i zapisze w zmiennej `totalItems` tekst `'24'`. Istnieje kilka rozwiązań tego problemu.

Po pierwsze, można dodać znak + przed *łańcuchem znaków* zawierającym liczbę:

```
var numOfShoes = '2';
var numOfSocks = 4;
var totalItems = +numOfShoes + numOfSocks;
```

Dodanie znaku + przed zmienną (elementów tych nie może oddzielać odstęp) nakazuje interpreterowi próbę konwersji łańcucha znaków na liczbę. Jest to możliwe, jeśli łańcuch zawiera liczbę, na przykład `'2'`. Wynikiem dodawania będzie wtedy 6 (2+4). Inna technika polega na użyciu polecenia `Number()`:

```
var numOfShoes = '2';
var numOfSocks = 4;
var totalItems = Number(numOfShoes) + numOfSocks;
```

Polecenie `Number()` przekształca łańcuch na liczbę (jeśli jest to możliwe). Jeżeli łańcuch zawiera same litery, polecenie to zwróci wartość `NaN`, aby poinformować, że nie może przekształcić liter na liczbę.

Liczby w łańcuchach znaków najczęściej pojawiają się przy pobieraniu danych od użytkowników za pomocą formularza. Dlatego jeśli program ma dodawać wartości wpisane przez internautów, warto najpierw przekazać pobrane dane do polecenia `Number()`.

Uwaga: Problem ten pojawia się wyłącznie w przypadku dodawania liczby do łańcucha znaków zawierającego liczbę. Przy próbie pomnożenia zmiennej numOfShoes przez zmienną zawierającą liczbę — shoePrice — interpreter JavaScript skonwertuje łańcuch przechowywany w zmiennej numOfShoes na liczbę i pomnoży ją przez wartość zmiennej shoePrice.

Zmienianie wartości zmiennych

Zmienne są przydatne, ponieważ mogą przechowywać wartości zmieniające się w czasie działania programu, na przykład wynik gry. Jak można zmodyfikować wartość zmiennej? Jeśli ma to być nowa wartość, można po prostu przypisać ją do zmiennej:

```
var score = 0;
score = 100;
```

Jednak często programista chce zachować wartość zmiennej i dodać coś do niej lub zmodyfikować ją w inny sposób. W czasie gry zwykle nie należy przypisywać do zmiennej nowego wyniku, a jedynie dodawać punkty do poprzedniej wartości lub odejmować je. Aby dodać liczbę do zmiennej, należy w działaniu użyć jej nazwy:

```
var score = 0;
score = score + 100;
```

Ostatni wiersz kodu może wydawać się skomplikowany, jednak użyto tu bardzo popularnej techniki. Wszystkie operacje zachodzą najpierw po prawej stronie znaku =. Program pobiera wartość zapisaną w zmiennej `score` (0) i dodaje do niej `100`. Wynik tej operacji jest *następnie* zapisywany w zmiennej `score`. Efekt działania tego kodu to przypisanie wartości `100` do zmiennej `score`.

W ten sam sposób działają też inne operacje matematyczne, na przykład odejmowanie, dzielenie i mnożenie:

```
score = score - 10;
score = score * 10;
score = score / 10;
```

Programiści niezwykle często wykonują działania na zmiennej i zapisują w niej wynik, dlatego dostępny jest skrócony zapis tego procesu dla czterech podstawowych operatorów matematycznych. Opis tych skrótów znajdziesz w tabeli 2.3.

Tabela 2.3. *Skróty do wykonywania operacji matematycznych na zmiennych*

Operator	Działanie	Jak go używać	Odpowiednik
`+=`	Dodaje wartość podaną po prawej stronie znaku równości do zmiennej podanej po stronie lewej.	`score += 10;`	`score = score + 10;`
`-=`	Odejmuje wartość podaną po prawej stronie znaku równości od zmiennej podanej po stronie lewej.	`score -= 10;`	`score = score – 10;`
`*=`	Mnoży wartość podaną po prawej stronie znaku równości przez zmienną podaną po stronie lewej.	`score *= 10;`	`score = score * 10;`
`/=`	Dzieli wartość podaną po prawej stronie znaku równości przez zmienną podaną po stronie lewej.	`score /= 10;`	`score = score / 10;`
`++`	Jeśli znajduje się bezpośrednio przed nazwą zmiennej, dodaje do niej 1.	`score++;`	`score = score + 1;`
`--`	Jeśli znajduje się bezpośrednio przed nazwą zmiennej, odejmuje od niej 1.	`score--;`	`score = score - 1;`

Te same reguły obowiązują przy dołączaniu do zmiennej łańcucha znaków. Jeśli na przykład zmienna zawiera łańcuch, można dodać do niej kilka następnych łańcuchów znaków:

```
var name = 'Maciej';
var message = 'Witaj';
message = message + ' ' + name;
```

Podobnie jak przy operacjach na liczbach, dostępny jest operator skrócony, który służy do dołączania łańcuchów znaków do zmiennej. Operator += dodaje łańcuch znaków podany po prawej stronie znaku = na koniec łańcucha zapisanego w danej zmiennej. Dlatego ostatni wiersz ostatniego przykładu można zapisać także w następujący sposób:

```
message += ' ' + name;
```

Z operatora += będziesz często korzystał przy używaniu łańcuchów znaków i w trakcie czytania tej książki.

Przykład — używanie zmiennych do tworzenia komunikatów

W tym przykładzie użyjesz zmiennych do wyświetlenia (czyli zapisania) komunikatu na stronie WWW.

Uwaga: Aby wykonać przykłady z tego rozdziału, pobierz pliki ze strony poświęconej książce w witrynie *helion.pl*. Szczegółowe informacje o tych plikach znajdziesz na stronie 46.

1. **W edytorze tekstu otwórz plik *use_variable.html* z katalogu *R02*.**

   Ta strona to prosty plik HTML wzbogacony o styl CSS. Dokument nie zawiera jeszcze kodu JavaScript. W następnych krokach użyjesz zmiennych, aby wyświetlić komunikat na stronie.

2. **Znajdź znacznik `<h1>` (na początku drugiej połowy pliku), a następnie dodaj otwierający i zamykający znacznik `<script>`:**

   ```
   <h1>Używanie zmiennych</h1>
   <script>

   </script>
   ```

   Prawdopodobnie znasz już ten kod HTML. Przygotowuje on na stronie miejsce na skrypt.

Uwaga: Na tej stronie korzystamy z deklaracji doctype typowej dla języka HTML5. Gdybyśmy używali języków XHTML 1.0 lub HTML 4.01, konieczne byłoby dodanie do znacznika <script> atrybutu type="text/javascript": <script type="text/javascript">. Modyfikacja ta nie jest konieczna do zapewnienia poprawności działania skryptu, a jedynie po to, by strona przechodziła testy zgodności ze standardem wykonywane przy użyciu narzędzia W3C Validator (więcej informacji na ten temat można znaleźć na stronie 23).

3. **Między znacznikami `<script>` wpisz następujący kod:**

   ```
   var firstName = 'Ciasteczkowy';
   var lastName = 'Potwór';
   ```

   Właśnie utworzyłeś dwie pierwsze zmienne — `firstName` i `lastName` — oraz zapisałeś w nich łańcuchy znaków. Później dodasz do siebie (lub połączysz) te łańcuchy i wyświetlisz wynik na stronie.

4. **Pod deklaracjami zmiennych wpisz następujący kod:**

   ```
   document.write('<p>');
   ```

   Jak dowiedziałeś się w rozdziale 1., polecenie `document.write()` dodaje tekst bezpośrednio do strony. Tu posłużyło ono do zapisania znacznika HTML. Do polecenia należy przekazać łańcuch (`'<p>'`), a zostanie on zapisany na stronie, tak jakbyś sam umieścił go w kodzie HTML. Umieszczanie znaczników HTML w poleceniu `document.write()` jest poprawną techniką. Tu JavaScript doda otwierający znacznik akapitu, w którym znajdzie się wyświetlany na stronie tekst.

Uwaga: Polecenie `document.write()` stanowi prosty sposób, by przyzwyczaić się do programowania w języku JavaScript, choć istnieją inne, bardziej wydajne metody dodawania kodu HTML do stron. Kiedy zdobędziesz nieco więcej doświadczenia, będziesz mógł przejść do bardziej zaawansowanych technik, w tym także kilku naprawdę wygodnych, udostępnianych przez bibliotekę jQuery (poznasz je na stronie 157).

5. **Wciśnij klawisz *Enter* i wpisz następujący kod JavaScript:**

   ```
   document.write(firstName + ' ' + lastName);
   ```

   W tej instrukcji wykorzystano wartości zapisane w zmiennych w kroku 3. Operator + umożliwia połączenie kilku łańcuchów znaków w jeden długi łańcuch,

który polecenie `document.write()` zapisuje w kodzie HTML strony. Tu program łączy wartość zmiennej `firstName` (`'Ciasteczkowy'`) z odstępem i wartością zmiennej `lastName` (`'Potwór'`). Wynik to jeden łańcuch znaków: `'Ciasteczkowy Potwór'`.

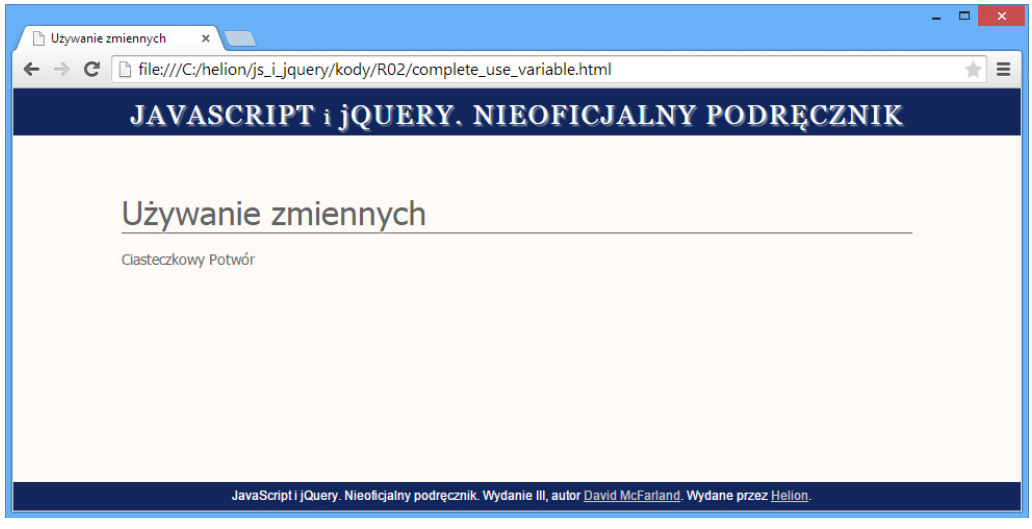
6. **Ponownie wciśnij klawisz *Enter* i wpisz instrukcję `document.write('</p>');`.**

   Gotowy skrypt powinien wyglądać następująco:

```
<script type="text/javascript">
var firstName = 'Ciasteczkowy';
var lastName = 'Potwór';
document.write('<p>');
document.write(firstName + ' ' + lastName);
document.write('</p>');
</script>
```

7. **Wyświetl stronę w przeglądarce, aby zobaczyć efekty swej pracy (patrz rysunek 2.2).**

   Słowa „Ciasteczkowy Potwór" powinny pojawić się pod nagłówkiem „Używanie zmiennych". Jeśli nie widzisz tekstu, prawdopodobnie w kod wkradła się literówka. Porównaj przedstawiony wcześniej skrypt z wprowadzonym kodem i przypomnij sobie wskazówki ze strony 51, dotyczące diagnozowania skryptów w przeglądarkach Firefox, Safari, Chrome oraz IE9.



Rysunek 2.2. *Choć pewnie nie sięgnąłeś po tę książkę, aby nauczyć się wyświetlać tekst „Ciasteczkowy Potwór", przykładowy skrypt lustruje ważne zagadnienie — tworzenie i używanie zmiennych w języku JavaScript*

8. **Ponownie otwórz edytor tekstu i zmień drugi wiersz skryptu na:**

```
var lastName = 'Wieczór';
```

Zapisz stronę i wyświetl ją w przeglądarce. Gotowe. Teraz komunikat to „Ciasteczkowy Wieczór". Działająca wersja tego skryptu z komunikatem „Ciasteczkowy Potwór" znajduje się w pliku *complete_use_variable.html*.

Przykład — pobieranie informacji

Poprzedni skrypt ilustrował tworzenie zmiennych, jednak nie dowiedziałeś się, jak za pomocą zmiennych wyświetlać użytkownikom wyjątkowe, dopasowane wiadomości. W tym przykładzie zobaczysz, jak używać polecenia `prompt()` do pobierania danych od użytkownika i zmieniać treść strony na podstawie uzyskanych informacji.

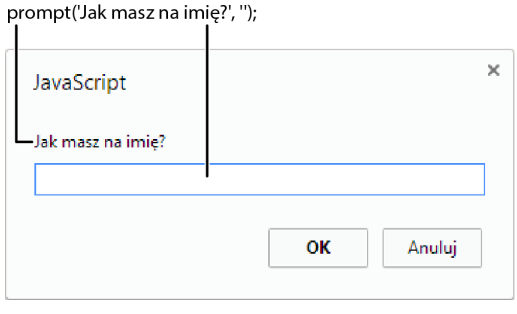
1. **Otwórz w edytorze tekstu plik *prompt.html* z katalogu *R02*.**

   Aby przyspieszyć pracę, w pliku umieszczono już znaczniki `<script>`. Zauważ, że dokument zawiera dwie pary takich znaczników. Jedna znajduje się w sekcji nagłówkowej, a druga — w ciele strony. Kod JavaScript ma wykonywać dwie operacje. Po pierwsze, otwierać okno dialogowe z prośbą do użytkownika o udzielenie odpowiedzi. Po drugie, wyświetlać w ciele strony komunikat dostosowany do odpowiedzi internauty.

2. **Między pierwszymi znacznikami <script> (w sekcji nagłówkowej) wpisz pogrubiony kod:**

```
<script>
var name = prompt('Jak masz na imię?', '');
</script>
```

   Funkcja `prompt()` wyświetla okno dialogowe podobne do okienka otwieranego przez funkcję `alert()`. Jednak funkcja `prompt()` nie tylko wyświetla komunikat, ale też pobiera odpowiedź (patrz rysunek 2.3). Ponadto aby użyć funkcji `prompt()`, należy podać w nawiasach dwa łańcuchy znaków rozdzielone przecinkiem. Na rysunku 2.3 pokazano, do czego służą te łańcuchy. Pierwszy pojawia się jako pytanie (tu jest to „Jak masz na imię?").



Rysunek 2.3. *Wydanie polecenia prompt() to jeden ze sposobów na pobranie danych od użytkownika. W tym poleceniu należy podać dwa łańcuchy znaków. Pierwszy z nich pojawia się jako pytanie, a drugi jest widoczny w polu odpowiedzi*

Drugi łańcuch pojawia się w polu, w którym użytkownik wprowadza dane. Tu użyto *pustego łańcucha znaków*, czyli dwóch apostrofów, dlatego pole tekstowe jest puste (zgodnie z informacjami podanymi na stronie 61, łańcuchy znaków można zapisywać zarówno w apostrofach, jak i cudzysłowach). Można jednak podać przydatne instrukcje, na przykład: „Tu wpisz swoje imię", a program wyświetli je we wspomnianym polu. Niestety, użytkownik będzie musiał najpierw usunąć ten tekst przed podaniem odpowiedzi.

Funkcja `prompt()` pobiera łańcuch znaków z tekstem wprowadzonym przez użytkownika w oknie dialogowym. Dodany wiersz kodu JavaScript zapisuje ten tekst w nowej zmiennej — `name`.

Uwaga: Wiele funkcji *zwraca* wartość. Po polsku oznacza to, że funkcja przekazuje pewne informacje po zakończeniu działania. Można pominąć te dane lub zapisać je w zmiennej w celu ich późniejszego wykorzystania. Tu funkcja `prompt()` zwraca łańcuch znaków, a program zapisuje go w zmiennej `name`.

3. **Zapisz stronę i wyświetl ją w przeglądarce.**

   W czasie wczytywania strony zobaczysz okno dialogowe. Do momentu jego wypełnienia i kliknięcia przycisku *OK* nic się nie stanie — nie pojawi się nawet strona WWW. Ponadto także po kliknięciu przycisku *OK* program nie wykonuje na razie żadnych operacji, ponieważ wprowadzony fragment tylko pobiera i zapisuje odpowiedź. Teraz trzeba dodać kod, który wykorzystuje te dane na stronie.

4. **Wróć do edytora tekstu. Znajdź drugą parę znaczników `<script>` i dodaj kod wyróżniony pogrubieniem:**

   ```
   <script>
   document.write('<p>Witaj, ' + name + '</p>');
   </script>
   ```

   Ten wiersz używa informacji podanych przez użytkownika. Podobnie jak skrypt ze strony 73, ten kod łączy kilka łańcuchów znaków — otwierający znacznik akapitu, tekst, wartość zmiennej i zamykający znacznik akapitu — i wyświetla efekt tej operacji na stronie.

5. **Zapisz stronę i wyświetl ją w przeglądarce.**

   Kiedy zobaczysz okno dialogowe, podaj imię, a następnie kliknij przycisk *OK*. Tym razem wpisany tekst pojawi się na stronie (patrz rysunek 2.4). Odśwież stronę i wprowadź nowe dane, a tekst się zmieni. Właśnie tak powinny działać zmienne.

Rysunek 2.4. *Oto siła zmiennych — strona dostosowuje tekst do odpowiedzi użytkownika*

Tablice

Proste zmienne, takie jak opisane w poprzednim podrozdziale, przechowują tylko pojedyncze informacje, na przykład liczby lub łańcuchy znaków. Takie zmienne doskonale nadają się do zapisywania pojedynczych wartości, na przykład wyniku, wieku lub łącznej ceny zakupów. Jednak jeśli program ma przechowywać grupę powiązanych elementów, na przykład nazwy dni tygodnia lub listę rysunków na stronie WWW, proste zmienne nie są zbyt wygodne.

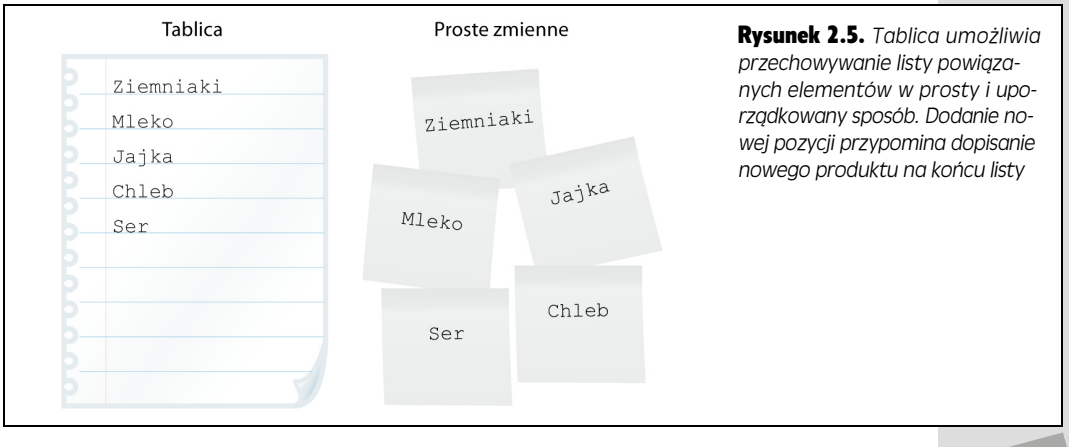
Wyobraź sobie, że masz utworzyć w języku JavaScript system obsługi koszyka zakupów, który zapisuje produkty zamawiane przez użytkownika. Gdyby program miał przechowywać każdy towar za pomocą prostych zmiennych, kod wyglądałby następująco:

```
var item1 = 'Xbox 360';
var item2 = 'Buty do tenisa';
var item3 = 'Bony podarunkowe';
```

Co się jednak stanie, jeśli użytkownik zechce kupić więcej rzeczy? Trzeba wtedy dodać następne zmienne — `item4`, `item5` i tak dalej. Ponieważ nie wiadomo, ile produktów internauta zechce kupić, nie można ustalić, ile zmiennych będzie potrzebnych.

Na szczęście JavaScript udostępnia *tablice*, które umożliwiają wygodne przechowywanie list elementów. Tablica pozwala zapisać w jednym miejscu więcej niż jedną wartość i przypomina nieco listę zakupów. Kiedy idziesz do sklepu, siadasz i przygotowujesz listę produktów. Kilka dni wcześniej lista mogła zawierać tylko kilka rzeczy, ale jeśli lodówka jest pusta, liczba pozycji na liście może być dość duża. Zawsze jednak lista jest tylko jedna.

Bez tablic każdemu elementowi listy musi odpowiadać odrębna zmienna. Wyobraź sobie, że nie możesz zapisać wszystkich produktów na jednej liście, ale musisz nosić stos kartek z zapisanymi pojedynczymi rzeczami. Jeśli chcesz dodać nowy produkt, musisz użyć następnej kartki. Musisz też kontrolować je wszystkie w trakcie zakupów (patrz rysunek 2.5). W ten sposób działają proste zmienne, natomiast tablice umożliwiają utworzenie jednej listy elementów oraz dodawanie, usuwanie i modyfikowanie ich w dowolnym momencie.



Rysunek 2.5. *Tablica umożliwia przechowywanie listy powiązanych elementów w prosty i uporządkowany sposób. Dodanie nowej pozycji przypomina dopisanie nowego produktu na końcu listy*

Tworzenie tablic

Aby utworzyć tablicę i zapisać w niej elementy, trzeba najpierw zadeklarować nazwę tablicy (podobnie jak przy tworzeniu zmiennych), a następnie podać listę oddzielonych przecinkami wartości. Każda wartość reprezentuje jeden element listy. Tablicę możesz nazwać w dowolny sposób (podobnie jak zmienną), jednak musisz przestrzegać reguł podanych na stronie 71. Przy tworzeniu tablicy należy umieścić listę elementów między otwierającym a zamykającym nawiasem kwadratowym — `[]`. Aby na przykład przygotować tablicę ze skrótowymi nazwami dni tygodnia, można użyć następującego kodu:

```
var days = ['Pn.', 'Wt.', 'Śr.', 'Czw.', 'Pt.', 'Sob.', 'Ndz.'];
```

Nawiasy kwadratowe — `[]` — są tu bardzo istotne. Informują interpreter o tym, że natrafił na tablicę. Można też utworzyć pustą tablicę, która nie zawiera żadnych elementów:

```
var playList = [];
```

Tworzenie pustych tablic przypomina deklarowanie zmiennych w sposób opisany na stronie 63. Pusta tablica powstaje, jeśli program nie dodaje do niej elementów do momentu uruchomienia kodu. Powyższa tablica może posłużyć na przykład do zapisywania piosenek wybranych z listy na stronie WWW. Nie wiadomo z góry, które utwory zaznaczy użytkownik, dlatego należy przygotować pustą tablicę, a następnie zapełnić ją elementami. (Dodawanie danych do tablic opisano na stronie 80).

Uwaga: W czasie analizy cudzych programów w języku JavaScript możesz natrafić na inny sposób tworzenia tablic, który wymaga użycia słowa kluczowego `Array`:

```
var days = new Array('Pn.', 'Wt.', 'Śr.');
```

To prawidłowa metoda, jednak technikę użytą w tej książce (*literały tablicowe*) preferują profesjonaliści, gdyż jest bardziej zwięzła, wymaga mniej pisania, a dodatkowo jest elegantsza.

W tablicach można przechowywać wartości różnego typu. Oznacza to, że w jednej tablicy mogą znajdować się liczby, łańcuchy znaków i wartości logiczne:

```
var prefs = [1, -30.4, 'www.oreilly.com', false];
```

Uwaga: W tablicy można umieszczać nawet inne tablice i obiekty. Technika ta pomaga przechowywać złożone dane.

Wcześniejsze tablice tworzył jeden wiersz kodu. Jednak jeśli programista chce dodać wiele pozycji lub elementy to długie łańcuchy znaków, umieszczenie całej instrukcji w jednym wierszu utrudni czytanie programu. Dlatego wielu programistów tworzy tablice w kilku wierszach:

```
var authors = [ 'Ernest Hemingway',
                'Charlotte Bronte',
                'Dante Alighieri',
                'Emily Dickinson'
              ];
```

Jak wspomniano w ramce na stronie 67, interpreter pomija dodatkowe odstępy i znaki karetki, dlatego choć kod zajmuje pięć wierszy, jest traktowany jak instrukcja jednowierszowa, na co wskazuje średnik na jej końcu.

Wskazówka: Aby zapisać imiona i nazwiska jedno pod drugim, wpisz pierwszy wiersz (`var authors = [ 'Ernest Hemingway',`), wciśnij klawisz *Enter*, a następnie wciśnij kilka razy klawisz spacji, aby odpowiednio umiejscowić następną wartość (`'Charlotte Bronte',`).

Używanie elementów tablicy

Dostęp do zawartości prostych zmiennych można uzyskać przez podanie ich nazw. Na przykład instrukcja `alert(lastName)` otwiera okno dialogowe z wartością zapisaną w zmiennej `lastName`. Jednak tablica może zawierać wiele wartości, zatem nie wystarczy użyć jej nazwy, aby uzyskać dostęp do poszczególnych elementów. Pozycję każdego elementu w tablicy określa niepowtarzalna liczba — *indeks*. Aby uzyskać dostęp do wybranego elementu, należy podać jego indeks. W celu wyświetlenia okna dialogowego z pierwszym elementem tablicy ze skrótami nazw dni tygodnia należy użyć następującego kodu:

```
var days = ['Pn.', 'Wt.', 'Śr.', 'Czw.', 'Pt.', 'Sob.', 'Ndz.'];
alert(days[0]);
```

Ten kod otwiera okno dialogowe ze skrótem „Pn.". Tablice *są indeksowane od zera*, co oznacza, że *pierwszy* element ma indeks `0`, a *drugi* — indeks `1`. Dlatego trzeba odjąć 1 od miejsca elementu na liście, aby uzyskać jego indeks. Indeks elementu piątego to 5 – 1, czyli `4`. Indeksowanie od zera może być mylące dla początkujących programistów, dlatego w tabeli 2.4 znajdziesz indeksy i wartości elementów tablicy `days` z poprzedniego przykładu oraz sposób uzyskania do nich dostępu.

Tabela 2.4. *Aby uzyskać dostęp do elementów tablicy, należy użyć indeksu. Odpowiada on pozycji elementu na liście pomniejszonej o 1*

Wartość indeksu	Element	Dostęp do elementu
0	Pn.	`days[0]`
1	Wt.	`days[1]`
2	Śr.	`days[2]`
3	Czw.	`days[3]`
4	Pt.	`days[4]`
5	Sob.	`days[5]`
6	Ndz.	`days[6]`

Aby zmodyfikować element tablicy, należy przypisać mu nową wartość za pomocą indeksu. Na przykład w celu umieszczenia nowej wartości w pierwszym elemencie tablicy `days` trzeba użyć następującej instrukcji:

```
days[0] = 'Poniedziałek';
```

Ponieważ indeks ostatniego elementu jest zawsze o 1 mniejszy od liczby wszystkich pozycji tablicy, aby uzyskać dostęp do ostatniej wartości, wystarczy ustalić długość tablicy. Jest to łatwe zadanie, ponieważ każda tablica ma właściwość `length` (długość), określającą liczbę elementów. Aby uzyskać dostęp do tej właściwości, należy dodać kropkę i słowo `length` po nazwie tablicy. Na przykład wyrażenie `days.length`

zwraca liczbę elementów tablicy o nazwie `days`. Jeśli utworzyłeś tablicę o innej nazwie, takiej jak `playList`, możesz sprawdzić jej długość za pomocą instrukcji `playList.length`. Dlatego aby uzyskać dostęp do wartości ostatniego elementu listy, można użyć pomysłowego kodu:

```
days[days.length-1]
```

Ten fragment pokazuje, że indeks nie musi być literałem liczbowym, takim jak `0` w instrukcji `days[0]`. Można też podać równanie, które zwraca poprawną liczbę. Takim krótkim równaniem jest na przykład `days.length - 1`. Program najpierw pobiera liczbę elementów tablicy `days` (tu jest to 7), a następnie odejmuje od niej 1. Dlatego instrukcja `days[days.length-1]` oznacza `days[6]`.

Jako indeksu można też użyć zmiennej liczbowej:

```
var i = 0;
alert(days[i]);
```

Ostatni wiersz tego kodu to odpowiednik instrukcji `alert(days[0]);`. Technika ta jest szczególnie przydatna w pętlach, co opisano w następnym rozdziale (patrz strona 111).

Dodawanie elementów do tablicy

Załóżmy, że program zawiera tablicę do zapisywania elementów klikniętych przez użytkownika na stronie WWW. Każde kliknięcie ma powodować dodanie elementu do tablicy. JavaScript umożliwia dodawanie danych do tablicy na kilka sposobów.

Dodawanie elementów na koniec tablicy

Aby dodać element na koniec tablicy, można użyć notacji indeksowej omówionej na stronie 79 i podać indeks o 1 większy od ostatniej pozycji. Przyjrzyj się tablicy `properties`:

```
var properties = ['red', '14px', 'Arial'];
```

Na tym etapie tablica ma trzy elementy. Pamiętaj, że ostatni element ma indeks o 1 mniejszy od liczby wszystkich wartości. W przykładowej tablicy dostęp do takiego elementu zapewnia instrukcja `properties[2]`. Aby dodać nową wartość, należy użyć poniższej składni:

```
properties[3] = 'bold';
```

Ten wiersz kodu przypisuje wartość `'bold'` do czwartej pozycji, co powoduje utworzenie tablicy o czterech elementach: `['red', '14px', 'Arial', 'bold']`. Zauważ, że do dodania nowego elementu posłużył indeks o wartości równej liczbie wszystkich elementów tablicy. Dlatego można zawsze dodać wartość na koniec tablicy przez podanie jako indeksu właściwości `length`. Ostatnią instrukcję można przekształcić w następujący sposób:

```
properties[properties.length] = 'bold';
```

Można też użyć polecenia `push()`, które dodaje na koniec tablicy wartość podaną w nawiasach. Polecenie `push()`, podobnie jak właściwość `length`, należy podać po nazwie tablicy i kropce. Oto następny sposób na dodanie elementu na koniec tablicy `properties`:

```
properties.push('bold');
```

Ta instrukcja dodaje wartość umieszczoną w nawiasach (tu jest to łańcuch znaków `'bold'`) na koniec tablicy. W ten sposób można dodać wartość dowolnego rodzaju: łańcuch znaków, liczbę, wartość logiczną, a nawet zmienną.

Zaletą polecenia `push()` jest to, że umożliwia dodanie grupy elementów. Jeśli chcesz wstawić na koniec tablicy `properties` trzy nowe wartości, możesz to zrobić w następujący sposób:

```
properties.push('bold', 'italic', 'underlined');
```

Dodawanie elementów na początek tablicy

Jeśli chcesz dodać element na początek tablicy, możesz użyć polecenia `unshift()`. Poniższy kod dodaje wartość `'bold'` na początek tablicy `properties`:

```
var properties = ['red', '14px', 'Arial'];
properties.unshift('bold');
```

Po wykonaniu tego kodu tablica `properties` będzie zawierać cztery elementy: `['bold', 'red', '14px', 'Arial']`. Polecenie `unshift()`, podobnie jak `push()`, pozwala dodać kilka elementów:

```
properties.unshift('bold', 'italic', 'underlined');
```

Uwaga: Pamiętaj o podaniu *nazwy* tablicy i kropki przed metodą, której chcesz użyć. Instrukcja `push('nowy element')` jest nieprawidłowa, gdyż interpreter JavaScriptu nie otrzymał informacji, do której tablicy należy dodać element. Trzeba najpierw podać nazwę tablicy, następnie kropkę, a dopiero na końcu metodę, na przykład `authors.push('Stephen King');`.

Wybór sposobu dodawania elementów

Do tej pory poznałeś trzy sposoby dodawania elementów do tablicy. W tabeli 2.5 znajduje się porównanie tych technik. Wszystkie pełnią podobne funkcje, dlatego wybór jednej z nich zależy od specyfiki programu. Jeśli kolejność elementów w tablicy jest nieistotna, można użyć dowolnej metody. Wyobraź sobie stronę ze zdjęciami produktów, które należy kliknąć, aby dodać przedmiot do koszyka zakupów i zapisać go w tablicy. Kolejność produktów w koszyku (i tablicy) nie ma znaczenia, dlatego można użyć każdej z omówionych technik.

Tabela 2.5. *Różne sposoby dodawania elementów do tablicy*

Metoda	Wyjściowa tablica	Przykładowy kod	Wynikowa tablica	Opis
Właściwość **`length`**	`var p = [0,1,2,3]`	`p[p.length]=4`	`[0,1,2,3,4]`	Dodaje jedną wartość na koniec tablicy.
`push()`	`var p = [0,1,2,3]`	`p.push(4,5,6)`	`[0,1,2,3,4,5,6]`	Dodaje jeden lub więcej elementów na koniec tablicy.
`unshift()`	`var p = [0,1,2,3]`	`p.unshift(4,5)`	`[4,5,0,1,2,3]`	Wstawia jeden lub więcej elementów na początek tablicy.

PORADNIA DLA ZAAWANSOWANYCH

Tworzenie kolejek

Metody używane do dodawania (`push()` i `unshift()`) oraz usuwania (`pop()` i `shift()`) elementów są często stosowane razem, aby zapewnić dostęp do elementów w kolejności dołączania ich do tablicy. Klasycznym przykładem jest lista odtwarzania. Powstaje ona przez dodawanie utworów, a każda wysłuchana piosenka jest usuwana z listy. Nagrania są odtwarzane w kolejności dodawania do listy, dlatego program najpierw uruchamia pierwszy utwór, a następnie go usuwa. W taki sposób funkcjonują kolejki, na przykład w kinie. Kiedy widz przychodzi do kina, zajmuje miejsce na końcu kolejki. Gdy drzwi się otwierają, pierwsza osoba w kolejce jako pierwsza wchodzi na salę.

W świecie programowania to podejście ma nazwę kolejka FIFO (ang. *First In, First Out*, czyli pierwszy wchodzi — pierwszy wychodzi). Można zasymulować taki system za pomocą poleceń `push()` i `shift()`. Aby na przykład dodać nowy utwór na koniec tablicy `playlist`, można użyć polecenia `push()`:

```
playlist.push('Yellow Submarine');
```

Aby przejść do piosenki, którą program ma odtworzyć jako następną, należy pobrać pierwszy element listy:

```
nowPlaying = playlist.shift();
```

Ten kod powoduje usunięcie z tablicy pierwszego elementu i zapisanie go w zmiennej `nowPlaying`. Kolejki FIFO są przydatne przy tworzeniu różnych kolejek i zarządzaniu nimi; przykładem mogą być listy odtwarzania, zadania do wykonania lub pokazy slajdów.

Jeśli jednak w tablicy trzeba uwzględnić kolejność elementów, używana metoda jest istotna. Załóżmy, że strona umożliwia użytkownikom tworzenie list odtwarzania przez wybór na stronie tytułów piosenek. Ponieważ takie listy zawierają utwory podane w kolejności ich odtwarzania, uporządkowanie elementów jest istotne. Dlatego za każdym razem, kiedy użytkownik kliknie tytuł nagrania, piosenkę należy umieścić na końcu listy (aby był to ostatni odtwarzany utwór). Wymaga to użycia metody `push()`.

Polecenia `push()` i `unshift()` zwracają wartość. Po wykonaniu zadania przekazują nową liczbę elementów w tablicy:

```
var p = [0,1,2,3];
var totalItems = p.push(4,5);
```

Ten fragment przypisze do zmiennej `totalItems` wartość `6`, ponieważ w tablicy `p` znajdzie się sześć elementów.

Usuwanie elementów z tablicy

Do usuwania elementów z początku i końca tablicy służą polecenia `pop()` i `shift()`. Oba usuwają z tablicy jedną wartość, przy czym polecenie `pop()` z końca, a polecenie `shift()` — z początku. Porównanie tych metod znajduje się w tabeli 2.6.

Tabela 2.6. *Dwa sposoby usuwania elementów z tablicy*

Metoda	Wyjściowa tablica	Przykładowy kod	Wynikowa tablica	Opis
pop()	`var p = [0,1,2,3]`	**p.pop()**	`[0,1,2]`	Usuwa z tablicy ostatni element.
shift()	`var p = [0,1,2,3]`	**p.shift()**	`[1,2,3]`	Usuwa z tablicy pierwszy element.

Polecenia `pop()` i `shift()` po zakończeniu działania zwracają wartość. Co więcej, okazuje się, że jest to wartość elementu usuniętego z tablicy. Dlatego poniższy kod usuwa wartość i zapisuje ją w zmiennej `removedItem`:

```
var p = [0,1,2,3];
var removedItem = p.shift();
```

Po zakończeniu działania kodu wartość zmiennej `removedItem` to `0`, a tablica `p` zawiera liczby `[1,2,3]`.

Uwaga: Wśród przykładowych plików do tego rozdziału znajduje się strona, która umożliwia przetestowanie różnych poleceń związanych z tablicami. Jej nazwa to *array_methods.html*, a plik ten znajduje się w katalogu *Przykłady/testy*. Możesz otworzyć ten plik w przeglądarce i wypróbować różne przyciski, aby zobaczyć, jak działają poszczególne metody. (Przy okazji, interaktywność tej strony to efekt zastosowania języka JavaScript oraz biblioteki jQuery).

Przykład — zapisywanie danych na stronie za pomocą tablic

Tablice są częścią wielu skryptów omawianych w dalszej części książki. Aby szybko zapoznać się z tworzeniem i używaniem tablic, prześledź ten krótki przykład.

Uwaga: Informacje o przykładowych plikach znajdziesz w uwadze na stronie 46.

1. **Otwórz w edytorze tekstu plik *arrays.html* z katalogu *R02*.**

   Najpierw utworzysz tablicę zawierającą cztery łańcuchy znaków. Plik, podobnie jak w poprzednim przykładzie, zawiera już znaczniki `<script>` w sekcji nagłówkowej i ciele strony.

2. **Między pierwszą parą znaczników `<script>` wpisz kod wyróżniony pogrubieniem:**

   ```
   <script>
   var authors = [ 'Ernest Hemingway',
                   'Charlotte Bronte',
                   'Dante Alighieri',
                   'Emily Dickinson'
                 ];
   </script>
   ```

   Ten kod zawiera jedną instrukcję języka JavaScript, jednak zajmuje pięć wierszy. Aby go wprowadzić, wpisz pierwszy wiersz (`var authors = [ 'Ernest Hemingway',`), wciśnij klawisz *Enter*, a następnie wciskaj klawisz spacji do momentu umieszczenia kursora pod znakiem `'` (16 uderzeń) i wpisz łańcuch `'Charlotte Bronte',`.

Uwaga: Większość edytorów wyświetla kody HTML i JavaScript za pomocą czcionek *o stałej szerokości znaków*. Są to na przykład czcionki Courier i Courier New. Wszystkie znaki mają w nich identyczną szerokość, dlatego wyrównywanie kolumn (nazwisk autorów w przykładowym kodzie) jest łatwe. Jeśli edytor nie udostępnia czcionki Courier lub podobnej, precyzyjne wyrównanie tekstu może być niemożliwe.

Na stronie 78 dowiedziałeś się, że przy tworzeniu tablic o wielu elementach można zwiększyć czytelność kodu przez podział instrukcji na kilka wierszy. O tym, że instrukcja jest tylko jedna, informuje brak średników po czterech pierwszych wierszach.

Podana instrukcja tworzy tablicę `authors` i zapisuje w niej nazwiska czterech autorów (cztery łańcuchy znaków). Następny fragment kodu użyje tych elementów.

3. **Znajdź drugą parę znaczników `<script>` i dodaj kod wyróżniony pogrubieniem:**

```
<script>
document.write('<p>Pierwszy autor to <strong>');
document.write(authors[0] + '</strong></p>');
</script>
```

Pierwszy wiersz rozpoczyna nowy akapit z tekstem i otwierającym znacznikiem `<strong>`. Znajduje się tu tylko zwykły kod HTML. Następny wiersz dodaje wartość pierwszego elementu tablicy `authors` oraz zamykające znaczniki `</strong>` i `</p>`, co tworzy kompletny akapit w kodzie HTML. Aby uzyskać dostęp do pierwszego elementu tablicy, należy użyć indeksu `0` (`authors[0]`), a nie `1`.

Na tym etapie możesz zapisać plik i wyświetlić go w przeglądarce. Na ekranie powinien pojawić się tekst „Pierwszy autor to **Ernest Hemingway**". Jeśli ten fragment jest niewidoczny, możliwe, że w kodzie wprowadzonym w krokach 2. i 3. pojawiły się literówki.

Uwaga: Pamiętaj, że do zlokalizowania źródła błędów w kodzie JavaScript możesz użyć opisanej na stronie 51 konsoli błędów przeglądarki.

---

4. **Ponownie otwórz edytor tekstu i dodaj do skryptu dwa poniższe wiersze kodu:**

```
document.write('<p>Ostatni autor to <strong>');
document.write(authors[4] + '</strong></p>');
```

Ten krok przypomina poprzedni, jednak tym razem skrypt wyświetla inny element tablicy. Zapisz stronę i wyświetl ją w przeglądarce. Zamiast imienia i nazwiska autora zobaczysz wartość „undefined" (patrz rysunek 2.6). Nie przejmuj się — kod powinien tak działać. Czy potrafisz wyjaśnić, dlaczego ten kod nie działa?

Pamiętaj, że indeksy tablicy rozpoczynają się od `0`, dlatego indeks ostatniej wartości to liczba wszystkich elementów minus 1. Używana tablica ma cztery wartości, dlatego dostęp do ostatniego elementu da wyrażenie `authors[3]`.

Uwaga: Jeśli spróbujesz wczytać wartość za pomocą nieistniejącego indeksu, otrzymasz wartość `undefined` języka JavaScript. Oznacza ona, że na pozycji określonej przez indeks nie ma zapisanej wartości.

Na szczęście istnieje łatwa technika dostępu do ostatniej wartości tablicy — niezależnie od liczby jej elementów.

5. **Wróć do edytora tekstu i zmodyfikuj wprowadzony wcześniej kod. Usuń indeks `4` i wpisz zamiast niego kod wyróżniony pogrubieniem:**

```
document.write('<p>Ostatni autor to <strong>');
document.write(authors[authors.length-1] + '</strong></p>');
```

Rysunek 2.6. *Jeśli spróbujesz uzyskać dostęp do nieistniejącego elementu tablicy, otrzymasz wartość „undefined"*

W punkcie „Dodawanie elementów do tablicy" na stronie 80. dowiedziałeś się, że właściwość `length` tablic przechowuje liczbę elementów. Dlatego kod `authors.length` zwraca sumę elementów tablicy `authors`. W tym miejscu skryptu jest to liczba 4.

Ponieważ indeks ostatniej wartości tablicy jest zawsze o 1 mniejszy od liczby elementów, wystarczy odjąć od tej liczby 1 (`authors.length-1`), aby uzyskać potrzebny indeks. Przy próbie dostępu do ostatniej wartości tablicy można podać to krótkie działanie jako wartość indeksu — `authors[authors.length-1]`.

Teraz dodasz nowy element na początek tablicy.

6. **Dodaj następny wiersz kodu po fragmencie wprowadzonym w 5. kroku:**

```
authors.unshift('Stan Lee');
```

Na stronie 81 napisano, że metoda `unshift()` dodaje elementy na początek tablicy. Po wykonaniu tej instrukcji tablica `authors` będzie rozpoczynała się od wartości `['Stan Lee', 'Ernest Hemingway', 'Charlotte Bronte', 'Dante Alighieri', 'Emily Dickinson']`.

Kolejny fragment skryptu wyświetli nowy element na stronie.

7. **Dodaj trzy wyróżnione pogrubieniem wiersze, aby kod wyglądał następująco:**

```
document.write('<p>Pierwszy autor to <strong>');
document.write(authors[0] + '</strong></p>');
document.write('<p>Ostatni autor to <strong>');
document.write(authors[authors.length-1] + '</strong></p>');
authors.unshift('Stan Lee');
document.write('<p>Prawie zapomniałem o autorze <strong>');
document.write(authors[0]);
document.write('</strong></p>');
```

Zapisz plik i wyświetl go w przeglądarce. Efekt powinien przypominać stronę z rysunku 2.7. Jeśli ekran wygląda inaczej, pamiętaj, że konsola błędów przeglądarki pomoże Ci znaleźć błąd (patrz strona 51).

Rysunek 2.7. *To prawda, Stan Lee może nie jest wielkim pisarzem, ale pomaga zrozumieć, jak działają tablice*

Krótka lekcja o obiektach

Dotychczas dowiedziałeś się, że na stronie można coś zapisywać przy użyciu funkcji `document.write()`. Wiesz także, że w celu sprawdzenia liczby elementów tablicy należy podać jej nazwę, kropkę oraz słowo `length`, tak jak w `days.length`. Zastanawiasz się zapewne, po co te kropki. Do tej pory uczyłeś się języka JavaScript bez wchodzenia w szczegóły tego elementu składni, jednak właśnie nadszedł czas, by przyjrzeć mu się dokładniej.

Wiele elementów języka JavaScript, podobnie jak wiele elementów stron WWW, można sobie wyobrazić w postaci *obiektów*. Oczywiście, nasz rzeczywisty świat także jest wypełniony obiektami, takimi jak psy czy samochody. Większość z nich składa się z różnych części. Każdy pies ma ogon, głowę i cztery łapy; a samochody mają drzwi, koła, światła, klaksony i tak dalej. Obiekty mogą także wykonywać pewne czynności — samochód może transportować pasażerów, a pies — szczekać. Co więcej, nawet poszczególne części obiektów mogą coś robić; na przykład ogon psa może machać, a klakson może trąbić. W tabeli 2.7 zaprezentowano jeden ze sposobów przedstawienia związków pomiędzy obiektami, ich częściami oraz wykonywanymi przez nie akcjami.

Tabela 2.7. *Uproszczona prezentacja świata*

Obiekt	Części	Akcje
`pies`		`szczekaj`
	`ogon`	`machaj`
`samochód`		`transportuj`
	`klakson`	`trąb`

Także świat języka JavaScript jest wypełniony obiektami, takimi jak okno przeglądarki, dokument, łańcuchy znaków, liczby oraz daty. Podobnie jak obiekty w rzeczywistym świecie, także obiekty języka JavaScript składają się z wielu różnych części. W terminologii programistycznej te części obiektów to *właściwości*. Z kolei akcje, które obiekty mogą wykonywać, są nazywane *metodami*; to funkcje (podobne do wbudowanej funkcji `alert()`) charakterystyczne dla konkretnego obiektu (patrz tabela 2.8).

Tabela 2.8. *Przykładowe metody i właściwości dwóch obiektów JavaScript: document oraz array*

Obiekt	Właściwość	Metoda
`document`	`title`	
	`url`	
		`write()`
`['Kasia', 'Edward', 'Janek']`	`length`	
		`push()`
		`pop()`
		`unshift()`

Uwaga: Metody można bardzo łatwo odróżnić od właściwości obiektów, gdyż zawsze kończą się parą nawiasów, na przykład tak jak `write()`.

Każdy obiekt w języku JavaScript ma własne właściwości i metody. Przykładowo obiekt tablicy dysponuje właściwością `length`, a obiekt `document` udostępnia metodę `write()`. Aby odwołać się do właściwości obiektu lub wywołać jego metodę, używa się zapisu z kropką — i właśnie stąd te kropki! Łączą one obiekty z ich właściwościami lub metodami. Przykładowy fragment kodu w postaci `document.write()` oznacza: wykonaj metodę `write()` obiektu `document`. Gdyby podobnie można było korzystać z obiektów w rzeczywistym świecie, machanie przez psa ogonem zapisalibyśmy w następujący sposób: `pies.ogon.machaj()`. (Oczywiście, psi sposób machania ogonem działa znacznie lepiej).

Podobnie jak można posiadać kilka psów, tak i w programach JavaScript może być używanych wiele wersji tego samego obiektu. Załóżmy na przykład, że przy użyciu poniższego fragmentu kodu utworzymy dwie proste zmienne:

```
var first_name = 'Jacek';
var last_name = 'Kowalski';
```

W rzeczywistości utworzyliśmy dwa różne obiekty *łańcuchów* znaków. Obiekty te dysponują zbiorem własnych właściwości i metod, różniących się od właściwości i metod innych obiektów, takich jak daty (niektóre z nich zostały przedstawione na stronie 593). Po utworzeniu obiektu (czasami operację tę nazywa się także tworzeniem *egzemplarza* lub *instancji* obiektu) można uzyskać dostęp do wszystkich jego właściwości i metod.

Uwaga: Miałeś już okazję spotkać się z innym obiektem — obiektem `window` — reprezentującym okno przeglądarki. Można by go uznać za rodzaj pojemnika, przechowującego stronę WWW wraz z całą zawartością. Przykładowo `alert()` oraz `prompt()` są metodami obiektu `window` i można je także wywoływać w następujący sposób: `window.alert()` oraz `window.prompt()`. Niemniej jednak, ze względu na to, że obiekt `window` zawsze jest dostępny na stronie WWW, odwołanie do niego można pominąć; a zatem wywołania `alert('Witamy')` oraz `window.alert('Witamy')` będą miały taki sam efekt.

Podczas tworzenia nowej zmiennej i zapisywania w niej wartości powstaje nowy egzemplarz konkretnego obiektu. A zatem każda z poniższych instrukcji JavaScript tworzy obiekty różnych typów:

```
var first_name = "Janek"; // obiekt łańcucha znaków (String)
var age = 32; // obiekt liczby
var valid = false; // obiekt Boolean (wartość logiczna)
```

Jeśli zmieni się typ informacji przechowywanej w zmiennej, zmianie ulegnie także typ obiektu. Kiedy na przykład utworzymy zmienną o nazwie `data`, zawierającą początkowo tablicę, a następnie zapiszemy w niej liczbę, zmienimy typ zmiennej z obiektu tablicy na obiekt liczby:

```
var data = false; // obiekt Boolean
data = 32; // zmiana na obiekt liczby
```

Na pierwszy rzut oka pojęcia obiektów, właściwości, metod oraz zapisu z kropką mogą się wydawać dosyć dziwne i trudne. Ponieważ jednak należą one do podstawowych zagadnień związanych z działaniem języka JavaScript, a dodatkowo są powszechnie używane w skryptach korzystających z biblioteki jQuery, bardzo szybko się do nich przyzwyczaisz.

Wskazówka: Język JavaScript definiuje specjalne słowo kluczowe służące do określania typu obiektu (łańcuchów znaków — String, liczb — number, wartości logicznych — Boolean i tak dalej). Jest to operator `typeof`. Umieszcza się go przed nazwą zmiennej, której typ zawartości chcemy poznać. Oto przykład:

```
var data = 32;
alert(typeof data); // w okienku dialogowym zostanie wyświetlone
                    // słowo "number"
data = 'Potwierdzam, zrozumiałem';
alert(typeof data); // w okienku dialogowym zostanie wyświetlone
                    // słowo "string"
```

Podczas dalszej lektury książki powinieneś pamiętać o kilku następujących zagadnieniach.

- W świecie języka JavaScript używanych jest wiele różnych rodzajów obiektów.
- Każdy obiekt ma swoje własne właściwości i metody.
- W celu odwołania się do właściwości obiektu lub wywołania jednej z jego metod używany jest zapis z kropką, na przykład `document.write()`.

Komentarze

Zazwyczaj podczas tworzenia kodu doskonale rozumiesz, jak działa program. Każdy wiersz kodu ma sens i — co najważniejsze — działa! Jednak po miesiącu lub dwóch, kiedy przełożony albo klient proszą o wprowadzenie zmian i dodanie nowych

funkcji do świetnego skryptu, niegdyś znajomy kod JavaScript będzie mniej zrozumiały. Możesz się zastanawiać, do czego służy dana zmienna, dlaczego napisałeś kod w taki, a nie inny sposób, lub co dzieje się w określonym fragmencie programu.

Łatwo zapomnieć, jak działa program lub dlaczego napisałeś skrypt w dany sposób. Na szczęście większość języków programowania umożliwia dodawanie do kodu notatek na użytek własny i innych programistów analizujących skrypt. JavaScript pozwala dodawać do kodu *komentarze*. Jeśli używałeś ich w kodach HTML i CSS, znasz już tę funkcję. Komentarz to po prostu wiersz (lub kilka wierszy) uwag. Interpreter języka JavaScript ignoruje je, jednak można w nich umieścić przydatne informacje na temat działania programu.

Aby utworzyć komentarz jednowierszowy, należy poprzedzić go dwoma ukośnikami:

```
// To komentarz.
```

Można też dodać komentarz po instrukcji języka JavaScript:

```
var price = 10; // Ustawianie początkowego kosztu kontrolki.
```

Interpreter JavaScript ignoruje wszystkie znaki zaczynające do dwóch ukośników (`//`), aż do końca wiersza.

Można też tworzyć komentarze wielowierszowe, które rozpoczynają się od sekwencji `/*`, a kończą sekwencją `*/`. (Dokładnie takie same komentarze są używane w arkuszach stylów CSS). Interpreter ignoruje cały tekst między tymi symbolami. Załóżmy, że chcesz na początku kodu opisać działanie programu. Możesz to zrobić w następujący sposób:

```
/*
  Pokaz slajdów w języku JavaScript
  Program automatyzuje wyświetlanie
  rysunków w oknie wyskakującym.
*/
```

Sekwencji `/*` i `*/` nie trzeba umieszczać w odrębnych wierszach. Przy ich użyciu można też utworzyć komentarz jednowierszowy:

```
/* To komentarz jednowierszowy. */
```

Jeśli chcesz dodać krótki, jednowierszowy komentarz, powinieneś raczej użyć sekwencji `//`. Do tworzenia komentarzy wielowierszowych warto stosować kombinację `/*` i `*/`.

Kiedy używać komentarzy?

Komentarz to nieocenione narzędzie w długich lub złożonych programach, których chcesz używać (lub je modyfikować) w przyszłości. Choć proste skrypty przedstawione do tej pory składają się z tylko kilku wierszy kodu, wkrótce zaczniesz tworzyć dłuższe i dużo bardziej skomplikowane programy. Aby szybko sobie przypomnieć, jak działa skrypt, warto dodać komentarze. Pomogą one zrozumieć ogólną logikę programu i jego szczególnie skomplikowane fragmenty.

Wielu programistów dodaje blok komentarzy na początku zewnętrznych plików JavaScript. Takie komentarze informują o przeznaczeniu skryptu, dacie jego utworzenia, numerze wersji (jeśli kod często się zmienia) i prawach autorskich.

Uwaga: Dodawanie wielu komentarzy zwiększa rozmiar skryptu i wydłuża jego wczytywanie. Jednak ogólnie rzecz biorąc, komentarze dodawane do skryptów nie będą się znacząco przyczyniać do wzrostu wielkości plików JavaScript. Jeśli jednak chcesz usunąć ze swych plików każdy niepotrzebny bajt, informacje na temat zmniejszania plików JavaScript i skracania czasu ich pobierania znajdziesz na stronie 609.

Na przykład w pliku JavaScript biblioteki jQuery znajduje się następujący komentarz:

```
/*!
 * jQuery JavaScript Library v1.11.0
 * http //jquery.com/
 *
 * Includes Sizzle.js
 * http //sizzlejs.com/
 *
 * Copyright 2005, 2014 jQuery Foundation, Inc. and other contributors
 * Released under the MIT license
 * http //jquery.org/license
 *
 * Date  2014-01-23T21 02Z
 */
```

Na początku skryptu można też zamieścić instrukcję jego używania, obejmującą zmienne, które trzeba ustawić, wymagane specjalne elementy w kodzie HTML i tak dalej.

Komentarz warto dodać także przed grupą skomplikowanych operacji programistycznych. Jeśli na przykład skrypt wyświetla animowany rysunek, poruszający się w oknie przeglądarki, fragment programu musi określać aktualną pozycję obrazka. Może to wymagać umieszczenia kilku skomplikowanych wierszy kodu. Warto dodać komentarz przed taką sekcją programu, aby w przyszłości można było ustalić, do czego służy dany fragment:

```
// Określanie współrzędnych x i y rysunku na ekranie.
```

Zgodnie z praktyczną regułą należy dodawać komentarze wszędzie tam, gdzie mogą być w przyszłości przydatne. Jeśli dany wiersz jest w pełni zrozumiały, prawdopodobnie nie wymaga opisu. Nie ma sensu dodawanie komentarzy do prostego kodu, na przykład `alert('Witaj')`, ponieważ jego działanie jest oczywiste (instrukcja ta wyświetla okno dialogowe ze słowem „Witaj").

Komentarze w tej książce

Komentarze są bardzo przydatne także przy objaśnianiu kodu JavaScript. W tej książce często opisują, do czego służy dany wiersz lub jaki jest efekt wykonania instrukcji. Komentarz w poniższym fragmencie określa wynik działania polecenia `alert`:

```
var a = 'Jan';
var b = 'Kowalski';
alert( a + ' ' + b); // 'Jan Kowalski'
```

Trzeci wiersz kończy się komentarzem, który informuje, jaki tekst kod wyświetli w oknie przeglądarki. Jeśli chcesz przetestować kod z tej książki przez umieszczenie go na stronie i wyświetlenie w przeglądarce, możesz pominąć podobne komentarze przy przepisywaniu instrukcji. Uwagi tego rodzaju mają jedynie pomóc zrozumieć działanie kodu w trakcie czytania książki.

Kiedy zaczniesz poznawać bardziej złożone polecenia języka JavaScript, nauczysz się manipulować danymi zapisanymi w zmiennych. Komentarze do takiego kodu często informują, jaką wartość powinna zawierać zmienna po wykonaniu instrukcji. Na przykład polecenie `charAt()` umożliwia pobranie znaku z określonego miejsca łańcucha. Przy opisie tego polecenia możesz natrafić na kod podobny do poniższego:

```
var x = "Nadszedł czas dobrych programistów.";
alert(x.charAt(2)); // 'd'
```

Komentarz `// 'd'` na końcu drugiego wiersza określa, co powinieneś zobaczyć w oknie dialogowym po uruchomieniu kodu w przeglądarce. Naprawdę jest to litera „d". Gdy liczymy litery w łańcuchu znaków, pierwsza z nich ma numer 0, a zatem wywołanie `charAt(2)` pobiera *trzeci* znak łańcucha. Czasem programowanie jest trudne do zrozumienia.

3
ROZDZIAŁ

Dodawanie struktur logicznych i sterujących

Poznałeś już podstawowe cegiełki języka JavaScript. Jednak samo tworzenie zmiennych oraz zapisywanie w nich łańcuchów znaków i liczb nie daje zbyt wielu możliwości. Także wieloelementowa tablica nie będzie przydatna, jeśli nie będziesz mógł wygodnie poruszać się po wszystkich jej wartościach. W tym rozdziale dowiesz się, jak sprawić, aby programy inteligentnie reagowały na zdarzenia i działały bardziej wydajnie dzięki instrukcjom warunkowym, pętlom i funkcjom.

Programy reagujące inteligentnie

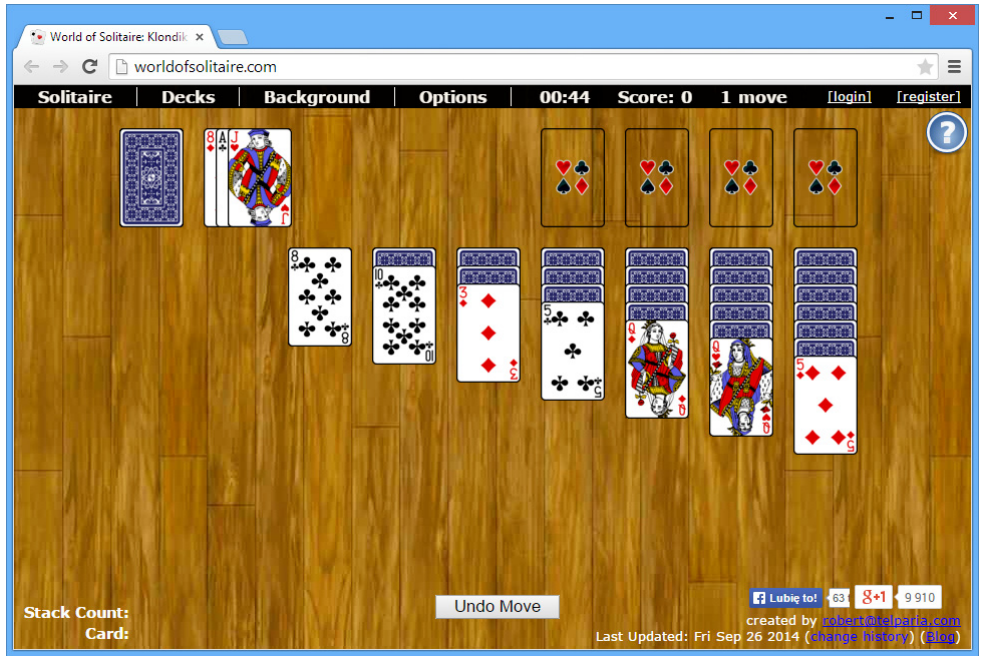
W życiu nieustannie stajemy przed wyborami. „W co powinienem się ubrać?", „Co mam zjeść na obiad?", „Gdzie spędzić piątkowy wieczór?". Wiele decyzji zależy od sytuacji. Załóżmy, że na piątkowy wieczór planujesz wyjście do kina. Prawdopodobnie zadasz sobie kilka pytań: „Czy grają jakiś dobry film?", „Czy seans rozpoczyna się o odpowiedniej godzinie?", „Czy mam wystarczająco dużo pieniędzy, aby kupić bilet (no i dużą porcję popcornu)?".

Załóżmy, że film *jest* wyświetlany o odpowiedniej godzinie. Następne pytanie to: „Czy mam wystarczającą ilość pieniędzy?". Jeśli tak, możesz wyruszać do kina. Jeżeli nie, zostaniesz w domu. Gdy do następnego piątku zgromadzisz fundusze, wybierzesz się do kina. Ten scenariusz to po prostu ilustracja wpływu warunków na podejmowane decyzje.

JavaScript udostępnia podobną funkcję podejmowania decyzji — *instrukcje warunkowe*. W najprostszej wersji taka instrukcja sprawdza odpowiedź na pytanie typu „tak-nie". Jeśli odpowiedź jest twierdząca, program wykona określone operacje. Jeżeli odpowiedź to „nie", skrypt wykona inne polecenia. Instrukcje warunkowe to jeden z najbardziej użytecznych elementów programistycznych. Umożliwiają one programom reagowanie na różne sytuacje i działanie w inteligentny sposób. W swoich programach będziesz ich używał niezliczoną ilość razy. A poniżej znajdziesz kilka przykładów i przekonasz się, jak bardzo są użyteczne.

- **Walidacja formularzy**. Jeśli program ma sprawdzić, czy użytkownik wypełnił wszystkie pola formularza („Nazwisko", „Adres", „E-mail" i tak dalej), można użyć do tego instrukcji warunkowych, na przykład: „Jeśli pole »Nazwisko« jest puste, nie przesyłaj formularza".
- **Przeciąganie i upuszczanie**. Jeśli strona umożliwia przenoszenie jej części, warto sprawdzać, gdzie użytkownik umieścił dany element. Na przykład po przeciągnięciu rysunku na ikonę kosza należy usunąć zdjęcie ze strony.
- **Sprawdzanie wprowadzonych danych**. Jeśli program wyświetla w oknie wyskakującym pytanie typu „Czy zechcesz odpowiedzieć na kilka pytań na temat jakości strony?", powinien reagować w różny sposób — w zależności od udzielonej odpowiedzi.

Na rysunku 3.1 przedstawiono przykładową aplikację, w której wykorzystano instrukcje warunkowe.



Rysunek 3.1. *Zabawa wymaga czasem dużo pracy. Podobna do pasjansa gra napisana w języku JavaScript (http://worldofsolitaire.com) dostosowuje swe działanie do warunków. Kiedy na przykład gracz przeciągnie kartę, skrypt musi określić miejsce jej upuszczenia, a następnie wykonać odpowiednie operacje*

Podstawy instrukcji warunkowych

Instrukcje warunkowe są także nazywane instrukcjami „jeśli, to", ponieważ wykonują operacje tylko wtedy, gdy odpowiedź na pytanie jest twierdząca: „*Jeśli* mam pieniądze, *to* pójdę do kina". Podstawowa struktura takich instrukcji wygląda następująco:

```
if ( warunek ) {
  // Tu operacje.
}
```

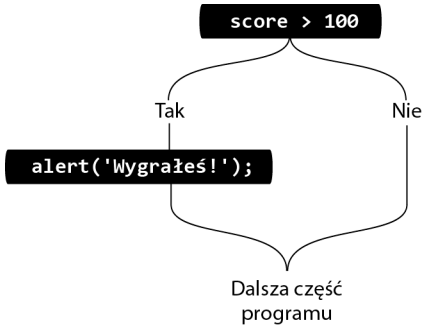
Ta instrukcja składa się z trzech części. Słowo kluczowe `if` informuje, że dalszy kod to instrukcja warunkowa. Nawiasy zawierają pytanie typu „tak-nie", nazywane *warunkiem* (więcej na ten temat dowiesz się już za chwilę). Nawiasy klamrowe ({}) wyznaczają początek i koniec kodu JavaScript, wykonywanego, jeśli warunek jest spełniony.

Uwaga: W podanym kodzie fragment „// Tu operacje." to komentarz w języku JavaScript. Nie jest to uruchamiany kod, a jedynie uwaga w programie, która w tym przypadku informuje Cię o tym, co powinno znaleźć się w danym miejscu. Więcej wiadomości o komentarzach znajdziesz na stronie 89.

Warunek to często porównanie dwóch wartości. Załóżmy, że gracz wygrywa, jeśli osiągnie wynik wyższy niż 100 punktów. W takim programie potrzebna jest zmienna przechowująca wynik i instrukcja sprawdzająca, czy liczba zdobytych punktów przekroczyła 100. Potrzebny kod JavaScript może wyglądać następująco:

```
if (score > 100) {
  alert('Wygrałeś!');
}
```

Zwróć uwagę na fragment `score > 100`. Jest to warunek, który sprawdza, czy wartość zmiennej `score` jest większa od `100`. Jeśli tak jest, pojawi się okno dialogowe z informacją „Wygrałeś!". Jeżeli gracz nie przekroczył jeszcze 100 punktów, interpreter pominie polecenie `alert` i przejdzie do następnej części programu. Na rysunku 3.2 przedstawiono cały ten proces graficznie.



Rysunek 3.2. *Przy użyciu prostej instrukcji warunkowej kod umieszczony wewnątrz nawiasów klamrowych zostanie wykonany wyłącznie wtedy, gdy podany warunek przyjmie wartość true. Jeśli warunek będzie mieć wartość false, kod zostanie pominięty i będzie wykonywana dalsza część programu*

Obok symbolu > (większy niż) w porównaniach można używać także kilku innych operatorów (patrz tabela 3.1).

Wskazówka: Przed każdym wierszem kodu JavaScript w nawiasach klamrowych dodaj dwa odstępy (lub kliknij raz klawisz tabulacji). Wcięcia w wierszach ułatwiają dostrzeżenie początkowego i końcowego nawiasu oraz ustalenie, który fragment kodu należy do instrukcji warunkowej. Stosowanie dwóch odstępów to standardowe rozwiązanie, jednak możesz używać także wcięć o długości na przykład czterech odstępów, jeśli uznasz, że poprawiają czytelność kodu. W przykładach w tej książce kod w nawiasach klamrowych zawsze ma wcięcia.

Tabela 3.1. *Operatory porównywania służą do oceny wartości w instrukcjach warunkowych*

Operator porównywania	Działanie
`==`	**Równa się**. Sprawdza, czy dwie wartości są takie same. Służy do porównywania liczb i łańcuchów znaków.
`!=`	**Nie równa się**. Sprawdza, czy dwie wartości są **różne**. Służy do porównywania liczb i łańcuchów znaków.
`===`	**Identyczny.** Operator porównuje nie tylko wartości, lecz także typy wartości. Innymi słowy, aby operator uznał, że porównywane elementy są sobie równe, muszą one mieć także te same typy (oba muszą być liczbami, łańcuchami znaków lub wartościami logicznymi). Choć na przykład wyrażenie `'2' == 2` będzie prawdziwe, to wyrażenie `'2' === 2` już nie będzie, gdyż pierwsza wartość została zapisana w apostrofach (czyli jest łańcuchem znaków), natomiast druga jest liczbą. Wielu programistów preferuje ten operator, gdyż gwarantuje on, że porównywane będą dane tych samych typów. Jednak w przypadku pobierania wartości liczbowej z pola tekstowego (na przykład z okna dialogowego wyświetlanego przez metodę `prompt()`), uzyskamy łańcuch znaków, taki jak '2', a nie liczbę. Dlatego przed porównywaniem takiej wartości należy ją skonwertować na liczbę — więcej informacji o konwertowaniu łańcuchów znaków na liczby znajdziesz na stronie 70.
`!==`	**Nieidentyczny.** Podobnie jak opisany powyżej operator dokładnie równy, także i ten porównuje zarówno wartości, jak i typy, na przykład wyrażenie `'2' != 2` zwraca wartość `false`, natomiast `'2' !== 2` zwraca `true`, bo choć porównywane wartości są takie same, to jednak ich typy są różne.
`>`	**Większy niż**. Sprawdza, czy liczba po lewej stronie znaku jest większa od liczby po prawej stronie. Wyrażenie `2 > 1` jest prawdziwe, ponieważ liczba 2 jest większa od 1, jednak warunek `2 > 3` jest fałszywy, ponieważ liczba 2 nie jest większa od 3.
`<`	**Mniejszy niż**. Sprawdza, czy liczba po lewej stronie znaku jest mniejsza od liczby po prawej stronie. Wyrażenie `2 < 3` jest prawdziwe, ponieważ liczba 2 jest mniejsza od 3, jednak warunek `2 < 1` jest fałszywy, ponieważ liczba 2 nie jest mniejsza od 1.
`>=`	**Większy lub równy**. Sprawdza, czy liczba po lewej stronie znaku jest większa od liczby po prawej stronie lub jej równa. Wyrażenie `2 >= 2` jest prawdziwe, ponieważ liczba 2 jest równa 2, jednak warunek `2 >= 3` jest fałszywy, ponieważ liczba 2 nie jest większa od 3 ani jej równa.
`<=`	**Mniejszy lub równy**. Sprawdza, czy liczba po lewej stronie znaku jest mniejsza od liczby po prawej stronie lub jej równa. Wyrażenie `2 <= 2` jest prawdziwe, ponieważ liczba 2 jest równa 2, jednak warunek `2 <= 1` jest fałszywy, ponieważ liczba 2 nie jest mniejsza od 1 ani jej równa.

Częściej warunki służą do sprawdzania, czy dwie wartości są sobie równe. Na przykład w quizie napisanym w języku JavaScript może znaleźć się pytanie: „Ile księżyców ma Saturn?”. Jeśli program zapisuje odpowiedź w zmiennej `answer`, można użyć następującej instrukcji warunkowej:

```
if (answer == 31) {
  alert('Prawidłowa odpowiedź. Saturn ma 31 księżyców.');
}
```

Dwa znaki równości (==) to nie literówka. Ta sekwencja nakazuje interpreterowi porównanie dwóch wartości i sprawdzenie, czy są sobie równe. Zgodnie z tym, czego dowiedziałeś się w poprzednim rozdziale, w języku JavaScript jeden znak równości to *operator przypisania*, który służy do zapisywania wartości w zmiennych:

```
var score = 0; // Zapisuje 0 w zmiennej score.
```

Ponieważ jeden znak równości ma dla interpretera specjalne znaczenie, trzeba użyć dwóch takich symboli przy sprawdzaniu, czy dwie wartości są sobie równe.

Sekwencji == (*operatora równości*) można użyć także do sprawdzenia, czy dwa łańcuchy znaków są takie same. Załóżmy, że użytkownik może wpisać w formularzu nazwę koloru. Jeśli wpisze `'red'`, program zmieni kolor tła strony na czerwony. Można to zrobić za pomocą instrukcji warunkowej:

```
if (enteredColor == 'red') {
  document.body.style.background='red';
}
```

Uwaga: Nie musisz na razie wiedzieć, jak przebiega zmiana koloru w powyższym kodzie. Dynamiczne modyfikowanie właściwości stylów CSS za pomocą kodu JavaScript opisane jest na stronie 163.

Możesz też sprawdzić, czy dwie wartości różnią się od siebie. Służy do tego operator nierówności:

```
if (answer != 31) {
  alert("Błąd! Liczba księżyców Saturna jest inna.");
}
```

Wykrzyknik odpowiada tu słowu „nie", dlatego sekwencja `!=` oznacza „nie równa się". Jeśli w zmiennej `answer` zapisana jest wartość różna od 31, uczestnik gry zobaczy nieprzyjemny komunikat.

Kod wykonywany przy spełnionym warunku w poprzednich przykładach miał tylko jeden wiersz, jednak między otwierającym i zamykającym nawiasem klamrowym można umieścić dowolną liczbę wierszy kodu JavaScript. W przykładowym quizie można też przechowywać liczbę prawidłowych odpowiedzi udzielonych przez gracza.

A zatem, kiedy użytkownik poprawnie poda liczbę księżyców Saturna, należy zwiększyć sumę jego punktów o 1. Można to zrobić w instrukcji warunkowej:

```
if (answer == 31) {
  alert('Prawidłowa odpowiedź. Saturn ma 31 księżyców.');
  numCorrect += 1;
}
```

Uwaga: Zgodnie z informacjami podanymi na stronie 72 drugi wiersz kodu umieszczony w instrukcji warunkowej — `numCorrect += 1` — dodaje 1 do aktualnej wartości zmiennej `numCorrect`.

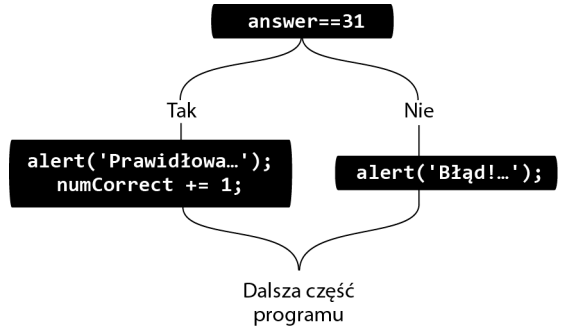
Między nawiasy można wstawić więcej dodatkowych wierszy kodu, który program ma uruchomić, jeśli warunek będzie spełniony.

Uwzględnianie planu awaryjnego

Co zrobić, jeśli warunek nie jest spełniony? Podstawowa instrukcja warunkowa z poprzedniego punktu nie obejmuje planu awaryjnego, używanego, jeśli warunek jest fałszywy. Kiedy zastanawiasz się, co robić w piątkowy wieczór, i nie masz pieniędzy na kino, prawdopodobnie wymyślisz coś *innego*. W instrukcjach `if` można użyć podobnego planu awaryjnego, nazywanego *klauzulą* `else`. Załóżmy, że program do obsługi quizu ma powiadamiać gracza, czy udzielił dobrej, czy złej odpowiedzi. Można to zrobić w następujący sposób:

```
if (answer == 31) {
  alert('Prawidłowa odpowiedź. Saturn ma 31 księżyców.');
  numCorrect = numCorrect + 1;
} else {
  alert("Błąd! Liczba księżyców Saturna jest inna.");
}
```

W tym kodzie występuje sytuacja „albo-albo". Może się pojawić tylko jeden z dwóch komunikatów (co pokazano na rysunku 3.3). Jeśli zmienna `answer` ma wartość `31`, użytkownik dowie się, że podał prawidłową odpowiedź. Inna wartość spowoduje wyświetlenie informacji o błędzie.



Rysunek 3.3. *Podczas stosowania instrukcji warunkowej z klauzulą else konieczne jest podanie dwóch bloków kodu, przy czym zostanie wykonany tylko jeden z nich. Jeśli warunek przyjmie wartość true, zostanie wykonany fragment podany w nawiasach klamrowych umieszczonych bezpośrednio za instrukcją if (lewa gałąź na rysunku); gdy natomiast warunek przyjmie wartość false, zostanie wykonany blok kodu umieszczony wewnątrz nawiasów klamrowych za słowem kluczowym else (prawa gałąź na rysunku)*

Aby utworzyć klauzulę `else`, wystarczy wpisać słowo „else" po zamykającym nawiasie klamrowym instrukcji warunkowej, a następnie dodać drugą parę takich nawiasów. W nowych nawiasach należy umieścić kod wykonywany wtedy, gdy warunek nie jest spełniony. Także w klauzuli `else` można wpisać dowolną liczbę wierszy kodu.

Sprawdzanie kilku warunków

Czasem trzeba sprawdzić kilka warunków i powiązać z nimi różne operacje. Wyobraź sobie grę, w której prezenter pyta uczestnika: „Czy wybiera pan bramkę numer 1, 2 czy 3?", przy czym można wskazać tylko jedną nagrodę. W życiu ludzie często stykają się z podobnymi wyborami.

PORADNIA DLA ZAAWANSOWANYCH

Jeszcze o wartościach logicznych

Na stronie 63 poznałeś wartości logiczne — `true` i `false`. Na pozór nie są one zbyt przydatne, jednak w instrukcjach warunkowych odgrywają kluczową rolę. Ponieważ warunek to pytanie typu „tak-nie", odpowiedzią na nie jest wartość logiczna. Przyjrzyj się poniższemu fragmentowi:

```
var x = 4;
if ( x == 4 ) {
  // Instrukcje.
}
```

Pierwszy wiersz kodu zapisuje w zmiennej `x` liczbę `4`. Warunek w następnym wierszu to proste pytanie: „Czy wartość zmiennej `x` jest równa 4?". Warunek ten jest spełniony, dlatego skrypt wykona kod JavaScript w nawiasach klamrowych. Interpreter przekształca warunek podany w nawiasach na wartość logiczną. (W żargonie programistycznym można powiedzieć, że interpreter *sprawdza* warunek). Jeśli warunek ma wartość `true` (odpowiedź na pytanie to „tak"), uruchamiany jest kod w nawiasach klamrowych. Jeżeli jednak warunek ma wartość `false`, program pomija instrukcje w nawiasach klamrowych.

Jednym ze standardowych zastosowań wartości logicznych jest tworzenie *flag*, czyli zmiennych, które określają, czy dany warunek jest spełniony. Na przykład na potrzeby walidacji formularza można utworzyć zmienną `valid` o wartości logicznej `true`. Programista używa tej wartości, ponieważ początkowo zakłada, iż formularz jest wypełniony prawidłowo. Jeśli przy analizie pól formularza okaże się, że nie zawierają informacji lub podane wartości mają niewłaściwy typ, należy zmienić wartość zmiennej `valid` na `false`. Po przejrzeniu wszystkich pól formularza należy sprawdzić zmienną `valid`. Jeśli ma ona wartość `true`, można przesłać formularz. Jeżeli ma wartość `false`, przynajmniej jedno pole formularza zawiera błędne dane, dlatego trzeba wyświetlić komunikat o błędzie i wstrzymać przesyłanie formularza:

```
var valid = true;
// Tu liczne operacje.
/* Jeśli dane są błędne, należy
ustawić wartość zmiennej valid
na false. */
if (valid) {
  // Przesyłanie formularza.
} else {
  // Wyświetlanie komunikatów o błędzie.
}
```

Przypomnij sobie rozważania nad piątkowym wieczorem. Możesz zwiększyć liczbę możliwych rozrywek i uzależnić wybór jednej z nich od ilości pieniędzy i humoru. Możesz na przykład zacząć od stwierdzenia: „Jeśli mam ponad 100 złotych, wybiorę się na dobrą kolację i film (i wystarczy mi na popcorn)". Jeśli nie masz 100 złotych, możesz postawić następny warunek: „Jeśli mam ponad 50 złotych, pójdę na dobrą kolację". Jeżeli nie masz 50 złotych, możesz stwierdzić: „Jeśli mam ponad 25 złotych, pójdę do kina". Jeżeli nie możesz wydać 25 złotych, uznajesz, że zostaniesz w domu i pooglądasz telewizję. Piątek otwiera przed Tobą naprawdę wiele możliwości!

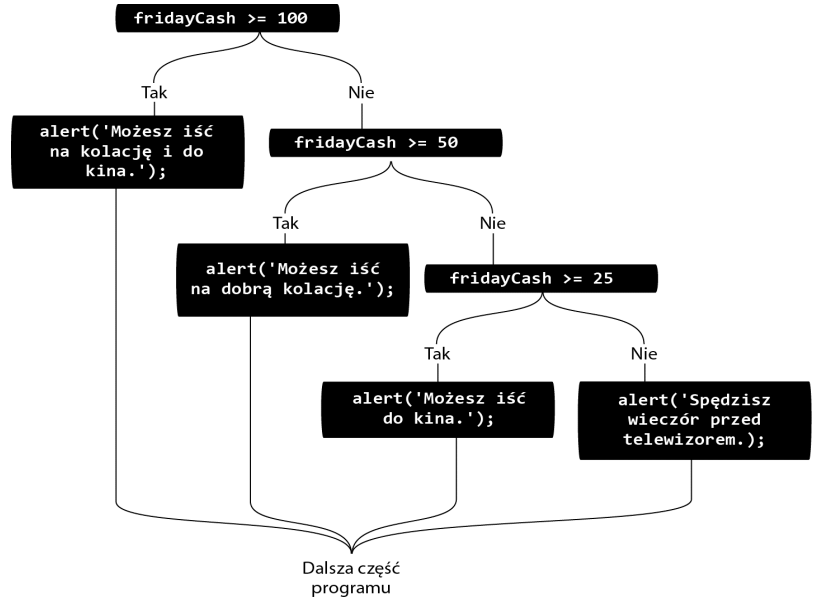
JavaScript umożliwia sprawdzenie serii warunków za pomocą instrukcji `else if`. Struktura ta działa w następujący sposób: instrukcja `if` jest powiązana z możliwością numer 1, a dalsze instrukcje `else if` pozwalają zadać dodatkowe pytania, które prowadzą do dodatkowych operacji. Na końcu jako możliwość rezerwową można umieścić klauzulę `else`. Podstawowa postać tej struktury w języku JavaScript wygląda następująco:

```
if (warunek) {
  // Bramka 1.
} else if (warunek2) {
  // Bramka 2.
} else {
  // Bramka 3.
}
```

Ta struktura wystarczy do zbudowania w języku JavaScript programu do planowania piątkowego wieczoru. Skrypt ten pyta użytkownika o ilość pieniędzy, a następnie określa możliwe rozrywki na piątkowy wieczór (brzmi znajomo, prawda?). Do pobrania odpowiedzi można użyć polecenia `prompt()`, które poznałeś na stronie 74, a do ustalenia planu wieczoru posłuży seria instrukcji `if` i `else if`:

```
var fridayCash = prompt('Ile chcesz wydać?', '');
if (fridayCash >= 100) {
  alert('Możesz iść na kolację i do kina.');
} else if (fridayCash >= 50) {
  alert('Możesz iść na dobrą kolację.');
} else if (fridayCash >= 25) {
  alert('Możesz iść do kina.');
} else {
  alert('Spędzisz wieczór przed telewizorem.');
}
```

Pierwszy wiersz programu otwiera okno dialogowe z pytaniem o ilość pieniędzy. Dane podane przez użytkownika są zapisywane w zmiennej `fridayCash`. Następny wiersz sprawdza warunek: czy użytkownik wpisał wartość równą 100 lub większą? Jeśli tak, pojawia się okno dialogowe z informacją, że użytkownik może iść na kolację i do kina. W tym momencie cała instrukcja warunkowa jest już wykonana. Interpreter pomija dwie instrukcje `else if` i końcową instrukcję `else`. W instrukcjach warunkowych uruchamiany jest tylko jeden zestaw poleceń, dlatego kiedy interpreter natrafi na warunek o wartości `true`, wykonuje powiązany z nim kod JavaScript i pomija wszystkie pozostałe warunki (patrz rysunek 3.4).



Rysunek 3.4. *Gdy korzystamy z prostych instrukcji warunkowych, kod umieszczany wewnątrz nawiasów klamrowych jest wykonywany wyłącznie wtedy, gdy warunek będzie spełniony. Jeśli warunek nie zostanie spełniony, kod jest pomijany, a program kontynuuje działanie. Działający przykład zilustrowanego tu kodu znajdziesz w przykładach dołączonych do książki, a konkretnie w pliku friday_night.html w katalogu R03*

Uwaga: Podczas przetwarzania warunków interpreter JavaScriptu poszukuje „prawdy". Okazuje się, że w języku JavaScript stosowane jest pojęcie „prawdy" i „fałszu" (ang: *truthy* oraz *falsy*), które mają nieco szersze znaczenie niż wartości `true` oraz `false`. Masz wrażenie, że to skomplikowane? Więcej o tych zaawansowanych zagadnieniach dowiesz się, czytając dalej tę książkę.

Załóżmy, że użytkownik podał wartość 30. Pierwszy warunek nie jest spełniony, ponieważ 30 to mniej niż 100. Dlatego interpreter pomija kod w nawiasach klamrowych przy pierwszym warunku i przechodzi do pierwszej instrukcji `else if`: „Czy 30 jest równe 50 lub większe?". Ponieważ odpowiedź to „nie", interpreter pomija kod powiązany z tym warunkiem i dochodzi do drugiej instrukcji `else if`: „Czy 30 jest równe 25 lub większe?". Odpowiedź to „tak", dlatego program wyświetla okno dialogowe z wiadomością: „Możesz iść do kina" i przerywa działanie, pomijając końcową klauzulę `else`.

Kolejność, w jakiej zostaną zapisane instrukcje warunkowe, może mieć wpływ na przebieg działania programu. Załóżmy na przykład, że zmieniliśmy kolejność instrukcji warunkowych w poprzednim przykładzie:

```
var fridayCash = prompt('Ile chcesz wydać?', '');
if (fridayCash >= 25) {
  alert('Możesz iść do kina.');
} else if (fridayCash >= 50) {
  alert('Możesz iść na dobrą kolację.');
} else if (fridayCash >= 100) {
  alert('Możesz iść na kolację i do kina.');
} else {
  alert('Spędzisz wieczór przed telewizorem.');
}
```

W tym przypadku — niezależnie od tego, jak dużo będziesz mieć pieniędzy — nigdy nie pójdziesz na dobrą kolację, ani na kolację i do kina. Jeśli będziesz mieć 100 złotych, program najpierw sprawdzi warunek „czy 100 > 12". Oczywiście, warunek ten będzie spełniony, a zatem program wyświetli komunikat `Możesz iść do kina.` , a pozostałe instrukcje warunkowe zostaną pominięte, bez względu na to, że będziemy mieli wystarczająco dużo pieniędzy, by pójść zarówno do kina, jak i na kolację.

Z problemami tego typu można się zetknąć podczas porównywania wartości liczbowych przy użyciu operatorów < lub >. Ponieważ bardzo wiele liczb może być większych bądź mniejszych od innych liczb, zatem możliwości spełnienia takiego warunku także może być wiele. Z kolei w przypadku sprawdzania, czy zmienna jest równa pewnej wartości liczbowej, warunek będzie spełniony tylko w jednym przypadku. Zatem sekwencje instrukcji warunkowych `if / else if / then` z takimi warunkami można zapisywać w dowolnej kolejności.

Wskazówka: Istnieje też inny sposób na utworzenie serii instrukcji warunkowych sprawdzających wartość jednej zmiennej, takiej jak `fridayCash`, w przykładowym kodzie. Służy do tego instrukcja `switch`, którą poznasz na stronie 603.

Bardziej skomplikowane warunki

Kiedy trzeba sprawdzić wartości wielu zmiennych, często niezbędne są jeszcze bardziej złożone instrukcje warunkowe. Przy walidacji pola na adres e-mail w formularzu należy sprawdzić, czy pole nie jest puste i czy zawiera adres, a nie przypadkowe znaki. Na szczęście JavaScript umożliwia przetestowanie także takich warunków.

Sprawdzanie, czy spełnionych jest kilka warunków

Często przy podejmowaniu decyzji należy wziąć pod uwagę kombinację czynników. Na przykład możesz zechcieć pójść do kina, jeśli masz wystarczającą ilość pieniędzy **i** grany jest film, który Cię interesuje. Oznacza to, że spełnione muszą być dwa warunki. Jeśli jeden z nich jest fałszywy, zrezygnujesz z kina. W języku JavaScript do łączenia warunków służy *logiczny operator I*, który ma postać dwóch ampersandów (`&&`). Można podać go między dwoma warunkami w jednej instrukcji warunkowej. Aby sprawdzić, czy wartość to liczba większa od 1 i mniejsza od 10, można użyć następującego kodu:

```
if (a > 1 && a < 10) {
  // Wartość pomiędzy 1 a 10.
  alert("Wartość " + a + " jest większa od 1 i mniejsza od 10.");
}
```

Ten kod testuje dwa warunki. Fragment `a > 1` sprawdza, czy wartość zmiennej `a` jest większa od `1`. Drugi warunek, `a < 10`, to pytanie: „Czy wartość `a` jest mniejsza od `10`?". Kod JavaScript spomiędzy nawiasów klamrowych zadziała tylko wtedy, gdy *oba* warunki będą spełnione. Jeśli zmienna `a` ma wartość `0`, pierwszy warunek, `a < 10`, jest prawdziwy (0 jest mniejsze od 10), ale drugi warunek jest fałszywy (0 nie jest większe od 1).

Nie trzeba ograniczać się do dwóch warunków. Przy użyciu operatora `&&` można połączyć ich dowolną liczbę:

```
if (b>0 && a>0 && c>0) {
  // Wszystkie trzy zmienne są większe od 0.
}
```

Ten kod sprawdza, czy wartość każdej z trzech zmiennych jest większa od 0. Jeśli przynajmniej jedna z nich ma wartość 0 lub mniejszą, program nie uruchomi kodu zapisanego w nawiasach klamrowych.

Sprawdzanie, czy spełniony jest przynajmniej jeden warunek

Czasem wystarczy, że prawdziwy jest jeden warunek z grupy. Załóżmy, że program umożliwia użytkownikom przechodzenie między zdjęciami w galerii za pomocą klawiatury. Kiedy internauta wciśnie klawisz *N*, pojawia się następny rysunek. Program ma reagować w ten sposób zarówno po wpisaniu małej litery *n*, jak i po wprowadzeniu dużej litery *N* (przy włączonym klawiszu *Caps Lock*). Trzeba więc sprawdzić, czy użytkownik podał jedną **lub** drugą literę. Służy do tego *logiczny operator LUB*, który ma postać dwóch symboli potoku (`||`):

```
if (key == 'n' || key == 'N') {
  // Przejdź do następnego zdjęcia.
}
```

Uwaga: Aby wprowadzić symbol potoku, wciśnij kombinację *Shift+*. Klawisz służący do wpisywania ukośnika i symbolu potoku znajduje się zwykle nad klawiszem *Enter*.

Program uruchamia kod JavaScript w nawiasach klamrowych, jeśli prawdziwy jest choć jeden warunek podany obok operatora LUB.

Operator LUB, podobnie jak operator I, umożliwia sprawdzanie kilku warunków. Załóżmy, że napisałeś za pomocą języka JavaScript grę w wyścigi samochodowe. Gracz ma ograniczoną ilość czasu i paliwa oraz liczbę samochodów (każdy wypadek powoduje utratę jednego pojazdu). Aby utrudnić grę, można ją przerwać, kiedy skończy się choć jeden z trzech zasobów:

```
if (gas <= 0 || time <= 0 || cars <= 0) {
  // Koniec gry.
}
```

Przy testowaniu wielu warunków czasem trudno zrozumieć działanie instrukcji warunkowej. Niektórzy programiści umieszczają każdy warunek w nawiasach, aby wyraźniej opisać logikę programu:

```
if ((key == 'n') || (key == 'N')) {
  // Przejście do następnego zdjęcia.
}
```

W czasie czytania takiego kodu można traktować każdą grupę jak odrębny test. Wynik operacji w każdej parze nawiasów to zawsze `true` lub `false`.

Negowanie warunków

Fani Supermena prawdopodobnie znają postać Bizarro. Jest to czarny charakter, który mieszka na sześciennej planecie Htrae (ang. *Earth*, czyli Ziemia, pisane wstecz), chodzi w kostiumie z odwróconą literą S i jest przeciwieństwem Supermena. Kiedy Bizarro mówi „Tak", ma na myśli „Nie", a kiedy mówi „Nie", chce powiedzieć „Tak".

Język JavaScript udostępnia operator *NIE*, który ma postać wykrzyknika (`!`) i działa w podobny sposób. Widziałeś już, jak używać tego operatora wraz ze znakiem równości. Ta sekwencja, `!=`, oznacza „nie równa się". Jednak operatora NIE można użyć także do całkowitego odwrócenia wyników instrukcji warunkowej, czyli zmiany wartości `false` na `true` i `true` na `false`.

Operatora NIE można użyć, jeśli program ma uruchomić kod dla fałszywego warunku. Załóżmy, że zmienna `valid` przyjmuje wartość logiczną `true` lub `false` (patrz ramka na stronie 99). Program może używać tej zmiennej do śledzenia, czy użytkownik prawidłowo wypełnił formularz. Kiedy internauta spróbuje przesłać formularz, kod JavaScript sprawdzi każde pole formularza pod kątem określonych wymagań (na przykład pole nie może być puste i musi zawierać adres e-mail). Jeśli wystąpi problem, na przykład pole będzie puste, można ustawić zmienną `valid` na wartość `false` (`valid = false`).

Aby program po wykryciu problemu wyświetlał informacje o błędzie i wstrzymywał przesyłanie formularza, można użyć następującej instrukcji warunkowej:

```
if (! valid) {
  // Wyświetlanie błędów i blokowanie przesyłania formularza.
}
```

Warunek `! valid` oznacza „jeśli nie valid". Jeśli zmienna `valid` ma wartość `false`, cały *warunek* ma wartość `true`. Aby ustalić, czy warunek jest spełniony, należy sprawdzić jego wartość z pominięciem operatora NIE, a następnie ją odwrócić. Dlatego jeśli warunek ma wartość `true`, operator `!` zmieni ją na `false`, a program nie uruchomi instrukcji warunkowych.

Działanie operatora NOT jest bardzo proste do zrozumienia (tłumaczenie słów Bizarro: ten operator jest skomplikowany, jeśli jednak będziesz długo go używał, przyzwyczaisz się do niego).

Zagnieżdżanie instrukcji warunkowych

Programowanie w dużej części polega na podejmowaniu decyzji na podstawie informacji podanych przez użytkownika i warunków, które wystąpiły w programie. Im więcej wyborów dokonuje skrypt, tym więcej możliwych skutków i wyższa „inteligencja" programu. *Po* przetworzeniu jednej instrukcji warunkowej często okazuje się, że trzeba podjąć następne decyzje.

Załóżmy, że chcesz rozbudować program wyświetlający propozycje rozrywki na piątkowy wieczór o obsługę pozostałych dni tygodnia. Taki skrypt musi najpierw ustalić, jaki jest dzień, a następnie określić, co użytkownik może zrobić danego wieczoru. Jedna instrukcja warunkowa może sprawdzać, czy dany dzień to piątek. Jeśli tak, następna seria instrukcji warunkowych określa rozrywkę na wieczór:

```
if (dayOfWeek == 'Piątek') {
 var fridayCash = prompt('Ile chcesz wydać?', '');
 if (fridayCash >= 100) {
   alert('Możesz iść na kolację i do kina.');
 } else if (fridayCash >= 50) {
   alert('Możesz iść na dobrą kolację.');
 } else if (fridayCash >= 25) {
   alert('Możesz iść do kina.');
 } else {
   alert('Spędzisz wieczór przed telewizorem.');
 }
}
```

Pierwszy warunek sprawdza, czy wartość zapisana w zmiennej `dayOfWeek` to łańcuch znaków `'Piątek'`. Jeśli tak jest, pojawia się okno dialogowe, które pobiera od użytkownika informacje. Następnie program uruchamia dalsze instrukcje warunkowe. Pierwszy warunek, `(dayOfWeek == 'Piątek')`, to wstęp do serii następnych instrukcji warunkowych. Jeśli zmienna `dayOfWeek` ma inną wartość, warunek jest fałszywy i skrypt pominie zagnieżdżone instrukcje.

Wskazówki na temat pisania instrukcji warunkowych

Zagnieżdżone instrukcje warunkowe w poprzednim punkcie mogą wydawać się skomplikowane. Zawierają wiele par `()` i `{}` oraz instrukcji `else` i `if`. Jeśli popełnisz pomyłkę w jednym z istotnych fragmentów instrukcji warunkowych, skrypt przestanie działać. Są jednak techniki, które ułatwiają korzystanie z takich instrukcji.

- **Dodawaj oba nawiasy klamrowe przed wpisaniem kodu wewnątrz nich**. Jednym z najczęstszych błędów jest pominięcie końcowego nawiasu klamrowego w instrukcji warunkowej. Aby uniknąć takiej pomyłki, najpierw dodaj warunek i oba nawiasy, a następnie wpisz kod JavaScript, uruchamiany, jeśli dany warunek jest spełniony. Możesz zacząć od następującego fragmentu:

```
if (dayOfWeek=='Piątek') {

}
```

  Najpierw należy wpisać klauzulę `if` i pierwszy nawias klamrowy, dwukrotnie wcisnąć klawisz *Enter*, a następnie dodać końcowy nawias klamrowy. Po poprawnym przygotowaniu podstawowej struktury kliknij pusty wiersz między nawiasami klamrowymi i dodaj kod JavaScript.

- **Stosuj wcięcia w kodzie w nawiasach klamrowych**. Struktura instrukcji warunkowej jest lepiej widoczna po wcięciu całego kodu JavaScript w nawiasach klamrowych:

```
if (a < 10 && a > 1) {
  alert("Wartość " + a + " jest większa od 1 i mniejsza od 10.");
}
```

  Wcięcie kodu przez dodanie kilku odstępów (lub wciśnięcie klawisza tabulacji) pomaga dostrzec, który fragment ma uruchomić program w ramach instrukcji warunkowej. W instrukcjach zagnieżdżonych warto dodawać wcięcia na każdym poziomie:

```
if (a < 10 && a > 1) {
  // Pierwszy poziom wcięć dla pierwszego warunku.
  alert("Wartość " + a + " jest większa od 1 i mniejsza od 10.");
    if (a==5) {
      // Drugi poziom wcięć dla drugiego warunku.
      alert(a + " to połowa z 10.");
    }
}
```

- **Używaj sekwencji == do porównywania wartości**. Przy sprawdzaniu, czy dwie wartości są sobie równe, pamiętaj o używaniu operatora równości:

```
if (name == 'Robert') {
```

  Często spotykany błąd to użycie jednego znaku równości:

```
if (name = 'Robert') {
```

  Pojedynczy znak równości zapisuje wartość w zmiennej, dlatego łańcuch znaków `'Robert'` zostanie przypisany do zmiennej `name`. Niestety, interpreter potraktuje tę operację jako prawdziwą, dlatego powyższy warunek zawsze będzie miał wartość `true`.

Przykład — używanie instrukcji warunkowych

Instrukcje warunkowe należą do zestawu codziennych narzędzi programisty języka JavaScript. W tym przykładzie użyjesz takich instrukcji do sterowania działaniem skryptu.

Uwaga: Informacje o przykładowych plikach znajdziesz w uwadze na stronie 46.

1. **Otwórz w edytorze tekstu plik *conditional.html* z katalogu *R03*.**

   Program ma najpierw pobierać liczbę od użytkownika. Wspomniany plik zawiera już znaczniki `<script>` w sekcji nagłówkowej i w ciele strony.

2. **W pierwszej parze znaczników `<script>` (w sekcji nagłówkowej) wpisz kod wyróżniony pogrubieniem:**

```
<script>
var luckyNumber = prompt('Jaka jest Twoja szczęśliwa liczba?','');
</script>
```

   Ten wiersz kodu otwiera okno dialogowe, zadaje pytanie i zapisuje w zmiennej `luckyNumber` dane wpisane przez użytkownika. Następnie należy dodać instrukcję warunkową, która sprawdza, jakie dane internauta podał w oknie dialogowym.

3. **Znajdź drugą parę znaczników `<script>` (w ciele strony) i dodaj w nich kod wyróżniony pogrubieniem:**

```
<script>
if (luckyNumber == 7) {
</script>
```

   Nie zapomnij o użyciu podwójnego znaku równości (==), który pozwala sprawdzić, czy dwie wartości są równe. To początek instrukcji warunkowej. Ten fragment sprawdza, czy użytkownik wpisał liczbę 7.

4. **Wciśnij dwukrotnie klawisz *Enter* i dodaj zamykający nawias klamrowy, aby kod wyglądał następująco:**

```
<script>
if (luckyNumber == 7) {

}
</script>
```

   Zamykający nawias klamrowy kończy instrukcję warunkową. Kod JavaScript umieszczony między takimi nawiasami zadziała tylko wtedy, gdy warunek będzie spełniony.

Uwaga: Na stronie 105 dowiedziałeś się, że warto dodać zamykający nawias klamrowy przed wpisaniem kodu uruchamianego w instrukcji warunkowej.

---

5. **Kliknij pusty wiersz nad zamykającym nawiasem klamrowym. Wciśnij dwa razy klawisz spacji i dodaj poniższy kod:**

```
document.write("7 to także moja szczęśliwa liczba!");
```

   Dwa odstępy przed kodem dodają wcięcie w wierszu. Pozwala to szybko dostrzec, że kod jest częścią instrukcji warunkowej. Użyty tu kod JavaScript nie jest niczym nowym. Instrukcja ta wyświetla wiadomość na stronie.

6. **Zapisz plik i wyświetl go w przeglądarce. Kiedy pojawi się okno dialogowe, wpisz 7.**

   Powinieneś zobaczyć wiadomość „7 to także moja szczęśliwa liczba!" pod nagłówkiem wczytanej strony. Jeśli jej nie ma, sprawdź, czy poprawnie przepisałeś kod (na stronie 51 znajdziesz wskazówki na temat analizy niedziałających skryptów). Odśwież stronę, lecz tym razem podaj inną liczbę. Pod nagłówkiem nie powinna pojawić się żadna informacja. Do wyświetlania innych wiadomości posłuży *klauzula* `else`.

CZĘSTO ZADAWANE PYTANIA

Dlaczego dwa zestawy znaczników skryptu?

Dlaczego znaczniki <script> pojawiają się dwa razy — najpierw w sekcji nagłówka strony, a następnie w jej treści?

Podczas stosowania metody `document.write()` w celu dodawania zawartości do strony jej wywołanie należy umieścić dokładnie w tym miejscu, w którym chcemy, by pojawił się komunikat — w naszym przypadku jest to główna część strony, poniżej znacznika `<h1>`. Pierwszy zestaw znaczników skryptu został umieszczony w sekcji nagłówka strony, gdyż chcemy, by okienko z pytaniem pojawiło się wcześniej. Jeśli wywołanie metody `prompt()` przesuniemy z nagłówka do głównej części strony (dalej — sam wypróbuj, co się stanie), przed pojawieniem się okienka z pytaniem zostanie wyświetlona tylko część zawartości. Ponieważ na tym etapie kod JavaScript jest wykonywany bezzwłocznie, przeglądarka, zanim wyświetli dalszą część zawartości strony, będzie musiała poczekać, aż użytkownik poda odpowiedź w okienku dialogowym. Innymi słowy, strona będzie wyglądać dziwnie.

Gdy natomiast umieścimy wywołanie metody `prompt()` w sekcji `<head>`, w momencie wyświetlania okienka dialogowego strona będzie zupełnie pusta. Takie rozwiązanie jest nieco lepsze. W następnym rozdziale dowiesz się, jak można dodawać nowe treści i umieszczać je w dowolnym miejscu strony bez korzystania z metody `document.write()`. Kiedy już poznasz tę technikę, będziesz mógł zamieszczać cały kod JavaScript w jednym miejscu strony.

7. **Ponownie otwórz edytor kodu i dodaj do strony kod wyróżniony pogrubieniem:**

```
<script>
if (luckyNumber == 7) {
  document.write("<p>7 to także moja szczęśliwa liczba!<p>");
} else {
  document.write("<p>Twoja szczęśliwa liczba to " + luckyNumber
  ➥+ "!</p>");
}
</script>
```

Klauzula `else` zawiera rezerwowy komunikat. Jeśli użytkownik nie wpisze wartości `7`, zobaczy inny komunikat z informacją o szczęśliwej liczbie. Aby uzupełnić przykład, można dodać instrukcję `else if`, która sprawdza dalsze wartości i wyświetla następną wiadomość.

8. **Dodaj do skryptu wiersze wyróżnione pogrubieniem:**

```
<script>
if (luckyNumber == 7) {
  document.write("<p>7 to także moja szczęśliwa liczba!</p>");
} else if (luckyNumber == 13 || luckyNumber == 24) {
  document.write("<p>Naprawdę " + luckyNumber + "? To pechowa
  ➥liczba!</p>");
} else {
  document.write("<p>Twoja szczęśliwa liczba to " + luckyNumber +
  ➥"!</p>");
}
</script>
```

W tej wersji skrypt najpierw sprawdza, czy zmienna `luckyNumber` ma wartość `7`. Jeśli jest to inna liczba, program uruchamia blok `else if`. Ta instrukcja warunkowa składa się z dwóch warunków — `luckyNumber == 13` i `luckyNumber == 24`. Sekwencja `||` (logiczny operator LUB) sprawia, że cała instrukcja jest prawdziwa, jeśli spełniony jest choć jeden z dwóch warunków. Dlatego jeżeli użytkownik poda wartość `13` **lub** `24`, na stronie pojawi się informacja o pechowej liczbie.

Uwaga: Aby dodać logiczny operator LUB (||), należy dwukrotnie wpisać kombinację *Shift+*.

Wyświetl stronę w przeglądarce i wpisz w oknie dialogowym liczbę 13. Odśwież stronę i wprowadź inne wartości, a także litery lub inne znaki. Zauważ, że jeśli podasz słowo albo znaki nieliczbowe, program uruchomi końcową klauzulę `else` i doda wiadomość typu: „Twoja szczęśliwa liczba to asdfg!". Ponieważ nie ma to sensu, należy wyświetlić drugie okno dialogowe, jeżeli użytkownik za pierwszym razem nie wpisze liczby.

9. **Wróć do edytora tekstu i znajdź pierwszą parę znaczników `<script>` w sekcji nagłówkowej. Dodaj kod wyróżniony pogrubieniem:**

```
<script>
var luckyNumber = prompt('Jaka jest Twoja szczęśliwa liczba?','');
luckyNumber = parseInt(luckyNumber, 10);
</script>
```

Nowy wiersz kodu przekazuje wartość zmiennej `luckyNumber` do funkcji `parseInt()`. To polecenie języka JavaScript przyjmuje wartość i próbuje przekształcić ją na liczbę całkowitą (na przykład -20, 0, 1, 5 lub 100). Więcej o tej funkcji dowiesz się na stronie 587, na razie jednak zapamiętaj, że jeśli użytkownik wpisze tekst typu „ha, ha", polecenie `parseInt()` nie zdoła przekształcić go na liczbę. W zamian zwróci specjalną wartość języka JavaScript, `NaN` (czyli nie liczbę). Tej informacji można użyć do wyświetlenia następnego okna dialogowego, jeśli użytkownik nie wpisał liczby.

10. **Dodaj do skryptu kod wyróżniony pogrubieniem:**

```
<script type="text/javascript">
var luckyNumber = prompt('Jaka jest Twoja szczęśliwa liczba?','');
luckyNumber = parseInt(luckyNumber, 10);
if (isNaN(luckyNumber)) {
  luckyNumber = prompt('Proszę, podaj swoją szczęśliwą liczbę','');
}
</script>
```

Także tu przydatna była instrukcja warunkowa. W warunku `isNaN(lucky` ↪`Number)` użyto następnego polecenia JavaScript. Sprawdza ono, czy dany element jest liczbą, a dokładniej — czy wartość zmiennej `luckyNumber` liczbą **nie** jest. Jeśli ta wartość to nie liczba (użytkownik wpisał na przykład „asklsdkl"), pojawi się drugie okno dialogowe z nowym pytaniem. Jeśli użytkownik wprowadzi liczbę, program pominie wyświetlanie dodatkowego okna.

Zapisz stronę i wyświetl ją w przeglądarce. Tym razem wpisz w oknie dialogowym słowo i kliknij przycisk *OK*. Powinno pojawić się drugie okno dialogowe. Wpisz w nim liczbę. Oczywiście zakładamy, że użytkownik rzeczywiście pomylił się przy wprowadzaniu pierwszej wartości i nie powtórzy błędu. Niestety, jeśli internauta poda słowo także za drugim razem, wystąpi znany już problem. W następnym podrozdziale dowiesz się, jak go rozwiązać.

Uwaga: Kompletną wersję tego przykładu znajdziesz w pliku *complete_conditional.html* w katalogu *R03*.

Obsługa powtarzających się zadań za pomocą pętli

Czasem skrypt musi wielokrotnie powtórzyć te same operacje. Załóżmy, że formularz zawiera 30 pól tekstowych. Kiedy użytkownik chce przesłać formularz, należy sprawdzić, czy wypełnił wszystkie pola. Oznacza to, że trzeba przeprowadzić tę samą operację (sprawdzić, czy pole nie jest puste) 30 razy. Ponieważ komputery dobrze nadają się do wykonywania takich zadań, język JavaScript powinien udostępniać narzędzia do szybkiego wielokrotnego wykonywania tych samych zadań.

W języku technicznym powtarzanie operacji jest nazywane wykonywaniem zadań w *pętli*. Ponieważ pętle występują w kodzie JavaScript niezwykle często, dostępnych jest kilka ich wersji. Wszystkie pełnią tę samą funkcję, jednak działają w nieco odmienny sposób.

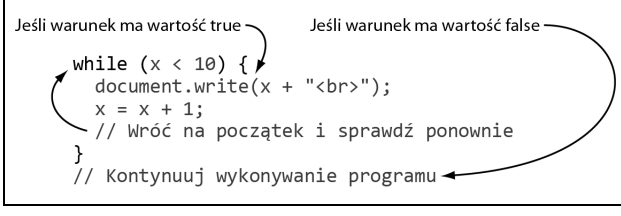
Pętle while

Pętla `while` wykonuje fragment kodu, dopóki dany warunek (warunek pętli `while`) jest spełniony. Podstawowa struktura tej pętli wygląda następująco:

```
while (warunek) {
  // Powtarzany kod JavaScript.
}
```

Pierwszy wiersz rozpoczyna instrukcję `while`. Podobnie jak w instrukcjach warunkowych, warunek należy umieścić w nawiasach po słowie kluczowym, którym tu jest `while`. Warunkiem może być dowolne wyrażenie, którego można użyć w instrukcji warunkowej, na przykład `x > 10` lub `answer == 'tak'`. Także, podobnie jak w instrukcjach warunkowych, interpreter wykonuje cały kod zapisany pomiędzy otwierającym i zamykającym nawiasem klamrowym, *jeśli* warunek jest spełniony.

Różnica polega na tym, że kiedy interpreter dojdzie do zamykającego nawiasu klamrowego instrukcji `while`, nie przechodzi do następnego wiersza programu, ale wraca na początek instrukcji `while` i ponownie sprawdza warunek. Jeśli nadal jest on spełniony, interpreter jeszcze raz uruchamia kod JavaScript podany między nawiasami klamrowymi. Proces ten kończy się, kiedy warunek przyjmuje wartość `false`. Wtedy program przechodzi do pierwszej instrukcji pod pętlą (patrz rysunek 3.5).



Rysunek 3.5. *Pętla while uruchamia kod JavaScript zapisany między nawiasami klamrowymi, jeśli warunek (tu jest to x < 10) ma wartość true*

Załóżmy, że chcesz wyświetlić na stronie liczby od 1 do 5. Można to zrobić w następujący sposób:

```
document.write('Liczba 1 <br>');
document.write('Liczba 2 <br>');
document.write('Liczba 3 <br>');
document.write('Liczba 4 <br>');
document.write('Liczba 5 <br>');
```

Zauważ, że każdy wiersz kodu jest prawie identyczny — zmienia się tylko liczba. Pętla umożliwia bardziej wydajne osiągnięcie tego samego efektu:

```
var num = 1;
while (num <= 5) {
  document.write('Liczba ' + num + '<br>');
  num += 1;
}
```

Pierwszy wiersz kodu (`var num = 1;`) nie jest częścią pętli `while`. Ten fragment tworzy zmienną, która będzie przechowywać liczbę wyświetlaną na stronie. Drugi wiersz to początek pętli. W tym miejscu należy podać warunek. Dopóki wartość zmiennej `num` jest mniejsza od lub równa 5, skrypt uruchamia kod zapisany między nawiasami klamrowymi. Przy pierwszym sprawdzaniu warunku zmienna `num` ma wartość `1`, dlatego warunek jest spełniony (liczba 1 jest mniejsza od 5), program wykonuje polecenie `document.write()` i dodaje do strony kod `'Liczba 1<br>'` (`<br>` to znacznik końca wiersza w języku HTML, który powoduje, że każda liczba znajdzie się w odrębnym wierszu).

Wskazówka: Instrukcję `num += 1` (która dodaje 1 do wartości zapisanej w zmiennej `num`) można zapisać także w bardziej zwięzły sposób:

```
num++
```

Ten skrócony zapis także dodaje 1 do zmiennej `num` (więcej informacji o tej notacji znajdziesz w tabeli 2.3 na stronie 72).

Ostatni wiersz pętli (`num += 1`) jest bardzo istotny. Nie tylko zwiększa wartość zmiennej `num` o 1, co umożliwia wyświetlenie następnej liczby (na przykład 2), ale sprawia, że w pewnym momencie warunek przestanie być prawdziwy (jeśli operator += wydaje Ci się dziwny, zajrzyj na stronę 72, gdzie znajdziesz więcej informacji o sposobie jego działania). Ponieważ kod JavaScript w instrukcji `while` działa, dopóki warunek jest spełniony, trzeba zmienić jedną z wartości w warunku, aby zatrzymać pętlę i przejść do dalszej części skryptu. Jeśli warunek będzie zawsze prawdziwy, powstanie *pętla nieskończona*, a program nigdy nie przestanie działać. Zastanów się, co się stanie po usunięciu z pętli ostatniego wiersza:

```
var num = 1;
while (num <= 5) { // To pętla nieskończona.
  document.write('Liczba ' + num + '<br>');
}
```

Przy pierwszym uruchamianiu pętli program sprawdza, czy 1 jest mniejsze od lub równe 5. Odpowiedź to „tak", dlatego wykonywane jest polecenie `document.write()`. Na końcu pętli (ostatni nawias klamrowy) interpreter wraca na początek pętli i ponownie sprawdza warunek. Zmienna `num` wciąż ma wartość 1, dlatego warunek nadal jest prawdziwy i skrypt uruchamia polecenie `document.write()`. Interpreter znów wraca na początek pętli i trzeci raz sprawdza warunek. Łatwo się domyślić, że doprowadzi to do wyświetlenia nieskończonej liczby wierszy z napisem „Liczba 1".

Ten prosty przykład ilustruje też elastyczność, jaką zapewniają pętle. Załóżmy, że chcesz wyświetlić liczby z przedziału od 1 do 100, a nie od 1 do 5. Zamiast dodawać wiele nowych wierszy z poleceniem `document.write()`, wystarczy zmienić warunek:

```
var num = 1;
while (num <= 100) {
  document.write('Liczba ' + num + '<br>');
  num = num + 1;
}
```

Ta pętla zadziała 100 razy i wyświetli na stronie 100 wierszy.

Pętle i tablice

Pętle są przydatne przy obsłudze standardowej struktury języka JavaScript — tablicy. Jak możesz przeczytać na stronie 77, tablica to kolekcja danych, która przypomina listę zakupów. Kiedy wybierasz się do sklepu, działasz w pewnego rodzaju pętli. Szukasz elementu z listy, a kiedy go znajdziesz, wkładasz towar do koszyka i przechodzisz do poszukiwania następnego produktu — i tak dalej, aż dodasz ostatnią pozycję na liście. Wtedy zakończyłeś zakupy (przypomina to wyjście z pętli) i możesz udać się do kasy (czyli przejść do następnego etapu programu).

Pętli w języku JavaScript można używać do poruszania się po elementach tablicy i wykonywania operacji na każdym z nich. Załóżmy, że tworzysz program wyświetlający kalendarz, który został zbudowany w całości za pomocą języka JavaScript. Chcesz, aby w kalendarzu znalazła się nazwa każdego dnia tygodnia. Możesz zacząć od zapisania tych nazw w tablicy:

```
var days = ['Poniedziałek', 'Wtorek', 'Środa', 'Czwartek',
    ↪'Piątek', 'Sobota', 'Niedziela'];
```

Uwaga: Symbol ↪ w powyższym fragmencie informuje, że cały kod to jedna instrukcja. Ponieważ szerokość strony książki czasem uniemożliwia zapisanie całej instrukcji w jednym wierszu, symbol ↪ określa, że połączony nim kod stanowi całość. Jeśli będziesz wprowadzał taki fragment w edytorze tekstu, powinieneś wpisać kod w jednym wierszu (i pominąć symbol ↪).

Następnie można przejść w pętli po wszystkich elementach tablicy i wyświetlić je na stronie. Pamiętaj, że aby uzyskać dostęp do wartości tablicy, należy podać indeks. Na przykład pierwszy element tablicy `days` (`Poniedziałek`) można pobrać za pomocą instrukcji `days[0]`. Drugi element to `days[1]` i tak dalej.

Aby wyświetlić wszystkie wartości tablicy za pomocą pętli `while`, można użyć następującego kodu:

```
var counter = 0;
while (counter < days.length) {
  document.write(days[counter] + ', ');
  counter++;
}
```

Pierwszy wiersz, `var counter = 0`, **inicjuje** zmienną `counter`, która jest używana zarówno w warunku, jak i jako indeks elementów tablicy. Sam warunek, `counter < days.length`, pozwala sprawdzić, czy zmienna `counter` ma wartość mniejszą niż liczba elementów tablicy (na stronie 80 dowiedziałeś się, że liczbę tę można sprawdzić za pomocą właściwości `length` tablicy). Przykładowy skrypt sprawdza, czy

zmienna `counter` ma wartość mniejszą od 7 (to liczba dni tygodnia). Jeśli tak jest, program uruchamia kod w pętli i wyświetla na stronie nazwę dnia tygodnia, przecinek oraz odstęp, a następnie zwiększa wartość zmiennej o 1 (instrukcja `counter++` działa tak samo jak `counter += 1`; patrz wskazówka na stronie 72). Po wykonaniu kodu w pętli skrypt ponownie sprawdza warunek. Pętla działa, dopóki warunek jest spełniony. Proces ten przedstawiono na rysunku 3.6.

```
var counter = 0;
while (counter < days.length) {
   document.write(days[counter] + ', ');
   counter++;
}
```

Wartość zmiennej `counter` przed sprawdzeniem warunku	Warunek	Wejście w pętlę?	`days [counter]`	Wartość zmiennej `counter` po instrukcji `counter++`
0	0 < 7	Tak	days [0]	1
1	1 < 7	Tak	days [1]	2
2	2 < 7	Tak	days [2]	3
3	3 < 7	Tak	days [3]	4
4	4 < 7	Tak	days [4]	5
5	5 < 7	Tak	days [5]	6
6	6 < 7	Tak	days [6]	7
7	7 < 7	Nie		

Rysunek 3.6. *W tej pętli warunek jest sprawdzany osiem razy. Ostatni test sprawdza, czy 7 jest mniejsze od 7. To nieprawda, dlatego interpreter kończy wykonywanie instrukcji while i przechodzi do dalszej części skryptu. Ostateczny efekt uruchomienia tego programu to tekst: „Poniedziałek, Wtorek, Środa, Czwartek, Piątek, Sobota, Niedziela". Warto zauważyć, że po niedzieli także zostanie wyświetlony przecinek. Aby uniknąć tego niepotrzebnego przecinka, można by użyć metody join() obiektu tablicy; to zaawansowane rozwiązanie zostało opisane na stronie 606*

Pętle for

Język JavaScript udostępnia także *pętlę* `for`, która jest bardziej zwięzła (i trochę bardziej skomplikowana). Pętle `for` służą zwykle do powtarzania danej operacji określoną liczbę razy, dlatego mają licznik, warunek i metodę zmieniania wartości licznika. Pętle `for` często umożliwiają wykonanie tych samych zadań, co pętle `while`, jednak w mniejszej liczbie wierszy. Przypomnij sobie pętlę `while` ze strony 109:

```
var num = 1;
while (num <= 100) {
  document.write('Liczba ' + num + '<br>');
  num += 1;
}
```

Przy użyciu pętli `for` ten sam efekt można uzyskać w trzech wierszach kodu:

```
for (var num=1; num<=100; num++) {
  document.write('Liczba ' + num + '<br>');
}
```

Początkowo pętle `for` mogą wydawać się skomplikowane, jednak kiedy zrozumiesz poszczególne części instrukcji `for`, nie powinieneś mieć problemów z ich używaniem. Każda pętla tego typu zaczyna się od słowa kluczowego `for`, po którym następują

nawiasy z trzema elementami i para nawiasów klamrowych. Podobnie jak w pętlach `while`, fragment w nawiasach klamrowych (tu jest to `document.write('Liczba ' + num + '<br>')`) to kod JavaScript uruchamiany w ramach pętli.

W tabeli 3.2 znajdziesz opis trzech elementów podawanych w nawiasach; jednak tutaj krótko je przedstawię: są to kolejno inicjalizacja licznika, sprawdzenie warunku oraz zmiana wartości licznika. Pierwsza część (`var num=1;`) inicjuje zmienną licznika. Ta operacja ma miejsce tylko raz, w momencie uruchamiania instrukcji. Druga część to warunek sprawdzany przed uruchomieniem kodu w pętli. Fragment trzeci to operacja wykonywana po zakończeniu działania tego kodu. Zwykle polega ona na zmianie wartości licznika, dlatego w pewnym momencie warunek przyjmuje wartość `false`, a pętla kończy działanie.

Tabela 3.2. *Elementy pętli for*

Element pętli	Znaczenie	Czas działania
`for`	Rozpoczyna pętlę `for`.	
`var num = 1;`	Przypisuje zmiennej `num` wartość `1`.	Jeden raz przy uruchamianiu pętli.
`num <= 100;`	Czy `num` ma wartość mniejszą od lub równą 100? Jeśli tak, należy powtórzyć pętlę. Jeśli nie, należy ją pominąć i kontynuować wykonywanie skryptu.	Na początku instrukcji i przed każdym powtórzeniem pętli.
`num++;`	Dodaje 1 do zmiennej `num`. Działa tak samo jak `num = num + 1` oraz `num += 1`.	Po każdym powtórzeniu pętli.

Ponieważ pętle `for` umożliwiają łatwe powtarzanie serii operacji określoną liczbę razy, dobrze nadają się do przechodzenia po elementach tablicy. Na rysunku 3.5 widoczna jest pętla `while`, która zapisuje na stronie wszystkie elementy tablicy. To samo zadanie można wykonać za pomocą pętli `for`:

```
var days = ['Poniedziałek', 'Wtorek', 'Środa', 'Czwartek',
    ↪'Piątek', 'Sobota', 'Niedziela'];
for (var i = 0; i<days.length; i++) {
  document.write(days[i] + ', ');
}
```

Wskazówka: Doświadczeni programiści często używają bardzo krótkich nazw zmiennych licznika w pętlach `for`. W powyższym kodzie taką nazwą jest `i`. Pojedyncze litery (na przykład `i`, `j` i `z`) można szybko wpisać, a ponieważ zmienne tego typu są używane tylko w pętli, nie muszą mieć bardziej opisowych nazw, takich jak `licznik`.

Przykłady przedstawione do tej pory zwiększały liczbę do określonej wartości, a następnie kończyły działanie pętli, jednak można też odliczać wstecz. Załóżmy, że chcesz wyświetlić elementy tablicy od końca. Można to zrobić w następujący sposób:

```
var example = ['pierwszy', 'drugi', 'trzeci', 'ostatni'];
for (var j = example.length - 1; j >= 0; j--) {
  document.write(example[j] + '<br>');
}
```

W tym fragmencie wartość zmiennej `j` to początkowo liczba elementów tablicy pomniejszona o `1` (4–1=3). (A dlaczego pomniejszona o `1`? Dlatego, że indeks elementu w tablicy jest zawsze o jeden mniejszy od numeru miejsca, który dany

element zajmuje: pierwszy element tablicy ma indeks o wartości `0`, drugi — indeks o wartości `1`, a indeksem ostatniego jest długość tablicy pomniejszona o `1`. Innymi słowy, aby odwołać się do ostatniego elementu naszej przykładowej tablicy, musielibyśmy użyć wyrażenia `example[3]`).

Przy każdym powtórzeniu pętli należy sprawdzić, czy wartość tej zmiennej jest większa lub równa `0`. Jeśli tak, skrypt uruchamia kod między nawiasami klamrowymi, odejmuje 1 od `j` (`j--`) i ponownie sprawdza warunek. A zatem powyższa pętla przegląda zawartość tablicy, zaczynając od jej końca (elementu o indeksie `3`) i zmierzając w kierunku początku (elementu o indeksie `0`).

Pętle do-while

Istnieje też inny, mniej popularny rodzaj pętli — `do-while`. Pętle tego typu działają prawie identycznie z pętlami `while`. Ich podstawowa struktura wygląda następująco:

```
do {
  // Powtarzany kod JavaScript.
} while (warunek);
```

Pętle tego typu sprawdzają warunek na *końcu*, po wykonaniu pętli. Dlatego kod JavaScript w nawiasach klamrowych zawsze jest uruchamiany *przynajmniej raz*. Nawet jeśli warunek nie jest spełniony, skrypt sprawdza go dopiero po pierwszym wykonaniu kodu.

Takie rozwiązanie jest potrzebne stosunkowo rzadko, jednak jest bardzo przydatne, jeśli program ma prosić użytkownika o podanie danych. Dobrym przykładem jest program, który napisałeś we wcześniejszej części rozdziału (patrz strona 105). Skrypt ten prosi użytkownika o wpisanie liczby. Kod ma wbudowany system zabezpieczający, dlatego jeśli internauta nie poda odpowiednich danych, program powtarza prośbę. Jeżeli jednak wyjątkowo uparty użytkownik ponownie wpisze błędne informacje, na stronie pojawi się bezsensowny komunikat.

Przy użyciu pętli `do-while` można ponawiać prośbę o wpisanie liczby do czasu uzyskania odpowiednich danych. Aby zobaczyć, jak działa to podejście, zmodyfikuj kod utworzony na stronie 108.

1. **Otwórz w edytorze tekstu plik *conditional.html* utworzony na stronie 108.**

   Jeśli nie wykonałeś wspomnianego przykładu, możesz otworzyć plik *complete_conditional.html*. Najpierw musisz zastąpić kod w początkowej części strony pętlą `do-while`.

2. **Znajdź kod między znacznikami `<script>` w sekcji nagłówkowej strony i usuń kod wyróżniony pogrubieniem:**

   ```
   var luckyNumber = prompt('Jaka jest Twoja szczęśliwa liczba?','');
   luckyNumber = parseInt(luckyNumber, 10);
   if (isNaN(luckyNumber)) {
     luckyNumber = prompt('Proszę, podaj swoją szczęśliwą liczbę','');
   }
   ```

   Usunięty kod wyświetlał drugie okno dialogowe, które nie będzie już potrzebne. W zamian należy umieścić pozostały kod w pętli `do-while`.

3. **Umieść kursor przed pierwszym wierszem kodu (`var luckyNumber...`) i wpisz:**

```
do {
```

Ten kod to początek pętli. Teraz trzeba dokończyć pętlę i dodać sprawdzany warunek.

4. **Kliknij koniec ostatniego wiersza w tej sekcji i wpisz fragment `} while (isNaN(luckyNumber));`. Gotowy blok kodu powinien wyglądać następująco:**

```
do {
var luckyNumber = prompt('Jaka jest Twoja szczęśliwa liczba?','');
luckyNumber = parseInt(luckyNumber, 10);
} while (isNaN(luckyNumber));
```

Zapisz plik i wyświetl go w przeglądarce. Spróbuj wpisać w oknie dialogowym tekst lub inne symbole nieliczbowe. To samo okno będzie pojawiało się do czasu wprowadzenia liczby.

Słowo kluczowe `do` informuje interpreter, że rozpoczyna się pętla `do-while`. Następnie uruchamiane są dwa dalsze wiersze, które wyświetlają okno dialogowe i przekształcają odpowiedź na liczbę całkowitą. Dopiero potem skrypt sprawdza warunek, który jest taki sam jak w kodzie ze strony 108 (sprawdza, czy wprowadzone dane „nie są liczbą"). Jeśli użytkownik nie podał liczby, pętla jest uruchamiana ponownie, aż do wprowadzenia liczby. Korzystną cechą tego podejścia jest to, że okno pojawia się przynajmniej raz, dlatego jeśli użytkownik poda liczbę, skrypt od razu wyjdzie z pętli.

Pełny, działający kod tego przykładu można znaleźć w pliku *complete_do-while.html* w katalogu *R03*.

Funkcje — wielokrotne korzystanie z przydatnego kodu

Wyobraź sobie, że przydzielono Ci asystenta do pomocy w wykonywaniu wszelkich zadań (chyba pora umieścić tę książkę na półce z literaturą science-fiction). Załóżmy, że masz ochotę na pizzę. Asystent nie zna jeszcze okolicy, dlatego musisz udzielić mu szczegółowych wskazówek: „Wyjdź przez drzwi, skręć w prawo, wejdź do windy, zjedź na parter, wyjdź z budynku…" i tak dalej. Asystent wykonuje polecenie i przynosi pizzę. Po paru godzinach masz ochotę na następny kawałek. Nie musisz wtedy powtarzać wszystkich instrukcji („Wyjdź przez drzwi, skręć w prawo…"). Asystent wie już, gdzie znajduje się pizzeria, dlatego wystarczy wydać polecenie: „Przynieś kawałek pizzy", a pomocnik dostarczy zamówienie.

Oznacza to, że wystarczy podać szczegółowe instrukcje *jeden raz*. Asystent zapamięta wszystkie kroki i po usłyszeniu polecenia „Przynieś kawałek pizzy" zniknie, a po pewnym czasie pojawi się z zamówieniem. Język JavaScript udostępnia podobny mechanizm — *funkcje*. Funkcja to zbiór instrukcji podany na początku skryptu i przypominający szczegółowe polecenia wydane asystentowi. Instrukcje funkcji nie są wykonywane w miejscu jej utworzenia. Przeglądarka zapisuje je w pamięci, a programista może wywołać funkcję w dowolnym momencie, kiedy chce wykonać określone operacje.

Funkcje są nieocenionym narzędziem, które pozwala w wydajny sposób wielokrotnie wykonywać powtarzalne operacje. Załóżmy, że chcesz utworzyć stronę z galerią fotografii, która ma zawierać 50 miniatur. Kiedy użytkownik kliknie jeden z małych obrazków, program ma przyciemnić tło strony oraz wyświetlić tytuł i większą wersję wybranego zdjęcia. Za każdym razem, kiedy internauta wybierze rysunek, proces ten się powtarza. Dlatego na stronie z 50 miniaturami skrypt musi wykonać serię tych samych operacji 50 razy. Na szczęście nie trzeba pisać 50 wersji prawie takiego samego kodu, aby utworzyć galerię fotografii. W zamian wystarczy napisać funkcję z wszystkimi operacjami i uruchamiać ją przy każdym kliknięciu miniatury. Kod funkcji wystarczy napisać raz, a następnie można wielokrotnie ją uruchamiać.

Podstawowa struktura funkcji wygląda następująco:

```
function nazwaFunkcji() {
  // Uruchamiany kod JavaScript.
}
```

Słowo kluczowe `function` informuje interpreter o tym, że natrafił na funkcję (podobne przeznaczenie mają inne słowa kluczowe: `if` rozpoczyna instrukcję `if-else`, a `var` tworzy zmienną). Następnie należy podać nazwę funkcji, którą — podobnie jak nazwę zmiennej — programista może określić samodzielnie. Obowiązują przy tym te same reguły, o jakich należy pamiętać przy tworzeniu nazw zmiennych (zasady te znajdziesz na stronie 64). Nazwy funkcji często zawierają czasownik, na przykład `obliczPodatek`, `pobierzWysokoscEkranu`, `zaktualizujStrone` lub `wygasRysunek`. Aktywne wyrażenia podkreślają, że kod wykonuje pewne zadanie, i pomagają odróżnić nazwy funkcji od nazw zmiennych.

Bezpośrednio po nazwie znajduje się para nawiasów, które są cechą charakterystyczną funkcji. Po nawiasach następuje odstęp, nawias klamrowy, kod w języku JavaScript i końcowy, zamykający nawias klamrowy. Podobnie jak w instrukcjach `if`, nawiasy klamrowe wyznaczają początek i koniec kodu JavaScript funkcji.

Uwaga: Funkcje, podobnie jak instrukcje `if-else`, są bardziej czytelne, jeśli kod w nawiasach klamrowych ma wcięcia. Programiści dodają przeważnie dwa odstępy lub tabulację na początku każdego wiersza.

Oto bardzo prosta funkcja, która wyświetla bieżącą datę w formacie „Sun May 12 2008":

```
function printToday() {
  var today = new Date();
  document.write(today.toDateString());
}
```

Nazwa tej funkcji to `printToday`. Zawiera ona tylko dwa wiersze kodu JavaScript, który pobiera bieżącą datę, przekształca ją na zrozumiały format (polecenie `toDate` ↪`String()`), a następnie wyświetla na stronie efekt tej operacji za pomocą standardowej metody `document.write()`. Na razie nie musisz znać sposobów obsługi dat. Poznasz je w dalszej części książki, na stronie 592.

Programiści zazwyczaj umieszczają funkcje na początku skryptu, aby można było używać ich w dalszym kodzie. Pamiętaj, że funkcja nie jest uruchamiana w miejscu jej utworzenia. Jej dodanie przypomina poinformowanie asystenta o tym, jak ma

dojść do pizzerii, bez natychmiastowego wysyłania go po posiłek. Kod funkcji jest zapisywany w pamięci przeglądarki i oczekuje na uruchomienie w odpowiednim momencie.

Jak można uruchomić funkcję? Programiści używają słowa *wywoływanie* na określenie procesu uruchomienia funkcji, kiedy ta ma wykonać swe zadania. Wywołanie funkcji polega na podaniu jej nazwy wraz z parą nawiasów. Aby na przykład uruchomić funkcję `printToday`, należy wpisać następującą instrukcję:

```
printToday();
```

Jak widać, funkcje można uruchamiać w zwięzły sposób. Jest to jedna z zalet funkcji. Po ich utworzeniu nie trzeba wpisywać wiele kodu, aby uzyskać pożądane efekty.

Uwaga: Przy wywoływaniu funkcji pamiętaj o dodaniu nawiasów. Ten element umożliwia wywołanie funkcji. Instrukcja `printToday` nie wywoła żadnej reakcji skryptu, natomiast kod `printToday()` spowoduje uruchomienie funkcji.

Krótki przykład

Ponieważ funkcje są niezwykle istotne, warto wykonać kilka zadań, aby przećwiczyć tworzenie i używanie funkcji na działającej stronie WWW.

1. **Otwórz w edytorze tekstu plik *print_date.html*.**

   Najpierw należy dodać funkcję w sekcji nagłówkowej dokumentu.

2. **Znajdź kod między znacznikami `<script>` w sekcji nagłówkowej strony i dodaj poniższy fragment:**

   ```
   function printToday() {
     var today = new Date();
     document.write(today.toDateString());
   }
   ```

   Prosta funkcja jest już gotowa, jednak na razie nie wykonuje swych zadań.

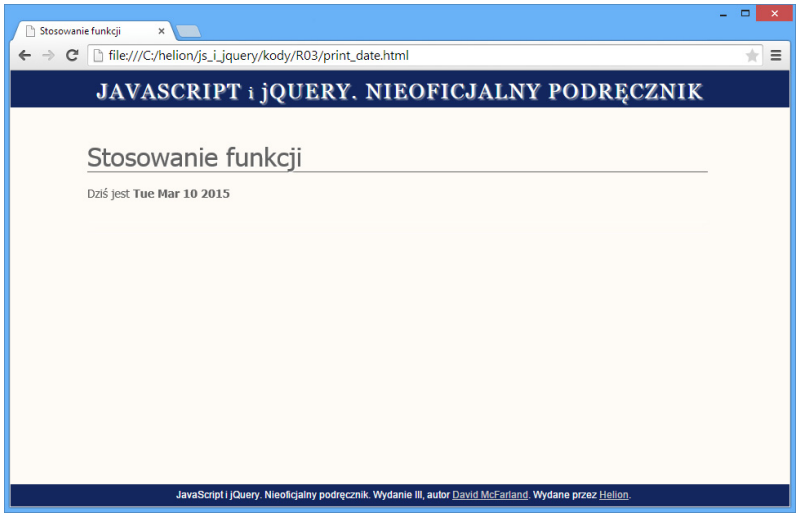
3. **Zapisz plik i wyświetl go w przeglądarce.**

   Nic się nie stanie — przynajmniej nic takiego, co można zauważyć. Przeglądarka wczytuje funkcję do pamięci i oczekuje na jej wywołanie, które dodasz w następnym kroku.

4. **Wróć do pliku *print_date.html* otwartego w edytorze tekstu. Znajdź znacznik `<p>` z tekstem rozpoczynającym się od słów „Dziś jest" i dodaj między znacznikami `<strong>` kod wyróżniony pogrubieniem:**

   ```
   <p>Dziś jest <strong>
   <script>printToday();</script>
   </strong></p>
   ```

Zapisz stronę i wyświetl ją w przeglądarce (wyniki zostały przedstawione na rysunku 3.7). Na stronie pojawi się aktualna data. Jeśli zechcesz wyświetlić ją także na dole strony, możesz powtórnie wywołać tę samą funkcję.



Rysunek 3.7. *Finalny efekt działania Twojej pierwszej funkcji. Oczywiście data, którą zobaczysz w przeglądarce, będzie inna niż widoczna na rysunku. To jedna z cech dynamicznej natury funkcji — działają one wtedy, gdy zostaną wywołane, a zatem każdego dnia, kiedy wywołasz funkcję printToday, wyświetli ona inną datę*

Przekazywanie danych do funkcji

Funkcje są jeszcze bardziej przydatne, kiedy przyjmują dane. Pomyśl ponownie o asystencie, którego wysyłasz po pizzę. Pierwsza „funkcja", opisana na stronie 115, zawiera instrukcje potrzebne do dotarcia do pizzerii, dokonania zakupu i powrotu do biura. Kiedy chcesz pizzę, „wywołujesz" funkcję przez wydanie asystentowi polecenia: „Przynieś mi pizzę!". Czasem chcesz otrzymać pizzę z peperoni, innym razem z dodatkowym serem lub oliwkami. Aby uwzględnić to w instrukcjach, możesz powiedzieć asystentowi, na którą pizzę masz ochotę.

Funkcje JavaScript także mogą przyjmować informacje. Są one przekazywane w *parametrach*, których funkcje używają do wykonywania zadań. Jeśli na przykład funkcja ma obliczyć łączną cenę zakupów z koszyka, musi wiedzieć, ile kosztuje każdy przedmiot i ile sztuk klient zamawia.

Najpierw, w momencie tworzenia funkcji, należy umieścić w nawiasach nową zmienną. Jest to *parametr*. Podstawowa struktura takiej funkcji wygląda następująco:

```
function nazwaFunkcji(parametr) {
  // Uruchamiany kod JavaScript.
}
```

Parametr to po prostu zmienna, dlatego musi mieć poprawną nazwę (wskazówki na ten temat znajdziesz na stronie 64). Załóżmy, że chcesz skrócić kod potrzebny do dodawania tekstu do strony. Możesz użyć do tego prostej funkcji, która zastępuje polecenie `document.write()` krótszą nazwą:

```
function print(message) {
  document.write(message);
}
```

Nowa funkcja ma nazwę `print` i przyjmuje jeden parametr, `message`. W momencie wywołania funkcja przyjmuje informacje (wyświetlaną wiadomość), a następnie używa polecenia `document.write()` do dodania komunikatu do strony. Funkcja

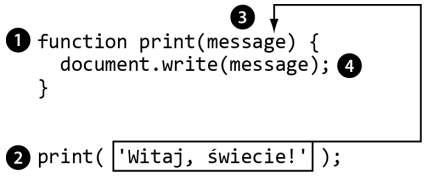
oczywiście nie wykonuje żadnych operacji do momentu jej wywołania, dlatego w innym miejscu strony należy ją uruchomić:

```
print('Witaj, świecie!');
```

W momencie uruchomienia tego kodu skrypt wywołuje funkcję `print` i przekazuje do niej tekst — łańcuch `'Witaj, świecie!'`. Następnie funkcja dodaje tekst „Witaj, świecie!" do strony. W języku technicznym proces przesyłania informacji do funkcji jest nazywany „przekazywaniem argumentów". W omawianej instrukcji tekst „Witaj, świecie!" to *argument*. Argumenty są wartościami przekazywanymi do funkcji i odpowiadają parametrom definiowanym podczas jej tworzenia.

Działanie nawet prostych funkcji może być nieco skomplikowane dla początkujących programistów. Poniżej znajdziesz szczegółowy opis wszystkich operacji, które ilustruje rysunek 3.8.

1. **Interpreter wczytuje funkcję i zapisuje ją w pamięci. Ten etap przygotowuje przeglądarkę do późniejszego uruchomienia funkcji.**
2. **Skrypt wywołuje funkcję i przekazuje do niej informacje — tekst „Witaj, świecie!".**
3. **Skrypt zapisuje informacje przekazane do funkcji w zmiennej `message`. Ten krok odpowiada instrukcji `var message = 'Witaj, świecie!';`.**
4. **Funkcja jest uruchamiana i wyświetla na stronie wartość zapisaną w zmiennej `message`.**



Rysunek 3.8. *Programiści zwykle tworzą funkcje przed miejscem ich wywołania. Tu funkcja print() znajduje się w trzech pierwszych wierszach kodu, natomiast jej kod jest uruchamiany dopiero w ostatnim wierszu*

Nasze możliwości nie ograniczają się jednak do jednego parametru — funkcja może ich mieć dowolnie wiele. Trzeba tylko określić w funkcji każdy z nich:

```
function nazwaFunkcji(parametr1, parametr2, parametr3) {
  // Uruchamiany kod JavaScript.
}
```

Następnie można wywołać funkcję, podając tę samą liczbę argumentów w odpowiedniej kolejności:

```
nazwaFunkcji(argument1, argument2, argument3);
```

Przy wywołaniu funkcji `nazwaFunkcji` wartość `argument1` jest zapisywana w zmiennej `parametr1`, `argument2` trafia do zmiennej `parametr2` i tak dalej. Załóżmy, że chcesz rozwinąć wcześniejszą funkcję `print` o możliwość przekazania znacznika HTML, który ma obejmować tekst dodawany do strony. Dzięki temu będzie można wyświetlać informacje w formie nagłówków lub akapitów. Nowa funkcja powinna wyglądać następująco:

```
function print(message,tag) {
  document.write('<' + tag + '>' + message + '</' + tag + '>');
}
```

A oto przykładowe wywołanie tej funkcji:

```
print('Witaj, świecie!', 'p');
```

Ta instrukcja przekazuje do funkcji dwa argumenty: `'Witaj, świecie!'` i `'p'`. Wartości te są zapisywane w dwóch zmiennych funkcji: `message` i `tag`. Efekt to nowy akapit dodany do strony: `<p>Witaj, świecie!</p>`.

Funkcje przyjmują dowolne zmienne i wartości języka JavaScript, a nie tylko łańcuchy znaków. Jako argument można przekazać na przykład tablicę, zmienną, liczbę lub wartość logiczną.

Pobieranie informacji z funkcji

Czasem funkcja po prostu wykonuje zadanie, na przykład wyświetla wiadomość na stronie, przenosi obiekt po ekranie lub sprawdza poprawność pól formularza. Jednak często programista chce pobrać dane z funkcji. W końcu funkcja „Przynieś mi pizzę!" nie będzie zbyt przydatna, jeśli jej wywołanie nie doprowadzi do otrzymania pysznego kawałka pizzy. Także funkcja obliczająca łączny koszt zakupów nie będzie wartościowa, jeśli nie będzie można pobrać tej wartości.

Niektóre opisane wcześniej wbudowane funkcje języka JavaScript zwracają wartość. Na przykład polecenie `prompt()` (patrz strona 74) wyświetla okno dialogowe z polem tekstowym, a następnie zwraca dane wpisane przez użytkownika. Wiesz już, jak zapisać zwróconą wartość w zmiennej i użyć jej:

```
var answer = prompt('W jakim miesiącu się urodziłeś?', '');
```

Skrypt zapisuje odpowiedź użytkownika w zmiennej `answer`. Następnie można sprawdzić wartość zmiennej w instrukcji warunkowej lub użyć jej do wielu innych operacji możliwych w języku JavaScript.

Aby zwracać wartość we własnych funkcjach, należy użyć słowa kluczowego `return` i zwracanej wartości:

```
function nazwaFunkcji(parametr1, parametr2) {
  // Uruchamiany kod JavaScript.
  return wartość;
}
```

Załóżmy, że chcesz obliczyć łączny koszt zakupów z uwzględnieniem podatku. Można to zrobić za pomocą następującego skryptu:

```
var TAX = .08; // 8-procentowy podatek od sprzedaży.
function calculateTotal(quantity, price) {
  var total = quantity * price * (1 + TAX);
  var formattedTotal = total.toFixed(2);
  return formattedTotal;
}
```

Pierwszy wiersz zapisuje wysokość podatku w zmiennej `TAX`. Umożliwia to szybką zmianę tej wartości przez zaktualizowanie jednego wiersza kodu. Trzy następne wiersze to definicja funkcji. Na razie nie musisz rozumieć, jak działa ten kod. Więcej informacji o operacjach na liczbach znajdziesz na stronie 587. Tu ważny jest czwarty wiersz funkcji — instrukcja `return`, która zwraca wartość zapisaną w zmiennej `formattedTotal`.

Aby użyć zwróconej wartości, zwykle należy zapisać ją w zmiennej. Utworzoną wcześniej funkcję można wywołać w następujący sposób:

```
var saleTotal = calculateTotal(2, 16.95);
document.write('Łączny koszt: $' + saleTotal);
```

W tym wywołaniu do funkcji przekazano wartości `2` i `16.95`. Pierwsza z nich określa liczbę produktów, a druga — ich cenę. Funkcja wylicza koszt całkowity oraz podatek i zwraca wartość sumaryczną. Wynik ten jest zapisywany w nowej zmiennej (`saleTotal`), używanej następnie w wywołaniu `document.write()` do wyświetlenia łącznej ceny zakupów z uwzględnieniem podatku.

Uwaga: Słowo kluczowe `return` powinno być ostatnią instrukcją funkcji, bo zaraz po tym, gdy tylko interpreter JavaScript je napotka, funkcja zostanie zakończona. Żaden kod umieszczony po tym słowie kluczowym nigdy nie zostanie wykonany.

Jednak zwracanej wartości nie trzeba zapisywać w zmiennej. Funkcji można użyć bezpośrednio w innej instrukcji:

```
document.write('Suma: $' + calculateTotal(2, 16.95));
```

Tu skrypt wywołuje funkcję, dodaje zwróconą przez nią wartość do łańcucha `'Suma: $'` i zapisuje tekst w dokumencie. Początkowo ten sposób używania funkcji może być mało czytelny, dlatego można wykonać dodatkową operację w postaci zapisania wyniku działania funkcji w zmiennej, a następnie użyć tej zmiennej w skrypcie.

Uwaga: Funkcje mogą zwracać tylko jedną wartość. Jeśli chcesz przekazać większą ich liczbę, zapisz wyniki w tablicy i zwróć ją. Więcej informacji o tablicach możesz znaleźć na stronie 77.

Unikanie konfliktów między nazwami zmiennych

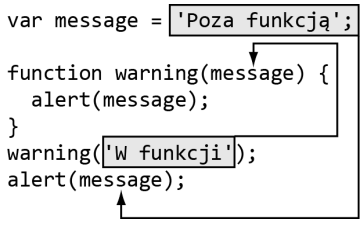
Jedną z wielkich zalet funkcji jest to, że zmniejszają ilość kodu, który trzeba napisać. Prawdopodobnie odkryjesz, że pewnych wartościowych funkcji używasz w wielu projektach. Na przykład funkcja doliczająca do ceny koszty wysyłki i podatek będzie przydatna w każdym formularzu z zamówieniami, dlatego można skopiować ją i wkleić w innych skryptach witryny lub wykorzystać w następnych projektach.

Kiedy programista przeniesie funkcję do gotowego skryptu, może pojawić się problem. Co się stanie, jeśli zmienna z tego programu i zmienna w funkcji mają tę samą nazwę? Której z nich użyje skrypt? Oto przykład:

```
var message = 'Poza funkcją';
function warning(message) {
  alert(message);
}
warning('W funkcji'); // 'W funkcji'
alert(message); // 'Poza funkcją'
```

Zauważ, że zmienna `message` pojawia się zarówno poza funkcją (wiersz 1.), jak i jako jej parametr. Parametr to zmienna, która przyjmuje wartość podaną w momencie wywołania funkcji. Tu kod `warning('W funkcji');` przekazuje do funkcji łańcuch znaków, który zostaje zapisany w zmiennej `message`. W skrypcie znajdują się więc dwie wersje zmiennej `message`. Co się stanie z wartością pierwszej zmiennej `message`, utworzonej w pierwszym wierszu programu?

Może się wydawać, że skrypt zastąpi pierwotną wartość zmiennej `message` nowym łańcuchem, `'W funkcji'`. To nieprawda. W momencie uruchamiania skryptu pojawią się dwa okna dialogowe. Pierwsze z nich wyświetli tekst „W funkcji", a drugi — łańcuch znaków „Poza funkcją". W kodzie pojawiają się dwie zmienne o nazwie `message`, jednak znajdują się one w innych miejscach (patrz rysunek 3.9).



Rysunek 3.9. *Parametr funkcji jest dostępny tylko w niej, dlatego jej pierwszy wiersz, function warning(message), tworzy nową zmienną message, widoczną tylko wewnątrz funkcji. Po zakończeniu jej działania zmienna znika*

Interpreter traktuje zmienne funkcji w inny sposób niż zmienne zadeklarowane i utworzone poza funkcjami. W języku technicznym można powiedzieć, że każda funkcja ma określony *zasięg*. Zasięg ten działa jak mur otaczający funkcję. Zmienne utworzone wewnątrz niej nie są widoczne w kodzie spoza tego muru. Zrozumienie działania zasięgu może początkowo sprawiać problemy, jednak jest to bardzo przydatny mechanizm. Ponieważ funkcje mają określony zasięg, nie trzeba się martwić, że parametry wywołają konflikty między nazwami zmiennych lub modyfikację wartości zmiennych spoza funkcji.

Do tej pory omówiono jedynie używanie zmiennych podanych jako parametry, jednak zmienne w funkcji można tworzyć także w standardowy sposób:

```
var message = 'Poza funkcją';
function warning() {
  var message = 'W funkcji';
  alert( message );
}
warning(); // 'W funkcji'
alert( message ); // 'Poza funkcją'
```

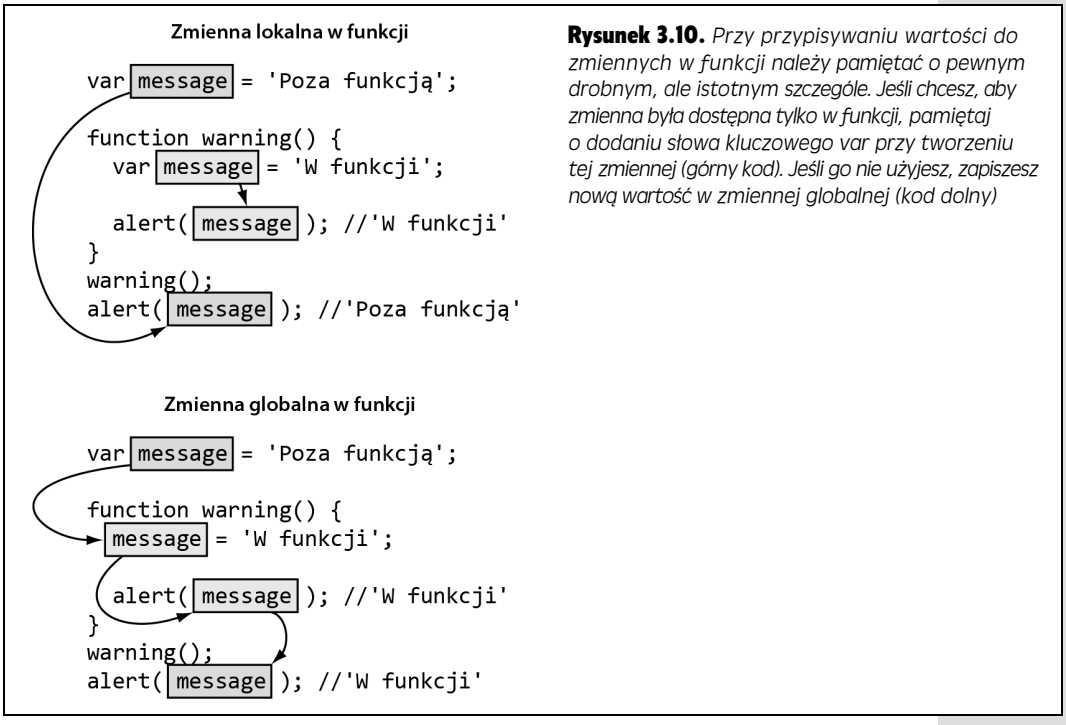
Ten kod dwukrotnie tworzy zmienną `message`: w pierwszym wierszu skryptu i w pierwszym wierszu funkcji. Kod działa podobnie jak w przykładzie, w którym użyto parametrów. Wpisanie w funkcji instrukcji `var message` powoduje utworzenie w zasięgu funkcji nowej zmiennej. Jest to tak zwana *zmienna lokalna*, dostępna tylko w ramach funkcji. Główny skrypt i pozostałe funkcje nie widzą tej zmiennej i nie mają do niej dostępu.

Jednak zmienne utworzone w głównej części skryptu (poza funkcją) istnieją w *zasięgu globalnym*. Wszystkie funkcje w skrypcie mają dostęp do zmiennych utworzonych w głównym bloku programu. Poniższy kod tworzy zmienną `message` w pierwszym wierszu skryptu. Jest to *zmienna globalna*, dlatego funkcja ma do niej dostęp:

```
var message = 'Zmienna globalna';
function warning() {
  alert( message );
}
warning(); // 'Zmienna globalna'
```

Ta funkcja nie ma parametrów i nie zawiera definicji zmiennej `message`, dlatego po uruchomieniu instrukcji `alert(message)` funkcja szuka zmiennej globalnej o takiej nazwie. W skrypcie znajduje się taka zmienna, dlatego pojawi się okno dialogowe z tekstem „Zmienna globalna".

Zmienne lokalne i globalne mają pewną specyficzną cechę. Zmienna istnieje w zasięgu funkcji tylko wtedy, jeśli jest parametrem lub została zadeklarowana wewnątrz niej przy użyciu słowa kluczowego `var`. Ilustruje to rysunek 3.10. Górny fragment kodu pokazuje, jak dwie zmienne `message` (zmienna globalna i zmienna lokalna funkcji) mogą współistnieć obok siebie. Kluczowy jest pierwszy wiersz funkcji — `var message = 'W funkcji';`. W celu utworzenia zmiennej lokalnej konieczne jest zastosowanie słowa kluczowego `var`.



Rysunek 3.10. *Przy przypisywaniu wartości do zmiennych w funkcji należy pamiętać o pewnym drobnym, ale istotnym szczególe. Jeśli chcesz, aby zmienna była dostępna tylko w funkcji, pamiętaj o dodaniu słowa kluczowego var przy tworzeniu tej zmiennej (górny kod). Jeśli go nie użyjesz, zapiszesz nową wartość w zmiennej globalnej (kod dolny)*

Porównaj to z kodem z dolnej części rysunku 3.10. Tym razem w funkcji nie użyto słowa kluczowego `var`. Dlatego kod `message='W funkcji';` nie tworzy zmiennej lokalnej, lecz zapisuje nową wartość w globalnej zmiennej `message`. Efekt? Funkcja modyfikuje zmienną globalną, zastępując jej pierwotną wartość.

Zagadnienie zasięgu zmiennych jest skomplikowane, dlatego wcześniejszy opis może być nie w pełni zrozumiały. Na razie zapamiętaj, że jeśli zmienne w skrypcie mają nieoczekiwaną wartość, może to wynikać z problemów z ich zasięgiem. Jeśli natrafisz na taki problem, jeszcze raz przeczytaj uważnie ten punkt.

Przykład — prosty quiz

Pora wykorzystać wiedzę przedstawioną w tym rozdziale i utworzyć kompletny program. W tym przykładzie utworzysz prosty system do obsługi quizu. Program będzie zadawał pytania i oceniał odpowiedzi graczy. Na początku podrozdziału znajdziesz kilka możliwych rozwiązań tego problemu i omówienie skutecznych technik programistycznych.

Pierwszy krok polega jak zawsze na precyzyjnym określeniu zadań programu. Tworzony skrypt ma wykonywać kilka operacji. Oto one.

- **Zadawanie pytań**. Program do obsługi quizu musi mieć funkcję zadawania pytań. Do tej pory poznałeś jeden prosty sposób pobierania informacji na stronach WWW — użycie polecenia `prompt()`. Ponadto trzeba przygotować listę pytań. Ponieważ tablice dobrze się nadają do przechowywania list informacji, pytania można zapisać właśnie w nich.
- **Informowanie użytkowników o poprawności odpowiedzi**. Po pobraniu odpowiedzi od użytkownika trzeba ustalić, czy jest ona prawidłowa. Posłuży do tego instrukcja warunkowa. Następnie należy poinformować gracza o poprawności odpowiedzi. Można użyć do tego polecenia `alert()`.
- **Wyświetlanie wyników quizu**. Potrzebny jest sposób na śledzenie wyników gracza. Posłuży do tego zmienna przechowująca liczbę poprawnych odpowiedzi. Do wyświetlenia ostatecznego wyniku quizu można użyć poleceń `alert()` lub `document.write()`.

Opisany program można napisać na wiele sposobów. Niektórzy początkujący programiści mogą zastosować najprostsze podejście i powtarzać ten sam kod przy każdym pytaniu. Na przykład kod JavaScript do zadania dwóch pierwszych pytań quizu może wyglądać następująco:

```
var answer1=prompt('Ile książyców ma Ziemia?','');
if (answer1 == 1) {
  alert('Prawidłowa odpowiedź!');
} else {
  alert('Błąd. Prawidłowa odpowiedź to 1.');
}
var answer2=prompt('Ile książyców ma Saturn?','');
if (answer2 == 31) {
  alert('Prawidłowa odpowiedź!');
} else {
  alert('Błąd. Prawidłowa odpowiedź to 31.');
}
```

To rozwiązanie wydaje się logiczne, ponieważ program ma zadawać pytania jedno po drugim. Jednak nie jest to wydajny sposób programowania. Kiedy w programie trzeba wielokrotnie powtórzyć te same operacje, warto zastanowić się nad użyciem pętli lub funkcji. W następnym skrypcie użyjesz obu tych struktur. Pętla posłuży do przejścia w pętli przez wszystkie pytania quizu, a funkcja będzie je zadawać.

1. **Otwórz w edytorze tekstu plik *quiz.html*.**

   Na początku należy dodać kilka zmiennych, które są przeznaczone do śledzenia liczby pytań i poprawnych odpowiedzi.

2. **Znajdź kod między znacznikami `<script>` w sekcji nagłówkowej strony i wpisz poniższy kod:**

```
var score = 0;
```

Ta zmienna będzie przechowywać liczbę prawidłowych odpowiedzi. Na początku quizu, przed zadaniem pierwszego pytania, należy przypisać zmiennej wartość `0`. Następnie trzeba przygotować listę pytań i odpowiedzi.

3. **Wciśnij klawisz *Enter*, aby dodać nowy wiersz, i wpisz kod `var questions = [`.**

Wszystkie pytania znajdą się w tablicy, czyli zmiennej, która może przechowywać wiele elementów. Dodany kod to pierwsza część instrukcji tworzącej tablicę. Elementy tablicy można podać w wielu wierszach, co opisano na stronie 78.

4. **Wciśnij dwukrotnie klawisz *Enter*, aby dodać dwa nowe wiersze, i wpisz sekwencję `];`. Kod powinien wyglądać następująco:**

```
var score = 0;
var questions = [

];
```

Ponieważ quiz składa się z wielu pytań, warto zapisać każde z nich w tablicy. Przy zadawaniu pytań wystarczy wtedy przejść po elementach tablicy. Jednak każde pytanie ma inną odpowiedź, dlatego trzeba zapisać także odpowiedzi.

Jedną z możliwości jest utworzenie nowej tablicy, na przykład `answers[]`, i zapisanie w niej wszystkich odpowiedzi. Aby zadać pierwsze pytanie, należy wtedy pobrać pierwszy element tablicy `questions`, a w celu sprawdzenia, czy odpowiedź jest poprawna, trzeba użyć pierwszej wartości z tablicy `answers`. Wadą tego rozwiązania jest potencjalny brak synchronizacji między listami, który może wynikać ze wstawienia nowego pytania w środku tablicy `questions`, a odpowiedzi — na początku tablicy `answers`. Wtedy pierwszy element tablicy z pytaniami nie będzie dopasowany do pierwszej wartości tablicy odpowiedzi.

Lepsze podejście polega na użyciu *tablicy zagnieżdżonej*, nazywanej też *tablicą wielowymiarową*. Wymaga to utworzenia tablicy z pytaniem **i** odpowiedzią oraz zapisania jej jako jednego elementu tablicy `questions`. Powstanie w ten sposób lista, której każdy element to tablica.

5. **Kliknij pusty wiersz między znakami `[` i `];`, a następnie dodaj kod wyróżniony pogrubieniem:**

```
var questions = [
  ['Ile księżyców ma Ziemia?', 1],
];
```

Kod `['Ile księżyców ma Ziemia?', 1]` to dwuelementowa tablica. Pierwszy element to pytanie, a drugi — odpowiedź. Ta tablica to pierwszy element tablicy `questions`. Tablice zagnieżdżone nie mają nazw. Przecinek na końcu wiersza oznacza koniec pierwszego elementu tablicy `questions` i informuje o tym, że pojawią się dalsze dane.

6. **Wciśnij klawisz *Enter*, aby utworzyć nowy, pusty wiersz. Następnie dodaj do skryptu dwa wiersze wyróżnione pogrubieniem:**

```
var questions = [
  ['Ile księżyców ma Ziemia?', 1],
```

```
    ['Ile księżyców ma Saturn?', 31],
    ['Ile księżyców ma Wenus?', 0]
];
```

Dodałeś do quizu dwa nowe pytania. Zauważ, że po ostatnim elemencie tablicy *nie ma* przecinka. Umieszczenie wszystkich pytań w jednej tablicy zapewnia dużą elastyczność. Jeśli zechcesz dodać nowe pytanie, możesz dołączyć następną zagnieżdżoną tablicę z pytaniem i odpowiedzią.

Po przygotowaniu podstawowych zmiennych quizu pora zastanowić się nad zadawaniem pytań. Są one zapisane w tablicy, a program ma wyświetlić każde z nich. Na stronie 111 napisałem, że doskonałym narzędziem do poruszania się po tablicach są pętle.

7. **Kliknij kolumnę za sekwencją `];` (za tablicą `questions`) i wciśnij klawisz *Enter*, aby utworzyć nowy, pusty wiersz. Dodaj w nim następujący kod:**

```
for (var i=0; i<questions.lenght; i++) {
```

Ten wiersz to pierwsza część pętli `for` (patrz strona 112). Wykonuje on trzy operacje. Po pierwsze, tworzy nową zmienną, `i`, której przypisuje wartość `0`. Ta zmienna to licznik potrzebny do śledzenia liczby powtórzeń pętli. Drugi element, `i<questions.length`, to warunek podobny do warunku w instrukcjach `if-else`. Sprawdza on, czy wartość zmiennej `i` jest mniejsza od liczby elementów tablicy `questions`. Jeśli jest to prawda, należy powtórnie uruchomić pętlę. Kiedy wartość zmiennej `i` będzie równa liczbie elementów tablicy (lub większa od niej), skrypt zakończy działanie pętli. Trzecia część, `i++`, zwiększa wartość zmiennej `i` o 1 przy każdym powtórzeniu pętli.

Teraz pora dodać najważniejszą część pętli — kod JavaScript uruchamiany przy każdym jej powtórzeniu.

8. **Wciśnij klawisz *Enter*, aby utworzyć nowy, pusty wiersz. Wpisz w nim poniższy kod:**

```
askQuestion(questions[i]);
```

Zamiast umieszczać w pętli cały kod do obsługi zadawania pytań, można dodać funkcję, która wykonuje tę operację. Funkcja (utworzysz ją za chwilę) będzie nosić nazwę `askQuestion()`. Przy każdym powtórzeniu pętli kod przekaże do tej funkcji jeden element z tablicy `questions` (`questions[i]`). Pamiętaj, że dostęp do wartości tablicy zapewnia indeks, dlatego wyrażenie `questions[0]` oznacza pierwszy element, `questions[1]` to element drugi i tak dalej.

Użycie funkcji do zadawania pytań zwiększa elastyczność programu. W przyszłości będziesz mógł przenieść tę funkcję do innego skryptu i ponownie ją wykorzystać. Teraz należy dokończyć kod pętli.

9. **Wciśnij klawisz *Enter*, aby dodać nowy, pusty wiersz, i wpisz znak `}`, aby zakończyć pętlę. Oto gotowa pętla:**

```
for (var i=0; i<questions.length; i++) {
  askQuestion(questions[i]);
}
```

To już cała prosta pętla, która wywołuje funkcję i przekazuje do niej każde pytanie quizu. Teraz trzeba utworzyć serce quizu — funkcję `askQuestion()`.

10. **Utwórz pusty wiersz przed dodaną wcześniej pętlą `for`.**

    Funkcję należy umieścić między dwiema początkowymi instrukcjami definiującymi podstawowe zmienne a dodaną przed chwilą pętlą. Choć funkcje można definiować w dowolnym miejscu skryptu, większość programistów umieszcza je na początku kodu. W wielu skryptach najpierw znajdują się definicje zmiennych globalnych (takich jak `score` i `questions` w przykładzie), co ułatwia ich przeglądanie i modyfikowanie. Po zmiennych następują funkcje, które są podstawą większości programów, a na końcu znajdują się wymienione krok po kroku operacje, takie jak pętla w przykładowym kodzie.

11. **Dodaj poniższy fragment:**

```
function askQuestion(question) {

}
```

    Ten kod wyznacza ciało funkcji. Zawsze warto najpierw wpisać początkowy i końcowy nawias klamrowy funkcji, a dopiero potem dodać skrypt między nimi. Zapobiega to pominięciu zamykającego nawiasu klamrowego.

    Nowa funkcja przyjmuje jeden argument i zapisuje go w parametrze `question`. Nie jest to tablica `questions[]` utworzona w kroku 6. W zmiennej `question` skrypt będzie zapisywał elementy z tablicy `questions[]`. Jak wiesz z kroku 8., każdy element tej tablicy to lista zawierająca dwie wartości, czyli pytanie i odpowiedź.

12. **Dodaj wiersz wyróżniony pogrubieniem:**

```
function askQuestion(question) {
  var answer = prompt(question[0],'');
}
```

    Ten kod powinien wyglądać znajomo. Występuje tu znane już polecenie `prompt()`. Jedyna nowa część to wyrażenie `question[0]`. W ten sposób można uzyskać dostęp do pierwszego elementu tablicy. Przykładowa funkcja przyjmuje tablicę, która zawiera pytanie i odpowiedź. Przykładowo pierwsza taka tablica ma wartość `['Ile księżyców ma Ziemia?', 1]`. Dlatego wyrażenie `question[0]` zapewnia dostęp do pierwszego elementu, `'Ile księżyców ma Ziemia?'`; funkcja przekazuje go do polecenia `prompt()` jako pytanie, które pojawi się w oknie dialogowym.

    Program zapisuje w zmiennej `answer` wartość wpisaną przez gracza w oknie dialogowym. Następnie należy porównać dane wprowadzone przez użytkownika z prawidłową odpowiedzią.

13. **Uzupełnij funkcję przez dodanie kodu wyróżnionego pogrubieniem:**

```
function askQuestion(question) {
  var answer = prompt(question[0],'');
  if (answer == question[1]) {
    alert('Prawidłowa odpowiedź!');
    score++;
  } else {
    alert('Błąd. Prawidłowa odpowiedź to ' + question[1]);
  }
}
```

Ten kod to prosta instrukcja `if-else`. Warunek (`answer == question[1]`) pozwala sprawdzić, czy wartość podana przez użytkownika (`answer`) jest taka sama jak odpowiedź zapisana w drugim elemencie tablicy (`question[1]`). Jeśli obie liczby są takie same, gracz odpowiedział poprawnie. Pojawia się wtedy informacja o prawidłowej odpowiedzi, a skrypt zwiększa liczbę punktów o 1 (`score++`). Oczywiście przy błędnej odpowiedzi pojawia się okno dialogowe z właściwą wartością.

Na tym etapie quiz ma już wszystkie funkcje. Jeśli zapiszesz plik i wyświetlisz go w przeglądarce, będziesz mógł wziąć udział w quizie. Jednak brakuje jeszcze możliwości wyświetlenia wyników użytkownikowi. Posłuży do tego skrypt w ciele strony.

14. **Znajdź drugą parę znaczników `<script>` (w dolnej części strony) i wpisz w nich poniższy kod:**

```
var message = 'Liczba punktów: ' + score;
```

Ten kod tworzy nową zmienną i zapisuje w niej łańcuch znaków `'Liczba punktów: '` oraz wynik gracza. Dlatego jeśli użytkownik udzielił trzech dobrych odpowiedzi, zmienna `message` będzie miała wartość `'Liczba punktów: 3'`. Aby skrypt był bardziej czytelny, tworzenie dłuższego komunikatu warto podzielić na kilka wierszy.

15. **Wciśnij klawisz *Enter* i wpisz następujący kod:**

```
message += ' z ' + questions.length;
```

Ten wiersz dodaje łańcuch `' z '` i liczbę wszystkich pytań do zmiennej `message`. Na tym etapie zmienna ta ma wartość typu `'Liczba punktów: 3 z 3'`. Teraz można dokończyć komunikat i wyświetlić go na ekranie.

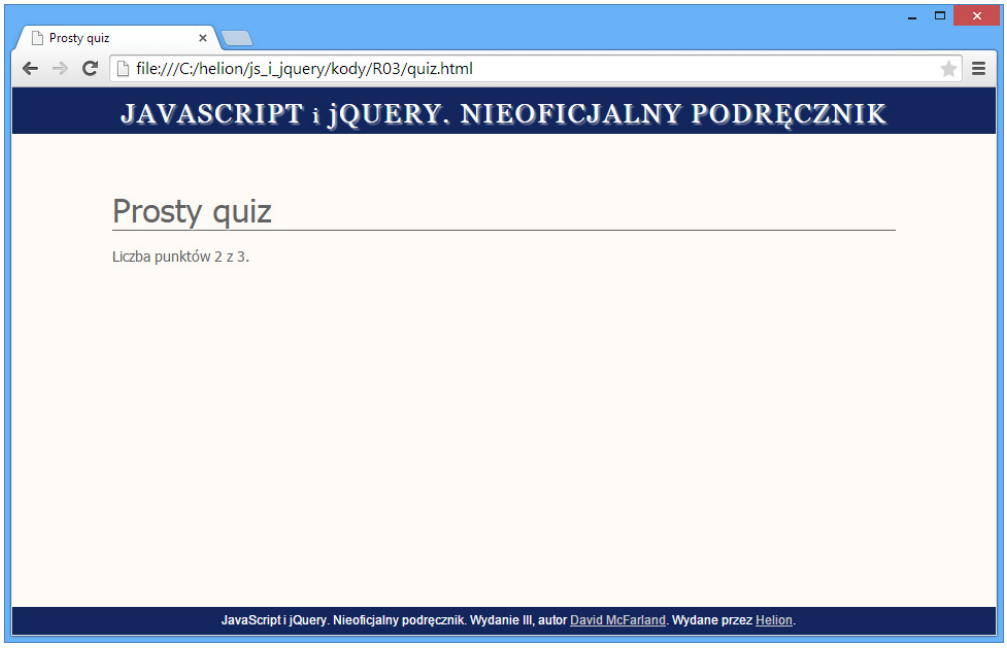
16. **Dodaj do skryptu wiersze wyróżnione pogrubieniem:**

```
var message = 'Liczba punktów: ' + score;
message += ' z ' + questions.length;
message += '.';
document.write('<p>' + message + '</p>');
```

Zapisz stronę i otwórz ją w przeglądarce. Odpowiedz na pytania i sprawdź, jaki wynik uzyskasz (patrz rysunek 3.11). Jeśli skrypt nie działa, wypróbuj techniki rozwiązywania problemów wymienione na stronie 51. Możesz też porównać skrypt z gotową, działającą wersją programu z pliku *complete_quiz.html*.

Aby wydłużyć quiz, spróbuj dodać dalsze pytania do tablicy `questions[]` z początku skryptu.

Teraz, kiedy już opanowałeś niektóre niezbyt fascynujące, lecz za to stanowiące wyzwanie intelektualne tajniki JavaScriptu, nadszedł czas, by zająć się czymś, co będzie naprawdę zabawne. W następnym rozdziale poznasz bibliotekę jQuery, dowiesz się, czym ona jest i jak jej używać, a co najważniejsze, jak się przy tym doskonale bawić i realizować wiele zadań programistycznych.



Rysunek 3.11. *Efekt działania prostego programu do obsługi quizów. Kiedy nauczysz się manipulować stronami WWW (patrz strona 157), reagować na zdarzenia zachodzące na stronach (patrz strona 182) oraz obsługiwać formularze umieszczane na stronach WWW (patrz strona 279), zobaczysz, jak zmodyfikować ten program, by pytania pojawiały się bezpośrednio na stronie, a wynik dynamicznie się zmieniał po udzieleniu każdej odpowiedzi. Wkrótce dowiesz się, jak zastąpić nieporęczne polecenie prompt()*

CZĘŚĆ

II

Wprowadzenie do biblioteki jQuery

ROZDZIAŁ 4

Wprowadzenie do jQuery

W trzech początkowych rozdziałach tej książki przedstawionych zostało wiele podstawowych informacji dotyczących języka programowania JavaScript; były to stosowane w nim słowa kluczowe, pojęcia oraz jego składnia. Wiele tych pojęć jest bardzo prostych („zmienna to takie pudełko, w którym można umieścić jakąś wartość"), jednak niektóre z nich mogły sprawić, że drapałeś się w zadumie po głowie lub sięgałeś po pastylkę aspiryny (na przykład pętle `for` opisane na stronie 109). Prawda jest taka, że dla większości osób pisanie kodu w języku JavaScript jest trudne. Rzeczywiście nawet w książce liczącej tysiąc stron nie udałoby się opisać wszystkich możliwych informacji dotyczących tego języka oraz sposobów jego działania we wszystkich dostępnych przeglądarkach.

Programowanie jest zagadnieniem trudnym. I właśnie z tego powodu w tej książce opisano zarówno język JavaScript, jak i bibliotekę jQuery. Jak się przekonasz w pierwszej części tego rozdziału, jQuery jest biblioteką języka JavaScript, która niezwykle przyspiesza i ułatwia programowanie, gdyż samoczynnie może wykonywać wiele złożonych zadań. Biblioteka ta, której mottem jest „pisz mniej, rób więcej", sprawia, że programowanie staje się zabawne, szybkie i satysfakcjonujące. Dzięki niej, za pomocą jednego wiersza kodu można zrobić to samo, co w innych przypadkach wymagałoby kilkudziesięciu wierszy zwyczajnego kodu JavaScript. Kiedy już zakończysz lekturę tego oraz kolejnego rozdziału, będziesz mógł zrobić na swoich stronach więcej, niż zrobiłbyś po przeczytaniu tysiącstronicowej książki poświęconej wyłącznie językowi JavaScript.

Kilka słów o bibliotekach JavaScript

Wielu programistów korzystających z języka JavaScript podczas tworzenia stron WWW wielokrotnie musi wykonywać dokładnie te same zadania, takie jak pobranie elementu strony, dodanie nowej zawartości, ukrycie lub wyświetlenie fragmentu strony, modyfikowanie atrybutów znaczników, określanie wartości pól formularzy, zapewnienie odpowiedniej reakcji programu na różnego rodzaju czynności wykonywane przez użytkownika. Szczegóły realizacji tych prostych zadań mogą być złożone zwłaszcza wtedy, kiedy chcemy, by nasz program działał we wszystkich

głównych przeglądarkach. Na szczęście *biblioteki* JavaScript stanowią rozwiązanie pozwalające ominąć problem mozolnego tworzenia kodu realizującego powtarzające się czynności programistyczne.

Biblioteki JavaScript to fragmenty kodu napisanego w tym języku, które zawierają rozwiązania wielu prozaicznych zadań wykonywanych każdego dnia przez programistów. Można je sobie wyobrazić jako kolekcje gotowych funkcji JavaScriptu, które wystarczy dodać do strony. Funkcje te ułatwiają wykonywanie najczęściej spotykanych zadań. Zdarza się dość często, że bardzo wiele wierszy naszego własnego kodu (oraz sporo godzin koniecznych do jego napisania) można zastąpić wywołaniem jednej funkcji takiej biblioteki. Istnieje bardzo dużo takich bibliotek, a wielu z nich używano podczas tworzenia największych i najbardziej znanych stron WWW, takich jak Yahoo!, Amazon, CNN, Apple oraz Twitter.

W tej książce użyto popularnej biblioteki jQuery (*http://www.jquery.com*). Dostępne są także inne biblioteki JavaScript (patrz ramka na stronie 135), jednak jQuery ma nad nimi przewagę i to z kilku powodów. Oto one.

- **Jest stosunkowo niewielka**. Skompresowana wersja biblioteki zajmuje jedynie około 96 kB w wersji 1.11 oraz 83 kB w wersji 2.1. (Jeśli serwer dodatkowo korzysta z kompresji „gzip", rzeczywistą wielkość przesyłanego pliku biblioteki można ograniczyć do około 38 kB!).
- **Jest przyjazna dla projektantów stron**. Biblioteka jQuery nie zakłada, że jesteś profesjonalnym informatykiem. Bazuje na znajomości CSS, a większość projektantów stron i tak już umie się nimi posługiwać.
- **Jest wypróbowana i sprawdzona**. Biblioteka jQuery jest używana na milionach stron, w tym także na wielu bardzo popularnych witrynach o bardzo dużym obciążeniu, takich jak Pinterest, MSN.com, Amazon, Microsoft.com, Craiglist czy też ESPN. Okazuje się, że jest ona używana przez ponad 57% wszystkich witryn na świecie (*http://w3techs.com/technologies/history_overview/javascript_library/all*). Popularność jQuery jest najlepszym świadectwem jej jakości.
- **Jest bezpłatna**. A tego nie można przebić!
- **Ma ogromną społeczność użytkowników**. Kiedy czytasz te słowa, cała rzesza ludzi pracuje nad projektem jQuery — pisze kod jej nowej wersji, poprawia błędy, dodaje nowe możliwości, aktualizuje witrynę z dokumentacją i poradnikami. Biblioteka JavaScript utworzona przez jednego programistę (lub udostępniana przez jednego autora) może zniknąć, kiedy ten zmęczy się prowadzeniem projektu. Jednak jQuery będzie dostępna przez bardzo długi czas dzięki temu, że wspiera ją bardzo wielu programistów z całego świata. Nawet bardzo duże firmy, takie jak Microsoft oraz Adobe, przydzielają swoich inżynierów do pracy nad jQuery i biorą udział w tworzeniu jej kodu. To tak, jakby spora grupa programistów pracowała dla nas, na dodatek za darmo.
- **Wtyczki, wtyczki i jeszcze raz wtyczki**. Biblioteka jQuery pozwala programistom tworzyć wtyczki — dodatkowe programy napisane w JavaScripcie i współpracujące z biblioteką w celu wykonywania określonych zadań, tworzenia efektów wizualnych i zapewniania nowych możliwości, z których w niezwykle prosty

WIEDZA W PIGUŁCE

Inne biblioteki JavaScript

jQuery nie jest jedyną dostępną biblioteką napisaną w języku JavaScript. Istnieje wiele, wiele innych. Niektóre z nich zostały zaprojektowane w celu wykonywania ściśle określonych zadań, inne natomiast mają ogólne przeznaczenie i służą do rozwiązywania wszelkich możliwych problemów, jakie mogą napotkać programiści JavaScript. Oto kilka najbardziej popularnych bibliotek.

- **Yahoo User Interface Library** (*http://yuilibrary.com/*) jest projektem prowadzonym przez Yahoo i w praktyce używanym na jej witrynie. Programiści firmy bezustannie poprawiają i rozwijają tę bibliotekę, udostępnili także jej bardzo dobrą dokumentację.
- **Dojo Toolkit** (*http://dojotoolkit.org/*) to kolejna biblioteka istniejąca już od bardzo dawna. Daje ogromne możliwości i stanowi kolekcję bardzo wielu plików JavaScript mogących posłużyć do rozwiązania niemal każdego problemu. Jest ona używana przez wielkie firmy, takie jak ADP, IBM oraz VMWare. Jest nieco złożona i przeznaczona przede wszystkim dla twórców aplikacji internetowych z dużym doświadczeniem.
- **Mootools** (*http://mootools.net/*) jest kolejną biblioteką służącą głównie do tworzenia płynnych animacji i innych efektów wizualnych.

Także inne biblioteki starają się udostępniać szkielety do tworzenia aplikacji internetowych. Najpopularniejsze są Ember.js (*http://emberjs.com/*), Angular.js (*http://angularjs.org/*) oraz Backbone.js (*http://backbonejs.org/*).

Niektóre biblioteki są małe i udostępniają proste, lecz przydatne narzędzia do programowania w języku JavaScript. Przykładowo Underscore.js (*http://underscorejs.org/*) jest niewielką biblioteką JavaScript udostępniającą bardzo dużo przydatnych funkcji, w której jednak nie znajdziesz narzędzi do obsługi efektów wizualnych, technologii AJAX ani modyfikacji kodu HTML strony, którymi dysponuje jQuery.

Istnieją także biblioteki o ściśle określonym przeznaczeniu i możliwościach, takie jak Raphaël (*http://raphaeljs.com/*), której jedynym celem jest ułatwienie tworzenia grafiki wektorowej w przeglądarkach.

Innymi słowy, nic nie stoi na przeszkodzie, by zaszaleć i korzystać z nawet 10 różnych bibliotek JavaScript. Jednak to jQuery jest najlepszym miejscem, od którego warto rozpocząć poznawanie bibliotek JavaScript. Później, kiedy zdobędziesz większą wiedzę i umiejętności, być może okaże się, że będziesz musiał skorzystać z możliwości, jakie dają inne biblioteki.

sposób można korzystać na swoich stronach. Czytając książkę, dowiesz się o wtyczkach umożliwiających weryfikację poprawności danych wpisywanych w formularzach, ułatwiających tworzenie rozwijanych menu oraz generowanie interaktywnych pokazów slajdów, których wykorzystanie zajmuje pół godziny, a nie stanowi odrębnego projektu o dwutygodniowym terminie realizacji. A takich wtyczek współpracujących z biblioteką jQuery są dosłownie tysiące.

W tej książce już wcześniej miałeś okazję skorzystać z jQuery. W przykładzie w rozdziale 1. (patrz strona 49) dodano do strony kilka wierszy kodu JavaScript, by utworzyć na niej efekt pojawiania się stopniowo.

Jak zdobyć jQuery?

Biblioteka jQuery to jedynie trochę kodu JavaScript umieszczonego w zewnętrznym pliku. Podobnie jak w przypadku wszystkich innych zewnętrznych plików JavaScript (patrz strona 49), także i plik jQuery należy dołączyć do swojej strony. Jednak, ze względu na ogromną popularność jQuery, można to zrobić na kilka sposobów: można pobrać wersję biblioteki udostępnianą na serwerach firm Google, Microsoft lub na serwerze jQuery.com (patrz rysunek 4.1) bądź też można skopiować plik biblioteki na własny komputer i umieścić go na witrynie.

Rysunek 4.1. *Strona główna witryny jQuery jest miejscem, z którego należy rozpocząć pobieranie biblioteki oraz zdobywanie informacji na temat jej API (zawiera ona alfabetyczną listę wszystkich funkcji biblioteki). Wiersz ikon widocznych w lewym, górnym rogu pozwala przejść na strony innych projektów jQuery: jQuery UI (który zostanie opisany w III części książki), jQuery Mobile (do tworzenia witryn przeznaczonych do oglądania na urządzeniach mobilnych) Sizzle (biblioteki wbudowanej w jQuery ułatwiającej wybieranie i manipulowanie fragmentami stron WWW) oraz QUnit (biblioteki do testowania programów JavaScript)*

Pierwsza z metod polega na skorzystaniu z *sieci dystrybucji treści* (ang. *content distribution network*, w skrócie *CDN*) — czyli innej witryny, która przechowuje bibliotekę i przesyła ją do każdego, kto o to poprosi. Takie rozwiązanie ma kilka zalet. Przede wszystkim można obniżyć obciążenie własnego serwera, gdyż to serwery Google, Microsoft lub jQuery będą udostępniać bibliotekę osobom przeglądającym naszą witrynę. Poza tym sieci tego typu zapewniają jeszcze jedną korzyść — ich serwery są rozmieszczone na całym świecie. A zatem, jeśli na naszą stronę wejdzie użytkownik, na przykład z Singapuru, pobierze plik biblioteki z serwera położonego zapewne znacznie bliżej swojego miejsca pobytu niż nasz; dzięki temu pobierze ten plik w krótszym czasie i odniesie wrażenie, że nasza witryna działa szybciej.

Jednak najważniejsze jest to, że z sieci dystrybucji korzysta także wielu innych programistów, zatem istnieje całkiem duże prawdopodobieństwo, że gdy użytkownik wejdzie na naszą stronę, w pamięci podręcznej jego przeglądarki będzie już zapisany odpowiedni plik jQuery. Ponieważ przeglądarka takiego użytkownika pobrała już bibliotekę z serwerów Google podczas przeglądania innych witryn, nie będzie musiała robić tego ponownie podczas wyświetlania naszej, co znacząco poprawi szybkość jej działania.

Jednak korzystanie z CDN ma także kilka wad. Przede wszystkim, aby metoda ta zadziałała, użytkownik musi być podłączony do internetu. Ma to znaczenie, gdy musimy zapewnić poprawność działania witryny bez podłączenia z internetem, na przykład w kioskach multimedialnych w muzeach lub podczas lekcji programowania w szkole, w klasie, w której nie ma dostępu do internetu. W takich sytuacjach konieczne będzie pobranie biblioteki z witryny jQuery.com (poniżej dowiesz się, jak to należy zrobić) i umieszczenie jej na własnej witrynie. Takie rozwiązanie ma także tę zaletę, że nasza witryna będzie działać nawet wtedy, gdyby zostały wyłączone serwery CDN. (Oczywiście, gdyby wyłączono serwery firmy Google, na świecie mogłyby się pojawić znacznie poważniejsze problemy niż ten, że nasza witryna przestała działać).

Dołączanie pliku jQuery z serwera CDN

Zarówno Microsoft, jQuery, jak i Google pozwalają korzystać na własnych witrynach z biblioteki jQuery pobieranej z ich serwerów. Aby na przykład skorzystać z wersji 1.11.0 biblioteki pobieranej z serwera CDN firmy Microsoft, w sekcji nagłówka naszej strony (tuż przed zamykającym znacznikiem `</head>`) musielibyśmy umieścić znacznik o następującej postaci:

```
<script src="http://ajax.aspnetcdn.com/ajax/jQuery/jquery-1.11.0.min.js">
➥</script>
```

Gdybyśmy chcieli skorzystać z serwera CDN jQuery, musielibyśmy użyć następującego kodu:

```
<script src="http://code.jquery.com/jquery-1.11.0.min.js"></script>
```

Poniższy kod pozwala użyć serwera CDN firmy Google:

```
<script src="http://ajax.googleapis.com/ajax/libs/jquery/1.11.0/
➥jquery.min.
js"></script>
```

Na naszej stronie powinniśmy umieścić tylko jeden z powyższych znaczników, zależnie od tego, z której sieci dystrybucji chcemy skorzystać. Najpopularniejsza jest chyba sieć firmy Google, jeśli zatem nie wiesz, której z nich użyć, wybierz właśnie tę.

Jeśli wolisz korzystać z biblioteki jQuery 2 (więcej informacji na jej temat znajdziesz w ramce na następnej stronie), w powyższych wierszach kodu zmień fragment `1.11.0` na `2.1.0` (bądź też inny numer aktualnie dostępnej najnowszej wersji jQuery, którą możesz sprawdzić na stronie *http://jquery.com/download/*). Aby na przykład pobrać jQuery 2.1.0 z serwera CDN firmy Google, należy użyć poniższego kodu:

```
<script src="//ajax.googleapis.com/ajax/libs/jquery/2.1.0/jquery.min.js">
➥</script>
```

Uwaga: Być może zauważyłeś, że odnośnik do serwera CDN firmy Google wygląda dość niezwykle. Nie zaczyna się od określenia protokołu — *http://* — jak w przypadkach serwerów CDN jQuery i Microsoftu. Adresy URL tego typu są określane jako zależne od protokołu (ang. *protocol-relative*), co oznacza tylko tyle, że przeglądarka pobierze je, korzystając z aktualnie używanego protokołu bezpieczeństwa. Jeśli na przykład witryna jest pobierana przy użyciu bezpiecznego protokołu https, to także biblioteka jQuery zostanie pobrana z wykorzystaniem tego protokołu. Więcej informacji na temat

adresów URL zależnych od protokołu znajdziesz na stronie *http://www.paulirish.com/2010/the-protocol-relative-url/*. Adresy tego typu przysparzają tylko jednego problemu: działają wyłącznie w przypadku, gdy strona jest pobierana z serwera WWW. Jeśli stronę zawierającą takie adresy URL otworzymy w przeglądarce jako plik lokalny, nie będą one działać.

Pobieranie pliku jQuery

Bez trudu można pobrać plik biblioteki jQuery i dodać go do wszystkich pozostałych stron oraz plików tworzących naszą witrynę. Plik ten wchodzi w skład przykładów dołączonych do tej książki, które pobrałeś z serwera FTP wydawnictwa Helion zgodnie z informacjami podanymi na stronie 46.; ponieważ jednak biblioteka ta jest regularnie aktualizowana, jej najnowszą wersję zawsze można znaleźć na stronie *http://jquery.com/download/*.

Aby pobrać najnowszą wersję biblioteki jQuery, wykonaj kolejno następujące kroki.

1. **Wejdź na stronę *http://jquery.com/download/*.**

   Strona ta zawiera informacje na temat kodu, listę CDN, o których wspomniano już wcześniej, oraz listę starszych wersji biblioteki.

2. **Wybierz wersje 1.x lub 2.x.**

   W tej książce używana jest wersja jQuery 1.11, warto jednak przeczytać informacje zamieszczone w ramce poniżej. Ogólnie rzecz biorąc, jeśli musisz obsługiwać wciąż popularną przeglądarkę Internet Explorer 8, powinieneś użyć wersji jQuery 1.11.

CZĘSTO ZADAWANE PYTANIA

Którą wersję wybrać: jQuery 1 czy 2?

Na witrynie jQuery dostępne są dwie wersje biblioteki — 1 i 2. Której z nich powinienem używać?

W czasie pisania tej książki wersje jQuery 1.11 oraz 2.1 odpowiadały sobie pod względem funkcjonalnym. Podstawowa różnica pomiędzy nimi — a jednocześnie podstawowy powód, dla którego zespół jQuery udostępnił wersję 2. — polega na porzuceniu wsparcia dla przeglądarek Internet Explorer 6, 7 oraz 8. Starsze wersje Internet Explorera często działają inaczej niż nowoczesne przeglądarki, a wykorzystanie w nich nowoczesnych możliwości wymaga dodatkowych starań. Obsługa tych przeglądarek wiąże się z koniecznością napisania dodatkowego kodu, co zwiększa rozmiar biblioteki jQuery.

Zespół jQuery, mając nadzieję, że przeglądarki IE 6, 7 i 8 kiedyś w końcu znikną z internetu, przygotował odchudzoną wersję biblioteki, z której usunięto kod obsługujący te przeglądarki. Jednak te wersje Internet Explorera wciąż są używane, dlatego też cały czas dostępne są wersje 1. biblioteki jQuery.

Jednak IE 8 wciąż jest najczęściej używaną wersją Internet Explorera. Z tego powodu w tej książce będziesz używał jQuery w wersji 1.11.0. Dysponuje ona tymi samymi możliwościami co jQuery 2, a jednocześnie wciąż działa w starszych wersjach Internet Explorera. Powinieneś używać najnowszej wersji biblioteki jQuery 1 (czyli obecnie 1.11) tak długo, jak Twoją witrynę odwiedzają osoby korzystające z przeglądarki Internet Explorer 8.

Jednak w przyszłości wszystkie nowe możliwości wprowadzane w bibliotece jQuery będą implementowane wyłącznie w jej wersji 2. W wersji 1. będą wprowadzane wyłącznie poprawki błędów. Nie obawiaj się jednak, wszystko, czego się nauczysz w tej książce, będzie działało we wszystkich wersjach jQuery. Jeśli jednak po zakończeniu lektury okaże się, że jakieś nowe, fantastyczne i absolutnie konieczne możliwości są dostępne tylko w jQuery 2, to być może warto, żebyś się zastanowił nad jej wykorzystaniem.

Na witrynie umożliwiającej pobieranie biblioteki jQuery jej pliki są udostępniane w dwóch wersjach — skompresowanej (ang. *compressed*) oraz nieskompresowanej (ang. *uncompressed*). Plik wersji nieskompresowanej jest bardzo duży (ma ponad 280 kB wielkości) i udostępniany wyłącznie po to, by można się było dowiedzieć czegoś więcej na temat biblioteki poprzez analizę jej kodu. W tej wersji pliku biblioteki znajduje się bardzo dużo komentarzy (patrz strona 90), które ułatwiają zrozumienie przeznaczenia poszczególnych fragmentów kodu. (Aby zrozumieć te komentarze, trzeba naprawdę *dobrze* znać język JavaScript).

Na swojej witrynie należy jednak używać wersji skompresowanej. Została ona *zminimalizowana*, co oznacza, że zajmuje znacznie mniej miejsca niż zwyczajne pliki z kodem JavaScript — nie zawiera żadnych komentarzy oraz niepotrzebnych znaków (takich jak znaki odstępu, nowego wiersza, tabulacji i tak dalej). Tę wersję biblioteki znacznie trudniej przeglądać i analizować, lecz przeglądarki mogą ją szybciej pobierać.

Uwaga: Zazwyczaj łatwo można stwierdzić, że dane pliki JavaScript zostały zminimalizowane, gdyż przeważnie do ich nazw dodawany jest ciąg znaków „min"; na przykład nazwa *jquery-1.11.0.min.js* oznacza, że plik zawiera zminimalizowaną wersję biblioteki jQuery.

3. **Kliknij odnośnik skompresowanej wersji biblioteki prawym przyciskiem myszy i z wyświetlonego menu kontekstowego wybierz opcję *Zapisz odnośnik jako*.**

   Jeśli ograniczymy się do zwyczajnego kliknięcia odnośnika, nie spowoduje to zamierzonego pobrania pliku biblioteki. Zamiast tego cały jej kod zostanie wyświetlony w oknie przeglądarki. Właśnie dlatego konieczne jest skorzystanie z metody *Zapisz jako*.

4. **Przejdź na swoim komputerze do katalogu, w którym są przechowywane zasoby witryny, i zapisz w nim plik.**

   Plik biblioteki jQuery można zapisać w dowolnym miejscu, jednak bardzo wielu projektantów umieszcza używane, zewnętrzne pliki JavaScript w osobnym katalogu. Zazwyczaj katalogom tym nadawane są takie nazwy jak *scripts*, *libs*, *js* bądź *_js*.

Dodawanie jQuery do strony

Jeśli używamy jednej z wersji jQuery udostępnianej przez sieci dystrybucji (patrz strona 137), możemy ją dodać do naszej strony, korzystając z jednego z fragmentów kodu zamieszczonych na stronie 137. Aby na przykład skorzystać z wersji dostępnej na serwerach Google, do sekcji nagłówka strony należy dodać poniższy znacznik `<script>`:

```
<script src="http://ajax.googleapis.com/ajax/libs/jquery/1.11.0/jquery.min.
js"></script>
```

CZĘSTO ZADAWANE PYTANIA

Wersje biblioteki jQuery

Jak widzę, w tej książce używana jest biblioteka jQuery w wersji 1.11.0, natomiast najnowszą wersją dostępną na witrynie jQuery jest 1.11 x. Czy to jakiś problem?

Biblioteka jQuery zmienia się cały czas. Często odnajdywane są nowe błędy, a zespół jej twórców skrupulatnie zabiera się do pracy nad ich poprawianiem. Co więcej, kiedy pojawiają się nowe wersje przeglądarek, dysponujące nowymi możliwościami i lepszą obsługą najnowszych standardów, zespół jQuery zabiera się do pracy, by zapewnić jak najbardziej efektywniejsze działanie biblioteki w takiej przeglądarce. W końcu, od czasu do czasu, także w bibliotece jQuery pojawiają się jakieś nowe możliwości, mające poprawić jej użyteczność. Właśnie z tych powodów jest całkiem prawdopodobne, że będziesz mógł znaleźć na witrynie jQuery wersję nowszą od używanej w tej książce. Jeśli faktycznie tak będzie, koniecznie powinieneś jej użyć.

Wraz z upływem lat biblioteka jQuery dojrzała i aktualnie jej podstawowe funkcjonalności zmieniają się bardzo nieznacznie. Choć programiści często modyfikują jej kod, by zwiększyć szybkość działania, zagwarantować, że będzie dobrze działać w wielu różnych przeglądarkach, a także po to, by poprawiać wykrywane błędy, jednak sam sposób korzystania z biblioteki zmienia się bardzo niewiele. Innymi słowy, choć programiści mogą modyfikować bibliotekę, by działała lepiej, sposób jej *używania* — nazwy funkcji, przekazywane do nich argumenty oraz zwracane przez nie wartości — zmienia się bardzo rzadko. Oznacza to, że wszystko, o czym przeczytasz w tej książce, będzie działać także w nowszej wersji biblioteki, tylko lepiej i szybciej.

Nie jest to jednak regułą. Na przykład sześć miesięcy po wydaniu poprzedniej wersji tej książki udostępniona została wersja jQuery 1.9. Usunięto z niej niektóre polecania używane w przykładach zamieszczonych w książce, więc czytelnicy, którzy korzystali z tej wersji biblioteki, odkryli, że niektóre z kodów nie działają.

Ilość różnic pomiędzy dwiema wersjami biblioteki można zazwyczaj określić na podstawie porównania numerów ich wersji. Pierwsza cyfra numeru reprezentuje tak zwaną bardzo ważną wersję, na przykład są to wersje 1. i 2. biblioteki jQuery. (Zgodnie z informacjami podanymi w ramce na stronie 109, wersja 2. ma dokładnie te same możliwości, co wersja 1.11, lecz nie obsługuje przeglądarki Internet Explorer 8 i wcześniejszych).

Po tej pierwszej cyfrze zapisywana jest kropka, a po niej kolejny numer, jak w jQuery 1.1, 1.2, 1.3 i tak dalej. Każda z wersji oznaczanych kolejnymi numerami udostępnia zazwyczaj jakieś nowe możliwości, modyfikacje zapewniające lepsze działanie starych funkcji i tak dalej. I w końcu ostatnia cyfra, np. 3 w jQuery 1.11.3, zazwyczaj odnosi się do kolejnej grupy poprawek wprowadzanych do biblioteki (w tym przypadku, do jej wersji 1.11). A zatem, jeśli używamy biblioteki w wersji 1.11.0, a pojawiła się wersja 1.11.3, warto z niej skorzystać, gdyż zazwyczaj będzie zawierała poprawki do błędów odnalezionych w kodzie biblioteki 1.11.0.

Aby zobaczyć, jakie zmiany wprowadzono w konkretnej wersji biblioteki, należy wejść na stronę umożliwiającą pobranie jQuery (*http://jquery.com/download/*) i poszukać na niej odnośnika *Release notes* (informacje o wersji) dla aktualnej wersji biblioteki. I tak na stronie *http://blog.jquery.com/2014/01/24/jquery-1-11-and-2-1-released/* można znaleźć informacje o zmianach wprowadzonych w wersjach 1.11.0 oraz 2.1 biblioteki jQuery. Po ich przeczytaniu można podjąć decyzję, czy warto aktualizować kod biblioteki, czy nie. Jeśli na przykład wszystkie zmiany są związane z możliwościami, z których nie korzystamy, zapewne będziemy mogli pominąć tę nową wersję; jeśli jednak zawiera ona poprawki możliwości, których używamy, zaktualizowanie biblioteki będzie zalecane. Jeśli na witrynie używamy wtyczek jQuery, aktualizację do najnowszej wersji biblioteki trzeba będzie przeprowadzać ostrożniej, chyba że będziemy absolutnie pewni, że stosowane wtyczki dobrze z nią działają.

Wskazówka: Podczas korzystania z sieci dystrybucji treści (CDN) firmy Google można pominąć fragmenty numeru wersji. Jeśli zamiast numeru 1.11.0 podamy w odnośniku numer 1.11 (`<script src="http://ajax.googleapis.com/ajax/libs/jquery/1.11/jquery.min.js"></script>`), Google udostępni najnowszą wersję biblioteki z rodziny 1.11 — na przykład wersję 1.11.2. Jeśli w przyszłości pojawi się aktualizacja o numerze 1.11.9, serwer Google będzie udostępniał właśnie ją. Technika ta jest całkiem sprytna, gdyż (zgodnie z informacjami podanymi w ramce powyżej) zmiany reprezentowane przez ostatni numer, na przykład z 1.11.0 na 1.11.2, zazwyczaj sprowadzają się do poprawek w kodzie, które mogą usprawnić działanie naszej witryny.

Jeśli pobrałeś plik biblioteki i umieściłeś na własnym komputerze, będziesz go musiał dodać do strony, na której chcesz używać biblioteki. Plik jQuery jest zwyczajnym zewnętrznym plikiem JavaScript, a zatem dodaje się go do strony dokładnie tak samo jak inne pliki tego typu, w sposób opisany na stronie 49. Załóżmy na przykład, że dysponujesz plikiem *jquery.js* umieszczonym w katalogu *js*, w głównym katalogu serwera WWW. Aby go dodać do strony głównej witryny, powinieneś umieścić w jej sekcji nagłówka znacznik w następującej postaci:

```
<script src="js/jquery-1.11.0.min.js"></script>
```

Po dołączeniu pliku jQuery do strony można już dodawać do niej własne skrypty, korzystające z zaawansowanych funkcji tej biblioteki. Następnym krokiem będzie dodanie kolejnej pary znaczników `<script>` zawierających nieco bardziej wymagający kod JavaScript:

```
<script src="js/jquery-1.11.0.min.js"></script>
<script>
$(document).ready(function() {
    // to jest miejsce na nasz kod
});
</script>
```

Wewnątrz drugiej pary znaczników `<script>` znajduje się cały kod, który chcemy umieścić na danej stronie WWW. Najprawdopodobniej jednak zastanawiasz się, co oznacza zapis `$(document).ready()`. Jest to wbudowana funkcja jQuery, która spowoduje wykonanie przekazanego do niej kodu po zakończeniu pobierania całej strony WWW.

A po co robić coś takiego? Wynika to z faktu, że przeważająca część programów pisanych w języku JavaScript jest związana z operowaniem na zawartości stron WWW; mogą to być takie czynności jak na przykład animowanie jakiegoś elementu `div`, stopniowe wyświetlenie niewidocznego początkowo obrazka, rozwinięcie menu po wskazaniu myszą konkretnego odnośnika i tak dalej. Aby z elementem strony zrobić coś interesującego i interaktywnego, program JavaScript musi go najpierw wybrać. Jednak nie jest to możliwe, dopóki przeglądarka go nie pobierze. Ponieważ przeglądarka wykonuje każdy kod JavaScript bezpośrednio po jego odnalezieniu, może się zdarzyć, że jakieś fragmenty kodu strony jeszcze nie będą pobrane. (Taki efekt występował w przykładzie z quizem zaprezentowanym w poprzednim rozdziale. W momencie wyświetlania quizu strona była pusta — jej zawartość pojawiała się dopiero po podaniu odpowiedzi. Działo się tak dlatego, że kod JavaScript obsługujący quiz był wykonywany, zanim przeglądarka zdążyła wyświetlić znaczniki HTML dalszej zawartości strony).

Innymi słowy, abyśmy mogli zrobić coś fajnego z zawartością strony, musimy poczekać, aż przeglądarka ją pobierze. I właśnie to zapewnia funkcja `$(document).ready()`: czeka na zakończenie pobierania całej zawartości dokumentu HTML i dopiero potem wykonuje wskazany kod JavaScript. Jeśli to wszystko wydaje się bardzo zagmatwane i trudne, wystarczy, byś zapamiętał, że zawsze wtedy, gdy umieszczasz skrypt w sekcji `<head>` strony (przed jej ciałem), powinieneś zrobić dwie rzeczy: zastosować funkcję `ready()` i umieścić swój kod dokładnie pomiędzy `$(document).ready(function() {` oraz zamykającą sekwencją znaków `});`.

Dodatkowo warto pamiętać także o kilku dodatkowych zagadnieniach.

- Odwołanie do pliku biblioteki jQuery musi być umieszczone przed jakimkolwiek kodem JavaScript, w którym biblioteka ta jest używana. Innymi słowy, przed znacznikami `<script>` odwołującymi się do jQuery nie należy umieszczać innych znaczników `<script>`.
- Własny kod JavaScript należy umieszczać *za* wszystkimi arkuszami stylów (zarówno dołączanymi, zewnętrznymi, jak i definiowanymi wewnątrz strony). Ponieważ kod korzystający z możliwości jQuery bardzo często odwołuje się do definicji stylów z CSS, dlatego też kod JavaScript należy umieszczać za arkuszami stylów, tak by w momencie jego wykonywania przeglądarka zdążyła je pobrać. Dobrą zasadą, do której można się stosować, jest umieszczanie znaczników `<script>` wewnątrz sekcji nagłówka strony, poniżej jakiejkolwiek innej zawartości, lecz — oczywiście — przed zamykającym znacznikiem `</head>`.
- Do własnego kodu JavaScript warto dodawać komentarze — na przykład komentarz `// koniec funkcji ready`, umieszczony po zamykającej sekwencji znaków `});`, może oznaczać miejsce, w którym kończy się wywołanie funkcji `ready()`. Oto kompletny przykład:

```
$(document).ready(function() {
  // To jest miejsce na nasz kod.
}); // Koniec funkcji ready.
```

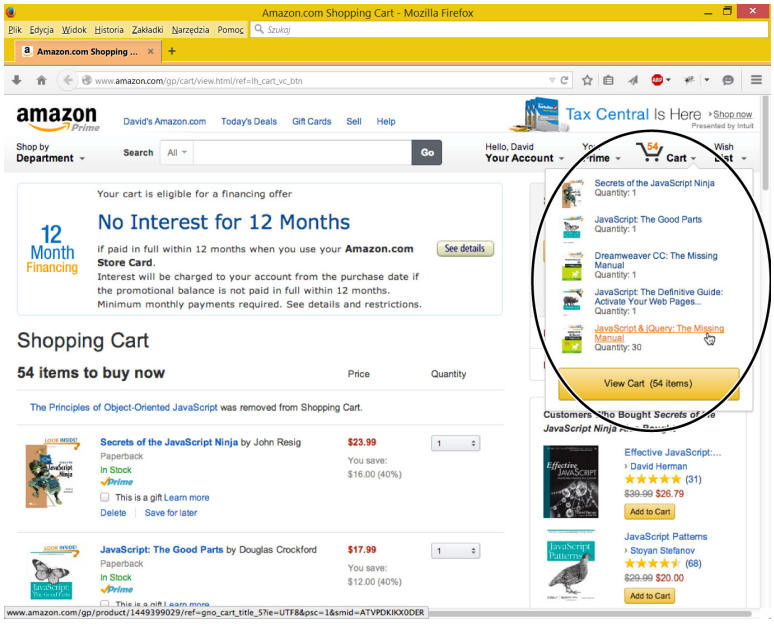
  Po umieszczeniu komentarza na końcu funkcji bardzo łatwo będzie można później określić, gdzie kończy się jej kod. Jak się przekonasz, korzystanie z biblioteki jQuery wymaga częstego stosowania tych krótkich sekwencji znaków składających się z zamykającego nawiasu klamrowego, okrągłego oraz średnika. Umieszczając za nimi komentarze, można sobie znacznie ułatwić określenie, do jakiego fragmentu kodu odnosi się dana sekwencja.

Wskazówka: jQuery udostępnia skrócony zapis wywołania funkcji `$(document).ready(function() {})`:

```
$(function() {
  // To jest miejsce na nasz kod.
}); // Koniec funkcji ready.
```

Podstawowe informacje o modyfikowaniu stron WWW

Język JavaScript umożliwia modyfikowanie stron WWW na oczach przeglądających je użytkowników. Korzystając z niego, można dodawać obrazki i teksty, usuwać fragmenty zawartości lub błyskawicznie zmieniać wygląd wybranych elementów strony. Dynamiczne modyfikowanie wyglądu stron jest cechą charakterystyczną najnowszej generacji witryn WWW, wykorzystujących możliwości języka JavaScript. Przykładowo witryna Google Maps (*http://maps.google.com/*) zapewnia dostęp do mapy całego świata, kiedy ją powiększamy lub przewijamy, oglądana mapa jest aktualizowana bez konieczności ponownego wczytywania strony. Kiedy na witrynie Netflix (*http://www.netflix.com/*) umieścimy wskaźnik myszy na tytule filmu, na stronie pojawi się wyskakujące okienko prezentujące dodatkowe informacje na jego temat (patrz rysunek 4.2). W obu tych przypadkach program napisany w języku JavaScript zmienia kod HTML pobrany początkowo przez przeglądarkę.

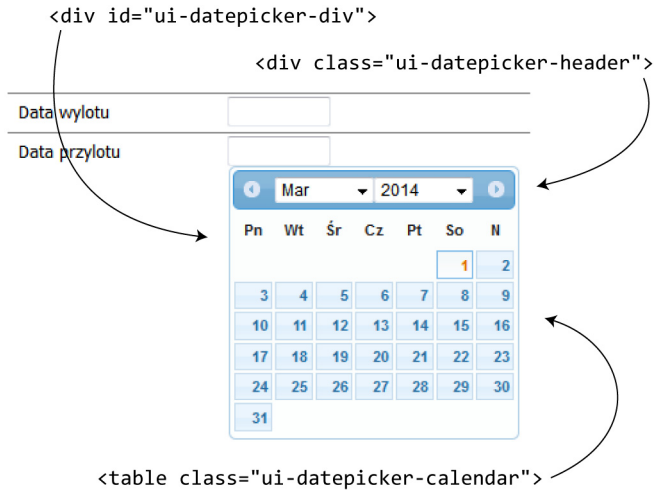


Rysunek 4.2. *JavaScript może ułatwić przeglądanie stron, pozwalając na wyświetlanie tylko tych ich fragmentów, które w danej chw li są potrzebne. Na przykład Amazon.com ukrywa pewne elementy aż do chwili, gdy wskażemy inne elementy strony myszą. Wskazanie myszą ikony koszyka wyświetla wszystkie wybrane produkty. Co? 30 egzemplarzy JavaScript & jQuery: The Missing Manual? To musi być świetna książka*

W tym rozdziale dowiesz się, jak można modyfikować zawartość stron przy użyciu języka JavaScript. Nauczysz się dodawać do strony nowe treści, znaczniki HTML oraz ich atrybuty, a także modyfikować oryginalną zawartość strony. Innymi słowy, nauczysz się używać języka JavaScrip do generacji nowego oraz modyfikowania już istniejącego kodu HTML strony.

Być może trudno w to uwierzyć, lecz okazuje się, że gdy wiemy, jak budować strony w językach HTML i CSS, dysponujemy już znaczną częścią wiedzy niezbędnej do efektywnego korzystania z języka JavaScript i tworzenia interaktywnych witryn WWW. Przykładowo popularna wtyczka Datepicker, należąca do pakietu jQuery UI, ułatwia użytkownikom wybieranie dat na formularzach (na witrynach do planowania lotów lub wydarzeń i tak dalej). Kiedy użytkownik kliknie specjalnie oznaczone pole tekstowe, na ekranie zostaje wyświetlony kalendarz (patrz rysunek 4.3). Choć efekt jest naprawdę świetny, a kalendarz bardzo ułatwia wybieranie dat, jednak język JavaScript odpowiada jedynie za interaktywną stronę całego rozwiązania — sam kalendarz powstaje z wykorzystaniem dobrze już znanych kodów HTML i CSS.

Jeśli dokładnie przeanalizujemy sposób utworzenia tego kalendarza, okaże się, że składowymi są znaczniki HTML, takie jak `<div>`, `<table>` oraz `<td>` posiadające specyficzne klasy CSS i identyfikatory (takie jak `ui-datepicker-month`, `ui-datepicker-↪div` i tak dalej). Arkusz stylów wykorzystujący te klasy i identyfikatory określa kolory, typografię oraz formatowanie kalendarza. Innymi słowy, korzystając z języków HTML i CSS, moglibyśmy sami utworzyć taki kalendarz. JavaScript sprawia jedynie, że kalendarz staje się interaktywny, bo zapewnia możliwość wyświetlenia go w odpowiedzi na kliknięcie pola formularza i ukrycia, gdy użytkownik wybierze już jakąś datę.



Rysunek 4.3. *Projekt jQuery UI (http://jqueryui.com) udostępnia bardzo użyteczne komponenty interfejsu użytkownika, które można stosować w aplikacjach internetowych. Przykładowo wtyczka Datepicker udostępnia łatwy i przyjazny dla użytkownika sposób wybierania dat. Poznasz ją dokładniej na stronie 375*

Jednym z możliwych sposobów nowoczesnego wykorzystania języka JavaScript, zwłaszcza w kontekście projektowania interfejsów użytkownika, jest automatyzacja tworzenia kodu HTML i stosowania stylów CSS. Na witrynie *Amazon.com*, przedstawionej na rysunku 4.2, kod JavaScript wyświetla animację odtwarzaną, kiedy użytkownik umieści wskaźnik myszy w obszarze przycisku, jednak najfajniejsza część całego tego rozwiązania (czyli utworzenie tego okienka) sprowadza się do napisania odpowiedniego kodu HTML i stylów CSS…, a to przecież już umiesz zrobić!

Znaczna część zadań, do jakich będziemy używali JavaScriptu, sprowadza się do manipulowania stronami WWW poprzez dodawanie do nich nowych treści, modyfikację ich oryginalnego kodu HTML bądź też przypisywanie stylów do wybranych elementów. Jakakolwiek zmiana zawartości strony — czy to kodu HTML, czy stylów CSS, taka jak dodanie paska nawigacyjnego z dynamicznym, rozwijanym menu, czy tworzenie interaktywnego pokazu slajdów, czy też bardzo proste stopniowe wyświetlanie elementu (co zrobiliśmy w ramach przykładu przedstawionego w rozdziale 1.) — będzie się składała z dwóch podstawowych etapów. Oto one.

1. **Wybranie odpowiedniego elementu strony**.

   Elementem tym może być dowolny istniejący znacznik; zanim będziemy mogli coś z nim zrobić, będziemy musieli *wybrać go* w kodzie JavaScript (jak to zrobić, dowiesz się w dalszej części rozdziału). Aby na przykład stopniowo wyświetlić zawartość strony, trzeba ją najpierw wybrać (znacznik `<body>`); aby z kolei wyświetlić menu kontekstowe po wskazaniu przycisku myszą, musimy wybrać ten przycisk. Nawet jeśli chcemy wykonać banalną operację dodania jakiegoś tekstu na końcu strony, to i tak musimy zacząć od wybrania znacznika, przed którym lub za którym tekst zostanie dodany.

2. **Wykonanie na tym elemencie pewnych operacji**.

   No dobrze, „pewne operacje" to niezbyt precyzyjne określenie tego, co należy zrobić. Jednak ilość potencjalnych zmian, które możemy *wprowadzić* w celu modyfikacji wyglądu lub sposobu działania strony, jest praktycznie nieskończona.

Znaczną część tej książki poświęcono przedstawieniu różnych rzeczy, jakie można robić z elementami stron WWW. Oto kilka przykładów.

- **Zmiana właściwości elementu**. Podczas animacji położenia elementu `<div>` na stronie modyfikowane są współrzędne określające jego położenie.
- **Dodawanie nowej zawartości**. Jeśli podczas wypełniania formularza użytkownik popełni jakiś błąd, popularnym rozwiązaniem jest wyświetlanie komunikatu o błędzie, takiego jak: „Proszę podać prawidłowy adres e-mail". W tym przypadku dodajemy nową zawartość strony, umiejscowioną względem konkretnego pola formularza.
- **Usunięcie elementu**. Na witrynie *Amazon.com*, przedstawionej na rysunku 4.2, okienko znika po usunięciu wskaźnika myszy z odnośnika do koszyka. W tym przypadku program JavaScript usuwa okienko ze strony.
- **Pobranie informacji o elemencie**. Może się zdarzyć, że będziemy chcieli zdobyć pewne informacje na temat wybranego znacznika. Aby na przykład sprawdzić poprawność informacji podanych w polu tekstowym, konieczne jest wybranie tego pola, a następnie pobranie tekstu, który został w nim podany. Inaczej mówiąc, konieczne jest pobranie wartości konkretnego pola.
- **Dodanie lub usunięcie atrybutu `class`**. Czasami będziemy chcieli zmienić wygląd jakiegoś elementu strony: wyświetlić na niebiesko tekst w wybranym akapicie lub oznaczyć błędnie wypełnione pole formularza, zmieniając kolor tła na czerwony. Choć takie wizualne modyfikacje można wprowadzać przy użyciu kodu JavaScript, jednak niejednokrotnie najprostszym rozwiązaniem będzie zastosowanie odpowiedniej klasy CSS, a przeglądarka wprowadzi wszelkie wizualne zmiany na podstawie kodu CSS zdefiniowanego w arkuszu stylów. W takim przypadku zmiana koloru tekstu w akapicie na niebieski sprowadza się do utworzenia odpowiedniej klasy z niebieskim kolorem tekstu i napisania kodu JavaScript, który dynamicznie użyje tej klasy w wybranym akapicie.

Niejednokrotnie będziemy wykonywali kilka powyższych czynności jednocześnie. Możemy na przykład upewnić się, że użytkownik nie zapomniał podać w polu formularza swojego adresu poczty elektronicznej. Jeśli użytkownik spróbuje przesłać formularz bez podania tej informacji, będziemy chcieli poinformować go o tym. Zadanie to może wymagać określenia, czy użytkownik wpisał cokolwiek w polu formularza (czyli pobrania informacji o konkretnym elemencie strony), wyświetlenia komunikatu o błędzie (czyli dodania do strony nowej zawartości) i wyróżnienia pola (poprzez przypisanie mu odpowiedniej klasy).

Pierwszym krokiem jest wybranie odpowiedniego elementu. Aby zrozumieć, jak to zrobić i jak zmodyfikować fragment strony przy użyciu kodu JavaScript, będziesz musiał poznać DOM — *Document Object Model* (model obiektów dokumentu).

Zrozumieć DOM

Kiedy przeglądarka wczyta dokument HTML, wyświetla jego zawartość na ekranie (oczywiście, po określeniu jego wyglądu na podstawie zastosowanych arkuszy stylów CSS). Jednak nie jest to jedyna czynność wykonywana przez przeglądarkę i związana ze znacznikami, ich atrybutami oraz zawartością dokumentu HTML. Oprócz tego

przeglądarka tworzy i zapamiętuje „model" danego dokumentu HTML. Innymi słowy, przeglądarka zapamiętuje znaczniki HTML, ich atrybuty oraz kolejność, w jakiej są zapisane w pliku — ta reprezentacja jest nazywana modelem obiektów dokumentu (ang. *Document Object Model*, w skrócie DOM).

DOM zawiera informacje niezbędne językowi JavaScript do operowania na elementach strony WWW. Oferuje także narzędzia pozwalające na poruszanie się po zawartości strony, modyfikowanie istniejących elementów HTML oraz dodawanie nowych. Sam DOM nie jest (w zasadzie) związany z językiem JavaScript — stanowi odrębny standard opracowany przez W3C (ang. *World Wide Web Consortium*), który większość producentów przeglądarek zaakceptowała i wykorzystuje w swoich programach. DOM pozwala skryptom pisanym w języku JavaScript komunikować się z kodem HTML strony oraz modyfikować go.

Aby dowiedzieć się, w jaki sposób działa DOM, przeanalizujmy przedstawiony poniżej przykład bardzo prostej strony WWW:

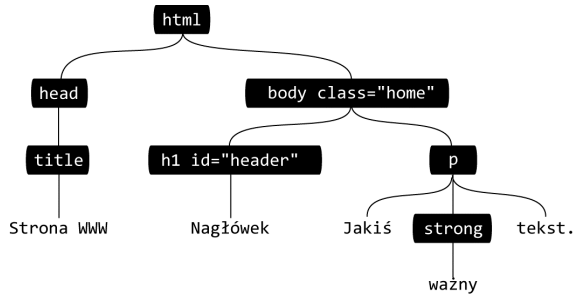
```
<!DOCTYPE HTML>
<html>
<head>
  <meta charset="UTF-8">
  <title>Strona WWW</title>
</head>
<body class="home">
  <h1 id="header">Nagłówek</h1>
  <p>Jakiś <strong>ważny</strong> tekst.</p>
</body>
</html>
```

Zarówno na tej, jak i na wszystkich innych stronach WWW jakieś znaczniki HTML otaczają inne znaczniki — na przykład wewnątrz znacznika `<html>` umieszczony jest znacznik `<body>`, wewnątrz którego z kolei są umieszczone inne znaczniki oraz treści wyświetlane wewnątrz okna przeglądarki. Wzajemne relacje pomiędzy poszczególnymi znacznikami można przedstawić za pomocą struktury przypominającej nieco drzewo genealogiczne (patrz rysunek 4.4). „Korzeniem" tego drzewa jest znacznik `<html>` — jest on takim prapradziadkiem wszystkich innych znaczników tworzących stronę; z kolei pozostałe znaczniki reprezentują „gałęzie" drzewa rodu; przykładem mogą tu być znaczniki `<head>` oraz `<body>`, z których każdy zawiera swój własny zbiór znaczników.

Przeglądarki, oprócz znaczników HTML, zapamiętują także teksty wyświetlane wewnątrz nich (takie jak `Nagłówek` umieszczony na rysunku 4.4 wewnątrz znacznika `<h1>`) oraz **atrybuty** przypisywane poszczególnym znacznikom (takie jak atrybut `class` w znaczniku `<body>` oraz `id` w znaczniku `<h1>` na przykładzie z rysunku 4.4). W rzeczywistości DOM traktuje te wszystkie znaczniki (nazywane także elementami), atrybuty i teksty jako odrębne jednostki, nazywane **węzłami** (ang. *node*).

JavaScript udostępnia kilka sposobów wybierania elementów strony tak, by później można było coś z nimi zrobić — na przykład stopniowo ukryć lub przesunąć w inne miejsce. Metoda `document.getElementById()` pozwala na pobranie elementu o konkretnym identyfikatorze (podanym w kodzie HTML znacznika). Jeśli zatem na stronie znajduje się znacznik `<div>` z atrybutem `id` o wartości `banner`, można go pobrać przy użyciu następującego wywołania:

```
document.getElementById('banner');
```



Rysunek 4.4. *Podstawowa, zagnieżdżona struktura strony WWW, w której jedne znaczniki są umieszczane wewnątrz innych, często jest przedstawiana w formie drzewa. Znaczniki otaczające inne znaczniki są czasami nazywane przodkami, natomiast umieszczane wewnątrz nich — potomkami*

Podobnie metoda `document.getElementsByTagName()` pobiera każdy egzemplarz podanego znacznika umieszczony na stronie — na przykład wywołanie `document.getElementsByTagName('a')` pobiera wszystkie dostępne na danej stronie znaczniki odnośników; dodatkowo niektóre przeglądarki udostępniają także metody pozwalające na pobieranie wszystkich elementów posiadających odpowiednią klasę bądź pobieranie elementów na podstawie reguł CSS.

Nowsze wersje przeglądarek zapewniają możliwość pobierania elementów DOM na podstawie selektorów CSS. Przykładowo metoda `document.getElementsByClassName()` pobiera wszystkie elementy o tej samej nazwie klasy. Poniższe wywołanie pozwala pobrać wszystkie elementy klasy `author`:

```
document.getElementsByClassName('author');
```

Nieco bardziej ogólną metodą jest `querySelectorAll()` — pozwala ona na pobieranie elementów strony przy użyciu dowolnego selektora CSS. Aby na przykład pobrać jedynie elementy `<span>` klasy `author`, należałoby użyć wywołania o postaci:

```
document.querySelectorAll('span.author');
```

Jak się przekonasz w następnym podrozdziale, także jQuery korzysta z selektorów CSS do pobierania elementów HTML, a co więcej, robi to w taki sposób, że można jej używać niemal we wszystkich przeglądarkach.

Pobieranie elementów stron na sposób jQuery

jQuery udostępnia niezwykle użyteczną technikę pozwalającą na pobieranie kolekcji elementów i operowanie na nich, są to selektory CSS. Jeśli zatem jesteś przyzwyczajony do określania postaci swoich stron przy użyciu kaskadowych arkuszy stylów, jesteś także gotów, by rozpocząć korzystanie z jQuery. Selektor CSS jest jedynie instrukcją informującą przeglądarkę, w których znacznikach należy zastosować dany styl. I tak `h1` jest bardzo prostym selektorem elementu, który odnosi się do wszystkich znaczników `<h1>`; z kolei `.copyright` jest selektorem klasy określającym postać dowolnych znaczników mających atrybut `class` o wartości `copyright`:

```
<p class="copyright">Wszelkie prawa zastrzeżone, 2011</p>
```

Podczas korzystania z jQuery można pobrać jeden lub większą liczbę elementów, posługując się specjalnym poleceniem nazywanym **obiektem jQuery**. Służy do tego kod o następującej postaci:

```
$('selektor')
```

W czasie tworzenia obiektu jQuery można używać niemal wszystkich selektorów CSS 2.1 oraz wielu selektorów CSS 3 (i to nawet wtedy, gdy nie są one rozumiane przez samą przeglądarkę — jak to się dzieje w przypadku pewnych selektorów CSS 3 w przeglądarkach Internet Explorer). Gdy na przykład chcemy pobrać znacznik o identyfikatorze (atrybucie `id`) `banner`, możemy użyć następującego fragmentu kodu:

```
$('#banner')
```

gdzie `#banner` jest selektorem CSS używanym do określania postaci znacznika, którego identyfikator ma wartość `banner` — znak # informuje, że interesuje nas właśnie atrybut `id`. Oczywiście, kiedy już pobierzemy jakieś elementy, będziemy chcieli coś z nimi zrobić — jQuery udostępnia wiele narzędzi pozwalających na przeprowadzanie różnego typu operacji na elementach. Załóżmy przykładowo, że chcielibyśmy zmienić kod HTML umieszczony wewnątrz elementu. Możemy to zrobić w następujący sposób:

```
$('#banner').html('<h1>Byłem tu, JavaScript</h1>');
```

Znacznie więcej informacji na temat operowania na elementach stron WWW znajdziesz w dalszej części książki, od strony 157. Jednak na początek musisz dowiedzieć się czegoś więcej na temat pobierania tych elementów przy użyciu biblioteki jQuery.

Proste selektory

Proste selektory CSS, takie jak te, które bazują na identyfikatorze elementu, nazwie klasy lub znacznika, stanowią podstawę kaskadowych arkuszy stylów. Są one doskonałym sposobem pobierania szerokiej gamy elementów przy użyciu jQuery.

Ponieważ czytanie informacji o selektorach nie jest najlepszym sposobem ich poznawania, zatem do książki dołączona została interaktywna strona WWW pozwalająca na ich testowanie. W przykładach do książki, w katalogu *testy* znajduje się plik o nazwie *selectors.html*. Otwórz go w przeglądarce. Aby przetestować selektor, wystarczy go wpisać w polu tekstowym *Selektor* i kliknąć przycisk *Zastosuj* (patrz rysunek 4.5).

Uwaga: Więcej informacji na temat przykładów z książki można znaleźć na stronie 46.

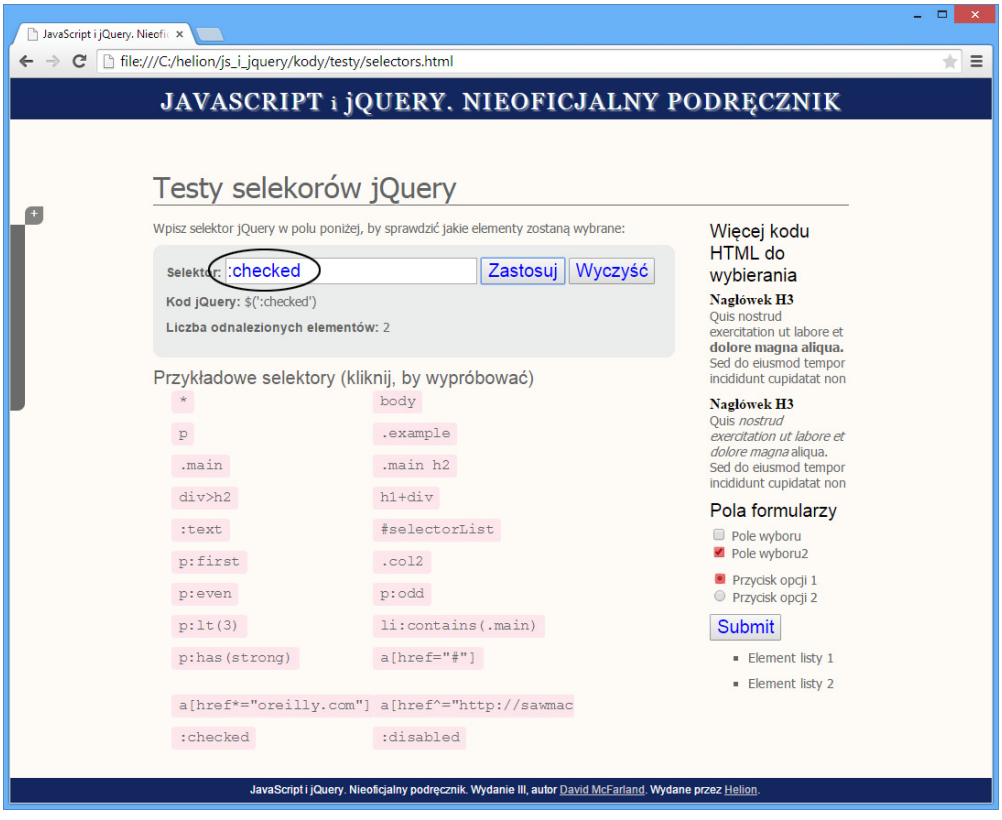
Selektory identyfikatorów

Każdy element strony, który ma określony identyfikator (wartość atrybutu `id`), można pobrać przy użyciu selektora identyfikatora. Załóżmy, że na stronie WWW znajduje się kod HTML w następującej postaci:

```
<p id="message">Komunikat specjalny</p>
```

Aby pobrać taki element przy użyciu biblioteki jQuery, należy użyć następującego wywołania:

```
var messagePara = $('#message');
```



Rysunek 4.5. *Plik selectors.html, dostępny w przykładach dołączonych do tej książki, pozwala testować selektory jQuery. Wystarczy wpisać selektor w polu Selektor (zakreślonym na rysunku) i kliknąć przycisk Zastosuj. Strona przekształci selektor na obiekt jQuery, a wszystkie pobrane przez niego elementy zostaną wyróżnione na czerwono. Poniżej pola prezentowany jest kod jQuery zastosowany do pobrania tych elementów oraz ich liczba. W przedstawionym przykładzie selektorem jest :checked, w związku z czym zostały wyróżnione wszystkie zaznaczone przyciski opcji i pola wyboru (konkretnie są to dwa pola widoczne w prawym dolnym rogu strony)*

Nie wystarczy tu podanie samego identyfikatora elementu (`'message'`) — należy użyć pełnego selektora CSS (`'#message'`). Innymi słowy, przed identyfikatorem elementu należy umieścić znak #, tworząc w ten sposób prawidłowy selektor identyfikatora CSS.

Selektory elementów

Biblioteka jQuery udostępnia także swój własny zamiennik metody DOM — `getElementsByTagName()`. By z niego skorzystać, wystarczy podać w wywołaniu jQuery nazwę znacznika. Aby na przykład pobrać wszystkie znaczniki `<a>` umieszczone na stronie z wykorzystaniem starej metody DOM, należałoby użyć poniższego wywołania:

```
var linksList = document.getElementsByTagName('a');
```

Po zastosowaniu jQuery analogiczne wywołanie ma następującą postać:

```
var linksList = $('a');
```

WIEDZA W PIGUŁCE

Zrozumieć CSS

Kaskadowe arkusze stylów są obszernym zagadnieniem pojawiającym się zawsze w ramach dyskusji o języku JavaScript. Aby korzyść z czytania tej książki była możliwie największa, konieczna jest przynajmniej pobieżna znajomość zasad projektowania stron WWW oraz wiedza dotycząca kaskadowych arkuszy stylów (CSS) i sposobu ich stosowania. CSS to najważniejsze z narzędzi, jakim dysponują projektanci stron i którego używają do tworzenia pięknych witryn; jeśli zatem jeszcze nie znasz tej technologii, najwyższy czas się jej nauczyć. Znajomość CSS nie tylko ułatwi korzystanie z biblioteki jQuery, lecz jednocześnie sprawi, że będziesz w stanie użyć kombinacji CSS i JavaScriptu do tworzenia na swoich stronach interaktywnych efektów wizualnych.

Jeśli potrzebujesz pomocy, by szybko rozpocząć pracę z CSS, możesz skorzystać z wielu dostępnych zasobów.

Podstawowe informacje na temat CSS możesz znaleźć na stronie *HTML Dog CSS Tutorials* (*http://www.htmldog.com/guides/css/*). Dostępne są na niej samouczki na poziomie podstawowym, średnim oraz zaawansowanym.

Można także kupić egzemplarz książki *CSS3. Nieoficjalny podręcznik. Wydanie III*, zawierającej wyczerpującą prezentację CSS (oraz wiele praktycznych przykładów, takich jak zamieszczone w tej książce).

Jednak podczas korzystania z biblioteki jQuery najważniejsza jest znajomość **selektorów CSS** — czyli narzędzia informującego przeglądarkę WWW, w jakich elementach strony należy zastosować dane reguły CSS. Do tego celu doskonale nadają się wszystkie publikacje wymienione w tej ramce. Istnieje także kilka doskonałych stron, na które można zajrzeć w celu odświeżenia informacji na temat różnych, dostępnych selektorów:

- *http://css.maxdesign.com.au/selectutorial/*,
- *https://developer.moz lla.org/en-US/docs/Web/Guide/CSS/Getting_started/Selectors*.

Wskazówka: Biblioteka jQuery obsługuje nawet większą liczbę różnego rodzaju selektorów niż te, które zostały tu przedstawione. W książce opisano wiele przydatnych selektorów jQuery, lecz ich pełną listę można znaleźć na stronie *http://api.jquery.com/category/selectors/*.

Selektory klas

Kolejnym przydatnym sposobem pobierania elementów stron jest zastosowanie nazwy klasy. Załóżmy, że chcemy utworzyć pasek nawigacyjny z rozwijanym menu — kiedy użytkownik przesunie wskaźnik myszy nad przyciskiem menu, ma pojawić się dodatkowe, rozwijane menu. Do utworzenia i kontroli działania takiego menu niezbędne jest zastosowanie kodu JavaScript oraz posiadanie możliwości zaprogramowania każdego z przycisków paska nawigacyjnego tak, by umieszczenie na nim wskaźnika myszy powodowało wyświetlenie menu.

Uwaga: Ponieważ pobieranie wszystkich elementów podanej klasy jest bardzo często wykonywaną operacją, zatem najnowsze wersje przeglądarek udostępniają specjalną metodę służącą do tego celu. Jednak ze względu na to, że nie wszystkie przeglądarki ją obsługują (przykładem jest choćby Internet Explorer 8 oraz jego wcześniejsze wersje), zastosowanie biblioteki, takiej jak jQuery, uwzględniającej różnice pomiędzy przeglądarkami, jest wprost nieocenione.

Jednym ze sposobów rozwiązania takiego problemu mogłoby być dodanie do każdego elementu głównego paska nawigacyjnego określonej klasy — na przykład `navButton` — oraz zastosowanie skryptu w celu pobrania *wyłącznie* elementów należących do danej klasy i wyposażenie ich w całą magię związaną z wyświetlaniem menu. Może się wydawać, że takie rozwiązanie jest złożone, jednak aktualnie najważniejsze jest to, że w celu zapewnienia prawidłowego działania paska nawigacyjnego potrzebny jest jakiś sposób pobrania tylko tych elementów, które należą do konkretnej klasy.

Na szczęście jQuery udostępnia bardzo prosty sposób pobierania wszystkich elementów należących do klasy o podanej nazwie. Wystarczy w tym celu użyć selektora klasy CSS, który ma następującą postać:

```
$('.submenu')
```

Także w tym przypadku warto zwrócić uwagę, że selektor klasy CSS użyty w wywołaniu jQuery wygląda jak selektor klasy CSS — czyli składa się z nazwy klasy poprzedzonej kropką. Po pobraniu znaczników można nimi operować przy użyciu innych możliwości jQuery. Aby na przykład ukryć wszystkie znaczniki należące do klasy `.submenu`, można użyć następującego wywołania:

```
$('.submenu').hide();
```

Więcej informacji na temat funkcji `hide()`podano na stronie 212., jednak przykład ten pokazuje, w jaki sposób używana jest biblioteka jQuery.

Selektory zaawansowane

Biblioteka jQuery pozwala także na stosowanie bardziej zaawansowanych selektorów, za pomocą których można precyzyjne wybrać potrzebne elementy. Nie warto jednak już teraz ich opanowywać: po przeczytaniu kilku kolejnych rozdziałów książki i lepszym poznaniu sposobów działania biblioteki jQuery oraz korzystania z niej w celu modyfikowania stron WWW zechcesz zapewne wrócić do tej części rozdziału i przyjrzeć się im ponownie.

- **Selektory elementów potomnych** pozwalają odwołać się do znacznika umieszczonego wewnątrz innego. Przykładowo załóżmy, że utworzyliśmy wypunktowaną listę odnośników i znacznikowi `<ul>` nadaliśmy identyfikator `navBar` — `<ul id= navBar >`. Wyrażenie jQuery w postaci `$('a')` pobierze wszystkie znaczniki `<a>` istniejące na stronie. Jeśli jednak chcemy pobrać wyłącznie odnośniki umieszczone wewnątrz listy, możemy to zrobić przy użyciu selektora w następującej postaci:

  ```
  $('#navBar a')
  ```

  Także w tym przypadku jest to standardowa postać selektora CSS, który składa się z selektora, umieszczonego za nim znaku odstępu oraz kolejnego selektora. Ostatni podany selektor (w tym przypadku jest to `a`) określa elementy docelowe, natomiast wszystkie selektory umieszczone na lewo od niego — elementy, wewnątrz których są umieszczone elementy docelowe.

- **Selektory dzieci** pozwalają pobierać elementy będące dziećmi innych elementów. „Dziecko" to bezpośredni element potomny innego elementu; na przykład w kodzie HTML zobrazowanym na rysunku 4.4 znaczniki `<h1>` oraz `<p>` są dziećmi znacznika `<body>`, natomiast znacznik `<strong>` nim nie jest (gdyż jest umieszczony wewnątrz znacznika `<p>`). Selektor dziecka tworzy się, zapisując najpierw element rodzica, następnie znak większości (`>`) i w końcu element dziecka. Aby na przykład pobrać znaczniki `<p>` będące dziećmi znacznika `<body>`, należałoby użyć wywołania:

  ```
  $('body > p')
  ```

- **Selektory elementów sąsiadujących** pozwalają pobierać znaczniki, które w kodzie HTML są umieszczone bezpośrednio za jakimiś innymi znacznikami. Załóżmy na przykład, że dysponujemy ukrytym panelem, wyświetlanym po kliknięciu konkretnej zakładki. W kodzie HTML taka zakładka mogłaby być reprezentowana przez jakiś znacznik nagłówka (na przykład `<h2>`), natomiast ukryty panel — przez znacznik `<div>` umieszczony bezpośrednio za nagłówkiem. Aby wyświetlić taki znacznik `<div>` (nasz panel), musimy mieć możliwość pobrania go. Przy użyciu jQuery oraz selektorów elementów sąsiadujących można to zrobić bardzo łatwo:

  ```
  $('h2 + div')
  ```

  Aby utworzyć taki selektor, wystarczy umieścić znak plusa (+) pomiędzy dwoma selektorami (przy czym mogą to być selektory dowolnego typu: identyfikatora, klasy bądź elementów). Selektor umieszczony z prawej strony określa elementy, jakie należy pobrać, przy czym muszą one być poprzedzone elementami pasującymi do selektora umieszczonego z lewej strony znaku +.

- **Selektory atrybutów** pozwalają pobierać elementy na podstawie tego, czy posiadają konkretne atrybuty, a nawet, czy atrybuty te posiadają ściśle określone wartości. Korzystając z takiego selektora, można odszukać wszystkie znaczniki `<img>`, w których użyto atrybutu `alt`, a nawet te spośród nich, w których w atrybucie `alt` podano odpowiedni łańcuch znaków. Można także odszukać wszystkie odnośniki, jakie odwołują się do stron spoza naszej witryny, a następnie dodać do nich kod, który sprawi, że strony te zostaną wyświetlone w nowym oknie przeglądarki.

  Selektor atrybutu jest umieszczany za nazwą elementu, którego atrybuty chcemy sprawdzać. Aby na przykład znaleźć wszystkie znaczniki `<img>`, w których został podany atrybut `alt`, można użyć następującego wyrażenia:

  ```
  $('img[alt]')
  ```

  Istnieje kilka różnych rodzajów selektorów atrybutów. Oto one.

  - `[atrybut]` pobierają elementy, w których kodzie HTML został podany konkretny atrybut; na przykład `$('a[href]')` znajdzie wszystkie znaczniki `<a>`, w których została podana wartość atrybutu `href`. Używając takiego selektora, można pominąć wszystkie nazwane odnośniki — `<a name= jakiesMiejsce ↪Strony ></a>` — czyli odnośniki używane do poruszania się w obrębie tej samej strony.
  - `[atrybut= wartość ]` pozwalają pobrać elementy, w których konkretny atrybut ma określoną wartość; na przykład poniższy selektor pozwala pobrać wszystkie pola tekstowe formularza:

    ```
    $('input[type="text"]')
    ```

    Ponieważ niemal wszystkie pola formularzy korzystają z tego samego znacznika — `<input>` — zatem jedynym sposobem określenia typu pola jest sprawdzenie jego atrybutu `type` (pobieranie pól formularzy na podstawie ich typu jest operacją wykonywaną tak często, że jQuery udostępnia specjalne selektory służące właśnie do tego celu, zostały one opisane na stronie 281).
  - `[atrybut^= wartość ]` odnajduje elementy, w których wartość określonego atrybutu *rozpoczyna się* od podanego ciągu znaków. Aby na przykład znaleźć wszystkie odnośniki wskazujące strony spoza naszej witryny, można użyć następującego kodu:

```
$('a[type^="http://"]')
```

Należy zwrócić uwagę, że nie cała wartość atrybutu musi pasować do łańcucha podanego w selektorze, a jedynie jej początek. A zatem selektor `href^= http://` odnajdzie odnośniki wskazujące strony *http://www.google.com*, *http://helion.pl* i tak dalej. Takiego selektora można także użyć do pobrania wszystkich odnośników służących do wysyłania wiadomości poczty elektronicznej; oto przykład:

```
$('a[href^="mailto:"]')
```

- `[atrybut$= wartość ]` odnajduje elementy, w których wartość określonego atrybutu *kończy się* podanym ciągiem znaków, co jest doskonałym sposobem odnajdywania rozszerzeń plików. Przy użyciu selektora tego typu można odszukać wszystkie odnośniki prowadzące do plików PDF (na przykład po to, by przy użyciu kodu JavaScript dodać do nich ikonę PDF lub automatycznie wygenerować odnośnik do strony firmy Adobe, z której użytkownik mógłby pobrać program Acrobat Reader). Selektor pozwalający na pobranie wszystkich odnośników do plików PDF ma następującą postać:

  ```
  $('a[href$=".pdf"]')
  ```

- `[atrybut*= wartość ]` pozwala pobrać wszystkie elementy, których określony atrybut zawiera podany ciąg znaków. Dzięki temu można odszukać na przykład odnośniki dowolnego typu prowadzące do domeny o określonej nazwie. Oto przykładowy selektor pozwalający na pobranie wszystkich odnośników do strony *missingmanuals.com* (*http://missingmanuals.com*):

  ```
  $('a[href*="missingmanuals.com"]')
  ```

  Powyższy selektor jest na tyle elastyczny, że pozwala nie tylko na pobieranie odnośników wskazujących stronę *http://www.missingmanuals.com*, lecz także kierujących na strony *http://missingmanuals.com* bądź *http://www.missingmanuals.com/library.html*.

Uwaga: Biblioteka jQuery udostępnia całą grupę selektorów bardzo użytecznych podczas pracy z formularzami. Pozwalają one na pobieranie takich elementów jak pola tekstowe, pola haseł czy też przyciski opcji. Więcej informacji na ich temat można znaleźć na stronie 281.

Filtry jQuery

Biblioteka jQuery zapewnia także możliwość filtrowania pobieranych elementów na podstawie ich pewnych cech charakterystycznych. I tak filtr `:even` pozwala pobrać każdy parzysty element kolekcji. Oprócz tego można wyszukiwać elementy zawierające podane inne elementy, określony tekst, elementy, które nie są aktualnie widoczne, a nawet elementy *niepasujące* do podanego selektora. Aby użyć takiego filtra, za głównym selektorem należy umieścić dwukropek i podać nazwę filtra. Aby na przykład odszukać wszystkie parzyste wiersze tabeli, należałoby użyć następującego selektora jQuery:

```
$('tr:even')
```

Powyższe wywołanie pobiera każdy parzysty znacznik <tr>. Aby dodatkowo zawęzić wybór, możemy zażądać pobrania wszystkich parzystych wierszy tabeli należącej do klasy stripped. Można to zrobić przy użyciu poniższego selektora:

```
$('.stripped tr:even')
```

Oto sposób, w jaki działają filtry, w tym także przedstawiony powyżej filtr :even.

- Filtry :even oraz :odd pobierają *co drugi* element z grupy. Filtry te działają nieco wbrew temu, co podpowiada intuicja; trzeba jednak pamiętać, że kolekcja elementów pobranych przez jQuery jest listą wszystkich elementów strony pasujących do podanego selektora. Pod tym względem przypominają one nieco tablice (opisane na stronie 77). Każdy element pobrany przez jQuery ma swój indeks, a trzeba pamiętać, że numeracja indeksów tablic w języku JavaScript zaczyna się od zera (patrz strona 79). A zatem, ponieważ filtr odrzuca każdy parzysty element kolekcji (0., 2., 4. i tak dalej), zatem w efekcie zwrócony zostanie element pierwszy, trzeci i tak dalej. Innymi słowy, filtr ten pobiera nieparzyste elementy kolekcji. Filtr :odd działa na tej samej zasadzie, przy czym odrzuca elementy nieparzyste (1., 3., 5. i tak dalej).
- Filtry :first oraz :last zwracają odpowiednio pierwszy i ostatni element z grupy. Aby na przykład pobrać pierwszy akapit strony, należy użyć następującego wywołania:

  ```
  $('p:first');
  ```

  Z kolei poniżej przedstawiono wywołanie pozwalające na pobranie ostatniego akapitu:

  ```
  $('p:last');
  ```

- Filtra :not() można użyć, by odszukać elementy, które *nie pasują* do podanego selektora. Załóżmy na przykład, że chcemy pobrać wszystkie znaczniki <a> z wyjątkiem tych, które należą do klasy navButton. Oto sposób, w jaki można to zrobić:

  ```
  $('a:not(.navButton)');
  ```

  W wywołaniu funkcji :not() przekazywany jest selektor, który chcemy ignorować. Użyty w powyższym przykładzie .navButton jest selektorem klasy, a zatem należy go zrozumieć tak: „elementy, które nie należą do klasy .navButton". Filtra :not() można używać wraz z większością innych filtrów jQuery oraz przeważającą liczbą jej selektorów; a zatem, by odszukać wszystkie odnośniki, których adresy nie zaczynają się od http://, można użyć następującego wywołania:

  ```
  $('a:not([href^="http://"])')
  ```

- Filtr :has() zwraca wszystkie elementy zawierające inny, podany selektor. Załóżmy, że interesują nas wszystkie znaczniki <li>, jednak wyłącznie wtedy, gdy wewnątrz nich umieszczony jest znacznik <a>. W tym celu możemy użyć następującego wywołania:

  ```
  $('li:has(a)');
  ```

  Takie rozwiązanie różni się znacząco od selektora elementów potomnych, gdyż pozwala pobrać nie elementy <a>, lecz elementy <li> zawierające wewnątrz jakieś odnośniki.

- Filtr `:contains()` zwraca wszystkie elementy zawierające podany tekst. Przykładowo poniższe wywołanie pozwala zwrócić wszystkie odnośniki z napisem `Kliknij mnie!`:

```
$('a:contains(Kliknij mnie!)');
```

- Filtr `:hidden()` odnajduje elementy ukryte, czyli takie, których właściwość `display` CSS ma wartość `none` (co oznacza, że nie są one wyświetlane na stronie), elementy ukryte przy użyciu funkcji `hide()` (opisanej dokładniej na stronie 212), elementy o wysokości lub szerokości wynoszącej zero oraz ukryte pola formularzy. (Filtr ten nie zwraca elementów, których właściwość `visibility` CSS ma wartość `invisible`). Przykładowo załóżmy, że ukryliśmy kilka elementów `<div>`. Oto sposób, w jaki przy użyciu jQuery można je pobrać i ponownie wyświetlić:

```
$('div:hidden').show();
```

  Powyższe wywołanie nie spowoduje żadnych zmian w znacznikach `<div>`, które w momencie wywoływania są widoczne. (Więcej informacji na temat funkcji `show()` można znaleźć na stronie 212).

- Filtr `:visible` jest przeciwieństwem filtra `:hidden`. Zwraca on wszystkie widoczne elementy strony.

Zrozumienie kolekcji jQuery

Wybierając elementy strony przy użyciu obiektu jQuery — na przykład: `$('navBar a')` — nie otrzymujemy w efekcie tradycyjnej listy węzłów DOM, takich jak zwracane przez metody `getElementById()` lub `getElementsByTagName()`. Zamiast tego zwracana jest specjalna, charakterystyczna dla biblioteki jQuery kolekcja elementów. Do elementów tych nie możesz wykorzystać tradycyjnych metod DOM, a zatem, jeśli już poznałeś metody DOM, czytając inną książkę, okaże się, że żadnej z nich nie będziesz mógł w bezpośredni sposób zastosować do elementów zwracanych przez wywołanie jQuery. Można uznać, że jest to ogromna wada biblioteki. Jednak okazuje się, że jQuery dysponuje odpowiednikami niemal wszystkich właściwości i metod DOM. Oznacza to, że można z nimi zrobić to wszystko, co przy użyciu tradycyjnych technik DOM, lecz szybciej, wygodniej i z wykorzystaniem krótszego kodu.

Niemniej jednak istnieją dwie, kluczowe, pojęciowe różnice pomiędzy działaniem kolekcji DOM i kolekcji jQuery. Biblioteka jQuery została napisana po to, by ułatwiać pisanie kodu JavaScript i skracać czas pisania aplikacji w tym języku. Jednym z jej podstawowych celów jest zapewnienie możliwości wykonywania wielu złożonych operacji przy użyciu możliwie krótkiego kodu. By tak się działo, jQuery używa dwóch, niezwykłych zasad: *automatycznych pętli* oraz *łańcuchów wywołań funkcji*.

Automatyczne pętle

Podczas korzystania ze standardowych metod DOM dysponujemy zazwyczaj grupą elementów strony, a następnie musimy utworzyć pętlę (patrz strona 109), by pobrać każdy z wybranych wcześniej węzłów i coś z nim zrobić. Jeśli na przykład chcemy pobrać wszystkie obrazki na stronie, a następnie je ukryć — a takie rozwiązanie może być niezbędne w przypadku tworzenia interaktywnego pokazu slajdów — najpierw należy pobrać obrazki, a następnie utworzyć pętlę, która je ukryje.

Ponieważ przetwarzanie elementów kolekcji w pętli jest tak często realizowaną czynnością, została ona wbudowana w funkcje biblioteki jQuery. Innymi słowy, wykonując jakąś funkcję jQuery na grupie elementów, nie musimy jawnie tworzyć pętli, gdyż funkcja wykona ją automatycznie.

Aby na przykład pobrać wszystkie obrazki umieszczone wewnątrz znacznika `<div>` o identyfikatorze `slideshow`, a następnie je ukryć, wystarczy wykonać następujące wywołanie jQuery:

```
$('#slideshow img').hide();
```

Kolekcja znaczników pobranych przez wywołanie `$('#slideshow img')` może liczyć na przykład 50 elementów. Funkcja `hide()` automatycznie pobierze każdy z nich i go ukryje. Rozwiązanie to jest tak wygodne (wystarczy sobie wyobrazić, jak wielu pętli `for` nie musimy pisać), że aż dziwi, dlaczego nie zostało wbudowane w JavaScript.

Łańcuchy wywołań funkcji

Czasami może się zdarzyć, że na kolekcji elementów będziemy chcieli wykonać sekwencję kilku różnych czynności. Załóżmy, że z poziomu kodu JavaScript chcemy określić szerokość i wysokość znacznika `<div>` (o identyfikatorze `popUp`). Normalnie musielibyśmy do tego celu użyć przynajmniej dwóch wierszy kodu. Gdy jednak zastosujemy bibliotekę jQuery, wystarczy tylko jeden:

```
$('#popUp').width(300).height(300);
```

Biblioteka jQuery korzysta z rozwiązania nazywanego *łańcuchami wywołań*, które pozwala zapisywać kilka wywołań funkcji jQuery jedno bezpośrednio za drugim. Wywołanie każdej funkcji jest połączone z następnym przy użyciu kropki (`.`), a każda funkcja operuje na tej samej kolekcji elementów, co poprzednia. A zatem powyższe wywołanie najpierw zmienia szerokość elementu o identyfikatorze `popUp`, a następnie jego wysokość. Możliwość takiego łączenia wywołań pozwala na świadome wykonanie liczby operacji. Załóżmy, że chcemy nie tylko zmienić szerokość i wysokość znacznika `<div>`, lecz także umieścić wewnątrz niego jakiś tekst i stopniowo wyświetlić go na stronie (przy założeniu, że aktualnie nie jest widoczny). Możemy to zrobić przy użyciu poniższego, zwięzłego kodu:

```
$('#popUp').width(300).height(300).text('Siema!').fadeIn(1000);
```

Kod ten wywołuje cztery funkcje jQuery — `width()`, `height()`, `text()` oraz `fadeIn()` — przy czym każda z nich modyfikuje elementy o identyfikatorze `popUp`.

Wskazówka: Długie sekwencje wywołań funkcji jQuery mogą być mało czytelne, dlatego też wielu programistów zapisuje je w osobnych wierszach:

```
$('#popUp').width(300)
           .height(300)
           .text('Siema!')
           .fadeIn(1000);
```

Jeśli tylko średnik zostanie umieszczony wyłącznie za *ostatnim* wywołaniem, interpreter JavaScriptu potraktuje taki kod jak jedną instrukcję.

Możliwość łączenia wywołań funkcji w łańcuch jest cechą biblioteki jQuery, innymi słowy, w takiej sekwencji nie można używać ani funkcji pisanych samodzielnie, ani wbudowanych funkcji języka JavaScript, a przynajmniej nie bez samodzielnego zapewnienia takich możliwości.

Dodawanie treści do stron

Biblioteka jQuery udostępnia wiele funkcji służących do wykonywania operacji na elementach oraz zawartości stron WWW, zaczynając od prostych operacji podmiany fragmentów kodu HTML, przez precyzyjne umiejscawianie jednych elementów w zależności od pozostałych, a na całkowitym usuwaniu znaczników i treści strony kończąc.

Uwaga: Przykładowy plik *content_functions.html* umieszczony w przykładach do książki, w katalogu *testy*, pozwala przetestować działanie każdej z tych funkcji. Aby przekonać się, jak one działają, wystarczy wyświetlić ten plik w przeglądarce, wpisać dowolny tekst w polu formularza i kliknąć dowolny z umieszczonych na stronie prostokątów.

Aby przeanalizować zamieszczone w dalszej części rozdziału przykłady tych funkcji, załóżmy, że dysponujemy stroną zawierającą następujący fragment kodu HTML:

```
<div id="container">
  <div id="errors">
    <h2>Błędy:</h2>
  </div>
</div>
```

Poniżej przedstawiono pięć najbardziej użytecznych funkcji do operowania na zawartości stron.

- Funkcja `.html()` może zarówno odczytać cały kod HTML umieszczony wewnątrz określonego elementu, jak i go zastąpić. Jest używana wraz z wywołaniem jQuery pobierającym jakieś elementy strony.

  Aby pobrać kod HTML umieszczony wewnątrz wybranego elementu, należy umieścić wywołanie funkcji `.html()` tuż za selektorem jQuery. I tak przy założeniu, że na stronie będzie się znajdował przedstawiony powyżej fragment kodu, moglibyśmy użyć następującego wywołania:

  ```
  alert($('#errors').html());
  ```

  Wywołanie to wyświetli okienko informacyjne JavaScript zawierające następujący łańcuch znaków: `<h2>Błędy:</h2>`. Podczas korzystania z tej funkcji w taki sposób można skopiować kod HTML umieszczony wewnątrz konkretnego elementu i wkleić go wewnątrz innego elementu.

  Jeśli w wywołaniu funkcji `.html()` podamy jakiś łańcuch znaków, zostanie on użyty jako zamiennik aktualnej zawartości elementu:

  ```
  $('#errors').html('<p>W formularzu odnaleziono cztery błędy.</p>');
  ```

  Powyższe wywołanie spowoduje zmianę kodu HTML umieszczonego wewnątrz elementu o identyfikatorze `errors`. Po wykonaniu wywołania przedstawiony wcześniej fragment kodu HTML przyjmie następującą postać:

```
<div id="container">
  <div id="errors">
    <p>W formularzu odnaleziono cztery błędy.</p>
  </div>
</div>
```

Warto zwrócić uwagę, że usunięte zostały także znaczniki <h2>, umieszczone wewnątrz elementu o identyfikatorze errors. Dzięki zastosowaniu innych spośród wymienionych niżej funkcji jQuery można uniknąć zmieniania całego kodu HTML.

Uwaga: W przypadku użycia funkcji html() lub text() do pobrania kodu HTML lub tekstu z kolekcji jQuery zawierającej większą liczbę elementów, kod HTML lub tekst zostanie pobrany wyłącznie z *pierwszego* elementu. Jeśli na przykład strona zawiera 10 znaczników i zostanie wykonany kod var divContents = $('div').html(), to w zmiennej divContents zostanie zapisana wyłącznie zawartość pierwszego elementu .

Jednak podczas użycia tych samych funkcji, czyli html() i text(), do wstawiania odpowiednio kodu HTML lub tekstu do kolekcji jQuery zmodyfikowane zostaną *wszystkie* jej elementy. Przykładowo kod $('div').html(' Witaj, świecie! ');, spowoduje dodanie akapitu z tekstem "Witaj, świecie!" do każdego elementu na stronie.

- Funkcja .text() działa podobnie jak funkcja .html(), jednak nie akceptuje znaczników HTML. Jest przydatna, gdy chcemy zamienić tekst umieszczony wewnątrz znacznika. Przykładowo we fragmencie kodu umieszczonym na początku tego podrozdziału znajduje się znacznik <h2> zawierający słowo Błędy: . Załóżmy, że po uruchomieniu programu w celu sprawdzenia, czy w formularzu nie ma już żadnych błędów, chcemy zmienić tekst umieszczony w tym nagłówku na: Nie znaleziono błędów! . Możemy to zrobić przy użyciu następującego wywołania:

  ```
  $('#errors h2').text('Nie znaleziono błędów!');
  ```

  W efekcie znacznik <h2> pozostanie w dotychczasowej postaci — zmieni się jedynie umieszczony wewnątrz niego tekst. jQuery koduje wszystkie znaczniki umieszczone w łańcuchu przekazywanym w wywołaniu funkcji .text(), na przykład zostanie przekształcony na <p>. Może się to przydać, gdy będziemy chcieli wyświetlić nawiasy kątowe i znaczniki w tekście *na* stronie. Funkcji tej można używać do wyświetlania na stronie fragmentów kodu HTML, tak by użytkownicy mogli je przeanalizować.

- Funkcja .append() dodaje przekazany w jej wywołaniu kod HTML jako ostatni element potomny wybranego wcześniej elementu. Załóżmy, że wybraliśmy wcześniej znacznik , lecz zamiast zmieniać jego zawartość, chcemy na jej końcu, tuż przed zamykającym znacznikiem , coś dodać. Funkcja .append() jest doskonałym sposobem dodawania nowych punktów na końcach list uporządkowanych (<ol>) lub wypunktowanych (<ul>). W ramach przykładu załóżmy, że na stronie zawierającej fragment kodu HTML przedstawiony na początku tego podrozdziału wykonaliśmy poniższe wywołanie:

  ```
  $('#errors').append('<p>W formularzu znaleziono cztery błędy.</p>');
  ```

  Po wykonaniu tej instrukcji kod HTML strony będzie mieć następującą postać:

```
<div id="container">
  <div id="errors">
    <h2>Błędy:</h2>
    <p>W formularzu znaleziono cztery błędy.</p>
  </div>
</div>
```

Należy zwrócić uwagę, że początkowy kod umieszczony wewnątrz znacznika `<div>` nie uległ zmianie, a nowy fragment kodu HTML został dodany bezpośrednio za nim.

- Funkcja `.prepend()` działa podobnie do `.append()`, ale powoduje dodanie przekazanego kodu HTML bezpośrednio za otwierającym znacznikiem wybranego elementu. Przykładowo załóżmy, że na tej samej stronie, co wcześniej, chcemy wykonać poniższą instrukcję:

  ```
  $('#errors').prepend('<p>W formularzu znaleziono cztery błędy.</p>');
  ```

  W efekcie kod strony przyjmie następującą postać:

  ```
  <div id="container">
    <div id="errors">
      <p>W formularzu znaleziono cztery błędy.</p>
      <h2>Błędy:</h2>
    </div>
  </div>
  ```

  Nowa treść została umieszczona bezpośrednio za otwierającym znacznikiem `<div>`.

- Jeśli chcemy dodać nowy kod HTML *poza* wybranym elementem, bądź to przed jego znacznikiem otwierającym, bądź bezpośrednio za znacznikiem zamykającym, możemy w tym celu użyć funkcji `.before()` oraz `.after()`. Często spotyka się rozwiązanie polegające na sprawdzaniu zawartości pola formularza, by przed jego przesłaniem zweryfikować, czy pole nie jest puste. Załóżmy, że kod HTML pola formularza ma następującą postać:

  ```
  <input type="text" name="userName" id="userName">
  ```

  A teraz załóżmy, że w momencie wysyłania formularza to pole będzie puste. Możemy napisać program, który je sprawdzi i doda za nim komunikat o błędzie. Aby dodać komunikat za polem (na razie nie będziemy zaprzątać sobie głowy, jak można sprawdzić, czy zawartość pola formularza jest prawidłowa — te informacje znajdziesz na stronie 299), można użyć funkcji `.after()`:

  ```
  $('#userName').after('<span class="error">Nazwa użytkownika jest
  wymagana</span>');
  ```

  Powyższa instrukcja sprawi, że na stronie pojawi się komunikat o błędzie, a jej kod będzie teraz wyglądał tak:

  ```
  <input type="text" name="userName" id="userName">
  <span class="error">Nazwa użytkownika jest wymagana</span>
  ```

  Funkcja `.before()` umieszcza przekazany kod HTML przed wybranym elementem. A zatem poniższy wiersz kodu:

  ```
  $('#userName').before('<span class="error">Nazwa użytkownika jest
  ↪wymagana</span>');
  ```

  wygeneruje następujący fragment kodu HTML:

  ```
  <span class="error">Nazwa użytkownika jest wymagana</span>
  <input type="text" name="userName" id="userName">
  ```

Uwaga: Funkcje opisane w tym podrozdziale — `html()`, `text()` i tak dalej — są najczęściej używanym sposobem dodawania i modyfikowania zawartości stron WWW; jednak istnieją także inne. Pełne zestawienie funkcji jQuery służących do manipulowania kodem HTML i zawartością stron WWW można znaleźć na stronie *http://api.jquery.com/category/manipulation/*.

Zastępowanie i usuwanie wybranych elementów

Może się zdarzyć, że będziemy chcieli całkowicie zastąpić lub usunąć wybrane elementy. Wyobraźmy sobie, że korzystając z JavaScriptu, utworzyliśmy pojawiające się okienko dialogowe (nie takie proste okienko tworzone przez funkcję `alert()`, lecz profesjonalnie wyglądające okno będące w rzeczywistości bezwzględnie umiejscowionym znacznikiem `<div>`, wyświetlonym na stronie). Kiedy użytkownik kliknie przycisk *Zamknij*, chcemy, by okienko zostało usunięte ze strony. Do tego celu możemy użyć funkcji jQuery o nazwie `.remove()`. Załóżmy, że znacznik `<div>` zawierający całe okienko ma identyfikator `popup`. W takim przypadku możemy usunąć okienko, używając następującego wywołania:

```
$('#popup').remove();
```

Możliwości funkcji `.remove()` nie ograniczają się do usuwania pojedynczych elementów. Równie dobrze można przy jej użyciu usunąć wszystkie znaczniki `<span>` należące do klasy `error`. Wystarczy skorzystać z następującego wywołania:

```
$('span.error').remove();
```

Dodatkowo mamy także możliwość całkowitego zastąpienia wybranych elementów zupełnie nową zawartością. Załóżmy, że dysponujemy stroną ze zdjęciami produktów oferowanych przez naszą firmę. Kiedy użytkownik kliknie zdjęcie produktu, zostaje on dodany do koszyka. Załóżmy też, że chcemy, by po kliknięciu obrazka znacznik `<img>` został zastąpiony jakimś tekstem (takim jak `Dodano do koszyka.`). W następnym rozdziale dowiesz się, jak sprawić, by konkretne elementy reagowały na zdarzenia (takie jak kliknięcie obrazka), na razie jednak przyjmijmy, że na stronie istnieje znacznik `<img>` o identyfikatorze `product101`, który chcemy zamienić na tekst. Oto sposób, w jaki możemy to zrobić za pomocą biblioteki jQuery:

```
$('#product101').replaceWith('<p>Dodano do koszyka.</p>');
```

Powyższe wywołanie usuwa ze strony znacznik `<img>` i zastępuje go znacznikiem `<p>`.

Uwaga: Biblioteka jQuery udostępnia także funkcję `clone()`, pozwalającą na zrobienie kopii wybranego elementu. Będziesz miał okazję przekonać się, jak działa, w przykładzie rozpoczynającym się na stronie 174.

Ustawianie i odczyt atrybutów znaczników

Dodawanie, usuwanie oraz modyfikowanie elementów strony to nie jedyne operacje, które z powodzeniem można wykonywać przy użyciu biblioteki jQuery; nie są to także jedyne czynności, które będziemy chcieli wykonywać na pobranych elementach. Bardzo często będziemy modyfikować wartości atrybutów elementów — na przykład dodawać do elementu nazwę klasy lub zmieniać jakąś właściwość CSS — a także pobierać wartości atrybutów przykładowo po to, by sprawdzić, na jaką stronę prowadzi konkretny odnośnik.

UWAGA! WYSOCE UŻYTECZNE NARZĘDZIA!

Problemy z podglądem źródła strony

Jednym z problemów spotykanych podczas korzystania z kodu JavaScript do manipulowania standardem DOM w celu dodawania, zmieniania, usuwania i reorganizowania elementów HTML są trudności, jakich nastręcza wyświetlenie kodu HTML po zakończeniu działania skryptu. Przykładowo polecenie *Pokaż źródło*, dostępne we wszystkich przeglądarkach, pokazuje jedynie kod HTML strony w takiej postaci, w jakiej został pobrany z serwera. Innymi słowy, pokazuje, jaki był kod HTML *przed* wprowadzeniem zmian przez skrypt. To w bardzo dużym stopniu utrudnia określenie, czy nasz skrypt generuje kod HTML w zamierzonej postaci. Gdybyśmy na przykład mogli oglądnąć kod HTML strony po programowym dodaniu do niego dziesięciu komunikatów o błędach odnalezionych w formularzu lub po utworzeniu rozbudowanego okna dialogowego z formularzem, znacznie ułatwiłoby to sprawdzenie, czy generowany kod HTML ma taką postać, jaką planowaliśmy.

Na szczęście wszystkie nowoczesne przeglądarki udostępniają zestaw narzędzi dla programistów, z których można skorzystać w celu przeanalizowania *wyświetlanego kodu HTML* — czyli kodu, który przeglądarka wyświetli po zakończeniu wykonywania skryptu. Zazwyczaj narzędzia te są prezentowane jako osobny panel umieszczony u dołu okna przeglądarki, poniżej wyświetlanej strony. Różne karty pozwalają na dostęp do kodów JavaScript, HTML, CSS oraz innych, przydatnych zasobów. Konkretne nazwy poszczególnych kart oraz metody wyświetlania tych narzędzi różnią się w poszczególnych przeglądarkach.

- W przeglądarce Chrome należy kliknąć przycisk *Dostosowywanie i kontrolowanie Google Chrom*, a następnie wybrać opcję *Więcej narzędzi/Narzędzia dla programistów*. W wyświetlonym u dołu okna panelu trzeba kliknąć przycisk *Elements*.
- W przeglądarce Firefox należy kliknąć przycisk *Otwórz menu*, a następnie ikonę Narzędzia i wybrać opcję *Inspektor*. Spowoduje to wyświetlenie u dołu okna panelu prezentującego kod HTML strony zmodyfikowany (lub, jak kto woli, zainfekowany) przez JavaScript.
- W przeglądarce Internet Explorer trzeba nacisnąć klawisz *F12*, co spowoduje wyświetlenie panelu *Narzędzia Deweloperskie*. W przeglądarce IE na karcie *HTML* początkowo wyświetlany jest kod HTML strony pobrany z serwera (czyli ten sam, który można zobaczyć, wybierając opcję *Pokaż źródło*). Jednak po kliknięciu ikony odświeżenia strony (lub naciśnięciu klawisza *F5*) na karcie zostanie wyświetlony kod HTML, zawierający także wszelkie zmiany wprowadzone przez kod JavaScript.
- W przeglądarce Safari należy upewnić się, że jest wyświetlone menu *Programowanie* (kliknij przycisk koła zębatego, wybierz opcję *Preferencje*, kliknij przycisk *Zaawansowane* i upewnij się, że jest zaznaczone pole wyboru *Pokazuj menu Programowanie w pasku menu*). Teraz wystarczy otworzyć wybraną stronę, wybrać z menu głównego opcję *Programowanie/Pokaż Inspektora www* i w wyświetlonym u dołu okna panelu kliknąć przycisk *Elements*.
- W przeglądarce Opera należy wybrać z menu głównego opcję *Narzędzia/Zaawansowne/Opera Dragonfly*. (Dragonfly to wbudowany w tę przeglądarkę zestaw narzędzi dla programistów). W panelu wyświetlonym u dołu okna przeglądarki trzeba kliknąć kartę *Dokumenty*.

Klasy

Kaskadowe arkusze stylów są technologią o bardzo dużych możliwościach, pozwalającą na stosowanie wyszukanych sposobów wizualnego formatowania kodu HTML. Jedna reguła CSS może dodać do strony kolorowe tło, a inna — całkowicie ukryć wybrany element. Istnieje możliwość tworzenia zaawansowanych efektów wizualnych już poprzez samo usuwanie, dodawanie i zmienianie klas stosowanych w elementach strony przy użyciu kodu JavaScript. Ponieważ przeglądarki WWW potrafią bardzo szybko i wydajnie przetwarzać oraz stosować reguły CSS, zatem już samo dodanie klasy do znacznika może spowodować całkowitą zmianę jego wyglądu, a nawet go ukryć.

Biblioteka jQuery udostępnia kilka funkcji służących do manipulowania nazwami klas w znacznikach HTML. Oto one.

- Funkcja `addClass()` dodaje do elementu podaną nazwę klasy. Funkcja ta wywoływana jest po wykonaniu selekcji elementów, a przekazywany do niej łańcuch znaków reprezentuje dodawaną nazwę klasy. Aby na przykład dodać klasę `externalLink` do wszystkich odnośników wskazujących stronę nienależącą do naszej witryny, można użyć następującego wywołania jQuery:

  ```
  $('a[href^="http://"]').addClass('externalLink');
  ```

  Takie wywołanie przekształci poniższy znacznik:

  ```
  <a href="http://helion.pl/">
  ```

  do następującej postaci:

  ```
  <a href="http://helion.pl/" class="externalLink">
  ```

  Aby ta funkcja była do czegokolwiek przydatna, przed jej użyciem trzeba zdefiniować odpowiednie style i dodać je do arkusza stylów używanych na stronie. Dzięki temu, kiedy kod JavaScript doda do znacznika nazwę klasy, przeglądarka będzie w stanie zastosować właściwości zdefiniowane w regule CSS.

Uwaga: W wywołaniach funkcji `addClass()` oraz `removeClass()` podawana jest sama nazwa klasy — należy przy tym pominąć kropkę zapisywaną przed tą nazwą w selektorach CSS. Przykładowo wywołanie `addClass('externalLink')` jest prawidłowe, natomiast `addClass('.externalLink')` już nie.

  Funkcja działa prawidłowo także w sytuacji, gdy w znaczniku jest już podana nazwa innej klasy — nie usuwa ona nazw klas już używanych w znaczniku, a jedynie dodaje do nich nową.

Uwaga: Dodawanie wielu nazw klas do jednego znacznika jest całkowicie poprawne, a niejednokrotnie stanowi bardzo wygodne rozwiązanie. Więcej informacji na temat tej techniki można znaleźć na stronie *http://www.cvwdesign.com/txp/article/177/use-more-than-one-css-class*.

- Funkcja `removeClass()` działa odwrotnie do `addClass()`. Usuwa podaną nazwę klasy z wybranych elementów. Aby na przykład usunąć klasę `highlight` ze znacznika `<div>` o identyfikatorze `alertBox`, można to zrobić przy użyciu następującego wywołania:

  ```
  $('#alertBox').removeClass('highlight');
  ```

- Może się także zdarzyć, że będziemy chcieli *przełączać* wykorzystanie konkretnej klasy — czyli dodawać ją, jeśli nie jest używana, oraz usuwać, jeśli jest. Przełączanie jest bardzo popularnym sposobem prezentowania elementu naprzemiennie w stanie włączonym i wyłączonym. Kiedy na przykład klikniemy pole wyboru, zostanie on zaznaczony (włączony), a kiedy klikniemy go po raz drugi — zaznaczenie zniknie (przycisk zostanie wyłączony).

  Załóżmy, że na stronie WWW dostępny jest przycisk, którego kliknięcie powoduje zmianę klasy używanej w znaczniku `<body>`. Dzięki temu można całkowicie zmienić wygląd strony, przygotowując drugi zestaw stylów wykorzystujących selektory elementów potomnych. Ponowne kliknięcie przycisku spowoduje usunięcie klasy ze znacznika `<body>`, zatem strona zostanie przywrócona do początkowego wyglądu. Na potrzeby tego przykładu załóżmy, że przycisk, który

użytkownik klika, by zmienić wygląd strony, ma identyfikator `changeStyle` i będzie przełączał klasę o nazwie `altStyle`. Oto kod, który zapewni takie działanie strony:

```
$('#changeStyle').click(function() {
  $('body').toggleClass('altStyle');
});
```

Aktualnie nie będziemy zaprzątać sobie głowy pierwszym oraz ostatnim wierszem powyższego kodu — mają one związek ze zdarzeniami pozwalającymi skryptowi reagować na czynności wykonywane przez użytkownika na stronie — takie jak klikanie przycisku. Funkcja `toggleClass()` została zastosowana w wierszu wyróżnionym pogrubioną czcionką; w odpowiedzi na kolejne kliknięcia przycisku, naprzemiennie, dodaje ona oraz usuwa z elementu podaną klasę.

Odczyt i modyfikacja właściwości CSS

Funkcja `css()` biblioteki jQuery pozwala na wprowadzanie bezpośrednich zmian we właściwościach CSS elementów. A zatem zamiast dodawania do elementu klas możemy dodać do niego obramowanie określonego koloru, określić jego szerokość lub położenie. Funkcji `css()` można używać na trzy sposoby: aby pobrać aktualną wartość właściwości CSS elementu, aby podać wartość konkretnej właściwości CSS oraz by w jednym wywołaniu podać wartości wielu właściwości CSS.

W celu odczytania bieżącej wartości właściwości CSS należy podać jej nazwę w wywołaniu funkcji. Załóżmy na przykład, że interesuje nas kolor tła znacznika `<div>` o identyfikatorze `main`:

```
var bgColor = $('#main').css('background-color');
```

Po wykonaniu powyższej instrukcji zmienna `bgColor` będzie zawierać łańcuch znaków stanowiący wartość koloru tła wskazanego elementu.

Uwaga: Może się zdarzyć, że wartości właściwości CSS zwracane przez jQuery nie będą zgodne z naszymi oczekiwaniami. W przypadku kolorów (takich jak kolor tła czy kolor tekstu) jQuery zawsze zwraca wartość RGB w postaci `rgb(255,0,10)` bądź też, jeśli w kolorze został określony poziom przezroczystości, zwraca wartość RGBA w postaci `rgba(255,10,10,.5)`. jQuery zwraca wartości RGB niezależnie od tego, czy w arkuszu stylów kolor został podany przy użyciu zapisu szesnastkowego (`#F4477A`), jako wartość RGB używająca wartości procentowych (`rgb(100%,10%,0%)`), czy też za pomocą zapisu HSL (`hsl(72,100%,50%)`). Dodatkowo jQuery przekształca wszystkie jednostki na piksele; a zatem, nawet jeśli w regule stylu określiliśmy wielkość czcionki elementu `<body>`, przypisując jej wartość `150%`, to w przypadku odczytu właściwości `font-size` jQuery zwróci wartość wyrażoną w pikselach.

Funkcja `css()` pozwala także ustawiać wartość właściwości CSS wybranego elementu. Aby użyć jej w taki sposób, należy przekazać w jej wywołaniu dwa argumenty: nazwę ustawianej właściwości CSS oraz jej wartość. Aby na przykład ustawić wielkość czcionki elementu `<body>` na `200%`, należałoby użyć następującego wywołania:

```
$('body').css('font-size', '200%');
```

Drugi z podawanych argumentów może być wartością łańcuchową, taką jak `'200%'`, bądź liczbową, którą jQuery przekształci na wartość wyrażoną w pikselach.

Aby zmienić wielkość wypełnienia wszystkich znaczników należących do klasy `pullquote` na 100 pikseli, można użyć następującego wywołania:

```
$('.pullquote').css('padding', 100);
```

Uwaga: Podczas określania wartości właściwości CSS przy użyciu funkcji `css()` jQuery można stosować właściwości skrótowe. Niżej pokazano, w jaki sposób można dodać do każdego akapitu należącego do klasy `highlight` czarne obramowanie o szerokości jednego piksela:

```
$('p.highlight').css('border', '1px solid black');
```

Często bardzo użyteczna może być możliwość zmieniania właściwości CSS na podstawie ich bieżącej wartości. Załóżmy, że chcemy umieścić na stronie przycisk *Powiększ czcionkę*, który użytkownik mógłby kliknąć i dwukrotnie zwiększyć wielkość używanej czcionki. Aby opracować takie rozwiązanie, należy odczytać wartość właściwości, a następnie odpowiednio ją zmienić. W tym przypadku musimy najpierw odczytać bieżącą wartość właściwości `font-size`, a następnie przypisać jej dwukrotnie większą wartość. Okazuje się jednak, że zadanie to jest nieco bardziej skomplikowane, niż można by sądzić. Poniżej przedstawiony został kod JavaScript realizujący to zadanie, a poniżej wyjaśnienie jego działania:

```
var baseFont = $('body').css('font-size');
baseFont = parseInt(baseFont,10);
$('body').css('font-size',baseFont * 2);
```

Pierwsza instrukcja powyższego fragmentu kodu pobiera wartość właściwości `font-size` znacznika `<body>`; zwrócona wartość jest łańcuchem znaków i ma postać `'16px'`. Ponieważ chcemy podwoić tę wartość — pomnożyć ją przez dwa — zatem musimy zamienić łańcuch znaków na liczbę, pozbywając się umieszczonych na końcu liter `'px'`. Właśnie tę operację wykonuje drugi wiersz kodu, w którym używamy metody `parseInt()` języka JavaScript, opisanej bardziej szczegółowo na stronie 587. Metoda ta usuwa wszystkie znaki zapisane po liczbie. A zatem po wykonaniu tego drugiego wiersza kodu w zmiennej `baseFont` będzie zapisana wartość liczbowa, taka jak `16`. I w końcu, w ostatnim wierszu kodu określamy nową wartość właściwości `font-size`, mnożąc w tym celu zmienną `baseFont` razy `2`.

Uwaga: Powyższy przykładowy kod zmodyfikuje wielkość czcionki tekstów wyświetlanych na stronie wyłącznie w przypadku, gdy wielkości zostaną podane przy użyciu jednostek względnych, takich jak `em`, bądź w formie wartości procentowych. Jeśli jednak wielkości czcionek w innych znacznikach zostaną podane przy użyciu jednostek bezwzględnych, takich jak piksele, zmiana wielkości czcionki znacznika `<body>` nie da żadnego widocznego efektu.

Jednoczesna zmiana wielu właściwości CSS

Gdy chcemy zmienić wartości większej liczby właściwości CSS, wcale nie musimy uciekać się do stosowania wielu wywołań funkcji `css()`. Aby na przykład dynamicznie wyróżnić znacznik `<div>` (choćby w odpowiedzi na jakąś czynność wykonaną przez użytkownika), można zmienić jego kolor tła *oraz* obramowanie, używając do tego następującego fragmentu kodu:

```
$('#hightlightedDiv').css('background-color','#FF0000');
$('#hightlightedDiv').css('border','2px solid #FE0037');
```

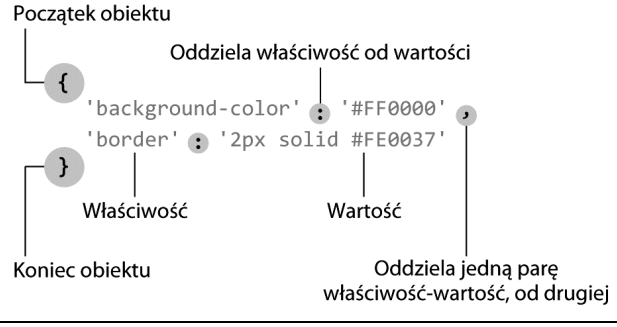
Innym rozwiązaniem jest przekazanie w wywołaniu metody `css()` tak zwanego **literału obiektowego**. Można go sobie wyobrazić jako listę par, na które składa się nazwa właściwości oraz odpowiadająca jej wartość. Tuż za nazwą właściwości należy zapisać dwukropek (`:`), a po nim wartość; poszczególne pary są od siebie oddzielone przecinkami, a całość zapisana wewnątrz pary nawiasów klamrowych (`{ }`). A zatem literał obiektowy definiujący dwie, przedstawione wcześniej właściwości CSS, będzie mieć następującą postać:

```
{ 'background-color' : '#FF0000', 'border' : '2px solid #FE0037' }
```

Ponieważ przeanalizowanie takiego literału zapisanego w jednym wierszu może być kłopotliwe, zapisuje się każdy jego element w osobnym wierszu. Poniższy literał jest dokładnym odpowiednikiem przedstawionego wcześniej:

```
{
  'background-color' : '#FF0000',
  'border' : '2px solid #FE0037'
}
```

Podstawowa struktura literałów obiektowych została zilustrowana na rysunku 4.6.



Rysunek 4.6. *Literały obiektowe języka JavaScript zapewniają możliwość tworzenia list właściwości i odpowiadających im wartości. JavaScript traktuje każdy literał obiektowy jako jeden blok informacji — podobnie jak tablicę stanowiącą listę wartości. Literały obiektowe, takie jak ten, będziemy często stosować podczas przekazywania opcji do wtyczek jQuery*

Aby użyć literału obiektowego w funkcji `css()`, wystarczy przekazać go w jej wywołaniu, co pokazano na poniższym przykładzie:

```
$('#highlightedDiv').css({
  'background-color' : '#FF0000',
  'border' : '2px solid #FE0037'
});
```

Warto dokładnie przeanalizować ten przykład, gdyż wygląda nieco inaczej, niż to, co widziałeś do tej pory, oraz dlatego, że w kolejnych rozdziałach podobne fragmenty kodu będą się pojawiać dosyć często. Pierwszą rzeczą, na jaką należy zwrócić uwagę, jest to, że powyższy kod stanowi jedną instrukcję JavaScriptu (w zasadzie jest to jeden wiersz kodu). Można to rozpoznać po tym, że średnik umieszczony jest na samym końcu ostatniego wiersza. Instrukcja ta została podzielona i zapisana w czterech wierszach, aby poprawić przejrzystość kodu.

Następnie należy zwrócić uwagę, że literał obiektowy jest argumentem (tak jak inny, pojedynczy element danych) wywołania funkcji `css()`. A zatem w łańcuchu znaków `css({` widocznym w powyższym kodzie otwierający nawias okrągły jest elementem wywołania funkcji, natomiast otwierający nawias klamrowy (`{`) oznacza początek literału obiektowego. Z kolei trzy znaki widoczne w ostatnim wierszu przykładu należy interpretować w następujący sposób: zamykający nawias klamrowy `}`

kończy literał obiektowy przekazywany w wywołaniu funkcji; nawias) kończy wywołanie funkcji — jest to ostatni nawias kodu `css()` — i wreszcie średnik (;) kończy całą instrukcję JavaScriptu.

Jeśli wszystkie te zagadnienia związane z literałami obiektowymi powodują ból głowy, nic nie stoi na przeszkodzie, być podawał wartości właściwości CSS po jednej, tak jak w poniższym przykładzie:

```
$('#highlightedDiv').css('background-color','#FF0000');
$('#highlightedDiv').css('border','2px solid #FE0037');
```

Choć lepszym rozwiązaniem byłoby wykorzystanie charakterystycznej cechy biblioteki jQuery, którą jest możliwość tworzenia łańcuchów wywołań (patrz strona 156). Polega ona na wywołaniu kilku funkcji jQuery operujących na jednej kolekcji elementów, przy czym kolejne wywołanie jest zapisywane za poprzednim i oddzielone od niego kropką (.):

```
$('#highlightedDiv').css('background-color','#FF0000')
                    .css('border','2px solid #FE0037');
```

Taki kod można czytać tak: odszukaj element o identyfikatorze `hightlightedDiv`, zmień jego kolor tła, a następnie zmień postać jego obramowania. Tworzenie łańcucha wywołań zapewnia lepszą wydajność niż dwukrotne pobieranie tego samego elementu — `$('#highlightedDiv')` — gdyż każda taka operacja wiąże się z koniecznością wykonania przez przeglądarkę całego kodu jQuery związanego z pobieraniem elementów stron. A zatem poniższy fragment kodu nie jest optymalny:

```
$('#highlightedDiv').css('background-color','#FF0000');
$('#highlightedDiv').css('border','2px solid #FE0037');
```

Zmusza on przeglądarkę do pobrania elementu, zmiany jego właściwości CSS, pobrania tego samego elementu po raz wtóry (co stanowi niepotrzebne marnotrawstwo czasu procesora) i określenie kolejnej właściwości CSS. Przy wykorzystaniu możliwości tworzenia łańcucha wywołań przeglądarka musi pobrać interesujący nas element strony tylko jeden raz, a następnie wykonać dwie funkcje modyfikujące właściwości CSS. Jednokrotne pobranie elementu zajmuje mniej czasu niż wykonanie tej czynności dwa razy.

Odczyt, ustawienia i usuwanie atrybutów HTML

Ponieważ modyfikowanie klas oraz wartości CSS przy użyciu kodu JavaScript to czynności wykonywane bardzo często, jQuery posiada wbudowane funkcje do ich obsługi. Jednak w rzeczywistości funkcje `addClass()` oraz `css()` są jedynie uproszczonymi sposobami modyfikowania atrybutów `class` i `style` znaczników HTML. Biblioteka jQuery udostępnia także funkcje ogólnego przeznaczenia służące do obsługi atrybutów HTML. Są to funkcje `attr()` oraz `removeAttr()`.

Funkcja `attr()` umożliwia odczyt wartości atrybutu znacznika HTML. Aby na przykład określić adres obrazka aktualnie wyświetlanego w znaczniku `<img>`, wystarczy przekazać w jej wywołaniu łańcuch znaków `'src'` (czyli pobrać wartość atrybutu `src` znacznika `<img>`):

```
var imageFile = $('#banner img').attr('src');
```

Funkcja `attr()` zwraca wartość atrybutu w takiej postaci, w jakiej został podany w kodzie HTML. Powyższy kod zwróci wartość atrybutu `src` pierwszego znacznika `<img>` umieszczonego wewnątrz innego znacznika o identyfikatorze `banner`; a zatem w zmiennej `imageFile` zostanie zapisana ścieżka podana w kodzie HTML strony; na przykład: `'images/banner.png'` lub `'http://www.jakaswitryna.pl/images/banner.png'`.

Uwaga: Podczas podawania nazwy atrybutu w wywołaniu funkcji `attr()` nie trzeba zwracać uwagi na używaną wielkość liter — można użyć dowolnego zapisu: `href`, `HREF`, `hReF` i tak dalej.

Jeśli w wywołaniu funkcji `attr()` przekazany zostanie drugi argument, spowoduje ustawienie wartości podanego atrybutu. Aby na przykład wyświetlić na stronie inny obrazek, wystarczy w następujący sposób zmienić wartość atrybutu `src` znacznika `<img>`:

```
$('#banner img').attr('src','images/newImage.png');
```

Jeśli trzeba całkowicie usunąć atrybut ze znacznika, można skorzystać z funkcji `removeAttr()`. I tak poniższe wywołanie usuwa ze znacznika `<body>` atrybut `bgColor`:

```
$('body').removeAttr('bgColor');
```

Wykonanie akcji na każdym elemencie kolekcji

Zgodnie z informacjami podanymi na stronie 155, jedną z unikalnych cech biblioteki jQuery jest to, że większość jej funkcji może wykonać zadaną czynność dla każdego z pobranych elementów. Aby na przykład stopniowo ukryć wszystkie znaczniki `<img>` na stronie, wystarczy jedno proste wywołanie jQuery:

```
$('img').fadeOut();
```

Funkcja `fadeOut()` powoduje powolne znikanie elementu, a gdy zastosujemy ją do kolekcji jQuery zawierającej większą liczbę elementów, sprawi, że zniknie każdy z nich. Istnieje wiele sytuacji, w których będziemy chcieli kolejno pobrać wszystkie elementy kolekcji i dla każdego z nich wykonać jakąś sekwencję czynności. Właśnie do tego celu służy funkcja `.each()` jQuery.

Przykładowo załóżmy, że chcemy zebrać wszystkie odnośniki prowadzące do zewnętrznych stron i umieszczone na danej stronie, a także wyświetlić je na dole w osobnej ramce z bibliografią, którą można by zatytułować „Inne strony wymieniane w artykule". (No dobrze, być może wcale nie chcesz tego robić, ale nie psuj zabawy). W każdym razie taką ramkę z bibliografią można utworzyć, wykonując następujące czynności.

1. **Pobierz wszystkie odnośniki wskazujące strony spoza naszej witryny.**
2. **Pobierz atrybuty HREF (czyli adresy URL) każdego z tych odnośników.**
3. **Dodaj każdy z tych adresów URL do listy odnośników umieszczonych w ramce z bibliografią.**

Biblioteka jQuery nie udostępnia wbudowanej funkcji wykonującej dokładnie te czynności, jednak możemy je wykonać samodzielnie, używając funkcji `each()`. Jest to zwyczajna funkcja jQuery, a zatem można ją dodać do wywołania jQuery pobierającego interesujące nas elementy strony:

```
$('selektor').each();
```

Funkcje anonimowe

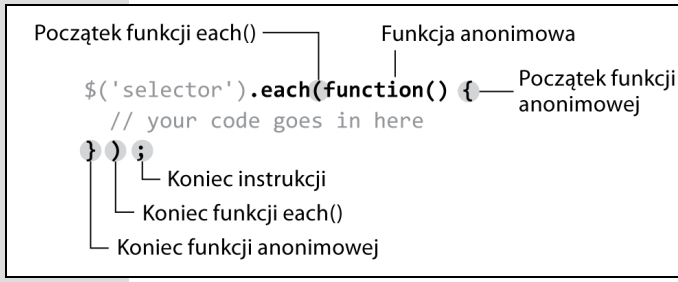
Aby skorzystać z funkcji `each()`, musimy przekazać do niej specjalny rodzaj argumentu — **funkcję anonimową**. Funkcja anonimowa jest zwyczajną funkcją zawierającą wszystkie czynności, jakie chcemy wykonać na pobranym elemencie strony. Określamy ją słowem *anonimowa*, gdyż, w odróżnieniu od zwyczajnych funkcji, które poznałeś na stronie 115, nie posiada nazwy. Poniżej przedstawiona została podstawowa struktura funkcji anonimowej:

```
function() {
  // tu jest umieszczany kod funkcji
}
```

Ponieważ funkcja anonimowa nie ma nazwy, zatem nie mamy jak jej wywołać. Podczas wywoływania zwyczajnej funkcji i przekazywania do niej argumentów używana jest jej nazwa — `calculateSalesTax()`. Funkcja anonimowa sama jest używana jako argument wywołania innej funkcji (jest to dziwne i trudne do pojęcia, ale właśnie tak jest!). A oto sposób, w jaki można użyć funkcji anonimowej jako argumentu funkcji `each()`:

```
$('selektor').each(function(){
  // tu jest umieszczony kod funkcji
});
```

Poszczególne elementy tej konstrukcji zostały pokazane i opisane na rysunku 4.7. Szczególnie kłopotliwy jest ostatni wiersz powyższego przykładu, gdyż zawiera trzy symbole kończące trzy różne części konstrukcji. Nawias klamrowy (}) stanowi zakończenie funkcji anonimowej (a jednocześnie koniec argumentu przekazywanego do funkcji `each()`), nawias ()) jest ostatnim znakiem wywołania funkcji `each()`; natomiast średnik (;) kończy instrukcję JavaScriptu. Innymi słowy, interpreter JavaScript potraktuje cały ten kod jak jedną instrukcję.



Rysunek 4.7. *Funkcja each() jQuery pozwala przejrzeć całą kolekcję wybranych elementów i na każdym z nich wykonać jakieś operacje. Kluczowym warunkiem, niezbędnym do stosowania tej funkcji jest dobre zrozumienie funkcji anonimowych*

Skoro zewnętrzna struktura funkcji jest już gotowa, czas umieścić coś wewnątrz funkcji anonimowej, a konkretnie — wszystkie czynności, jakie mają zostać wykonane dla każdego z wybranych elementów strony. Funkcja `each()` działa jak pętla — instrukcje umieszczone wewnątrz funkcji anonimowej zostaną wykonane jeden raz dla każdego z pobranych elementów. Załóżmy na przykład, że mamy stronę zawierającą pięćdziesiąt obrazków; dodamy do niej następujący kod JavaScript:

```
$('img').each(function() {
  alert('Znaleziono obrazek.');
});
```

Spowoduje to pięćdziesięciokrotne wyświetlenie okienka informacyjnego z komunikatem `Znaleziono obrazek.` . (To byłoby naprawdę denerwujące, więc nie próbuj tego robić).

Uwaga: To rozwiązanie może wydać się znajome. Zgodnie z tym, czego się dowiedziałeś na stronie 141, gdy dodajesz kod jQuery na stronie WWW, należy użyć funkcji `$(document).ready()`, by upewnić się, że kod HTML strony zostanie w całości pobrany, zanim przeglądarka wykona jakikolwiek kod JavaScript. Także do tej funkcji przekazywana jest jako argument funkcja anonimowa:

```
$(document).ready(function() {
  // wykonywany kod jest umieszczany
  // w tej funkcji anonimowej
});
```

this oraz $(this)

Oczywiste jest, że podczas korzystania z funkcji `each()` będziemy chcieli pobierać i ustawiać atrybuty każdego z przetwarzanych elementów, aby na przykład pobrać adres URL łącza do strony zewnętrznej. Żeby pobrać aktualny element wewnątrz pętli, używane jest specjalne słowo kluczowe `this`. Reprezentuje ono dowolny element wywołujący funkcję anonimową. A zatem podczas pierwszej iteracji pętli `this` będzie reprezentować pierwszy z elementów strony pobranych przez jQuery, natomiast podczas drugiej iteracji będzie to drugi element.

Biblioteka jQuery działa w taki sposób, że `this` odwołuje się do tradycyjnych elementów DOM, dzięki czemu można za jego pośrednictwem uzyskać dostęp do tradycyjnych właściwości DOM. Jednak, zgodnie z tym, czego już się dowiedziałeś, wyniki zwracane przez jQuery pozwalają na korzystanie ze wszystkich, cudownych funkcji tej biblioteki. Aby zatem skonwertować `this` na obiekt jQuery, należy użyć wywołania w postaci `$(this)`.

Myślisz zapewne, że całe to zamieszanie ze słowem kluczowym `this` zostało wymyślone tylko po to, by spowodować ból głowy. Nie jest to żart, jednak bez wątpienia rozwiązanie to jest nieco zagmatwane. Aby lepiej zrozumieć zasady korzystania z wyrażenia `$(this)`, przyjrzyjmy się ponownie zadaniu opisanemu na początku tego podrozdziału; chodziło w nim o utworzenie u dołu strony ramki z listą odnośników do stron zewnętrznych.

Załóżmy, że w kodzie strony znajduje się już znacznik `<div>`, gotowy do utworzenia takiej listy odnośników:

```
<div id="bibliography">
<h3>Inne strony wymieniane w artykule.</h3>
<ul id="bibList">
</ul>
</div>
```

Pierwszym krokiem jest pobranie listy wszystkich odnośników wskazujących strony spoza witryny. Możemy ją pobrać przy użyciu selektora atrybutu (patrz strona 152):

```
$('a[href^="http://"]')
```

Następnie, aby przetworzyć każdy z pobranych odnośników, musimy dodać do wywołania funkcję `each()`:

```
$('a[href^="http://"]).each()
```

Po czym w wywołaniu funkcji `each()` trzeba przekazać funkcję anonimową:

```
$('a[href^="http://"]).each(function(){

});
```

Pierwszą czynnością, jaką należy wykonać wewnątrz funkcji anonimowej, jest pobranie adresu URL podanego w odnośniku. Ponieważ każdy z odnośników może zawierać unikalny adres URL, musimy go pobierać z aktualnie przetwarzanego elementu podczas każdej iteracji pętli. Możemy to zrobić, posługując się wyrażeniem `$(this)`:

```
$('a[href^="http://"]).each(function(){
  var extLink = $(this).attr('href');
});
```

Wiersz kodu umieszczony wewnątrz funkcji anonimowej i wyróżniony pogrubieniem realizuje dwa podstawowe zadania — tworzy nową zmienną lokalną (`extLink`) i zapisuje w niej wartość właściwości `href` aktualnie przetwarzanego elementu. Podczas każdej iteracji pętli wyrażenie `$(this)` będzie się odnosić do innego odnośnika odnalezionego na stronie, a zatem podczas każdej z tych iteracji wartość zmiennej `extLink` będzie się zmieniać.

Teraz nasze zadanie sprowadza się tylko do dodania nowego punktu do znacznika `<ul>` (został on pokazany w kodzie HTML przedstawionym powyżej). Można to zrobić w następujący sposób:

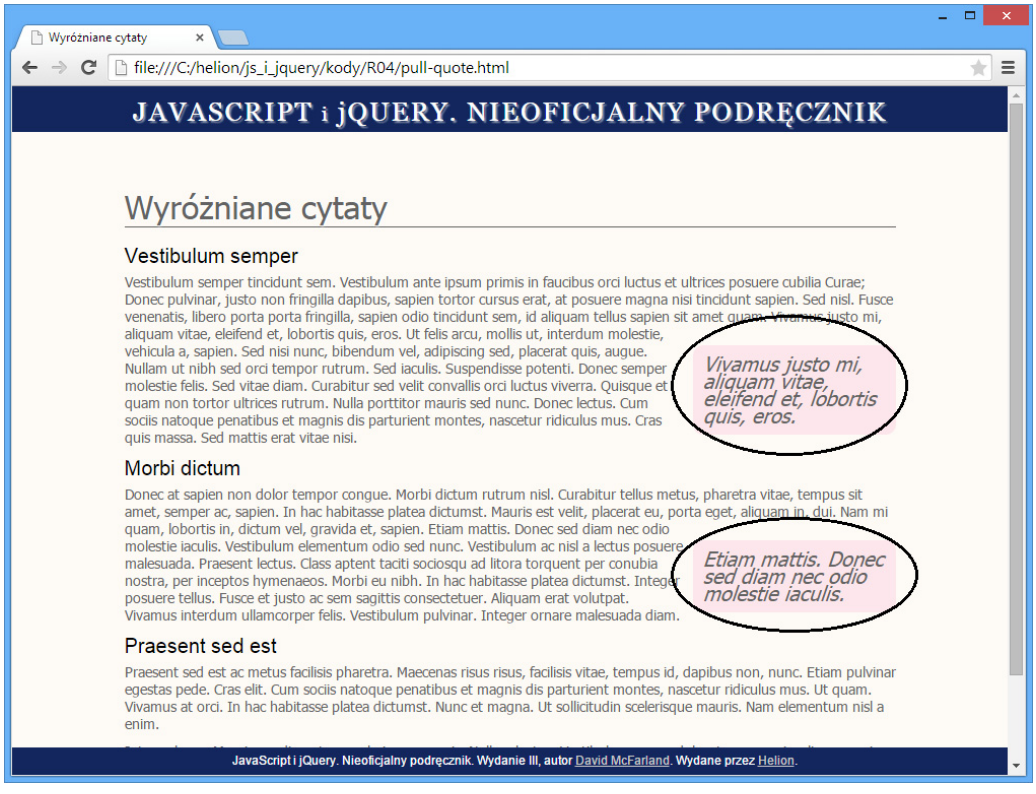
```
$('a[href^="http://"]).each(function(){
  var extLink = $(this).attr('href');
  $('#bibList').append('<li>' + extLink + '</li>');
});
```

Wyrażenia `$(this)` będziesz używał niemal zawsze wtedy, gdy będziesz korzystał z funkcji `each()`, więc po jakimś czasie stanie się Twoim drugim imieniem. Aby przyswoić je trochę lepiej i zdobyć nieco praktyki w posługiwaniu się nim, użyjemy go także w większym przykładzie zamieszczonym dalej w tym rozdziale.

Uwaga: Przykładowy skrypt zamieszczony w tym podrozdziale jest doskonałą ilustracją zastosowania wyrażenia `$(this)`, choć z drugiej strony, nie jest zapewne najlepszym sposobem tworzenia na stronie listy odnośników. Przede wszystkim, nawet jeśli na stronie nie będzie żadnych odnośników, znacznik `<div>` (umieszczony na stałe w kodzie HTML) i tak się pojawi na stronie, choć będzie pusty. Co więcej, jeśli użytkownik wyłączy w przeglądarce obsługę JavaScriptu, po wyświetleniu strony nie zobaczy listy odnośników, lecz jedynie pustą ramkę. Znacznie lepszym rozwiązaniem byłoby użycie skryptu, który tworzyłby nie tylko samą listę odnośników, lecz także znacznik `<div>`, wewnątrz którego ma się ona znaleźć. Takie rozwiązanie można znaleźć w pliku *bibliography.html* dołączonym do przykładów prezentowanych w tym rozdziale.

Automatycznie tworzone, wyróżniane cytaty

W pierwszym przykładzie zamieszczonym w tym podrozdziale napiszemy skrypt, który ułatwia tworzenie wyróżnianych cytatów (wyglądających tak jak przedstawione na rysunku 4.8). **Wyróżniany cytat** (ang. *pull quote*) to ramka zawierająca interesujący cytat wybrany z głównego tekstu publikowanego na danej stronie. Wszystkie gazety, czasopisma oraz witryny WWW używają ich, by wzbudzić zainteresowanie czytelników bądź skierować ich uwagę na ważne lub interesujące zagadnienia. Jednak ręczne tworzenie takich wyróżnianych cytatów wymagałoby powielania tekstu na stronie i umieszczania go w znacznikach — `<div>`, `<span>` bądź jeszcze innych.



Rysunek 4.8. *Ręczne dodawanie wyróżnionych cytatów jest bolesne, zwłaszcza gdy weźmiemy pod uwagę fakt, że przy użyciu języka JavaScript można bardzo łatwo zautomatyzować cały ten proces. W tym przykładzie, po prawej stronie widoczne są dwa wyróżnione cytaty utworzone przy użyciu kodu JavaScript*

Tworzenie kodu HTML wymaga czasu i powoduje powiększenie strony o powielające się fragmenty. Na szczęście z wykorzystaniem JavaScriptu można bardzo szybko utworzyć na stronie dowolną liczbę wyróżnionych cytatów, dodając do niej jedynie niewielkie fragmenty kodu HTML.

Opis rozwiązania

Skrypt, który napiszemy, będzie realizował kilka zadań.

1. **Odnajdzie na stronie wszystkie znaczniki `<span>` należące do klasy o nazwie `pq` (od angielskich słów *pull quote*).**

   Jedyną zmianą, jaką będziemy musieli wprowadzić w kodzie HTML strony, będzie dodanie kilku znaczników `<span>`, wewnątrz których umieścimy tekst, jaki ma zostać przekształcony na wyróżnione cytaty. Załóżmy na przykład, że na stronie znajduje się akapit, zawierający kilka słów, które chcemy pobrać i wyświetlić w formie wyróżnionego cytatu. Wystarczy umieścić te słowa wewnątrz znacznika `<span>`:

   ```
   <span class="pq">... i właśnie w taki sposób odkryłem potwora
   ↪z Loch Ness.</span>.
   ```

2. **Powieli każdy ze znaczników `<span>`.**

   Każdy wyróżniony cytat jest kolejnym znacznikiem `<span>` zawierającym dokładnie ten sam tekst, a zatem możemy skorzystać z JavaScriptu, by powielić istniejące na stronie znaczniki `<span>`.

3. **Usunie z powielonych znaczników `<span>` klasę `pq` i zastąpi ją klasą `pullquote`.**

   Za całą magię formatowania — utworzenie ramki, użycie większej czcionki, wyświetlenie obramowania i zmianę koloru tła — powodującą wizualne wyróżnienie wybranego cytatu nie odpowiada JavaScript. Arkusz stylów używanych na stronie zawiera definicję stylu o nazwie `pullquote`, odpowiadającego za zmianę wyglądu cytatu. A zatem całkowita zmiana wyglądu nowych znaczników `<span>` jest wyłącznie efektem użycia JavaScriptu do zmiany stosowanej w nich nazwy klasy.

4. **Doda powielony znacznik `<span>` do kodu strony.**

   W końcu powielone znaczniki `<span>` trzeba dodać do strony. (W kroku 2. utworzyliśmy kopię znacznika przechowywaną w pamięci przeglądarki, jednak aż do tej pory nie dodaliśmy jej do kodu strony. Takie rozwiązanie pozwala wprowadzać dodatkowe zmiany w wyglądzie powielanych znaczników, zanim zostaną one wyświetlone na stronie).

Kod rozwiązania

Skoro wiemy już, co chcemy zrobić i osiągnąć, nadszedł czas, by otworzyć edytor i zacząć wcielać pomysł w życie.

Uwaga: Informacje dotyczące pobierania przykładów do książki można znaleźć na stronie 46.

1. **W edytorze tekstów otwórz plik *pull-quote.html* umieszczony w katalogu *R04*.**

   Zaczniemy od dodania na początku pliku odwołania do zewnętrznego pliku biblioteki jQuery.

2. **Kliknij pusty wiersz umieszczony tuż powyżej zamykającego znacznika `</head>` i wpisz**:

```
<script src="../_js/jquery.min.js"></script>
```

Ten znacznik wczyta zewnętrzny plik jQuery przechowywany na naszej witrynie. Zwróć uwagę, że plik ten znajduje się w katalogu o nazwie *_js* (nie zapomnij o znaku podkreślenia na samym początku). Teraz musisz dodać drugą parę znaczników `<script>`, w której umieścisz kod JavaScript.

Uwaga: Zapewne zauważysz, że podczas dołączania pliku biblioteki jQuery do przykładowych stron WWW nie jest podawany jej numer wersji, choć informacje zamieszczone na stronie 141 sugerują, aby go zamieszczać, na przykład:

```
<script src="../_js/jquery.1.11.0.min.js"></script>
```

W przykładach prezentowanych w tej książce numer wersji jQuery jest pomijany celowo, aby łatwiej je było uaktualnić, kiedy pojawi się jej kolejna wersja. Przykładowo w czasie, kiedy powstawała ta książka, najnowszą wersją biblioteki jQuery 1 była 1.11.0; więcej informacji na temat różnic pomiędzy wersjami 1. oraz 2. biblioteki jQuery można znaleźć w ramce na stronie 138. Kiedy jednak książka trafi do Twoich rąk, najnowszą wersją jQuery może być 1.11.1 lub 1.12.0. W plikach przykładowych dostępna będzie najnowsza wersja biblioteki, a zatem właśnie jej będziesz używał.

---

3. **Naciśnij klawisz *Enter* (lub *Return*), by utworzyć poniżej nowy, pusty wiersz i wpisz w nim poniższy tekst wyróżniony pogrubieniem:**

```
1  <script src="../_js/jquery.min.js"><script>
2  <script>
3
4  </script>
```

Uwaga: Numery wyświetlone z lewej strony wierszy kodu są jedynie dla naszej informacji — nie wpisuj ich w kodzie strony.

Kolejnym krokiem będzie dodanie wywołania funkcji `$(document).ready()`.

4. **Kliknij pusty wiersz umieszczony pomiędzy znacznikami `<script>` i wpisz w nim poniższy tekst wyróżniony pogrubieniem:**

```
1  <script src="../_js/jquery.min.js"><script>
2  <script>
3  $(document).ready(function() {
4
5  }); // Koniec funkcji ready.
6  </script>
```

Komentarz `// koniec funkcji ready` okaże się szczególnie przydatny w przyszłości, kiedy nasz program stanie się znacznie większy i bardziej skomplikowany. W większych programach często będą występowały sekwencje znaków `});`, z których każda będzie oznaczać koniec funkcji anonimowej i wywołania jakiejś innej funkcji. Umieszczenie za nimi komentarzy umożliwia zidentyfikowanie każdej i sprawia, że kiedy w przyszłości wrócimy do takiego kodu, znacznie łatwiej zrozumiemy, o co w nim chodzi.

Czynności wykonane w punktach od 1. do 4. stanowią podstawowe przygotowania, które będziesz wykonywał, pisząc każdy program używający biblioteki jQuery, zatem koniecznie się upewnij, że dobrze je rozumiesz. Teraz zajmiemy

się najważniejszymi czynnościami, jakie ma wykonywać nasz program — zaczniemy do pobrania wszystkich znaczników zawierających teksty, które chcemy wyświetlić w formie wyróżnionych cytatów.

5. **Dodaj pogrubiony tekst z czwartego wiersza poniższego przykładu:**

```
1  <script src="../_js/jquery.min.js"><script>
2  <script>
3  $(document).ready(function() {
4    $('span.pq')
5  }); // Koniec funkcji ready.
6  </script>
```

Wyrażenie $('span.pq') to selektor jQuery pozwalający pobrać wszystkie znaczniki należące do klasy pq. Teraz dodamy kod niezbędny do przejrzenia znaczników i wykonania na nich operacji.

6. **Dodaj pogrubiony tekst z wierszy 4. i 6. poniższego przykładu:**

```
1  <script src="../_js/jquery.min.js"><script>
2  <script>
3  $(document).ready(function() {
4    $('span.pq').each(function() {
5
6    }); // Koniec funkcji each.
7  }); // Koniec funkcji ready.
8  </script>
```

Zgodnie z informacjami podanymi na stronie 167, each() jest funkcją jQuery pozwalającą przejrzeć kolekcję wybranych elementów strony. Wymaga ona przekazania jednego argumentu — funkcji anonimowej.

Teraz zaczniesz pisać kod funkcji, która będzie przetwarzać kolejne pobrane znaczniki. Pierwszym zadaniem będzie utworzenie kopii przetwarzanego elementu .

7. **Dodaj wyróżniony pogrubieniem kod, umieszczony w 5. wierszu poniższego przykładu:**

```
1  <script src="../_js/jquery.min.js"></script>
2  <script >
3  $(document).ready(function() {
4    $('span.pq').each(function() {
5      var quote=$(this).clone();
6    }); // Koniec funkcji each.
7  }); // Koniec funkcji ready.
8  </script>
```

Działanie tej funkcji rozpoczyna się od utworzenia nowej zmiennej o nazwie quote, zawierającej „klon" (czyli po prostu kopię) aktualnie przetwarzanego elementu (zajrzyj na stronę 169, jeśli zapomniałeś, jakie znaczenie ma wyrażenie $(this)). Funkcja .clone() biblioteki jQuery powiela aktualny element, włącznie z całym, umieszczonym wewnątrz niego kodem HTML. W tym przypadku tworzymy kopię znacznika , włącznie z umieszczonym wewnątrz niego tekstem, który chcemy wyświetlić w formie wyróżnionego cytatu.

Klonowanie elementów powoduje skopiowanie ich w całości, włącznie ze wszelkimi atrybutami. W naszym przypadku kopiowany znacznik należy do klasy o nazwie pq. W kolejnym kroku usuniemy tę klasę ze skopiowanego znacznika.

8. **Dodaj dwa wyróżnione pogrubieniem wiersze kodu (6. i 7.):**

```
<script src="../_js/jquery.min.js"></script>
<script>
$(document).ready(function() {
  $('span.pq').each(function() {
    var quote=$(this).clone();
    quote.removeClass('pq');
    quote.addClass('pullquote');
  }); // Koniec funkcji each.
}); // Koniec funkcji ready.
</script>
```

Zgodnie z informacjami podanymi na stronie 160, funkcja `removeClass()` usuwa podaną nazwę klasy ze wskazanego znacznika, natomiast funkcja `addClass()` dodaje do znacznika podaną nazwę klasy. W tym przypadku operację zamiany nazwy klasy wykonujemy na kopii znacznika `<span>`, zatem będziemy mogli użyć klasy CSS o nazwie `pullquote`, by sformatować skopiowany znacznik i nadać mu wygląd wyróżnionego cytatu.

Kolejną czynnością będzie dodanie znacznika do kodu strony WWW.

9. **Dodaj do skryptu pogrubiony wiersz kodu (8.):**

```
<script src="../_js/jquery.min.js"></script>
<script>
$(document).ready(function() {
  $('span.pq').each(function() {
    var quote=$(this).clone();
    quote.removeClass('pq');
    quote.addClass('pullquote');
    $(this).before(quote);
  }); // Koniec funkcji each.
 }); // Koniec funkcji ready.
</script>
```

Wyróżniony pogrubieniem wiersz kodu jest ostatnim elementem funkcji — do tej pory operowaliśmy na kopii znacznika `<span>` przechowywanej w pamięci przeglądarki. Użytkownik nie zobaczyłby funkcji aż do momentu dodania jej do modelu obiektów dokumentu strony.

W tym kroku dodajemy kopię znacznika `<span>`; umieszczamy ją w kodzie HTML strony bezpośrednio przed oryginalnym elementem. W efekcie wynikowa strona będzie zawierać kod HTML w następującej postaci:

```
<span class="pullquote">... i właśnie w taki sposób odkryłem potwora
➥z Loch Ness.</span><span class="pq">... i właśnie w taki sposób
➥odkryłem potwora z Loch Ness.</span>.
```

Choć kod może sugerować, że na stronie prezentowanej w przeglądarce te dwa fragmenty tekstu zostaną umieszczone tuż obok siebie, jednak zastosowane style CSS sprawią, że cytat zostanie wydzielony i wyświetlony przy prawej krawędzi strony.

Uwaga: Aby uzyskać wizualny efekt wyróżnionego cytatu, w stylu CSS używanym do określenia jego wyglądu została zastosowana właściwość `float`. Sformatowany przy jego użyciu element zostanie wyświetlony z prawej strony akapitu, w którym jest umieszczony, a pozostały tekst będzie go „opływał" z lewej strony. Jeśli nie znasz tej techniki, znacznie więcej informacji na temat działania właściwości `float` możesz znaleźć na stronie *http://css.maxdesign.com.au/floatutorial/*. Jeśli chcesz sprawdzić, jak wygląda definicja stylu `pullquote`, zajrzyj na początek pliku z kodem przykładu — zostały tam umieszczone wszystkie style oraz używane w nich właściwości.

W ten sposób udało się zakończyć tworzenie kodu JavaScript naszego przykładu. Jednak nie uda się zobaczyć żadnych wyróżnionych cytatów, dopóki nie wprowadzisz jeszcze pewnych zmian w kodzie HTML strony.

10. **Odszukaj w kodzie HTML strony pierwszy znacznik `<p>`, następnie odszukaj zdanie i umieść je wewnątrz znaczników `<span class="pq"></span>`, na przykład tak**:

```
<span class="pq">Nullam ut nibh sed orci tempor rutrum.</span>
```

Możesz powtórzyć powyższy proces, by dodać wyróżnione cytaty także do innych akapitów tekstu.

11. **Zapisz plik i wyświetl go w przeglądarce.**

Ostateczny wynik powinien wyglądać tak, jak pokazano na rysunku 4.8. Jeśli jednak nie zobaczysz wyróżnionego cytatu, upewnij się, że prawidłowo dodałeś do kodu strony znacznik `<span>`, zgodnie z informacjami podanymi w kroku 10. Dodatkowo przejrzyj podane na stronie 51 porady związane z poprawianiem niedziałających programów. Pełną wersję przykładu możesz znaleźć w pliku *complete_pull-quote.html*.

5
ROZDZIAŁ

Akcja i reakcja — ożywianie stron za pomocą zdarzeń

W rozmowach na temat języka JavaScript często pada słowo „interaktywny", na przykład: „JavaScript pozwala tworzyć interaktywne strony WWW". Oznacza to, że język JavaScript umożliwia reagowanie stron na działania użytkowników. Przeniesienie wskaźnika myszy nad przycisk nawigacyjny może powodować wyświetlenie menu z odnośnikami, zaznaczenie przycisku opcji — udostępnienie zestawu nowych pól formularza, a kliknięcie miniaturki zdjęcia — przyciemnienie całej strony i wyświetlenie na niej większej wersji tego samego zdjęcia.

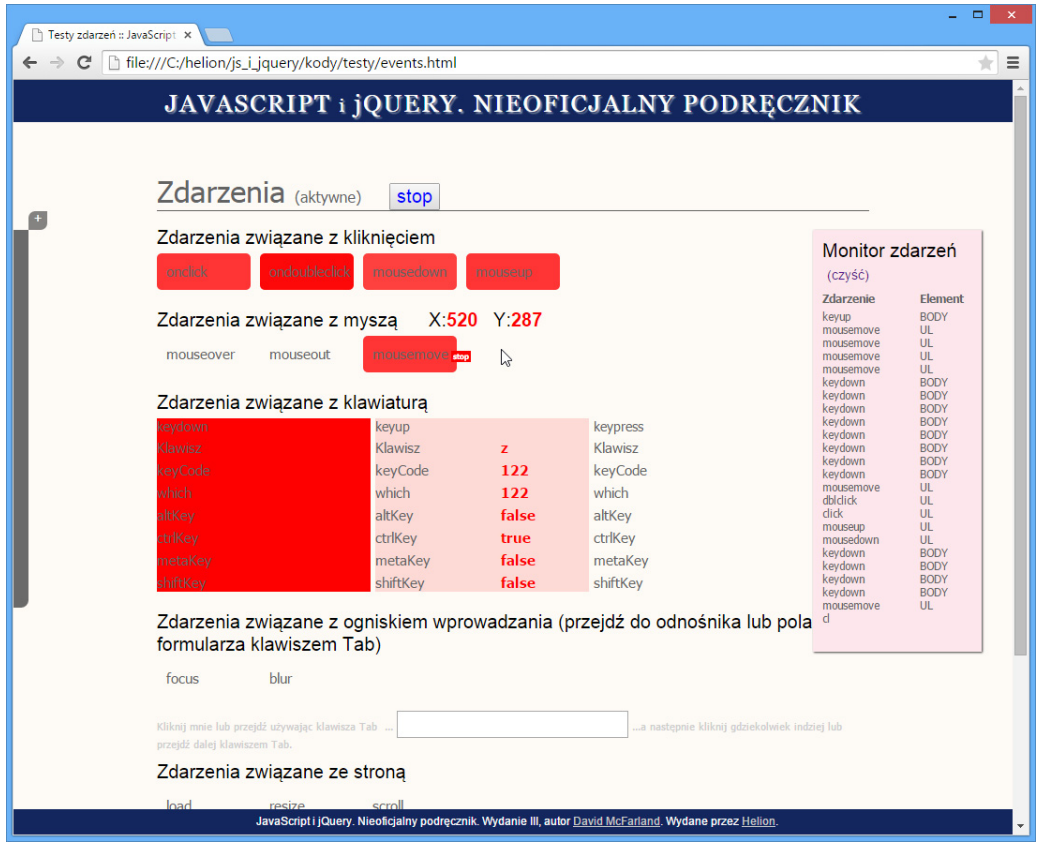
Różne działania użytkowników, na które strona może reagować, to tak zwane *zdarzenia*. JavaScript to język *sterowany zdarzeniami*. Bez tego mechanizmu strona nie może reagować na zachowania internautów ani wykonywać ciekawych operacji. Podobnie działają komputery. Kiedy włączysz system, nic się nie dzieje, dopóki nie zaczniesz uruchamiać programów, klikać plików, wybierać opcji z menu i ruszać wskaźnikiem myszy po ekranie.

Czym są zdarzenia?

Przeglądarki są zaprogramowane tak, aby wykrywały podstawowe zjawiska, takie jak wczytanie strony, przeniesienie wskaźnika, wpisanie znaku lub zmiana wielkości okna. Wszystkie zmiany związane ze stroną WWW to *zdarzenia*. Aby utworzyć interaktywną stronę, należy przygotować skrypty reagujące na zdarzenia. W ten sposób można sprawić, że element `<div>` zniknie lub pojawi się w wyniku kliknięcia, na stronie znajdzie się nowy rysunek po umieszczeniu wskaźnika myszy nad odnośnikiem lub skrypt sprawdzi zawartość pól tekstowych po kliknięciu przez użytkownika przycisku *Wyślij* formularza.

Zdarzenie reprezentuje moment zajścia określonego zjawiska. Kiedy na przykład klikniesz myszą element, w chwili zwolnienia przycisku przeglądarka zasygnalizuje wystąpienie *zdarzenia kliknięcia*. Programiści nazywają moment poinformowania przez przeglądarkę o zajściu zdarzenia jego *zgłoszeniem*.

Przy kliknięciu przeglądarki zgłaszają kilka zdarzeń. Najpierw, bezpośrednio po wciśnięciu przycisku, informują o zdarzeniu `mousedown`. Następnie, przy zwolnieniu przycisku, zgłaszają zdarzenie `mouseup`, a później — zdarzenie `click` (patrz rysunek 5.1).



Rysunek 5.1. *Choć może nie zdajesz sobie z tego sprawy, przeglądarki nieustannie zgłaszają zdarzenia w czasie wpisywania tekstu, poruszania wskaźnikiem myszy i klikania. Przykładowo dwukrotne kliknięcie myszą powoduje wygenerowanie dwóch zdarzeń mousedown i mouseup, zdarzenia dblclick, jak również zdarzenia click. Strona events.html (dostępna wśród przykładowych plików) lustruje działanie wielu zdarzeń*

Uwaga: Zrozumienie, kiedy i jak przeglądarka zgłasza zdarzenia, nie jest proste. Aby przetestować zdarzenia różnego rodzaju, możesz użyć strony demonstracyjnej, dostępnej wśród przykładowych plików. Otwórz w przeglądarce stronę *events.html* z katalogu *testy*. Następnie poruszaj wskaźnikiem myszy, kliknij dowolny element i wpisz tekst, aby zobaczyć niektóre z wielu zdarzeń nieustannie zachodzących na stronach WWW (patrz rysunek 5.1).

Zdarzenia związane z myszą

Od 1984 roku, kiedy to Steve Jobs zaprezentował komputer Macintosh, mysz stała się podstawowym urządzeniem we wszystkich komputerach osobistych (podobnie jak płytka dotykowa w laptopach). Użytkownicy korzystają z niej do uruchamiania aplikacji, przeciągania plików do katalogów, wybierania elementów z menu, a nawet do rysowania. Przeglądarki udostępniają wiele mechanizmów do śledzenia operacji wykonywanych przez użytkowników myszą w czasie korzystania ze stron. Oto one.

- **Zdarzenie `click`**. To zdarzenie jest zgłaszane po zwolnieniu przycisku myszy przy kliknięciu. Programiści często przypisują to zdarzenie do odnośników. Na przykład kliknięcie miniatury rysunku może powodować wyświetlenie jego pełnej wersji. Jednak można używać także innych elementów. Na kliknięcie może reagować każdy znacznik, a nawet cała strona.

Uwaga: Zdarzenie `click` jest uruchamiane także po wybraniu odnośnika za pomocą klawiatury. Jeśli przejdziesz do odsyłacza za pomocą klawisza *Tab*, a następnie wciśniesz klawisz *Enter*, przeglądarka zgłosi zdarzenie `click`.

- **Zdarzenie `dblclick`**. Kiedy użytkownik szybko dwukrotnie wciśnie i zwolni przycisk myszy, przeglądarka zgłosi zdarzenie dwukrotnego kliknięcia (`dblclick`). Ta operacja służy na przykład do otwierania katalogów i plików na pulpicie. Dwukrotne kliknięcie nie jest standardowym zachowaniem internauty, dlatego jeśli chcesz użyć go na stronie, powinieneś wyraźnie opisać, które elementy użytkownicy mogą kliknąć w taki sposób i do czego prowadzi ta operacja. Warto też pamiętać, że każde kliknięcie składające się na zdarzenie dwukrotnego kliknięcia generuje także zdarzenie `click`, dlatego też nie należy w tym samym znaczniku obsługiwać obu tych zdarzeń, czyli `click` i `dblclick`. Jeśli o tym zapomnisz, skrypt najpierw dwa razy uruchomi funkcję powiązaną z pojedynczym kliknięciem, a następnie wykona funkcję dla dwukrotnego kliknięcia.
- **Zdarzenie `mousedown`**. To zdarzenie odpowiada pierwszej części kliknięcia — przyciśnięciu przycisku myszy przed jego zwolnieniem. Jest ono przydatne przy przenoszeniu elementów na stronie. Można pozwolić użytkownikom na przeciąganie obiektów w podobny sposób, jak przenoszą ikony na pulpicie. Technika ta polega na kliknięciu elementu i — bez zwalniania przycisku — przeciągnięciu go, a następnie upuszczeniu w wyniku zwolnienia przycisku myszy (na stronie 421 dowiesz się, jak to zrobić przy użyciu bibliotek jQuery).
- **Zdarzenie `mouseup`**. To zdarzenie odpowiada drugiej części kliknięcia — zwolnieniu przycisku myszy. Pozwala ono wykryć moment upuszczenia przenoszonego elementu.
- **Zdarzenie `mouseover`**. Kiedy umieścisz wskaźnik myszy nad elementem strony, przeglądarka zgłosi zdarzenie `mouseover`. Przy jego użyciu można przypisać uchwyty zdarzeń do przycisków nawigacyjnych i wyświetlać menu rozwijane po umieszczeniu wskaźnika myszy nad danym przyciskiem. Jeśli używałeś kiedyś pseudoklasy `:hover` języka CSS, wiesz już, jak działa to zdarzenie.

- **Zdarzenie `mouseout`**. Przeniesienie wskaźnika myszy poza dany element wyzwala zdarzenie `mouseout`. Można go użyć do zasygnalizowania, że użytkownik przeniósł wskaźnik myszy poza stronę, lub do ukrycia menu po opuszczeniu jego obszaru przez wskaźnik.
- **Zdarzenie `mousemove`**. To zdarzenie jest wyzwalane przy każdym ruchu myszą, czyli prawie cały czas. Można go używać do sprawdzania aktualnej pozycji wskaźnika myszy na ekranie. Ponadto zdarzenie to można przypisać do określonego znacznika strony, na przykład `<div>`, i reagować tylko na ruchy myszą w obrębie danego elementu.

Uwaga: Zdarzenia `mousemove` są zgłaszane bardzo często (wiele razy w trakcie każdego przesunięcia wskaźnika myszy), dlatego obsługując to zdarzenie, należy zachować bardzo dużą ostrożność. Próby wykonywania długotrwałych operacji w ramach obsługi tego zdarzenia mogą doprowadzić do znacznego spowolnienia programu i pogorszenia szybkości reakcji strony na poczynania użytkownika.

Zdarzenia związane z dokumentem i oknem

Okno przeglądarki obsługuje liczne zdarzenia, zgłaszane od momentu wczytania strony do czasu jej zamknięcia przez użytkownika.

- **Zdarzenie `load`**. To zdarzenie jest zgłaszane po wczytaniu przez przeglądarkę wszystkich plików strony: pliku HTML, a także dołączonych rysunków, filmów we Flashu oraz zewnętrznych plików CSS i JavaScript. Projektanci stron WWW zwykli używać tego zdarzenia do uruchamiania programów JavaScript, które służą do manipulowania stroną. Jednak wczytywanie strony i wszystkich plików może trwać dość długo, jeśli trzeba pobrać wiele obrazków i dużych dokumentów zewnętrznych. Czasem oznacza to, że kod JavaScript jest uruchamiany długo po wyświetleniu strony w przeglądarce. Na szczęście jQuery udostępnia szybciej reagujący zastępnik zdarzenia `load` (patrz strona 190).
- **Zdarzenie `resize`**. Kiedy zmienisz rozmiar okna przeglądarki przez kliknięcie przycisku maksymalizacji lub przeciągnięcie krawędzi okna, przeglądarka zgłosi zdarzenie `resize`. Niektórzy projektanci używają go do modyfikowania układu strony po zmianie wielkości strony. Kiedy na przykład użytkownik zmieni rozmiar okna, warto sprawdzić jego szerokość. Jeśli jest duża, można zmodyfikować układ i zapełnić puste miejsce nowymi kolumnami.
- **Zdarzenie `scroll`**. To zdarzenie jest zgłaszane po przeciągnięciu suwaka albo użyciu klawiatury (strzałek w górę i dół, klawiszy *Home* lub *End* i tak dalej) lub rolki myszy do przewijania strony. Jeśli strona nie udostępnia pasków przewijania, zdarzenie to nie zachodzi. Niektórzy programiści używają go do wykrywania, które elementy są widoczne na stronie po jej przewinięciu.

Uwaga: Podobnie jak zdarzenie `mousemove` (przedstawione na stronie 180), także i zdarzenie `scroll` jest podczas przewijania strony zgłaszane bardzo często. Dlatego też i w jego przypadku należy zachować dużą ostrożność, jeśli obsługa tych zdarzeń wymaga złożonych operacji.

- **Zdarzenie `unload`**. Kiedy klikniesz odnośnik, aby przejść do nowej strony, albo zamkniesz kartę lub okno przeglądarki, zgłoszone zostanie zdarzenie `unload`. Jest to ostatnia informacja dla skryptu JavaScript, która umożliwia wykonanie końcowych operacji przed opuszczeniem strony przez użytkownika. Złośliwi programiści używali go do utrudniania zamykania strony. Każda próba opuszczenia strony prowadziła do pojawienia się jej w nowym oknie. Jednak tego zdarzenia można używać także w wartościowy sposób. Przykładowo program może ostrzegać użytkownika o tym, że nie wysłał częściowo uzupełnionego formularza, lub przesyłać dane z formularza na serwer w celu ich zapisania.

Zdarzenia związane z formularzami

Przed pojawieniem się języka JavaScript użytkownicy wchodzili w interakcje ze stronami głównie za pomocą klikania odnośników i wypełniania pól w formularzach HTML. Wprowadzenie informacji w polach formularza było jedynym sposobem na przesłanie danych do witryny. Ponieważ formularze wciąż są ważnym elementem sieci WWW, dostępnych jest wiele związanych z nimi zdarzeń.

- **Zdarzenie `submit`**. To zdarzenie jest zgłaszane przy przesyłaniu formularza. Użytkownik może to zrobić przez kliknięcie przycisku *Wyślij* lub wciśnięcie klawisza *Enter*, kiedy kursor znajduje się w polu tekstowym. Zdarzenie `submit` najczęściej używane jest do walidacji formularzy. Proces ten pozwala *przed* wysłaniem danych na serwer sprawdzić, czy wszystkie wymagane pola są prawidłowo wypełnione. Na stronie 299 dowiesz się, jak przeprowadzić walidację formularza.
- **Zdarzenie `reset`**. Choć przycisk *Wyczyść* obecnie nie jest tak popularny jak kiedyś, umożliwia anulowanie wszystkich zmian wprowadzonych w formularzu i przywrócenie jego wyjściowego stanu. Kiedy użytkownik spróbuje wyczyścić zawartość formularza, można uruchomić skrypt w odpowiedzi na wykrycie zdarzenia `reset`. Program ten powinien wyświetlać okno dialogowe z tekstem typu: „Czy na pewno chcesz usunąć zmiany?". Okno to powinno zawierać przycisk *Nie*, który umożliwia zrezygnowanie z usuwania danych z formularza.
- **Zdarzenie `change`**. Wiele pól formularza zgłasza zdarzenie `change` przy zmianie stanu, na przykład w wyniku kliknięcia przycisku opcji lub wybrania odnośnika z menu rozwijanego. Zdarzenie to umożliwia natychmiastowe sprawdzenie wybranego odsyłacza lub zaznaczonego przycisku.
- **Zdarzenie `focus`**. Kiedy klikniesz pole tekstowe lub przejdziesz do niego za pomocą klawisza *Tab*, *aktywujesz* je. Oznacza to, że przeglądarka „skoncentruje uwagę" na tym polu. Także zaznaczenie przycisku opcji lub kliknięcie pola wyboru powoduje ich aktywowanie i umożliwia zareagowanie w kodzie JavaScript na zgłoszenie zdarzenia `focus`. Załóżmy, że w polu tekstowym znajdują się pomocne instrukcje, na przykład „Wpisz imię i nazwisko". Kiedy użytkownik kliknie pole (aktywuje je), warto usunąć instrukcje, aby internauta mógł wprowadzić dane w pustym obszarze.
- **Zdarzenie `blur`**. Jest to przeciwieństwo zdarzenia `focus`. Przeglądarka zgłasza zdarzenie `blur`, kiedy użytkownik opuści aktywne pole za pomocą klawisza *Tab* lub kliknięcia myszą. Także to zdarzenie jest przydatne przy walidacji formularzy.

Kiedy użytkownik wpisze adres e-mail w polu tekstowym, a następnie przejdzie do następnego elementu, można natychmiast sprawdzić, czy wprowadzone dane to poprawny adres.

Uwaga: Zdarzenia `focus` i `blur` działają także dla odnośników. Kiedy wybierzesz odsyłacz za pomocą klawisza *Tab*, przeglądarka zgłosi zdarzenie `focus`. Kiedy ponownie wciśniesz ten klawisz (lub klikniesz myszą inny element), zajdzie zdarzenie `blur`.

Zdarzenia związane z klawiaturą

Przeglądarki śledzą też operacje wykonywane za pomocą klawiatury, dlatego można przypisać do poszczególnych klawiszy polecenia lub umożliwić użytkownikom kontrolowanie skryptów przy użyciu różnych znaków. Na przykład wciśnięcie klawisza spacji może uruchamiać i zatrzymywać animację wyświetlaną przez kod JavaScript.

Niestety, poszczególne przeglądarki obsługują zdarzenia związane z klawiaturą w odmienny sposób, dlatego trudno nawet ustalić, który klawisz wcisnął użytkownik! Opis jednej z technik sprawdzania użytych klawiszy znajdziesz we wskazówce na stronie 195.

- **Zdarzenie `keypress`**. To zdarzenie jest zgłaszane w momencie wciśnięcia klawisza. Wcale nie trzeba później zwalniać klawisza — zdarzenie to jest zgłaszane bezustannie tak długo, jak długo użytkownik trzyma klawisz wciśnięty. Z tego względu kontrola tego zdarzenia jest dobrym sposobem, aby się przekonać, czy użytkownik wciąż naciska klawisz. Jeśli na przykład będziemy opracowywać wyścigi samochodowe w przeglądarce, można przypisać jakiś klawisz pedałowi gazu. W takim przypadku wystarczy, że użytkownik wciśnie klawisz, a samochód będzie jechał.
- **Zdarzenie `keydown`**. To zdarzenie działa podobnie jak zdarzenie `keypress` — jest zgłaszane przy wciśnięciu klawisza. Zachodzi tuż *przed* zdarzeniem `keypress`. W Operze zdarzenie to jest zgłaszane tylko raz. W pozostałych przeglądarkach działa tak samo jak zdarzenie `keypress` — jest zgłaszane, dopóki użytkownik nie zwolni klawisza.
- **Zdarzenie `keyup`**. To zdarzenie przeglądarka zgłasza w momencie zwolnienia klawisza.

Obsługa zdarzeń przy użyciu jQuery

Programowa obsługa zdarzeń zawsze była trudna. Przez wiele lat zdarzenia w przeglądarce Internet Explorer były obsługiwane zupełnie inaczej niż we wszystkich innych przeglądarkach, co zmuszało programistów do tworzenia dwóch odrębnych wersji kodu (dla IE oraz dla pozostałych przeglądarek), by zapewnić prawidłowe działanie strony. Na szczęście w najnowszej wersji programu — w Internet Explorerze 9 — wykorzystywana jest już ta sama metoda obsługi zdarzeń, co w innych przeglądarkach, dzięki czemu programowanie stało się znacznie prostsze. Jednak wciąż sporo osób używa przeglądarki IE8, dlatego potrzebne jest jakieś dobre rozwiązanie, które

mogłoby ułatwić obsługę zdarzeń i zagwarantować ich spójną obsługę w wielu różnych przeglądarkach. Właśnie takim rozwiązaniem jest jQuery.

W poprzednim rozdziale dowiedziałeś się, że biblioteki języka JavaScript, na przykład jQuery, rozwiązują wiele problemów programistycznych. Między innymi pozwalają zapomnieć o niezgodnościach między przeglądarkami. Ponadto biblioteki często upraszczają podstawowe zadania związane z językiem JavaScript. JQuery sprawia, że przypisywanie zdarzeń i *funkcji obsługujących zdarzenia* jest banalnie proste.

Jak mogłeś się przekonać na stronie 142, korzystanie z biblioteki jQuery polega na wybraniu odpowiedniego elementu strony oraz wykonaniu na nim pewnej operacji. Ponieważ zdarzenia stanowią tak ważny i integralny element programowania w języku JavaScript, tworzenie skryptów wykorzystujących jQuery lepiej wyobrazić sobie jako proces trójetapowy.

1. **Pobieranie elementów strony.**

   W poprzednim rozdziale dowiedziałeś się, jak za pomocą jQuery i selektorów CSS pobrać elementy strony, którymi chcesz manipulować. Przy przypisywaniu zdarzeń potrzebne są znaczniki, z którymi użytkownik będzie wchodził w interakcję. Jakie elementy internauci będą klikać — odnośniki, komórki tabeli, a może rysunki? Nad którą częścią strony użytkownik ma umieścić wskaźnik myszy, aby uruchomić określony kod?

2. **Przypisywanie zdarzenia.**

   W jQuery większości zdarzeń modelu DOM odpowiadają funkcje biblioteczne o takiej samej nazwie. Dlatego aby przypisać zdarzenie do elementu, wystarczy dodać kropkę, nazwę zdarzenia i parę nawiasów. Jeśli na przykład chcesz dodać zdarzenie `mouseover` do każdego odnośnika na stronie, możesz to zrobić w następujący sposób:

   ```
   $('a').mouseover();
   ```

   Poniższy kod dodaje zdarzenie `click` do elementu o identyfikatorze `menu`:

   ```
   $('#menu').click();
   ```

   W ten sposób można używać wszystkich zdarzeń opisanych na stronach od 179 do 182. Ponadto dostępnych jest kilka zdarzeń specyficznych dla biblioteki jQuery, omówionych na stronie 192.

   Jednak dodanie zdarzenia to nie koniec pracy. Aby strona reagowała na zgłoszenie zdarzenia, trzeba przypisać do niego funkcję.

3. **Przekazywanie funkcji do zdarzenia.**

   W ostatnim kroku trzeba określić, co się stanie po zgłoszeniu zdarzenia. W tym celu należy przekazać do zdarzenia funkcję zawierającą polecenia wykonywane przy jego wystąpieniu. Może ona na przykład wyświetlać ukryty znacznik `<div>` lub wyróżniać element, nad którym użytkownik umieścił wskaźnik myszy.

   Do zdarzenia można przekazać nazwę wcześniej zdefiniowanej funkcji:

   ```
   $('#start').click(startSlideShow);
   ```

   Podczas przypisywania funkcji do zdarzeń należy pominąć nawiasy, standardowo umieszczane po nazwie funkcji w celu jej wywołania. Oznacza to, że poniższy zapis jest nieprawidłowy:

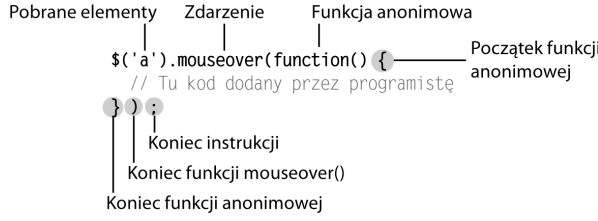
   ```
   $('#start').click(startSlideShow());
   ```

Jednak najczęściej używanym sposobem obsługi zdarzeń jest przypisywanie do nich *funkcji anonimowych*. Funkcje anonimowe poznałeś na stronie 168 — najprościej rzecz ujmując, można stwierdzić, że są to funkcje bez nazwy. Podstawowa struktura funkcji anonimowej wygląda następująco:

```
function() {
  // Tu kod funkcji.
}
```

Sposób przypisywania funkcji anonimowych do zdarzeń ilustruje rysunek 5.2.

Uwaga: Aby dowiedzieć się więcej o korzystaniu ze zdarzeń za pomocą biblioteki jQuery, odwiedź stronę *http://api.jquery.com/category/events/*.



Rysunek 5.2. *W jQuery zdarzenia działają jak funkcje, dlatego możesz przekazywać do nich argumenty. Funkcje anonimowe możesz traktować jak zwykłe argumenty, podobne do pojedynczych danych. Z tej perspektywy łatwiej zrozumieć działanie wszystkich znaków specjalnych. Na przykład w ostatnim wierszu symbol } oznacza koniec funkcji anonimowej (czyli argumentu przekazanego do funkcji mouseover), znak) kończy funkcję mouseover(), a średnik to ostatni symbol całej instrukcji rozpoczynającej się od selektora $('a')*

Oto prosty przykład. Załóżmy, że na stronie znajduje się odnośnik o identyfikatorze `menu`. Kiedy użytkownik umieści wskaźnik myszy nad tym odnośnikiem, skrypt ma wyświetlać listę dodatkowych odsyłaczy (lista ta jest zapisana w znaczniku o identyfikatorze `submenu`). Dlatego należy dodać do odnośnika `menu` zdarzenie `mouseover`, a następnie wywołać funkcję wyświetlającą znacznik `submenu`. Proces ten można podzielić na cztery kroki.

1. **Pobieranie odnośnika `menu`:**

   ```
   $('#menu')
   ```

2. **Dołączanie zdarzenia:**

   ```
   $('#menu').mouseover();
   ```

3. **Dodawanie funkcji anonimowej:**

   ```
   $('#menu').mouseover(function() {

   }); // Koniec funkcji mouseover.
   ```

   Często można zobaczyć kolekcje zamykających nawiasów klamrowych, zwyczajnych nawiasów i średników — `});` — które zazwyczaj reprezentują koniec funkcji anonimowej umieszczonej wewnątrz wywołania funkcji. Ponieważ występują stosunkowo często, dobrym pomysłem jest dodawanie do kodu komentarzy — w tym przypadku jest to `// koniec mouseover` — informujących o tym, które wywołanie kończy dana sekwencja znaków.

4. **Dodawanie potrzebnych operacji (tu jest to wyświetlanie listy submenu):**

```
$('#menu').mouseover(function() {
  $('#submenu').show();
});   // Koniec funkcji mouseover.
```

Wiele osób uważa zbitek znaków specjalnych potrzebny przy używaniu funkcji anonimowych za mało zrozumiały (sekwencja `});` na końcu instrukcji). Ten fragment *jest* złożony, jednak najlepszy sposób na przyzwyczajenie się do dziwnego świata języka JavaScript polega na ćwiczeniu, dlatego następny praktyczny przykład pomoże Ci utrwalić zdobytą wiedzę.

Uwaga: Funkcję `show()` opisano na stronie 212.

Przykład — prezentacja obsługi zdarzeń

W tym samouczku przedstawione zostało krótkie wprowadzenie do zagadnień obsługi zdarzeń. Utworzysz w nim stronę reagującą na kilka różnych typów zdarzeń; będziesz miał okazję zobaczyć, w jaki sposób obsługuje się zdarzenia przy użyciu biblioteki jQuery.

Uwaga: Na stronie 46 znajdziesz informacje o tym, skąd można pobrać gotową wersję przykładu.

1. **W edytorze otwórz plik *events_intro.html*, znajdujący się w katalogu *R05*.**

Zaczniemy do samego początku, czyli od dodania odwołania do biblioteki jQuery.

2. **Kliknij pusty wiersz, tuż powyżej zamykającego znacznika `</head>`, i wpisz w nim:**

```
<script src="../_js/jquery.min.js"></script>
```

Znacznik ten spowoduje wczytanie biblioteki jQuery z pliku dostępnego na tej samej witrynie. Zwróć uwagę, że katalog, w którym jest przechowywana biblioteka, nosi nazwę *_js* (nie zapomnij o znaku podkreślenia na samym początku). Teraz dodasz kolejny znacznik `<script>`, w którym umieścisz kod swojego skryptu.

3. **Naciśnij klawisz *Enter*, aby utworzyć nowy, pusty wiersz i dodaj do pliku poniższy fragment kodu, wyróżniony pogrubioną czcionką:**

```
<script src="../_js/jquery.min.js"></script>
<script>

</script>
```

A teraz dodaj wywołanie funkcji `document.ready()`.

4. **Kliknij pusty wiersz pomiędzy otwierającym i zamykającym znacznikiem `<script>`, a następnie wpisz poniższy kod wyróżniony pogrubioną czcionką:**

```
<script src="../_js/jquery.min.js"></script>
<script>
$(document).ready(function() {

}); // Koniec funkcji ready.
</script>
```

Nie zapomnij o dodaniu komentarza JavaScript za sekwencją znaków `});`. Choć dodawanie komentarzy wymaga dodatkowego pisania, później są one niezwykle pomocne przy identyfikowaniu różnych części programu. W ten sposób wykonałeś czynności, które będziesz powtarzał na wszystkich stronach wykorzystujących bibliotekę jQuery.

Teraz nadszedł czas, by dodać zdarzenie. Twoje pierwsze zadanie jest całkiem proste: masz wyświetlać okienko dialogowe, za każdym razem gdy użytkownik dwukrotnie kliknie w dowolnym miejscu strony. Na początek musisz wybrać element (w tym przypadku będzie nim cała strona), do którego chcesz dodać zdarzenie.

5. **Kliknij pusty wiersz wewnątrz funkcji `ready()` i wpisz poniższy kod wyróżniony pogrubioną czcionką:**

```
<script src="../_js/jquery.min.js"></script>
<script>
$(document).ready(function() {
  $('html')
}); // Koniec funkcji ready.
</script>
```

Wywołanie `$('html')` wybiera element HTML, czyli całe okno przeglądarki. Kolejną czynnością będzie dodanie zdarzenia.

6. **Za selektorem jQuery wpisz `.dblclick();`, by kod wyglądał tak, jak na poniższym przykładzie:**

```
<script src="../_js/jquery.min.js"></script>
<script>
$(document).ready(function() {
  $('html').dblclick(); // Koniec funkcji dblclick.
}); // Koniec funkcji ready.
</script>
```

Funkcja `.dblclick()` jest funkcją jQuery, która sprawia, że przeglądarka będzie gotowa do wykonania konkretnych czynności w momencie, gdy użytkownik dwukrotnie kliknie na stronie. Jedynym brakującym elementem są te „czynności", a określenie ich wymaga przekazania funkcji anonimowej jako argumentu w wywołaniu funkcji `dblclick()` (jeśli musisz przypomnieć sobie, jak działają funkcje oraz czym jest „przekazywanie argumentów", informacje na ten temat znajdziesz na stronie 115).

7. **Teraz dodaj funkcję anonimową, czyli wpisz poniższy kod wyróżniony pogrubioną czcionką:**

```
<script src="../_js/jquery.min.js"></script>
<script>
$(document).ready(function() {
  $('html').dblclick(function(){

  }); // Koniec funkcji dblclick.
}); // Koniec funkcji ready.
</script>
```

Nie przejmuj się, w dalszej części książki zagadnienia w przykładach nie będą opisywane w takim żółwim tempie. Jednak teraz bardzo ważne jest, byś dobrze zrozumiał, jaką rolę pełni każdy element kodu. Funkcja anonimowa `function(){}` jest jedynie zewnętrznym pojemnikiem, nie da żadnego efektu, dopóki nie umieścisz jakiegoś kodu wewnątrz nawiasów klamrowych `{ }`. A tym zajmiemy się w kolejnym kroku.

8. **Teraz dodaj funkcję `alert()`:**

```
<script src="../_js/jquery.min.js"></script>
<script>
$(document).ready(function() {
  $('html').dblclick(function(){
    alert('ała!');
  }); // Koniec funkcji dblclick.
}); // Koniec funkcji ready.
</script>
```

Jeśli wyświetlisz stronę w przeglądarce i dwukrotnie klikniesz w dowolnym jej miejscu, zostanie wyświetlone niewielkie okienko informacyjne JavaScriptu z komunikatem `ała!`. Gdyby się nie pojawił, powinieneś dokładnie sprawdzić cały wpisany kod i upewnić się, że niczego nie pominąłeś.

Uwaga: Po wykonaniu tych wszystkich przydługich czynności przygotowawczych wyświetlenie prostego komunikatu „ała!" może trochę rozczarować. Musisz jednak pamiętać, że samo wywołanie funkcji `alert()`, zastosowane w powyższym przykładzie, nie ma żadnego znaczenia — kluczowy jest cały pozostały kod, stanowiący podstawę do obsługi zdarzeń przy użyciu biblioteki jQuery. Kiedy już dowiesz się nieco więcej na temat korzystania z jQuery, bez najmniejszych problemów będziesz mógł zastąpić wywołanie funkcji `alert()` sekwencją innych wywołań, które (w odpowiedzi na podwójne kliknięcie strony) będą przesuwać elementy wyświetlane na stronie, wyświetlać interaktywny pokaz slajdów bądź uruchamiać napisaną w JavaScripcie grę wyścigową.

A teraz, kiedy poznałeś już podstawy, spróbujemy obsłużyć kilka innych zdarzeń.

1. **Dodaj fragment kodu wyróżniony pogrubioną czcionką, by skrypt wyglądał tak, jak na poniższym przykładzie:**

```
<script src="../_js/jquery.min.js"></script>
<script>
$(document).ready(function() {
  $('html').dblclick(function(){
    alert('ała!');
  }); // Koniec funkcji dblclick.
  $('a').mouseover(function(){

  }); // Koniec funkcji mouseover.
}); // Koniec funkcji ready.
</script>
```

Ten nowy fragment kodu wybiera wszystkie odnośniki na stronie (odpowiada za to wywołanie `$('a')`) i dodaje do nich funkcję anonimową, która będzie obsługiwać zdarzenia `mouseover`. Innymi słowy, coś się stanie, kiedy użytkownik umieści wskaźnik myszy w obszarze odnośnika.

2. **Do anonimowej funkcji utworzonej w poprzednim kroku dodaj dwie instrukcje JavaScriptu:**

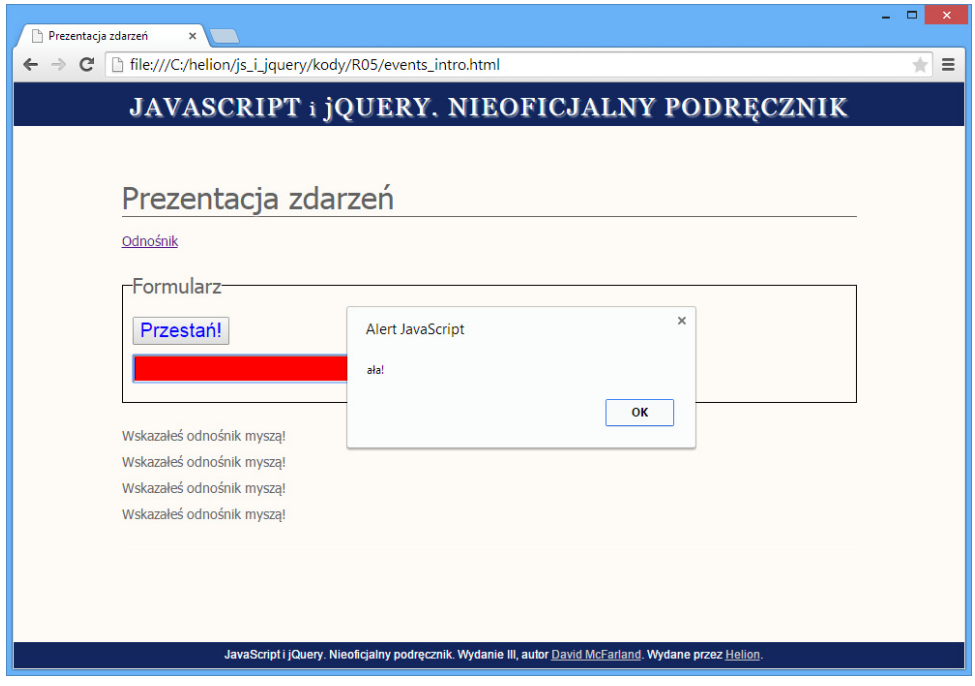
```
<script src="../_js/jquery.min.js"></script>
<script>
$(document).ready(function() {
  $('html').dblclick(function(){
    alert('ała!');
  }); // Koniec funkcji dblclick.
  $('a').mouseover(function(){
    var message = "<p>Wskazałeś odnośnik myszą!</p>";
```

```
        $('.main').append(message);
    }); // Koniec funkcji mouseover.
}); // Koniec funkcji ready.
</script>
```

Pierwszy wiersz kodu — `var message = <p>Wskazałeś odnośnik myszą!</p> ;` — definiuje nową zmienną o nazwie `message` i zapisuje w niej łańcuch znaków. Łańcuch ten zawiera znacznik akapitu i tekst umieszczony wewnątrz niego. Drugi wiersz wybiera określony element strony należący do klasy `main` (odpowiada za to wywołanie `$('.main')`), a następnie na jego końcu dodaje zawartość zmiennej `message`. Nasza przykładowa strona zawiera znacznik `<div>` należący do klasy `main`, a zatem wywołanie doda na końcu jego zawartości akapit z tekstem „Wskazałeś odnośnik myszą!" za każdym razem, gdy użytkownik umieści wskaźnik myszy w obszarze jakiegoś odnośnika. (Informacje na temat działania funkcji `append()` jQuery możesz znaleźć na stronie 158).

3. **Zapisz stronę, wyświetl ją w przeglądarce i spróbuj przesunąć wskaźnik muszy nad dowolnym z odnośników widocznych na stronie.**

   Zawsze wtedy, gdy umieścisz wskaźnik myszy w obszarze jakiegoś odnośnika, do strony zostanie dodany nowy akapit (patrz rysunek 5.3). Teraz dodasz do strony ostatni fragment kodu — kiedy użytkownik kliknie przycisk formularza, przeglądarka zmieni tekst wyświetlany na tym przycisku.



Rysunek 5.3. *Biblioteka jQuery znacznie ułatwia reagowanie na czynności wykonywane przez użytkownika na stronie, na przykład poprzez wyświetlenie okienka informacyjnego w odpowiedzi na dwukrotne kliknięcie strony, wyświetlenie tekstu po wskazaniu odnośnika myszą lub kliknięcie przycisku formularza*

4. **I w końcu dodaj poniższy kod wyróżniony pogrubioną czcionką, żeby skrypt wyglądał tak, jak na poniższym przykładzie:**

```
<script src="../_js/jquery.min.js"></script>
<script>
$(document).ready(function() {
  $('html').dblclick(function(){
    alert('ała!');
  }); // Koniec funkcji dblclick.
  $('a').mouseover(function(){
    var message = "<p>Wskazałeś odnośnik myszą!</p>";
    $('.main').append(message);
  }); // Koniec funkcji mouseover.
  $('#button').click(function(){
    $(this).val("Przestań!");
  }); // Koniec funkcji click.
}); // Koniec funkcji ready.
</script>
```

Oto podstawowy cel zastosowania powyższego fragmentu kodu: `$('#button')` wybiera element o identyfikatorze `button` (w naszym przypadku jest to przycisk formularza) i dodaje do niego funkcję obsługi zdarzenia `click`; kiedy zatem użytkownik kliknie przycisk, coś się stanie. W naszym przykładzie, w odpowiedzi na kliknięcie tekst wyświetlony na przycisku zostanie zmieniony na „Przestań!".

Na stronie 169 dowiedziałeś się, jak można używać wywołania w postaci `$(this)` wewnątrz pętli. Dokładnie w taki sam sposób działa ono wewnątrz funkcji obsługi zdarzenia: w tym przypadku `$(this)` odnosi się do elementu odpowiadającego na zdarzenie — czyli elementu, który wcześniej został wybrany i z którym skojarzyliśmy funkcję obsługi zdarzenia. W naszym przykładzie jest to przycisk formularza. (Więcej informacji na temat funkcji `val()` biblioteki jQuery znajdziesz na stronie 281, jednak najprościej rzecz ujmując, służy ona do odczytywania lub podawania wartości elementu. W naszym przykładzie przekazanie w wywołaniu funkcji `val()` łańcucha znaków `Przestań!` spowoduje zastosowanie go jako wartości przycisku).

5. **Ponownie zapisz stronę, wyświetl ją w przeglądarce i kliknij przycisk formularza.**

Tekst na przycisku powinien się natychmiast zmienić (patrz rysunek 5.3). W ramach dodatkowego ćwiczenia napisz fragment kodu, który sprawi, że kolor tła pola tekstowego zmieni się na czerwony, kiedy użytkownik je kliknie lub przejdzie do niego, używając klawisza *Tab*. A oto drobna podpowiedź: w tym celu musisz (a) wybrać pole tekstowe, (b) skorzystać z funkcji `focus()` (patrz strona 286), (c) użyć wywołania w postaci `$(this)` (tego samego, z którego korzystałeś w kroku 12.), by odwołać się do pola tekstowego wewnątrz funkcji anonimowej obsługującej zdarzenie oraz (d) skorzystać z funkcji `css()` jQuery (opisanej na stronie 163), by zmienić kolor tła pola. Odpowiedź (jak również kompletną wersję strony) możesz znaleźć w pliku *complete_events_intro.html* w katalogu *R05*.

Zdarzenia specyficzne dla biblioteki jQuery

Ponieważ zdarzenia są kluczowe przy tworzeniu interaktywnych stron WWW, jQuery zawiera kilka specyficznych funkcji, które ułatwiają programowanie i zwiększają interaktywność witryn.

Oczekiwanie na wczytanie kodu HTML

W czasie wczytywania strony przeglądarka próbuje natychmiast uruchomić napotkane skrypty. Dlatego kod umieszczony w sekcji nagłówkowej może zostać wykonany przed wyświetleniem strony. Taką sytuację widziałeś już w przykładzie przedstawionym na stronie 124, w którym strona WWW była pusta aż do momentu zakończenia działania skryptu zadającego pytania. Niestety, to zjawisko często powoduje problemy. Ponieważ wiele programów JavaScript manipuluje zawartością stron (wyświetla informacje po kliknięciu odnośnika, ukrywa elementy dokumentu, dodaje paski do wierszy tabeli i tak dalej), próba zmiany kodu strony przed jej wczytaniem i wyświetleniem w przeglądarce może prowadzić do błędów.

Najczęściej stosowanym rozwiązaniem tego problemu jest użycie zdarzenia `load`. Pozwala ono odroczyć uruchomienie kodu JavaScript do momentu wczytania i wyświetlenia całej strony. Niestety, oczekiwanie na uruchomienie kodu JavaScript do momentu wczytania całej strony prowadzi nieraz do dziwnych efektów. Przeglądarka zgłasza zdarzenie `load` dopiero *po* pobraniu wszystkich plików strony, w tym rysunków, filmów, zewnętrznych arkuszy stylów i tak dalej. Dlatego użytkownicy stron bogatych w grafikę mogą przez kilka sekund czekać na wczytanie rysunków *przed* uruchomieniem kodu JavaScript. Jeśli ten kod wprowadza wiele zmian na stronie (na przykład nadaje styl wierszom tabeli, ukrywa widoczne menu, a nawet zarządza układem elementów), strona będzie zmieniać się na oczach odwiedzających.

Na szczęście jQuery pozwala rozwiązać ten problem. Zamiast używać zdarzenia `load` do uruchamiania programów JavaScript, jQuery udostępnia funkcję `ready()`, która czeka na pobranie samego kodu HTML, a następnie wykonuje skrypty. To podejście umożliwia natychmiastowe manipulowanie stroną bez konieczności oczekiwania na wolno wczytujące się rysunki i filmy. To rozwiązanie jest skomplikowane, ale użyteczne, dlatego stanowi następny powód do używania bibliotek języka JavaScript.

Z funkcją `ready()` zetknąłeś się już w kilku przykładach w tej książce. Jej podstawowa struktura wygląda następująco:

```
$(document).ready(function() {
  // Tu uruchamiany kod.
});
```

Cały kod programu należy umieścić w tej funkcji. Ponieważ funkcja `ready()` jest tak ważna, prawdopodobnie będziesz używał jej na każdej stronie, na której będziesz korzystał z jQuery. Wystarczy użyć jej jeden raz. Zwykle funkcja ta jest pierwszym i ostatnim wierszem skryptu. Należy umieścić ją między otwierającym a zamykającym znacznikiem `<script>` (w końcu jest to kod JavaScript) i po elemencie `<script>... </script>` dołączającym do strony bibliotekę jQuery.

Dlatego w kontekście kompletnej strony WWW funkcja ta powinna wyglądać następująco:

```
<!DOCTYPE html>
<html>
<head>
<meta charset="UTF-8">
<title>Tytuł strony</title>
<script src="js/jquery.js"></script>
<script>
$(document).ready(function() {
    // Tu należy umieścić cały kod JavaScript.
}); // Koniec funkcji ready.
</script>
</head>
<body>
Zawartość strony...
</body>
</html>
```

Wskazówka: Ponieważ funkcja `ready()` znajduje się na prawie wszystkich stronach, w których kodzie używana jest biblioteka jQuery, dostępny jest skrótowy zapis. Możesz pominąć fragment `$(document)` i użyć poniższego kodu:

```
$(function() {
  // Tu można wykonać operacje na wczytanym dokumencie.

});
```

Alternatywa dla funkcji $(document).ready()

Umieszczanie funkcji `$(document).ready()` w sekcji nagłówka strony ma na celu opóźnienie wykonania kodu JavaScript aż do momentu, kiedy cała strona będzie już pobrana i wyświetlona. Jednak dokładnie ten sam efekt można uzyskać także w inny sposób: wystarczy umieścić kod JavaScript za kodem HTML. Przykładowo wielu twórców stron umieszcza swój kod JavaScript bezpośrednio przed zamykającym znacznikiem `</body>`, tak jak pokazano na poniższym przykładzie:

```
<!DOCTYPE html>
<html>
<head>
<meta charset="UTF-8">
<title>Tytuł strony</title>
</head>
<body>
Zawartość strony...
<script src="js/jquery.js"></script>
<script>
    // Tu można umieścić cały kod JavaScript.
</script>
</body>
</html>
```

W tym przypadku stosowanie funkcji `$(document).ready()` nie jest konieczne, gdyż w chwili wczytania skryptu dokument już będzie gotowy. Takie rozwiązanie może mieć spore zalety. Przede wszystkim nie trzeba wpisywać dodatkowego kodu, czyli wywołania funkcji `.ready()`. Poza tym wczytywanie i uruchamianie kodu JavaScript wstrzymuje działanie przeglądarki aż do momentu, gdy zostanie on wczytany i wykonany. Jeśli do strony dołączonych zostanie wiele zewnętrznych

plików JavaScript, których wczytywanie zajmuje sporo czasu, może ona działać nieprawidłowo. W takim przypadku osoby, które z niej korzystają, mogą odnieść wrażenie, że strona długo się wczytuje.

Na wielu blogach poświęconych projektowaniu i tworzeniu stron można wyczytać, że umieszczanie skryptów na końcu dokumentów HTML jest prawidłowym rozwiązaniem. Jednak także ono ma swoje wady. Czasami używany kod JavaScript wprowadza ogromne zmiany w wyglądzie strony. Może on na przykład całkowicie modyfikować złożoną tabelę HTML, poprawiając jej przejrzystość i ułatwiając nawigację. Ewentualnie skrypt może zmieniać prostą typografię stosowaną na stronie i przekształcać ją na coś fantastycznego (*http://letteringjs.com/*).

W takim przypadku, jeśli będziemy czekać, aż kod HTML zostanie wczytany i wyświetlony, zanim zostanie wczytana biblioteka jQuery i wykonany kod JavaScript, użytkownik najpierw zobaczy stronę w jej początkowej (nieprzetworzonej przez skrypty) postaci, a następnie, na jego oczach, zmieni ona wygląd. Taka „nagła zmiana wyglądu" może być niepokojąca. Co więcej, jeśli tworzymy aplikację internetową, która nie może działać bez wykorzystania kodu JavaScript, to wyświetlanie kodu HTML, a następnie zmuszanie użytkownika do czekania na wczytanie kodu JavaScript nie ma większego sensu — w końcu przyciski, widżety oraz wszelkie inne elementy interfejsu użytkownika obsługiwane przez skrypty będą jedynie bezużytecznymi fragmentami kodu HTML aż do momentu „ożywienia ich" przez JavaScript.

A zatem odpowiedź na pytanie, gdzie umieścić kod JavaScript, brzmi: „To zależy". Czasami strona będzie reagować sprawniej, kiedy kod JavaScript zostanie umieszczony za kodem HTML, a czasami, kiedy umieścimy go w sekcji nagłówka strony. Na szczęście dzięki temu, że przeglądarki korzystają z pamięci podręcznej, kiedy strona z naszej witryny wczyta już cały niezbędny kod JavaScript, pozostałe strony będą miały do niego błyskawiczny dostęp i nie będą musiały tracić czasu na jego ponowne pobieranie. Innymi słowy, nie ma powodu, żeby się tym specjalnie przejmować: jeśli uznamy, że nasza strona nie wyświetla się dostatecznie szybko, możemy spróbować przenieść skrypty na jej koniec. Jeśli to pomoże, dobrze. Jednak w wielu przypadkach to, czy użyjesz funkcji `.ready()` na początku strony, czy zrezygnujesz z niej i umieścisz kod na końcu strony, nie ma żadnego znaczenia.

Uwaga: Tworząc strony na własnym komputerze i wyświetlając je bezpośrednio w przeglądarce, nie zauważymy opisanych wcześniej problemów. Takie problemy związane z czasem pobierania i wyświetlania stron można zaobserwować wyłącznie wtedy, gdy witryna będzie umieszczona na serwerze WWW, a strony i wszystkie dodatkowe pliki będą pobierane przez stosunkowo wolne łącze.

Umieszczanie i usuwanie wskaźnika myszy z elementu

Zdarzenia `mousover` i `mouseout` często są używane razem. Kiedy na przykład użytkownik umieści wskaźnik nad przyciskiem, pojawia się menu, a gdy wskaźnik znajdzie się poza elementem, menu znika. Ponieważ łączenie tych dwóch zdarzeń jest tak powszechne, jQuery udostępnia skrótowy zapis uwzględniający oba. Funkcja `hover()` biblioteki jQuery działa tak, jak każde inne zdarzenie, jednak

jako argument przyjmuje dwie funkcje zamiast jednej. Pierwsza funkcja jest uruchamiana po umieszczeniu wskaźnika myszy nad elementem, a druga — kiedy użytkownik przeniesie wskaźnik myszy w inne miejsce. Podstawowa struktura tej funkcji wygląda następująco:

```
$('#selektor').hover(funkcja1, funkcja2);
```

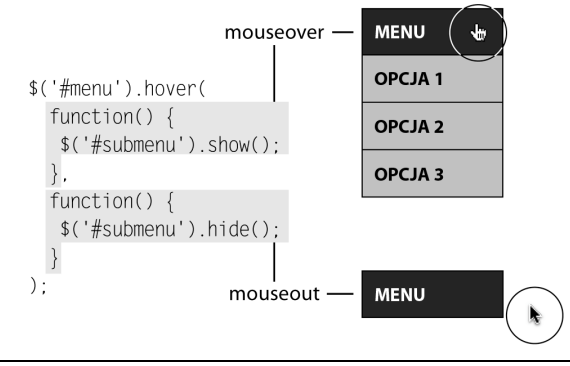
Często napotkasz funkcję `hover()`, używaną wraz z dwiema funkcjami anonimowymi. Taki kod wygląda dziwnie, jednak następny przykład powinien ułatwić jego zrozumienie. Załóżmy, że po najechaniu wskaźnikiem myszy na odnośnik o identyfikatorze `menu` skrypt ma wyświetlać niewidoczny początkowo znacznik `DIV` o identyfikatorze `submenu`. Przeniesienie wskaźnika myszy w inne miejsce ma powodować ponowne ukrycie znacznika. Ten efekt można uzyskać za pomocą funkcji `hover()`:

```
$('#menu').hover(function() {
  $('#submenu').show();
}, function() {
  $('#submenu').hide();
}); // Koniec funkcji hover.
```

Aby poprawić czytelność instrukcji z wieloma funkcjami anonimowymi, warto umieścić każdą funkcję w odrębnym wierszu. Oto nieco bardziej przejrzysta forma tego samego kodu:

```
$('#menu').hover(
  function() {
    $('#submenu').show();
  }, // Koniec funkcji mouseover.
  function() {
    $('#submenu').hide();
  } // Koniec funkcji mouseout.
); // Koniec funkcji hover.
```

Na rysunku 5.4 przedstawiono działanie tego kodu przy wystąpieniu zdarzeń `mouseover` i `mouseout`.



Rysunek 5.4. *Funkcja hover() biblioteki jQuery przyjmuje jako argumenty dwie funkcje. Pierwsza jest uruchamiana po umieszczeniu wskaźnika nad elementem, a druga — kiedy użytkownik przeniesie wskaźnik w inne miejsce*

Jeśli uważasz, że metoda oparta na funkcjach anonimowych jest zbyt skomplikowana, możesz uzyskać ten sam efekt za pomocą zwykłych funkcji (patrz strona 115). Najpierw utwórz standardową funkcję, którą skrypt ma uruchamiać po zajściu zdarzenia `mouseover`. W drugiej standardowej funkcji umieść kod wykonywany po zgłoszeniu zdarzenia `mouseout`. Następnie przekaż nazwy obu tych funkcji do funkcji `hover()`:

```
function showSubmenu() {
  $('#submenu').show();
}
function hideSubmenu() {
  $('#submenu').hide();
}
$('#menu').hover(showSubmenu, hideSubmenu);
```

Jeśli ta technika wydaje Ci się łatwiejsza, możesz jej używać. Oba podejścia działają tak samo, jednak część programistów ceni funkcje anonimowe, ponieważ pozwalają umieścić cały kod w jednym miejscu, bez dzielenia go na kilka odrębnych instrukcji.

Uwaga: Biblioteka jQuery w wersjach do 1.9 udostępniała przydatną funkcję `toggle()`. Działała ona podobnie jak funkcja hover() z tą różnicą, że obsługiwała zdarzenia `click`. Pierwszy zestaw kodu obsługiwał pierwsze kliknięcie, a drugi zestaw — drugie. Innymi słowy, pozwalała ona „przełączać" kod obsługujący kliknięcia — stanowiła przez to doskonałe narzędzie do wyświetlania wybranego elementu strony w ramach pierwszego kliknięcia i ukrywania go po drugim kliknięciu. Jednak funkcja ta nie jest już dostępna, dlatego w dalszej części rozdziału, w przykładzie rozpoczynającym się na stronie 204 nauczysz się odtwarzać jej możliwości funkcjonalne.

Obiekt reprezentujący zdarzenie

Kiedy przeglądarka zgłasza zdarzenie, rejestruje informacje na jego temat i zapisuje go w *obiekcie zdarzenia*. Obiekt ten zawiera dane zebrane w momencie wystąpienia zdarzenia, na przykład współrzędne wskaźnika myszy, element powiązany ze zdarzeniem lub informacje o tym, czy wciśnięty był klawisz *Shift*.

W jQuery obiekt zdarzenia jest dostępny w funkcji odpowiedzialnej za obsługę danego zdarzenia. Obiekt ten jest przekazywany do funkcji, dlatego aby uzyskać do niego dostęp, należy użyć parametru. Poniższy kod sprawdza, jakie były współrzędne X i Y wskaźnika w momencie kliknięcia dowolnego fragmentu strony:

```
$(document).click(function(evt) {
  var xPos = evt.pageX;
  var yPox = evt.pageY;
  alert('X:' + xPos + ' Y:' + yPos);
}); // Koniec funkci click.
```

Istotna jest tu zmienna `evt`. W momencie wywołania funkcji (w wyniku kliknięcia dowolnego miejsca w oknie przeglądarki) program zapisuje obiekt zdarzenia w zmiennej `evt`. W ciele funkcji można uzyskać dostęp do różnych właściwości tego obiektu dzięki używaniu notacji z kropką, na przykład wyrażenie `evt.pageX` zwraca współrzędną X wskaźnika, czyli liczbę pikseli od lewej krawędzi okna.

Uwaga: W tym kodzie `evt` to nazwa zmiennej podana przez programistę. Nie jest to słowo kluczowe języka JavaScript, a jedynie zmienna służąca do przechowywania obiektu zdarzenia. Możesz użyć też dowolnej innej nazwy, na przykład `event` lub `e`.

Obiekt zdarzenia ma wiele właściwości, jednak — niestety — ich lista jest różna w poszczególnych przeglądarkach. W tabeli 5.1 znajdziesz kilka właściwości obsługiwanych przez większość przeglądarek.

Tabela 5.1. *Każde zdarzenie związane jest z obiektem o różnych właściwościach, które można sprawdzić w funkcji obsługującej dane zdarzenie*

Właściwość zdarzenia	Opis
`pageX`	Odległość w pikselach wskaźnika myszy od lewej krawędzi okna przeglądarki.
`pageY`	Odległość w pikselach wskaźnika myszy od górnej krawędzi okna przeglądarki.
`screenX`	Odległość w pikselach wskaźnika myszy od lewej krawędzi monitora.
`screenY`	Odległość w pikselach wskaźnika myszy od górnej krawędzi monitora.
`shiftKey`	Ma wartość `true`, jeśli w momencie wystąpienia zdarzenia wciśnięty był klawisz *Shift*.
`which`	Należy jej używać w zdarzeniu `keypress`. Pozwala sprawdzić kod wciśniętego klawisza (patrz następna wskazówka).
`target`	Obiekt docelowy zdarzenia. Na przykład kliknięty element w zdarzeniu `click()`.
`data`	Obiekt jQuery użyty w funkcji `on()` do przekazania danych do funkcji obsługi zdarzenia (patrz strona 197).

Wskazówka: Za pomocą właściwości `which` obiektu zdarzenia `keypress()` można pobrać kod wciśniętego klawisza. Jeśli chcesz ustalić, jaki znak wcisnął użytkownik (a, K, 9 i tak dalej), musisz przekazać wartość właściwości `which` do metody języka JavaScript, która przekształci numer klawisza na literę, liczbę lub symbol:

```
String.fromCharCode(evt.which)
```

Blokowanie standardowych reakcji na zdarzenia

Niektóre elementy języka HTML mają wbudowane reakcje na zdarzenia. I tak użycie odnośnika powoduje przejście do nowej strony, a kliknięcie przycisku *Wyślij* przesyła dane formularza na serwer w celu ich przetworzenia. Czasem takie domyślne reakcje są niepożądane. Na przykład warto zablokować przesyłanie formularza (zdarzenie `submit()`), jeśli użytkownik pominął wymagane dane.

Aby wyłączyć standardową reakcję przeglądarki na zdarzenie, należy użyć funkcji `preventDefault()`. Jest to funkcja obiektu zdarzenia (patrz poprzedni punkt), dlatego można ją wywołać w funkcji obsługującej dane zdarzenie. Załóżmy, że na stronie znajduje się odnośnik o identyfikatorze `menu`. Prowadzi on do następnej strony z menu, co umożliwia dostęp do menu w przeglądarkach z wyłączoną obsługą języka JavaScript. Jednak na stronie znajduje się też kod JavaScript, który po kliknięciu tego odnośnika wyświetla menu w prawej części bieżącej strony. Domyślnie przeglądarka użyje odnośnika do przejścia do nowej strony z menu, dlatego należy zablokować tę reakcję:

```
$('#menu').click(function(evt){
  // Kod JavaScript.
  evt.preventDefault(); // Program nie przejdzie do strony z menu.
});
```

Inne rozwiązanie polega na zwróceniu wartości `false` w ostatnim wierszu funkcji. Poniższy fragment kodu działa tak samo jak wcześniejszy:

```
$('#menu').click(function(evt){
  // Kod JavaScript.
  return false; // Program nie przejdzie do strony z menu.
});
```

Usuwanie zdarzeń

Czasem trzeba usunąć zdarzenie przypisane wcześniej do znacznika. Można to zrobić za pomocą funkcji `off()`. Aby jej użyć, najpierw należy utworzyć obiekt jQuery, reprezentujący element z usuwanym zdarzeniem. Następnie wystarczy dodać funkcję `off()` i przekazać do niej łańcuch znaków z nazwą zbędnego zdarzenia. Jeśli chcesz sprawić, aby wszystkie znaczniki `tabButton` nie reagowały na kliknięcie, możesz wywołać poniższą instrukcję:

```
$('.tabButton').off('click');
```

Poniższy krótki skrypt ilustruje działanie funkcji `off()`:

```
1 $('a').mouseover(function() {
2   alert('Wskaźnik myszy znajduje się nade mną!');
3 });
4 $('#disable').click(function() {
5   $('a').off('mouseover');
6 });
```

Wiersze od 1. do 3. przypisują funkcję do zdarzenia `mouseover` wszystkich odnośników (znaczników `<a>`). Umieszczenie wskaźnika myszy nad odnośnikiem spowoduje wyświetlenie okna dialogowego z tekstem „Wskaźnik myszy znajduje się nade mną!". Jednak ponieważ ciągłe wyświetlanie tego komunikatu jest irytujące, wiersze od 4. do 6. umożliwiają wyłączenie powiadomień. Kiedy użytkownik kliknie odnośnik o identyfikatorze `disable` (na przykład przycisk formularza), skrypt odłączy zdarzenie `mouseover` od wszystkich odnośników, dlatego okno dialogowe przestanie się pojawiać.

Jeśli chcesz usunąć ze znacznika wszystkie obsługiwane w nim zdarzenia, wystarczy wywołać funkcję `off()` bez żadnych argumentów. Aby na przykład usunąć wszystkie zdarzenia — `mouseover`, `click`, `dblclick` i tak dalej — z przycisku przesyłającego formularz na serwer, mógłbyś użyć następującego fragmentu kodu:

```
$('input[type="submit"]').off();
```

To dość drastyczne rozwiązanie i w większości przypadków nie będziesz chciał usuwać z elementu wszystkich obsługiwanych zdarzeń.

Uwaga: Więcej informacji o funkcji `off()` biblioteki jQuery znajdziesz na stronie *http://api.jquery.com/off/*.

PORADNIA DLA ZAAWANSOWANYCH

Wstrzymywanie przekazywania zdarzeń

Internet Explorer oraz model zdarzeń organizacji W3C, używany w przeglądarkach Firefox, Safari i Opera, umożliwiają przekazywanie zdarzeń poza element, który zarejestruje zdarzenie jako pierwszy. Załóżmy, że przypisałeś funkcję obsługującą zdarzenia `click` odnośnika. Po jego kliknięciu przeglądarka zgłasza to zdarzenie i uruchamia funkcję. Jednak to jeszcze nie koniec. Na to samo kliknięcie zareaguje też każdy przodek (czyli element, wewnątrz którego jest umieszczony kliknięty element). Dlatego jeśli przypisałeś też funkcję obsługującą zdarzenia `click` znacznika `<div>`, w którym znajduje się wspomniany odnośnik, uruchomiona zostanie także ta funkcja.

Ten mechanizm (tak zwane *przekazywanie zdarzeń*) sprawia, że jedno zdarzenie może wywołać reakcję kilku elementów. Oto następny przykład. Do rysunku można dodać zdarzenie `click`, aby jego kliknięcie powodowało wyświetlenie nowego obrazka. Rysunek znajduje się w znaczniku `<div>`, który także reaguje na zdarzenie `click` (na przykład wyświetla okno dialogowe). Kliknięcie rysunku spowoduje uruchomienie funkcji określonych dla obu znaczników, ponieważ zdarzenie zajdzie także dla znacznika `<div>`.

Taka sytuacja zdarza się rzadko, jednak prowadzi zwykle do niepożądanych skutków. Nie chcesz przecież, aby znacznik `<div>` reagował na kliknięcie rysunku. Dlatego należy zablokować przekazywanie zdarzeń do znacznika `<div>`, a przy tym uruchomić funkcję przypisaną do zdarzenia `click` rysunku. Oznacza to, że kliknięcie obrazka powinno prowadzić do zmiany grafiki i zatrzymania zdarzenia `click`.

Funkcja `stopPropagation()` zatrzymuje przekazywanie zdarzenia do przodków. Jest to metoda obiektu zdarzenia (patrz strona 194), dlatego można jej użyć w funkcji obsługującej dane zdarzenie:

```
$('#theLink').click(function(evt){
  // Wykonywane operacje.
  evt.stopPropagation(); // Zatrzymuje
  // przekazywanie zdarzenia.
});
```

Zaawansowane zarządzanie zdarzeniami

Możesz napisać wiele programów, używając tylko metod i technik obsługi zdarzeń biblioteki jQuery, opisanych na poprzednich stronach. Jednak jeśli chcesz w pełni wykorzystać możliwości obsługi zdarzeń oferowane przez tę bibliotekę, powinieneś nauczyć się stosowania funkcji `on()`.

Uwaga: Jeśli wciąż przyswajasz sobie materiał przedstawiony w poprzednim podrozdziale, możesz pominąć ten fragment i przejść bezpośrednio do przykładu na stronie 204. Zawsze możesz wrócić do tego punktu, kiedy nabierzesz doświadczenia w obsłudze zdarzeń.

Metoda `on()` umożliwia bardziej elastyczne zarządzanie zdarzeniami niż specyficzne dla zdarzeń funkcje biblioteki jQuery, na przykład `click()` i `mouseover()`. Metoda ta nie tylko pozwala na określenie zdarzenia i reagującej na nie funkcji, ale też na przekazanie dodatkowych danych do funkcji obsługującej zdarzenie. Umożliwia to różnym elementom i zdarzeniom (na przykład kliknięciu odnośnika lub umieszczeniu wskaźnika myszy nad rysunkiem) przekazywanie odmiennych informacji do tej samej funkcji obsługi zdarzeń. Oznacza to, że jedna funkcja może działać w inny sposób, w zależności od tego, które zdarzenie obsługuje.

Podstawowa składnia funkcji `on()` wygląda następująco:

```
$('#selektor').on('click', selector, myData, functionName);
```

Uwaga: Wraz z rozwojem biblioteki jQuery zmieniały się także nazwy funkcji używanych do obsługi zdarzeń. Jeśli czytasz starsze książki lub wpisy na blogach, zapewne spotkałeś się z takimi funkcjami jak `bind()`, `live()` lub `delegate()`. Jednak obecnie wszystkie te funkcje służące do dodawania zdarzeń do elementów zostały zastąpione jedną funkcją `on()`. Oprócz tego funkcja `unbind()`, służąca wcześniej do usuwania zdarzeń z elementów, została zastąpiona funkcją `off()`.

Pierwszy argument to łańcuch znaków zawierający nazwę zdarzenia (na przykład `click`, `mouseover` lub dowolne inne zdarzenie wymienione na stronach od 179 do 182).

Drugi argument jest opcjonalny, więc podawanie jego wartości w wywołaniach funkcji `on()` nie jest konieczne. Jeśli jednak zdecydujesz się go podać, musi to być prawidłowy selektor, taki jak `tr`, `.callout` bądź `#alarm`.

Uwaga: Tego drugiego argumentu można używać do obsługi zdarzenia w innym elemencie umieszczonym wewnątrz elementu, który został wybrany. Technika ta jest określana jako *delegowanie zdarzeń* (ang. *event delegation*) i dowiesz się o niej na stronie 200.

Trzecim argumentem funkcji `on()` są dodatkowe dane, które mają zostać do niej przekazane. Może to być literał obiektowy lub zmienna zawierająca taki literał. Literały obiektowe (patrz strona 165) to listy nazw i wartości właściwości:

```
{
  firstName : 'Robert',
  lastName : 'Kowalski'
}
```

Literał obiektowy można zapisać w zmiennej w następujący sposób:

```
var linkVar = {message:'Pozdrowienia od odnośnika'};
```

Czwarty argument funkcji `on()` to następna funkcja. Jest ona uruchamiana po zgłoszeniu zdarzenia. Można użyć tu funkcji anonimowej lub standardowej, podobnie jak podczas używania zwykłych zdarzeń biblioteki jQuery, co opisano na stronie 182.

Uwaga: Przekazywanie danych w funkcji `on()` nie jest konieczne. Jeśli chcesz jej użyć do dołączenia zdarzenia i funkcji do elementu, możesz pominąć zmienną z danymi:

```
$('selektor').on('click', functionName);
```

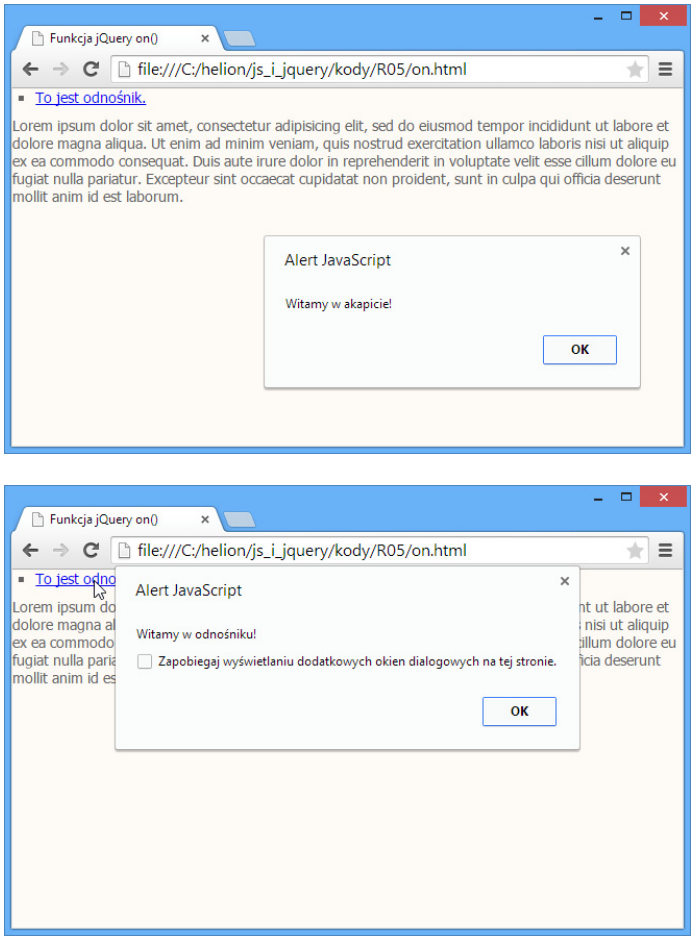
Ten kod działa tak samo jak poniższa instrukcja:

```
$('selektor').click(functionName);
```

Załóżmy, że chcesz wyświetlić okno dialogowe w reakcji na zgłoszenie zdarzenia, jednak komunikat ma być dopasowany do elementu powiązanego z tym zdarzeniem. Aby uzyskać ten efekt, można utworzyć zmienne przechowujące różne literały obiektowe, a następnie przekazywać te zmienne do funkcji `on()`, powiązanej z różnymi elementami:

```
var linkVar = { message:'Pozdrowienia od odnośnika'};
var pVar = { message:'Pozdrowienia od akapitu'};
function showMessage(evt) {
  alert(evt.data.message);
}
$('a').on('click',linkVar,showMessage);
$('p').on('mouseover',pVar,showMessage);
```

Na rysunku 5.5 pokazano działanie tego kodu. Skrypt tworzy dwie zmienne — `linkVar` w wierszu pierwszym i `pVar` w wierszu drugim. Obie zmienne zawierają literał obiektowy z właściwością o tej samej nazwie, `message`, ale z innym tekstem komunikatu. Funkcja `showMessage()` przyjmuje obiekt zdarzenia (patrz strona 194) i zapisuje go w zmiennej `evt`. Funkcja ta wyświetla wartość właściwości `message` (jest ona zapisana we właściwości `data` obiektu zdarzenia) za pomocą polecenia `alert()`. Warto pamiętać, że `message` to nazwa właściwości zdefiniowana w literale obiektowym.



Rysunek 5.5. *Funkcja on() biblioteki jQuery umożliwia przekazywanie danych do funkcji obsługujących zdarzenia. W ten sposób można użyć jednej standardowej funkcji do obsługi k lku elementów (a nawet różnych zdarzeń) i jednocześnie korzystać w tej funkcji z danych specyficznych dla funkcji obsługujących zdarzenia*

Inne sposoby stosowania funkcji on()

Funkcja `on()` biblioteki jQuery zapewnia bardzo dużą elastyczność. Oprócz technik opisanych w poprzednim podrozdziale, pozwala także wyznaczyć jedną funkcję do obsługi dwóch lub nawet większej liczby zdarzeń. Na przykład załóżmy, że piszemy program, który w odpowiedzi na kliknięcie miniaturki zdjęcia będzie wyświetlał na ekranie jego powiększoną wersję (jest to popularne rozwiązanie o nazwie „lightbox”,

które możemy znaleźć na tysiącach stron). Chcemy też, by powiększony obrazek zniknął, gdy użytkownik kliknie w dowolnym miejscu strony bądź naciśnie dowolny klawisz (udostępnienie obu tych możliwości sprawi, że ze strony wygodnie będą mogły korzystać zarówno osoby preferujące stosowanie myszy, jak i klawiatury). Oto kod, który zapewnia stosowną funkcjonalność:

```
$(document).on('click keypress', function() {
  $('#lightbox').hide();
}); // Koniec funkcji on.
```

Najważniejszym fragmentem powyższego przykładu jest pierwszy argument metody `bind()` — `'click keypress'`. Podając nazwy kilku zdarzeń, oddzielone od siebie znakami odstępu, informujemy jQuery, że *każde* z nich ma być obsługiwane przy użyciu przekazanej funkcji anonimowej. W naszym przypadku nastąpi to, gdy zostanie zgłoszone zarówno zdarzenie `click`, jak i `keypress`, skierowane do całego dokumentu.

Jeśli oprócz tego chcemy obsługiwać kilka zdarzeń i każdemu z nich przypisać inną funkcję obsługi, nie musimy w tym celu używać kilku odrębnych wywołań metody `on()`. Innymi słowy, jeśli chcemy, by po kliknięciu elementu została wykonana jedna czynność, a inna w momencie, gdy użytkownik umieści na nim wskaźnik myszy, moglibyśmy to zrobić za pomocą następującego fragmentu kodu:

```
$('#theElement').on('click', function() {
  // tu robimy coś interesującego
}); // Koniec funkcji on.
$('#theElement').on('mouseover', function() {
  // tu robimy coś interesującego
}); // Koniec funkcji on.
```

Jednak dokładnie to samo można uzyskać, przekazując w wywołaniu metody `on()` literał obiektowy (patrz strona 165) składający się z nazwy zdarzenia oraz podanej po dwukropku funkcji anonimowej. Poniżej przedstawiona została zmodyfikowana wersja powyższego fragmentu kodu, w której funkcja `on()` jest wywoływana tylko raz, a przy tym w jej wywołaniu jest przekazywany literał obiektowy (wyróżniony pogrubioną czcionką):

```
$('#theElement').on({
    'click' : function() {
        // Tu robimy coś interesującego.
     }, // Koniec funkcji click.
    'mouseover' : function() {
        // Tu robimy coś interesującego
     }; // Koniec funkcji mouseover.
}); // Koniec funkcji on.
```

Delegowanie zdarzeń przy użyciu funkcji on()

Zgodnie z informacjami podanymi na stronie 197, drugim, opcjonalnym argumentem metody `on()` może być selektor innego elementu:

```
$('#selektor).on('click', selector, myData, functionName);
```

Tym drugim argumentem może być dowolny, prawidłowy selektor jQuery, na przykład selektor identyfikatora, elementu lub dowolny spośród selektorów opisanych na stronie 147. Przekazanie takiego selektora w wywołaniu funkcji `on()`

znacząco zmienia jej działanie. Bez niego zdarzenie jest obsługiwane w elemencie wskazanym przez początkowy selektor, w powyższym przykładzie będzie to `$('selektor')`. Załóżmy, że umieściłeś na stronie następujący fragment kodu:

```
$('li').on('click', function() {
  $(this).css('text-decoration', 'line-through');
}); // Koniec funkcji on.
```

Powyższy kod powoduje, że każdy znacznik `<li>`, który klikniesz, zostanie przekreślony. Pamiętaj, że w tym przypadku `$(this)` reprezentuje element obsługujący zdarzenie, czyli kliknięty znacznik `<li>`. Innymi słowy, zdarzenie `click` zostało „powiązane" ze znacznikiem `<li>`. W większości przypadków właśnie o to będzie chodziło — o zdefiniowanie funkcji, która zostanie wykonana w momencie, gdy użytkownik wejdzie w interakcję z wybranym elementem strony. Dzięki temu można wykonywać określone funkcje, kiedy użytkownik kliknie odnośnik, wskaże myszą opcję w menu, prześle formularz i tak dalej.

Uwaga: Wciąż nie jesteś pewny, co oznacza `$(this)`? Szczegółowe informacje na ten temat znajdziesz na stronie 169, natomiast na stronie 189 dowiesz się wszystkiego o tym, w jaki sposób wyrażenie `$(this)` jest używane podczas obsługi zdarzeń.

Jednak z takim sposobem obsługi zdarzeń w elementach wiąże się jeden problem: można go zastosować wyłącznie w przypadku, gdy wybrany element już istnieje na stronie. Jeśli kod HTML zostanie dodany do strony dynamicznie, już po wcześniejszym określeniu funkcji obsługującej zdarzenia przy użyciu takich funkcji jQuery jak `click()`, `mouseover()` lub `on()`, to te nowe elementy nie będą reagowały na żadne zdarzenia. No dobrze, to całkiem rozsądne, jednak wymaga nieco dokładniejszego wyjaśnienia.

Załóżmy, że utworzyłeś aplikację pozwalającą użytkownikowi na zarządzanie listą zadań do zrobienia. Bezpośrednio po wczytaniu takiej aplikacji lista jest pusta (patrz punkt 1. na rysunku 5.6.). Użytkownik może wpisać coś w polu tekstowym i kliknąć przycisk *Dodaj zadanie*, aby dodać do listy nowe zadania (patrz punkt 2. na rysunku 5.6). Kiedy użytkownik wykona zadanie, może je kliknąć na liście, by oznaczyć, że zostało zrobione (patrz punkt 3. na rysunku 5.6).

Na pewno domyślasz się, że aby oznaczyć zadanie jako wykonane, musisz dodać do każdego znacznika `<li>` obsługę zdarzeń `click`. W przykładzie przedstawionym na rysunku 5.6 wykonane zadania są przekreślone i wyświetlone na szaro. Problem jednak polega na tym, że w momencie wczytywania i wyświetlania strony nie ma jeszcze żadnych znaczników `<li>`, zatem dodanie funkcji obsługującej zdarzenia `click` do tych właśnie znaczników `<li>` nie da żadnego efektu. Innymi słowy, kod przedstawiony na stronie 200 nie będzie działał.

Uwaga: Więcej informacji na temat delegowania zdarzeń możesz znaleźć na stronie *http://api.jquery.com/on/*.

Dlatego też konieczne jest *delegowanie* zdarzeń, czyli obsługa zdarzenia w elemencie umieszczonym wyżej w hierarchii (już istniejącym na stronie) i odbieranie zdarzeń związanych z konkretnym elementem potomnym. Ponieważ zdarzenie jest definiowane dla elementu, który już istnieje na stronie, zatem dodawanie nowych elementów potomnych nie będzie miało żadnego wpływu na jego obsługę.

Rysunek 5.6. *Czasami używany kod JavaScript będzie dodawał do strony WWW nowe elementy. W tym przypadku użytkownik może dodawać nowe elementy (zadania) do listy wypunktowanej. Po wyświetleniu strony lista jest pusta — zawiera jedynie pustą parę znaczników <ul> (widoczną u góry). Użytkownik może wpisywać zadania w polu tekstowym i klikać przycisk Dodaj zadanie, co będzie powodowało dodawanie nowych znaczników <li> (środkowy zrzut). Aby oznaczyć zadanie jako wykonane, wystarczy je kliknąć (dolny zrzut). Działającą aplikację listy zadań możesz znaleźć w pliku to-do-list.html w katalogu R05*

Innymi słowy, rozwiązanie to polega na *przekazaniu* obsługi zdarzenia do istniejącego elementu rodzica. Bardziej szczegółowe informacje na temat tego rozwiązania możesz znaleźć w ramce na stronie 205. Na razie jednak zajmiemy się wykorzystaniem funkcji `on()` do uruchomienia naszej przykładowej aplikacji:

```
$('li').on('click', 'li', function() {
  $(this).css('text-decoration', 'line-through');
}); // Koniec funkcji on.
```

Tworząc stronę, umieściliśmy na niej pusty znacznik `<ul>`, stanowiący pojemnik, do którego później będą dodawane znaczniki `<li>`. W efekcie po wczytaniu strony ten pusty znacznik `<ul>` jest już na niej dostępny. Zatem wykonanie przedstawionego powyżej fragmentu kodu dodaje do tego znacznika funkcję `on()`. Użytkownik nie dodał do swojej listy żadnych zadań, a więc na stronie nie ma jeszcze

znaczników <li>. Kiedy jednak dodamy do wywołania funkcji on() drugi argument, selektor 'li', stwierdzamy, że *nie* interesują nas zdarzenia związane ze znacznikiem <ul>, lecz z umieszczonymi w nim znacznikami <li>. To, kiedy te znaczniki zostaną dodane do strony, nie ma większego znaczenia, gdyż zdarzenia są obsługiwane przez istniejący znacznik <ul>.

Wpływ delegowania zdarzeń na wartość $(this)

Zgodnie z informacjami podanymi już wcześniej, wartością wyrażenia $(this) jest obiekt reprezentujący aktualnie przetwarzany element pętli (patrz strona 169) lub element związany z aktualnie obsługiwanym zdarzeniem (patrz strona 189). Zazwyczaj w funkcjach obsługujących zdarzenia wyrażenie $(this) reprezentuje zastosowany selektor, na przykład:

```
$('ul').on('click', function() {
  $(this).css('text-decoration': 'line-through');
}
```

W tym przykładzie $(this) odwołuje się do kliknięty znacznika <ul>. Jednak w przypadku korzystania z delegowania zdarzeń ten pierwszy selektor nie określa już elementu, z którym jest prowadzona interakcja (czyli który użytkownik kliknął, wskazał myszą i tak dalej), lecz element, który go zawiera. Przyjrzyjmy się jeszcze raz przykładowi wykorzystującemu delegowanie zdarzeń:

```
$('ul').on('click', 'li', function() {
  $(this).css('text-decoration', 'line-through');
}); // Koniec funkcji on.
```

W tym przypadku znacznikiem, który użytkownik klika, jest <li> i to właśnie on powinien odpowiedzieć na zdarzenie. Innymi słowy, znacznik <ul> pełni jedynie rolę kontenera, natomiast funkcja ma być wykonana w momencie kliknięcia znacznika <li>. A zatem w tym przykładzie wyrażenie $(this) będzie się odwoływało do znacznika <li>, a powyższa funkcja będzie przekreślać każdy kliknięty przez użytkownika element listy.

W bardzo wielu przypadkach stosowanie delegowania zdarzeń nie będzie potrzebne. Jednak technika ta okaże się przydatna, kiedy konieczne będzie dodawanie obsługi zdarzeń do elementów, których jeszcze nie ma na stronie w momencie jej wczytywania. Przykładowo podczas korzystania z technologii AJAX (opisanej w rozdziale 13.) konieczne będzie stosowanie delegowania do obsługi zdarzeń przez kod HTML dynamicznie pobierany z serwera i dodawany do strony.

Uwaga: W niektórych przypadkach delegowanie zdarzeń można także stosować w celu poprawienia wydajności działania kodu JavaScript. Jeśli ta sama funkcja obsługi zdarzeń jest używana w bardzo wielu znacznikach, na przykład w setkach komórek bardzo dużej tabeli, lepszym rozwiązaniem może być przekazanie ich do znacznika <table> w sposób pokazany poniżej:

```
$('table').on('click', 'td', function() {
  // kod obsługi kliknięcia
}
```

Dodając funkcję obsługującą zdarzenia do tabeli, unikamy konieczności dodawania jej do setek, a może nawet tysięcy komórek, co mogłoby dodatkowo obciążyć pamięć przeglądarki i doprowadzić do spadku wydajności jej działania.

Przykład — jednostronicowa lista FAQ

W sieci WWW znajduje się wiele stron z listami często zadawanych pytań (ang. *Frequently Asked Questions* — FAQ). Pozwalają one błyskawicznie uzyskać odpowiedź na pytanie — 24 godziny na dobę 7 dni w tygodniu, co usprawnia obsługę klienta. Niestety, większość list FAQ to albo bardzo długie strony pełne pytań i kompletnych odpowiedzi, albo krótkie strony z pytaniami w formie odnośników, które prowadzą do odrębnych stron z odpowiedziami. Oba te rozwiązania wydłużają wyszukiwanie informacji, gdyż w pierwszym przypadku zmuszają użytkownika do przewijania długiej strony w poszukiwaniu interesującego go pytania i odpowiedzi, a w drugim — oczekiwania na pobranie i wyświetlenie kolejnej strony.

W tym przykładzie rozwiążesz ten problem przez utworzenie strony z listą FAQ opartej na kodzie JavaScript. Wczytana strona będzie zawierać wszystkie pytania, dlatego użytkownik będzie mógł szybko znaleźć to, które go interesuje. Jednak odpowiedź ma być ukryta do momentu kliknięcia wybranego pytania. Wtedy skrypt powinien stopniowo wyświetlać pożądaną odpowiedź (patrz rysunek 5.7).

Rysunek 5.7. *Przy użyciu kilku wierszy kodu JavaScript można sprawić, że elementy strony będą pojawiać się i znikać w odpowiedzi na kliknięcie myszą*

Omówienie zadania

Kod JavaScript w tym zadaniu ma wykonywać kilka operacji:

- Wyświetlać ukrytą odpowiedź po kliknięciu powiązanego z nią pytania.
- Ukrywać widoczną odpowiedź po kliknięciu pytania.

CZĘSTO ZADAWANE PYTANIA

Magia delegowania zdarzeń?

No dobrze, rozumiem podstawy delegowania zdarzeń, ale jak naprawdę działa to rozwiązanie?

Zgodnie z informacjami podanymi w ramce na stronie 197, zdarzenie jest przekazywane ku górze hierarchii strony WWW. Kiedy klikniemy odnośnik umieszczony w akapicie, zdarzenie `click` najpierw trafi do odnośnika, następnie otrzyma je element rodzica odnośnika, czyli akapit, później element <body>, a na samym końcu element <html>. Innymi słowy, zdarzenie wywołane przez jeden element strony jest przekazywane ku górze, do każdego kolejnego rodzica.

Ten fakt może być bardzo pomocny w razie użycia delegowania zdarzeń. Jak już wspomniałem na stronie 200, może się zdarzyć, że będziemy chcieli obsługiwać zdarzenia w kodzie, który jeszcze nie istnieje — na przykład w zadaniach do zrobienia, które pojawią się dopiero po wczytaniu strony i dodaniu zadania przez użytkownika. Choć nie możemy dodać obsługi zdarzeń click do nieistniejących znaczników <li>, jednak możemy ją dodać do już istniejącego znacznika, takiego jak <ul> lub nawet , w którym <ul> jest umieszczony.

Zgodnie z informacjami podanymi na stronie 194, każdemu zdarzeniu towarzyszy obiekt zdarzenia, który przechowuje wiele różnych informacji. W kontekście delegowania zdarzeń najważniejszą z tych informacji jest właściwość target. Precyzyjnie określa ona element, który jako pierwszy odebrał zdarzenie. Jeśli na przykład klikniemy odnośnik, to właśnie on będzie zapisany we właściwości target. Jednak ze względu na przekazywanie zdarzeń zdarzenie może trafić także do elementu rodzica, który z kolei na podstawie właściwości target może określić, do którego elementu zdarzenie było skierowane.

A zatem delegowanie zdarzeń sprawia, że mogą one być odbierane przez elementy położone wyżej w hierarchii strony; na przykład element <ul> może odbierać zdarzenia click. Jeśli zdarzenie było kierowane do znacznika <li>, możemy na to odpowiednio zareagować — na przykład przekreślić tekst, oznaczając zadanie jako wykonane.

Ponadto kod JavaScript posłuży do ukrycia wszystkich odpowiedzi w momencie wczytywania strony. Dlaczego nie użyć do tego stylów CSS? Aby ukryć odpowiedzi, wystarczy przecież ustawić w stylu właściwość `display` na `none`. Jednak jeśli przeglądarka ma wyłączoną obsługę języka JavaScript, użytkownik nie zobaczy odpowiedzi ani nie będzie mógł ich wyświetlić. Aby strona była przydatna zarówno w przeglądarkach z włączoną, jak i wyłączoną obsługą skryptów, najlepiej ukryć jej zawartość przy użyciu kodu JavaScript.

Uwaga: Informacje o pobieraniu przykładowych plików znajdziesz na stronie 46.

Tworzenie kodu

1. **Otwórz w edytorze tekstu plik *faq.html* z katalogu *R05*.**

   Ten plik zawiera już funkcję `$(document).ready()` (patrz strona 190) i kod dołączający bibliotekę jQuery. Najpierw należy ukryć wszystkie odpowiedzi w czasie wczytywania strony.

2. **Kliknij pusty wiersz pod kodem `$(document).ready()` i dodaj fragment `$('.answer').hide();`.**

   Poszczególne odpowiedzi znajdują się w znacznikach `<div>` klasy `answer`. Dodany wiersz kodu pobiera wszystkie te znaczniki i ukrywa je (opis funkcji

`hide()` znajdziesz na stronie 212). Zapisz stronę i wyświetl ją w przeglądarce. Wszystkie odpowiedzi powinny być ukryte.

Uwaga: Elementy są ukrywane przy użyciu kodu JavaScript, a nie przez style CSS dlatego, że niektórzy użytkownicy mogą wyłączać w przeglądarkach obsługę języka JavaScript. W takim przypadku nie zobaczą oni fantastycznych efektów, które tworzysz w tym przykładzie, lecz przynajmniej będą w stanie obejrzeć wszystkie odpowiedzi.

Następny krok wymaga określenia, do których elementów program ma dodać odbiornik zdarzenia. Ponieważ odpowiedzi pojawiają się po kliknięciu pytania, trzeba pobrać wszystkie pytania z listy FAQ. Na tej stronie znajdują się one w znacznikach `<h2>` w elemencie o identyfikatorze `main`.

3. **Wciśnij klawisz *Enter*, aby dodać nowy wiersz, i wpisz w nim kod wyróżniony pogrubieniem:**

```
<script src="../_js/jquery.min.js"></script>
<script>
$(document).ready(function() {
  $('.answer').hide();
  $('.main h2')
}); // Koniec funkcji ready.
</script>
```

Nowy kod to prosty selektor potomków, który pobiera wszystkie znaczniki `<h2>` zapisane w elemencie klasy `main` (dlatego skrypt nie modyfikuje pozostałych znaczników `<h2>` znajdujących się na stronie). Teraz należy dodać zdarzenie. Dobrym kandydatem jest zdarzenie `click`, jednak aby wykonać postawione zadanie, będziesz musiał zrobić coś więcej. W tym przykładzie każde kliknięcie będzie wyświetlać odpowiedź bądź ją chować. To wygląda jak sytuacja, w której należało użyć instrukcji warunkowej. Konkretnie rzecz biorąc, będziesz chciał sprawdzić, czy znacznik `<div>` umieszczony za znacznikiem `<h2>` jest ukryty; jeśli jest, będziesz musiał go wyświetlić, a w przeciwnym razie, gdy jest widoczny, będziesz musiał go ukryć.

4. **Do znaczników <h2> dodaj funkcję anonimową wyróżnioną poniżej pogrubioną czcionką:**

```
$(document).ready(function() {
  $('.answer').hide();
  $('.main h2').click(function() {

  }); // Koniec funkcji click.
}); // Koniec funkcji ready.
```

Powyższy kod dodaje do znaczników <h2> zdarzenie `click`. Komentarz na końcu funkcji *click* nie jest konieczny, jednak, zgodnie z informacjami podanymi na stronie 89, może pomóc w określeniu, do czego należy sekwencja `)};`. Teraz dowolny kod, który umieścisz wewnątrz funkcji anonimowej, zostanie wykonany za każdym razem, gdy użytkownik kliknie jeden ze znaczników `<h2>`.

5. **W pustym wierszy wewnątrz funkcji anonimowej wpisz:**

```
var $answer = $(this).next('.answer');
```

Ten wiersz kodu tworzy zmienną — `$answer` — która będzie zawierać obiekt jQuery. Zgodnie z informacjami podanymi na stronie 189, `$(this)` odwołuje się do elementu odpowiadającego na zdarzenie — w tym przypadku jest

to konkretny element <h2>. Biblioteka jQuery udostępnia kilka funkcji znacznie ułatwiających poruszanie się po strukturze strony. Funkcja .next() odnajduje znacznik, który jest umieszczony bezpośrednio za aktualnie wybranym znacznikiem. Innymi słowy, w naszym przypadku odnajduje ona znacznik umieszczony bezpośrednio za znacznikiem <h2>. Przekazując w wywołaniu funkcji .next() dodatkowy selektor, można jeszcze dokładniej określić, o jaki znacznik chodzi; wywołanie .next('.answer') odnajdzie pierwszy znacznik umieszczony za <h2>, należący do klasy answer.

Inaczej mówiąc, zapisujesz w zmiennej odwołanie do znacznika umieszczonego bezpośrednio za znacznikiem <h2>. Znacznik ten zawiera odpowiedź na pytanie. Odwołanie zapisujesz w zmiennej, gdyż będziesz musiał użyć go kilka razy: aby sprawdzić, czy odpowiedź jest ukryta, oraz wyświetlić ją, jeśli jest schowana, bądź ukryć, jeśli jest widoczna. Za każdym razem, kiedy odwołujesz się do jQuery przy użyciu wywołania $( ), wykonujesz całkiem sporo kodu biblioteki. A zatem kod o postaci $(this).next('.answer') sprawia, że jQuery wykona trochę pracy. Zamiast kilkakrotnego wykonywania tych samych czynności lepiej będzie zapisać ich wynik w zmiennej i używać jej do wykonywania operacji na znaczniku , który chcesz ukryć lub wyświetlić.

Gdy trzeba wielokrotnie używać tych samych wyników jQuery, zawsze warto zapisać je za pierwszym razem w zmiennej, a następnie stosować do wykonywania dalszych operacji. Takie rozwiązanie poprawia wydajność działania programu, eliminuje konieczność wykonywania dodatkowych operacji przez przeglądarkę i poprawia szybkość reagowania strony na poczynania użytkownika.

Nazwa zmiennej rozpoczyna się od znaku $ (jak w $answer), by zaznaczyć, że zawiera obiekt jQuery (czyli wynik zwrócony przez wywołanie $( )). Dodanie tego znaku nie jest konieczne, stanowi jedynie konwencję, którą powszechnie stosują programiści używający biblioteki jQuery, aby pokazać, że dana zmienna pozwala na stosowanie wszystkich cudownych funkcji jQuery, takich jak .hide().

Uwaga: Funkcja .next() jest tylko jedną spośród wielu funkcji jQuery (nazywanych także *metodami*), pozwalających na poruszanie się po DOM strony. Aby poznać więcej funkcji zaliczanych do tej kategorii, zajrzyj na stronę *http://docs.jquery.com/Traversing*. Dodatkowe informacje na ten temat znajdziesz w podrozdziale „Poruszanie się po DOM", na stronie 554.

6. **Do kodu dodaj pustą instrukcję `if` z klauzulą `else`:**

```
$(document).ready(function() {
  $('.answer').hide();
  $('.main h2').click(function() {
    var $answer = $(this).next('.answer');
    if ( ) {

    } else {

    }
  }); // Koniec funkcji click.
}); // Koniec funkcji ready.
```

Doświadczeni programiści zazwyczaj nie wpisują instrukcji warunkowych w taki sposób, jednak kiedy się uczysz, takie tworzenie kodu krok po kroku może być naprawdę użyteczne. A teraz może warto sprawdzić, czy odpowiedź jest ukryta?

7. **Wewnątrz nawiasów instrukcji warunkowej wpisz `$answer.is(':hidden')`:**

```
$(document).ready(function() {
  $('.answer').hide();
  $('.main h2').click(function() {
    var $answer = $(this).next('.answer');
    if ($answer.is(':hidden')) {

    } else {

    }
  }); // Koniec funkcji click.
}); // Koniec funkcji ready.
```

Jak widać, zastosowałeś tu zmienną `$answer`, utworzoną w kroku 5. Zmienna ta zawiera element należący do klasy `answer`, umieszczony bezpośrednio za znacznikiem `<h2>` kliknięty przez użytkownika. Pamiętasz zapewne, że jest to znacznik `<div>` zawierający odpowiedź na pytanie umieszczone w znaczniku `<h2>`.

Aby sprawdzić, czy element jest ukryty, w instrukcji warunkowej użyłeś funkcji `is()` jQuery. Metoda ta sprawdza, czy bieżący element pasuje do podanego selektora, przy czym w jej wywołaniu można przekazać dowolny selektor CSS lub jQuery. Jeśli element pasuje do selektora, funkcja zwraca wartość `true`, w przeciwnym razie zwracana jest wartość `false`. Coś wspaniałego! Doskonale wiesz, że do działania instrukcji warunkowych potrzebne są wartości logiczne `true` lub `false` (patrz strona 99).

Zastosowany w powyższym kodzie selektor `:hidden` jest specjalnym selektorem jQuery reprezentującym ukryte elementy. W tym przypadku sprawdzasz, czy odpowiedź jest ukryta.

8. **Do kodu dodaj szósty wiersz widoczny na poniższym przykładzie:**

```
$(document).ready(function() {
  $('.answer').hide();
  $('.main h2').click(function() {
    var $answer = $(this).next('.answer');
    if ($answer.is(':hidden')) {
      $answer.slideDown();
    } else {

    }
  }); // Koniec funkcji click.
}); // Koniec funkcji ready.
```

Funkcja `slideDown()` jest jedną z funkcji jQuery służących do tworzenia animacji (więcej na ich temat dowiesz się w następnym rozdziale). Wyświetla ona ukryty wcześniej element, wsuwając go od góry na właściwe miejsce. Teraz możesz już sprawdzić efekty swojej ciężkiej pracy. Zapisz stronę i wyświetl ją w przeglądarce. Kliknij jedno z widocznych pytań. Odpowiedź powinna się wysunąć na miejsce (jeśli tego nie zrobi, dokładnie sprawdź kod skryptu, możesz przy tym skorzystać ze wskazówek dotyczących rozwiązywania problemów, zamieszczonych na stronie 51).

Kiedy spojrzysz na stronę, zauważysz niebieski znak „+", umieszczony z lewej strony nagłówka. Taki „plusik" to popularna ikona, która zazwyczaj ma znaczenie: hej, tutaj jest coś więcej. Teraz, aby pokazać, że użytkownik może ukryć odpowiedź, będziesz musiał zastąpić znak plus znakiem minus. Możesz to zrobić w bardzo prosty sposób, dodając do znacznika `<h2>` jakąś klasę.

9. **Poniżej wiersza kodu dodanego w poprzednim kroku wpisz:**

```
$(this).addClass('close');
```

Pamiętasz zapewne, że `$(this)` reprezentuje element, do którego zostało skierowanie zdarzenie (patrz strona 189). W tym przypadku jest to znacznik `<h2>`. A zatem powyższy wiersz kodu dodaje do klikniętego znacznika klasę o nazwie `close`. Ikona znaku minus jest zdefiniowana w stylu tej klasy jako obraz tła. (I ponownie zastosowanie CSS pozwoliło uprościć kod JavaScript).

W następnym kroku dokończymy drugą połowę efektu tworzonego efektu wizualnego — czyli chowanie odpowiedzi po ponownym kliknięciu pytania.

10. **Wewnątrz klauzuli `else` instrukcji warunkowej dodaj dwa wiersze wyróżnione pogrubioną czcionką. Pełny kod przykładu powinien mieć następującą postać:**

```
<script src="../_js/jquery.min.js"></script>
<script>
$(document).ready(function() {
  $('.answer').hide();
  $('.main h2').click(function() {
    var $answer = $(this).next('.answer');
    if ($answer.is(':hidden')) {
      $answer.slideDown();
      $(this).addClass('close');
    } else {
      $answer.fadeOut();
      $(this).removeClass('close');
    }
  }); // Koniec funkcji click.
}); // Koniec funkcji ready.
```

Ten fragment kodu odpowiada za ukrycie odpowiedzi. Choć mogłeś zastosować funkcję `slideUp()`, która ukrywa element, wysuwając go ku górze, jednak w tym przypadku, w celu uatrakcyjnienia i urozmaicenia rozwiązania, element będzie stopniowo wygaszany, aż całkowicie zniknie. Taki efekt zapewnia funkcja `fadeOut()` (więcej informacji na jej temat możesz znaleźć na stronie 214).

Druga z dodanych instrukcji usuwa klasę `close` z elementu `<h2>`: w ten sposób przy pytaniu ponownie zostanie wyświetlony znak „+".

Zapisz stronę i wyświetl ją w przeglądarce. Kiedy klikniesz jedno z pytań, nie tylko poniżej pojawi się odpowiedź, lecz dodatkowo zmieni się wyświetlona obok ikona (patrz rysunek 5.7).

Uwaga: Kiedy już skończysz prace nad stroną, spróbuj zastąpić funkcję `slideDown()` funkcją `fadeIn()`, a funkcję `fadeOut()` funkcją `slideUp()`. Która z tych animacji podoba Ci się najbardziej?

6

ROZDZIAŁ

Animacje i efekty

Podczas lektury dwóch poprzednich rozdziałów zdobyłeś podstawową wiedzę dotyczącą stosowania biblioteki jQuery: wiesz, jak można dodawać plik jQuery do swoich stron WWW, wybierać elementy strony oraz reagować na zdarzenia związane z czynnościami wykonywanymi przez użytkownika, takimi jak kliknięcie przycisku bądź wskazanie odnośnika myszą. Większość programów wykorzystujących bibliotekę jQuery wymaga wykonania trzech kroków; są to wybór elementów strony, dołączenie do nich procedur obsługi zdarzeń oraz odpowiedzi na te zdarzenia poprzez wykonanie odpowiednich czynności. W tym rozdziale dowiesz się czegoś na temat „wykonywania odpowiednich czynności", a konkretnie — poznasz wbudowane w bibliotekę jQuery efekty wizualne oraz animacje. Przypomnisz sobie także nieco wiadomości związanych z kilkoma ważnymi właściwościami CSS, związanymi z tworzeniem efektów wizualnych. Dodatkowo dowiesz się także, jak stosować animacje CSS3 z narzędziami biblioteki jQuery, by (bardzo łatwo) tworzyć płynne efekty animacji.

Efekty biblioteki jQuery

Częstym zadaniem realizowanym przy użyciu języka JavaScript jest ukrywanie i wyświetlanie elementów stron WWW. Rozwijane menu nawigacyjne, etykietki ekranowe i automatyczne pokazy slajdów — wszystkie te rozwiązania bazują na możliwości ukrywania i wyświetlania wybranych elementów stron w odpowiednim momencie.

Aby zastosować każdy z takich efektów, należy użyć go na elemencie wybranym przy użyciu jQuery, podobnie jak robimy podczas stosowania wszelkich innych funkcji tej biblioteki. Aby na przykład ukryć wszystkie znaczniki należące do klasy `submenu`, wystarczy wykonać następujący wiersz kodu:

```
$('.submenu').hide();
```

Każda funkcja realizująca jakieś efekty wizualne umożliwia także podanie dwóch opcjonalnych argumentów: czasu odtwarzania efektu oraz *funkcji zwrotnej* (ang. *callback function*). Szybkość określa długość okresu czasu, jaki zajmie wykonanie

efektu, natomiast funkcja zwrotna określa kod, który zostanie wykonany po zakończeniu tego efektu. (Więcej informacji na temat funkcji zwrotnych można znaleźć na stronie 223).

W celu ustalenia szybkości efektu można podać jeden z trzech predefiniowanych łańcuchów znaków — `'fast'`, `'normal'` lub `'slow'` — bądź też liczbę określającą czas trwania efektu wyrażony w milisekundach (czyli wartość `1000` to sekunda, `500` — pół sekundy i tak dalej). Przykładowo kod ukrywający stopniowo element mógłby wyglądać tak:

```
$('element').fadeOut('slow');
```

Gdybyśmy chcieli, żeby element zanikał *naprawdę* wolno — przez 10 sekund — miałby następującą postać:

```
$('element').fadeOut(10000);
```

Podczas stosowania efektu w celu ukrycia elementu nie jest on usuwany ze strony. Wciąż jest dostępny w DOM — modelu obiektów dokumentu (patrz strona 145). Kod HTML elementu wciąż jest przechowywany w pamięci przeglądarki, jednak nie ma *żadnej* wizualnej reprezentacji (określanej za pomocą właściwości `display` CSS; ukrycie elementu jest możliwe poprzez przypisanie jej wartości `none`). Właśnie dzięki temu nie zajmuje żadnego widocznego miejsca w treści strony, przez co miejsce to mogą zająć inne elementy. Wszystkie dostępne efekty jQuery można poznać na stronie *effects.html*, w przykładach dołączonych do książki, w katalogu *testy* (patrz rysunek 6.1).

Uwaga: Słowa kluczowe służące do określania szybkości odtwarzania efektów wizualnych — `'fast'`, `'normal'` oraz `'slow'` — odpowiadają wartościom liczbowym `200`, `400` oraz `600`. A zatem wywołanie:

```
$('element').fadeOut('slow');
```

da dokładnie taki sam efekt jak wywołanie:

```
$('element').fadeOut(600);
```

Podstawowe wyświetlanie i ukrywanie

Biblioteka jQuery udostępnia trzy funkcje służące do prostego ukrywania i wyświetlania elementów. Oto one.

- **`show()`** — funkcja ta sprawia, że element będzie widoczny. Nie powoduje żadnej zmiany, jeśli element już jest widoczny. Jeśli w jej wywołaniu nie zostanie określona szybkość efektu, element będzie wyświetlony bezzwłocznie. Jeśli jednak szybkość efektu zostanie określona — `show(1000)` — to pojawianie się elementu jest animowane — będzie rozwijany od lewego górnego do prawego dolnego wierzchołka.
- **`hide()`** — ta funkcja służy do ukrywania elementów. Nie powoduje żadnej zmiany, jeśli element już jest niewidoczny. Podobnie jak w funkcji `show()`, także i ona ukrywa element bezzwłocznie, jeśli nie zostanie określona szybkość efektu. Jeśli jednak szybkość zostanie podana, element będzie animowany — stopniowo zmniejszany.

Rysunek 6.1. *Wszystkie efekty wizualne jQuery można przetestować przy użyciu strony effects.html umieszczonej w katalogu testy. Wystarczy kliknąć element fadeOut('#photo'), by zobaczyć, w jaki sposób tekst oraz obrazki stopniowo zanikają, są wysuwane ze strony lub powoli się na niej pojawiają. Niektóre z elementów będą szare, co oznacza, że nie można ich użyć w odniesieniu do danego elementu, na przykład nie ma większego sensu, by starać się wyświetlić obrazek, który już jest widoczny*

- **`toggle()`** — funkcja ta zmienia stan elementu, naprzemiennie go ukrywając lub wyświetlając. Jeśli element jest aktualnie widoczny, wywołanie funkcji `toggle()` spowoduje jego ukrycie; jeśli natomiast element jest ukryty, wywołanie funkcji spowoduje jego wyświetlenie. Funkcja idealnie nadaje się, gdy chcemy, by jeden element sterujący (taki jak przycisk) naprzemiennie pokazywał i ukrywał element.

W przykładzie przedstawionym na stronie 205 w rozdziale 5. mogłeś zobaczyć funkcję `hide()`w działaniu. Funkcja ta ukrywała wszystkie odpowiedzi w momencie wyświetlania strony.

Wygaszanie oraz rozjaśnianie elementów

Aby uzyskać bardziej spektakularny efekt, można wygaszać (ukrywać) oraz rozjaśniać (wyświetlać) elementy poprzez regulację stopnia ich nieprzezroczystości. Biblioteka jQuery udostępnia trzy funkcje realizujące operacje tego typu. Oto one.

- **`fadeIn()`** powoduje, że początkowo ukryty element stopniowo się pojawia. W pierwszym etapie na stronie zostanie przydzielone miejsce dla jeszcze niewidocznego elementu (co może się wiązać z przesunięciem innych elementów strony), a później wybrany element stopniowo będzie się pojawiał. Funkcja ta nie powoduje żadnych zmian, kiedy element już jest widoczny. Jeśli w jej wywołaniu nie zostanie określona szybkość efektu, element zostanie rozjaśniony przy użyciu szybkości `'normal'` (czyli w czasie 400 milisekund).
- **`fadeOut()`** — ta funkcja sprawia, że element będzie się stawał coraz słabiej widoczny — jak duch — aż w końcu zniknie. Nie powoduje żadnych wizualnych zmian, kiedy element już jest niewidoczny i, podobnie jak w przypadku funkcji `fadeIn()`, pominięcie argumentu określającego szybkość efektu sprawi, że zostanie on wykonany w czasie 400 milisekund. Funkcja ta była używana do ukrywania elementów strony w przykładzie listy często zadawanych pytań przedstawionym na stronie 205.
- **`fadeToggle()`** — łączy funkcje `fadeIn()` oraz `fadeOut()`. Jeśli w momencie wywoływania funkcji element jest niewidoczny, zostanie stopniowo rozjaśniony, jeśli natomiast początkowo jest widoczny — funkcja go wygasi. Funkcji tej można używać, by wyświetlać i ukrywać na stronie ramkę z instrukcjami dla użytkownika. Załóżmy na przykład, że na naszej stronie jest umieszczony przycisk *Instrukcje*. Kiedy użytkownik go kliknie, na stronie stopniowo pojawi się element `<div>` zawierający instrukcje; kolejne kliknięcie przycisku spowoduje wygaszenie tego elementu. Aby przy użyciu tej funkcji naprzemiennie wyświetlać i ukrywać element w czasie pół sekundy (500 milisekund), wystarczy skorzystać z następującego fragmentu kodu:

  ```
  $('#button').click(function() {
    $('#instructions').fadeToggle(500);
  }); // Koniec funkcji click.
  ```

- **`fadeTo()`** — ta funkcja działa nieco inaczej niż trzy poprzednie. Powoduje stopniową zmianę nieprzezroczystości elementu, aż do osiągnięcia określonej wartości, za jej pomocą można na przykład częściowo wygasić obrazek, tak że stanie się półprzezroczysty. W odróżnieniu od poprzednich, w wywołaniu tej funkcji trzeba podać szybkość efektu. Co więcej, konieczne jest także podanie drugiego argumentu z zakresu od 0 do 1, określającego docelowy stopień nieprzezroczystości. By na przykład zmienić stopień nieprzezroczystości elementu do poziomu 75%, należałoby użyć następującego wywołania:

  ```
  $('p').fadeTo('normal', .75);
  ```

  Funkcja ta zmienia poziom nieprzezroczystości elementu niezależnie od tego, czy jest widoczny na stronie, czy nie. Przykładowo załóżmy, że zmienimy nieprzezroczystość ukrytego elementu do poziomu 50%, w takim przypadku element pojawi się i będzie półprzezroczysty. Jeśli ukryjemy półprzezroczysty element, a następnie go ponownie wyświetlimy, poziom jego nieprzezroczystości będzie taki sam jak wcześniej.

  Jeśli w wywołaniu funkcji `fadeTo()` przekażemy wartość `0`, wybrany element będzie niewidoczny, choć wciąż będzie zajmował miejsce na stronie. Innymi słowy, w odróżnieniu od innych efektów powodujących ukrycie elementu, w przypadku jego całkowitego wygaszenia miejsce, które zajmuje na stronie, i tak pozostanie zajęte.

Jeśli wygasimy element, a następnie go ukryjemy, element ten zniknie ze strony. Jeśli później taki element ponownie wyświetlimy, będzie „pamiętał" ustawienie nieprzezroczystości, a zatem, choć przeglądarka go wyświetli, jego poziom nieprzezroczystości wyniesie 50%.

WIEDZA W PIGUŁCE

Bezwzględne pozycjonowanie przy użyciu CSS

Zazwyczaj, kiedy ukrywamy jakiś element strony, pozostałe jej elementy są przesuwane tak, by wypełnić zajmowane przez niego dotychczas miejsce. Jeśli na przykład ukryjesz obrazek, zniknie on ze strony, a umieszczony pod nim tekst zostanie przesunięty do góry. Podobnie wyświetlanie elementu powoduje przesunięcie pozostałej zawartości strony tak, by powstało dla niego miejsce. Może się zdarzyć, że nie będziemy chcieli, by zawartość strony skakała w górę bądź w dół. W takim przypadku możemy skorzystać z arkuszy stylów CSS oraz możliwości **bezwzględnego pozycjonowania**, by usunąć wybrany element z normalnego rozkładu treści strony. Innymi słowy, możemy sprawić, że znacznik `<div>`, akapit lub obrazek pojawią się ponad pozostałą zawartością strony, tak jakby były umieszczone na osobnej warstwie. Efekt ten możemy uzyskać, stosując właściwość `position`.

Aby element został wyświetlony ponad pozostałą zawartością strony, w stylu określającym jego wygląd należy użyć właściwości `position` i przypisać jej wartość `absolute`. Następnie można określić położenie elementu na stronie, korzystając z właściwości `left`, `right`, `top` oraz `bottom`. Załóżmy, że na naszej stronie znajduje się znacznik `<div>` zawierający formularz do logowania. Normalnie formularz ten nie będzie widoczny, kiedy jednak użytkownik kliknie odpowiedni odnośnik, zostanie on wsunięty na stronę i umieszczony ponad pozostałą zawartością strony, na jej środku. Element `<div>` można by umieścić w wybranym miejscu strony przy użyciu następującej reguły CSS:

```
#login {
    position: absolute;
    left: 536px;
    top: 0;
    width: 400px;
}
```

Powyższa reguła umieszcza element na górze okna przeglądarki, 536 pikseli na prawo od jej lewej krawędzi. Można także rozmieścić element, odnosząc się do prawej krawędzi strony — w tym przypadku należałoby skorzystać z właściwości `right` oraz względem jej dolnej krawędzi użyć właściwości `bottom`.

Oczywiście, może się także zdarzyć, że będziemy chcieli rozmieszczać elementy, uwzględniając inną zawartość strony, a nie okno przeglądarki, na przykład etykietki ekranowe są zazwyczaj rozmieszczane obok innych elementów. Przy jakimś słowie wyświetlanym na stronie może zostać umieszczony znak zapytania, którego kliknięcie otwiera niewielką ramkę zawierającą na przykład definicję tego słowa. W takim przypadku etykieta ekranowa nie powinna nie być rozmieszczona względem którejś z krawędzi okna przeglądarki, lecz obok odpowiedniego słowa. Wtedy położenie bezwzględnie pozycjonowanego elementu należy określić względem nadrzędnego elementu strony, wewnątrz którego się on znajduje. W ramach przykładu przeanalizujmy następujący kod HTML:

```
<span class="word">Hefalump
<span class="definition">Nieistniejące
➥w rzeczywistości stworzenie,
➥podobne do słonia występujące
➥w książkach o Kubusiu Puchatku</span>
</span>
```

Aby znacznik `<span>` zawierający definicję został wyświetlony poniżej słowa, musimy najpierw pozycjonować zewnętrzny znacznik `<span>` względnie, a dopiero później skorzystać z bezwzględnego pozycjonowania znacznika wewnętrznego:

```
.word { position: relative; }
.definition {
    position: absolute;
    bottom: -30px;
    left: 0;
    width: 200px;
}
```

Więcej informacji na temat pozycjonowania bezwzględnego można znaleźć na stronie *http://www.elated.com/articles/css-positioning/* bądź w książce *CSS3: The Missing Manual*[1].

[1] Wydanie polskie: *CSS3. Nieoficjalny podręcznik*. Wydanie III, Helion, Gliwice 2013 — *przyp. tłum.*

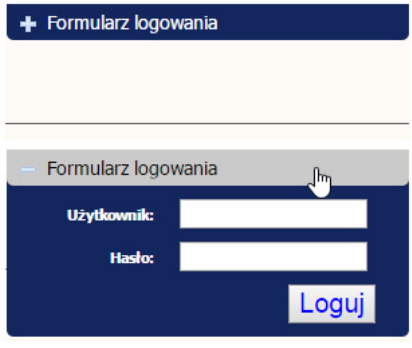
Przesuwanie elementów

Jeśli chcemy mieć na stronie nieco więcej ruchu, możemy użyć wbudowanych funkcji jQuery, by wysuwać i wsuwać elementy na stronę. Funkcje te działają podobnie do przedstawionych we wcześniejszym punkcie rozdziału, gdyż umożliwiają wyświetlanie i ukrywanie elementów oraz określanie czasu trwania efektu.

- **`slideDown()`** sprawia, że niewidoczne elementy są wsuwane na stronę. Najpierw pojawia się górna część elementu, a wszelkie inne elementy strony umieszczone poniżej niego zostają przesunięte w dół; następnie pojawia się pozostała część wyświetlanego elementu. Funkcja ta nie daje żadnego efektu, kiedy element już jest widoczny. Jeśli nie zostanie określona szybkość efektu, funkcja użyje wartości domyślnej `'normal'` (odpowiadającej 400 milisekundom). W przykładzie przedstawionym na stronie 208 funkcja ta została użyta do wyświetlania odpowiedzi na stronie.
- **`slideUp()`** usuwa element ze strony, ukrywając jego dolną część, a następnie przesuwając całą pozostałą zawartość strony ku górze, aż element całkowicie zniknie. Wywołanie tej funkcji nie powoduje żadnego widocznego efektu, kiedy element już jest ukryty. Podobnie jak w przypadku funkcji `slideDown()`, także i w tej, jeśli nie przekażemy argumentu określającego szybkość efektu, zostanie on wykonany w czasie 400 milisekund.
- **`slideToggle()`** — użycie tej funkcji powoduje wywołanie funkcji `slideDown()`, jeśli element jest aktualnie ukryty, oraz wywołanie funkcji `slideUp()`, jeśli aktualnie jest widoczny. Za pomocą tej funkcji można utworzyć na stronie jeden element sterujący (taki jak przycisk), którego klikanie będzie naprzemiennie wyświetlało i ukrywało element.

Przykład — wysuwany formularz logowania

W tym przykładzie nabierzesz nieco praktyki w stosowaniu efektów wizualnych udostępnianych przez bibliotekę jQuery; utworzysz popularny element interfejsu użytkownika, czyli panel, który się wysuwa i chowa po kliknięciu myszą odpowiedniego elementu strony (patrz rysunek 6.2).



Rysunek 6.2. *Najpierw go nie widać, a potem widać. Normalnie formularz jest ukryty (górna część rysunku), lecz kliknięcie nagłówka powoduje, że zostaje wyświetlony*

Zasada działania tego przykładu jest prosta.

1. **Pobierasz akapit zawierający nagłówek `"Formularz logowania"`.**

   Pamiętasz zapewne, że znaczna część programów jQuery rozpoczyna się od pobrania odpowiednich elementów strony. W tym przypadku interesuje nas akapit zawierający słowa `Formularz logowania`, który użytkownicy będą klikali.

2. **Dodajesz do akapitu procedurę obsługi zdarzeń `click`.**

   JavaScript nie zapewnia interaktywności bez wykorzystania zdarzeń: aby coś się stało, użytkownik musi rozpocząć interakcję z elementami strony (w naszym przypadku jest to akapit tekstu).

3. **Przełączasz widoczność formularza.**

   Poprzednie dwie czynności to jedynie przypomnienie (choć niezbędne w tak wielu programach pisanych z wykorzystaniem biblioteki jQuery). Jednak to właśnie w ramach tej ostatniej czynności wykorzystasz efekty jQuery opisywane w tym rozdziale. Możemy sprawić, by formularz pojawił się natychmiast (za pomocą funkcji `show()`), możemy wsunąć go (przy użyciu funkcji `slideDown()`) bądź stopniowo wyświetlić, gdy spowodujemy zmianę jego przezroczystość (z wykorzystaniem funkcji `fadeIn()`).

Uwaga: Informacje na temat pobierania przykładów dołączonych do tej książki można znaleźć na stronie 46.

Tworzenie kodu

1. **W edytorze tekstów otwórz plik *login.html* umieszczony w katalogu *R06*.**

   Plik zawiera już odwołanie do pliku jQuery oraz wywołanie funkcji `$(document).ready()` (patrz strona 190). Na początek zajmiesz się wybraniem akapitu zawierającego tekst `Formularz logowania`.

2. **Kliknij pusty wiersz widoczny poniżej funkcji `$(document).ready()` i wpisz w nim `$('#open')`.**

   Tekst `Formularz logowania` jest umieszczony wewnątrz akapitu znajdującego się wewnątrz odnośnika — `<a href= form.html ><p id= open >Formularz logowania</p></a>`. Ten odnośnik spowoduje wyświetlenie w przeglądarce nowej strony zawierającej formularz logowania. Odnośnik został zastosowany dlatego, gdyż inaczej użytkownicy, którzy wyłączyli w przeglądarce obsługę języka JavaScript, nie byliby w stanie zobaczyć ukrytego formularza. Dodając odnośnik, zapewniasz, że nawet osoby, które wyłączyły obsługę JavaScript, będą mogły dotrzeć do formularza logowania.

   Wyrażenie `$('#open')` powoduje wybranie akapitu tekstu. Teraz nadszedł już czas, by dodać procedurę obsługi zdarzeń.

Uwaga: Znacznik `<p>`, o którym była mowa w kroku 2 , jest umieszczony wewnątrz znacznika `<a>`. Osoby, które przez jakiś czas nie zajmowały się tworzeniem stron WWW, mogłyby przypuszczać, że taka konstrukcja jest nieprawidłowa. I owszem, tak właśnie było w języku HTML 4 i jego wcześniejszych wersjach. Jednak w przypadku zastosowania definicji typu dokumentu (`doctype`) HTML5 (patrz strona 21) umieszczanie elementów blokowych, takich jak `<p>`, `<h1>`, a nawet `<div>`, w odnośnikach jest prawidłowe. Takie rozwiązanie pozwala na tworzenie dużych, klikalnych obszarów stron.

3. **Dodaj kod wyróżniony w poniższym przykładzie pogrubioną czcionką, tak by miał następującą postać:**

```
$(document).ready(function() {
  $('#open').click(function(evt) {

  }); // Koniec funkcji click.
}); // Koniec funkcji ready.
```

Powyższy kod dodaje procedurę obsługi zdarzeń `click`, a zatem, za każdym razem gdy użytkownik kliknie akapit, coś się stanie. W naszym przypadku kliknięcie powinno powodować wyświetlenie formularza, który zniknie po ponownym kliknięciu; następne kliknięcie ponownie powinno go wyświetlić i tak w dalej. Innymi słowy, formularz ma być naprzemiennie wyświetlany i ukrywany. Biblioteka jQuery udostępnia trzy funkcje zapewniające takie możliwości: `toggle()`, `fadeToggle()` oraz `slideToggle()`. Jedyna różnica pomiędzy nimi polega na sposobie wyświetlania i ukrywania wybranego elementu.

Funkcja anonimowa umieszczona wewnątrz wywołania metody `click()` ma jeden argument. Zgodnie z informacjami podanymi na stronie 194, do każdego zdarzenia jest automatycznie przekazywany obiekt zawierający różne właściwości i metody. Będziesz potrzebował tego obiektu, by poinformować jQuery, że powinna uniemożliwić przeglądarce standardowe obsłużenie odnośnika i wyświetlenie strony z formularzem do logowania.

4. **Kliknij pusty wiersz wewnątrz funkcji `click()` i wpisz w nim**:

```
evt.preventDefault();
```

W tym przypadku wywołanie metody `preventDefault()` sprawi, że przeglądarka nie przejdzie na stronę wskazywaną przez odnośnik, wewnątrz którego został umieszczony akapit tekstu. Pamiętaj, że odnośnik został umieszczony w kodzie po to, by zapewnić możliwość wyświetlenia formularza do logowania osobom, które wyłączyły w przeglądarce obsługę języka JavaScript. Jednak we wszystkich pozostałych przypadkach musisz nakazać przeglądarce, by pozostała na tej samej stronie i wykonała dodatkowy kod JavaScript.

5. **W kolejnym wierszu kodu wpisz następujące wywołanie**:

```
$('#login form').slideToggle(300);
```

Ten kod powoduje pobranie formularza oraz wsunięcie go na stronę, jeśli nie jest widoczny, bądź jego ponowne wysunięcie i ukrycie. Kolejną czynnością będzie zmiana klasy używanej w akapicie, dzięki czemu akapit tekstu zmieni wygląd.

6. **Dodaj poniższy, wyróżniony pogrubieniem fragment kodu, by końcowy skrypt wyglądał tak, jak ten:**

```
1  $(document).ready(function() {
2    $('#open').click(function(evt) {
3      evt.preventDefault();
4      $('#login form').slideToggle(300);
5      $(this).toggleClass('close');
6    }); // Koniec funkcji click.
7  }); // Koniec funkcji ready.
```

Po przeczytaniu informacji zamieszczonych na stronie 189 wiesz już, że wewnątrz procedury obsługi zdarzeń można korzystać z wyrażenia `$(this)`, by odwołać się do elementu odpowiadającego na zdarzenie. W naszym przykładzie `$(this)` odwołuje się do akapitu kliknięty przez użytkownika — pobranego w 2. wierszu przy użyciu wywołania `$('#open')`. Funkcja `toggleClass()` dodaje lub usuwa klasę z elementu. Podobnie jak wszystkie inne funkcje przełączające, także i `toggleClass()` dodaje podaną nazwę klasy, jeśli jeszcze nie jest używana w wybranym elemencie, bądź też usuwa ją, kiedy aktualnie jest stosowana. W naszym przykładzie korzystamy z klasy o nazwie `close`, zdefiniowanej w arkuszu stylów umieszczonym na stronie. (Znajdziesz go w kodzie strony, wewnątrz sekcji `<head>`).

7. **Zapisz stronę i wyświetl ją w przeglądarce.**

   Koniecznie kliknij kilka razy akapit z tekstem `Formularz logowania`, by przekonać się, jak działa. Gotową wersję strony znajdziesz w pliku *complete_login.html*, w katalogu *R06*. Możesz także wypróbować inne efekty wizualne, zastępując funkcję `slideToggle()` funkcjami `toggle()` lub `fadeToggle()`.

A co zrobić, gdy będziesz chciał zastosować dwa różne efekty wizualne? Chcesz na przykład wysuwać formularz podczas jego wyświetlania, a wygaszać podczas ukrywania. W takim przypadku kod przedstawiony w kroku 5. powyższej listy nie spełni oczekiwań, gdyż funkcja `click()` nie zapewnia możliwości stosowania i wyboru jednej z dwóch różnych akcji. Musisz zatem zastosować tę samą sztuczkę, którą wykorzystałeś w przykładzie z listą pytań i odpowiedzi, przedstawionym w poprzednim rozdziale (na stronie 204): kiedy użytkownik kliknie odnośnik `Formularz logowania`, musisz sprawdzić, czy formularz jest ukryty. Jeśli jest, to go wyświetlisz, jeśli nie jest, to go ukryjesz.

Aby formularz został wsunięty na stronę, a następnie wygaszony podczas kolejnego kliknięcia, powinieneś użyć następującego kodu:

```
$(document).ready(function() {
  $('#open').click(function(evt) {
    evt.preventDefault();
    if ($('#login form').is(':hidden')) {
      $('#login form').fadeIn(300);
      $(this).addClass('close');
    } else {
      $('#login form').slideUp(600);
      $(this).removeClass('close');
    }
  }); // Koniec funkcji click.
}); // Koniec funkcji ready.
```

Uwaga: Powyższy kod możesz znaleźć w pliku *complete_login2.html* w katalogu *R06*.

Animacje

Nasze możliwości nie ograniczają się wyłącznie do wbudowanych efektów jQuery. Korzystając z funkcji `animate()`, można animować dowolną właściwość CSS akceptującą wartości liczbowe, na przykład wyrażone w pikselach, w jednostkach em lub wartości procentowe. Przykładowo można animować wielkość tekstu, położenie elementu na stronie, jego przezroczystość czy też szerokość obramowania.

Uwaga: Biblioteka jQuery nie umożliwia samodzielnego animowania kolorów — na przykład koloru tekstu, tła czy też krawędzi elementu. Jednak biblioteka jQuery UI udostępnia wiele dodatkowych efektów animacji, w tym także możliwość animowania kolorów. Informacje na temat możliwości tworzenia animacji w bibliotece jQuery UI znajdziesz w rozdziale 12.

Aby skorzystać z tej funkcji, należy przekazać w jej wywołaniu literał obiektowy (patrz strona 165) zawierający listę właściwości CSS, które mają być animowane, oraz ich wartości docelowych. Załóżmy, że animacja elementu ma polegać na przesunięciu go w miejsce oddalone o 650 pikseli od lewej krawędzi strony, zmianie poziomu jego nieprzezroczystości do wartości 50% oraz powiększeniu używanej w nim czcionki do 24 pikseli. Poniższy fragment kodu definiuje obiekt zawierający te właściwości i wartości:

```
{
    left: '650px',
    opacity: .5,
    fontSize: '24px'
}
```

Warto zwrócić uwagę, że wartości należy zapisywać w apostrofach wyłącznie wtedy, gdy zawierają jednostki miary, takie jak px, em lub %. W naszym przykładzie wartość `'650px'` musimy zatem zapisać w apostrofach, gdyż zawiera jednostkę `'px'`; natomiast w przypadku wartości .5, przypisywanej właściwości `opacity`, nie musimy tego robić, gdyż nie zawiera ona żadnych dodatkowych liter ani znaków. Opcjonalne jest także zapisywanie w apostrofach nazw właściwości (takich jak `left`, `opacity` oraz `fontSize`).

Uwaga: JavaScript nie pozwala na zapisywanie nazw właściwości CSS z łącznikami, na przykład `font-size` jest prawidłową nazwą właściwości CSS, jednak język JavaScript nie zrozumiałby jej prawidłowo, gdyż łącznik ma w nim specjalne znaczenie — jest używany jako operator odejmowania. A zatem podczas podawania nazw właściwości CSS w kodzie JavaScript należy pomijać łączniki, a pierwszy znak słowa umieszczonego po łączniku zapisać wielką literą. Przykładowo `font-size` należy zapisać jako `fontSize`, a `border-left-width` jako `borderLeftWidth`.

Jeśli jednak chcesz posługiwać się nazwami właściwości CSS (by uniknąć niepotrzebnego zamieszania), musisz je zapisywać w apostrofach, jak w poniższym przykładzie:

```
{
    'font-size': '24px',
    'border-left-width': '2%'
}
```

Załóżmy, że chcemy animować element o identyfikatorze `message` przy użyciu powyższych ustawień. W takim przypadku moglibyśmy wywołać funkcję `animate()` w następujący sposób:

```
$('#message').animate(
  {
    left: '650px',
    opacity: .5,
    fontSize: '24px'
  },
  1500
);
```

Funkcja `animate()` może przyjmować kilka argumentów. Pierwszym z nich jest literał obiektowy zawierający właściwości CSS, które chcemy animować. Drugim jest liczba określająca czas trwania animacji (wyrażona w milisekundach). W powyższym przykładzie animacja ma trwać 1500 milisekund, czyli 1,5 sekundy.

Uwaga: Jeśli chcemy animować położenie elementu, używając przy tym właściwości `left`, `right`, `top` oraz `bottom`, konieczne jest także przypisanie jego właściwości CSS `position` wartości `absolute` lub `relative`. Tylko w tych dwóch przypadkach możliwe jest określanie położenia elementu (patrz ramka na stronie 215).

Istnieje także możliwość określania docelowych wartości animowanych właściwości w odniesieniu do ich wartości bieżących. Służy do tego zapis += oraz -=. Załóżmy na przykład, że chcemy animować element, przesuwając go o 50 pikseli w prawo, za każdym razem gdy zostanie kliknięty. Oto sposób, w jaki można to zrobić:

```
$('#moveIt').click(function() {
  $(this).animate(
    {
      left:'+=50px'
    },
    1000); // Koniec funkcji animate.
}); // Koniec funkcji click.
```

Tempo animacji

Wszystkie funkcje jQuery tworzące efekty wizualne (`slideUp()`, `fadeIn()` i tak dalej) oraz funkcja `animate()` pozwalają na podanie dodatkowego argumentu kontrolującego *tempo animacji* (ang. *easing*) — określa on szybkość zmian podczas różnych etapów animacji. Przykładowo w czasie przesuwania elementu na stronie można zażądać, by ruch elementu początkowo był wolny, następnie przyspieszył, a w końcu, gdy zbliżał się będzie koniec animacji, ponownie zwolnił. Określanie tempa animacji sprawia, że efekty wizualne mogą być naprawdę interesujące i dynamiczne.

Biblioteka jQuery udostępnia dwie metody określania tempa animacji: `linear` (liniowa) oraz `swing` (zmienna). Pierwsza z nich zapewnia stałe tempo animacji, a zatem każdy jej krok będzie identyczny (jeśli na przykład przesuwamy element na ekranie, każdy krok animacji będzie go przesuwał o identyczny odcinek). Metoda `'swing'` jest nieco bardziej dynamiczna, gdyż animacja rozpoczyna się szybciej, a następnie zwalnia. Metoda ta jest stosowana domyślnie, a zatem, jeśli jawnie nie podamy sposobu określania tempa animacji, jQuery użyje właśnie jej.

Metoda określania tempa animacji jest podawana jako drugi argument funkcji tworzących efekty wizualne; aby zatem element został wysunięty ze strony ze stałą szybkością, możemy to zrobić przy użyciu następującego wywołania:

```
$('#element').slideUp(1000,'linear');
```

W funkcji `animate()` metoda określania tempa animacji jest podawana jako trzeci argument wywołania, za literałem obiektowym z właściwościami oraz całkowitym czasem trwania animacji. Aby na przykład wykorzystać metodę `linear` w wywołaniu funkcji `animate()` przedstawionym na stronie 221, należałoby użyć następującego kodu:

```
$('#message').animate(
{
  left: '650px',
  opacity: .5,
  fontSize: '24px'
},
1500,
'linear'
);
```

Nasze możliwości nie ograniczają się do stosowania tylko tych dwóch domyślnych metod, udostępnianych przez bibliotekę jQuery. Dzięki pracowitości innych programistów można także korzystać z licznej grupy innych metod — niektóre z nich zapewniają bardzo dramatyczne i interesujące efekty wizualne. Przykładowo biblioteka jQuery UI zawiera wiele dodatkowych sposobów określania tempa animacji. Poznasz ją dokładniej w trzeciej części książki, jednak nie ma powodu, byś już teraz nie mógł zacząć używać jej do poprawienia atrakcyjności tworzonych animacji.

Aby skorzystać z biblioteki jQuery UI (będącej zwyczajnym zewnętrznym plikiem JavaScript), trzeba dodać jej plik do strony tuż za znacznikiem dołączającym bibliotekę jQuery. Po dołączeniu biblioteki można będzie używać udostępnianych przez nią metod określania tempa animacji (ich pełną listę można znaleźć na stronie *http://api.jqueryui.com/easings/*). Załóżmy na przykład, że chcemy, by po kliknięciu element `div` umieszczony na stronie został powiększony. Aby dodatkowo animacja była bardziej interesująca, chcemy określać jej tempo metodą `easeInBounce`. Jeśli założymy, że animowany element ma identyfikator `animate`, kod strony realizujący taką animację może wyglądać następująco:

```
1   <script src="_js/jquery.min.js"></script>
2   <script src="_js/jquery-ui.min.js"></script>
3   <script>
4   $(document).ready(function() {
5     $('#animate').click(function() {
6       $(this).animate(
7         {
8           width: '400px',
9           height: '400px'
10        },
11        1000,
12        'easeInBounce'); // Koniec funkcji animate.
13    }); // Koniec funkcji click.
14  }); // Koniec funkcji ready.
15  </script>
```

Wiersze 1. oraz 2. powodują dołączenie do strony bibliotek jQuery oraz jQuery UI. W wierszu 4. znajduje się wszechobecna funkcja `ready()` (przedstawiona na stronie 190), natomiast w wierszu 5. przypisujemy interesującemu nas elementowi `div` procedurę obsługi zdarzeń `click`. Najważniejszym fragmentem tego przykładu jest jednak kod zapisany w wierszach od 6. do 12. Jak zapewne pamiętasz (była o tym mowa na stronie 189), wewnątrz procedury obsługi zdarzeń używane jest wyrażenie `$(this)`, pozwalające odwołać się do elementu odpowiadającego na zdarze-

nie — w naszym przypadku jest to znacznik . Innymi słowy, w odpowiedzi na kliknięcie tego elementu rozpoczynamy animację powodującą zmianę jego szerokości i wysokości (wiersze 8. i 9.). W wierszu 11. określamy, że animacja ma trwać 1 sekundę (czyli 1000 milisekund), a w wierszu 12. podajemy, że używaną metodą zmiany tempa ma być `easeInBounce` (zamiast niej można jednak wybrać dowolną inną spośród dostępnych metod, na przykład `easeInOutSine` bądź `easeInCubic`).

Uwaga: Powyższy fragment kodu można znaleźć w przykładach dołączonych do książki, umieszczonych w katalogu *R06*. Wystarczy otworzyć w przeglądarce plik *easing_example1.html*. Przykład *easing_example2.html* pokazuje, w jaki sposób można wykorzystać zdarzenie `toggle()`, by w jednym elemencie stosować dwa różne sposoby animacji: po pierwszym kliknięciu zostanie użyty pierwszy sposób, a po kolejnym — drugi.

Wykonywanie operacji po zakończeniu efektu

Czasami może się zdarzyć, że będziemy chcieli wykonać jakieś operacje po zakończeniu efektu. Załóżmy, że chcemy, by po stopniowym wyświetleniu na stronie jakiegoś obrazka poniżej niego został pokazany podpis. Zazwyczaj efekty nie są wykonywane jeden po drugim — ich realizacja rozpoczyna się dokładnie w momencie wywołania. A zatem, jeśli w naszym kodzie jeden wiersz odpowiada za stopniowe wyświetlenie obrazka, a następny za wyświetlenie podpisu, podpis pojawi się jeszcze w czasie wyświetlania obrazka.

Aby wykonać jakąś czynność po zakończeniu odtwarzania efektu, do funkcji generującej efekt można przekazać dodatkową *funkcję zwrotną*. Zostanie ona wykonana dopiero po zakończeniu prezentowania efektu. Zazwyczaj jest przekazywana jako drugi argument większości funkcji generujących efekty wizualne jQuery (trzeci argument funkcji `fadeTo()`).

Załóżmy, że na naszej stronie znajduje się obrazek o identyfikatorze `photo`, a poniżej niego akapit tekstu o identyfikatorze `caption`. Aby stopniowo wyświetlić obrazek, a następnie, w podobny sposób, wyświetlić jego podpis, musimy skorzystać z funkcji zwrotnej w następujący sposób:

```
$('#photo').fadeIn(1000, function() {
  $('#caption').fadeIn(1000);
});
```

Oczywiście, gdybyśmy chcieli wykonać taką funkcję w momencie wczytywania strony, najpierw zapewne ukrylibyśmy zarówno obrazek, jak i jego podpis, a dopiero później wywołali funkcję `fadeIn()`:

```
$('#photo, #caption').hide();
$('#photo').fadeIn(1000, function() {
      $('#caption').fadeIn(1000);
});
```

Podczas stosowania funkcji `animate()` funkcja zwrotna określana jest jako ostatnia, za wszelkimi innymi argumentami — obiektem zawierającym animowane właściwości CSS, czasem trwania animacji oraz metodą określania jej tempa. Metoda określania tempa animacji jest jednak argumentem opcjonalnym, a zatem do funkcji `animate()` wystarczy przekazać jedynie literał obiektowy z właściwościami,

czas trwania efektu oraz funkcję zwrotną. Przykładowo załóżmy, że nie interesuje nas jedynie stopniowe wyświetlenie obrazka, lecz jednocześnie powiększenie go od zera do pełnej wielkości (tworząc w ten sposób coś, co mogłoby przypominać efekt powiększania). W takim przypadku wszystkie zamierzone operacje można wykonać przy użyciu wywołania funkcji `animate()` przedstawionego w poniższym przykładzie:

```
1   $('#photo').width(0).height(0).css('opacity',0);
2   $('#caption').hide();
3   $('#photo').animate(
4      {
5         width: '200px',
6         height: '100px',
7         opacity: 1
8      },
9      1000,
10     function() {
11        $('#caption').fadeIn(1000);
12     }
13  ); // Koniec funkcji animate.
```

W 1. wierszu powyższego przykładu została przypisana wartość `0` właściwościom `width`, `height` oraz `opacity` obrazka. (W ten sposób obrazek jest ukrywany, a jednocześnie zostaje przygotowany do rozpoczęcia animacji). Wiersz 2. odpowiada za ukrycie elementu zawierającego podpis pod obrazkiem. Wiersze od 3. do 13. zawierają wywołanie funkcji `animate()` wraz z umieszczoną w wierszach od 10. do 12. funkcją zwrotną. Cały ten kod może się wydawać nieco przerażający, jednak funkcja zwrotna jest jedynym sposobem wykonania operacji (a także kolejnego efektu wizualnego operującego na zupełnie innym elemencie strony) po zakończeniu animacji.

Uwaga: Powyższy przykład można znaleźć w pliku *callback.html* umieszczonym w przykładach dołączonych do książki, w katalogu *R06*.

Funkcje zwrotne mogą się stać bardzo kłopotliwe, jeśli będziemy chcieli animować kilka elementów jeden po drugim; na przykład gdybyśmy chcieli przesunąć obrazek na środek strony, następnie stopniowo wyświetlić pod nim podpis, by w końcu ukryć oba elementy. Aby utworzyć taką animację, konieczne byłoby przekazanie jednej funkcji zwrotnej wewnątrz innej funkcji zwrotnej, w sposób podobny do przedstawionego poniżej:

```
$('#photo').animate(
   {
      left: '+=400px',
   },
   1000,
   function() { // Pierwsza funkcja zwrotna.
      $('#caption').fadeIn(1000,
       function() { // Druga funkcja zwrotna.
          $('#photo, #caption').fadeOut(1000);
      } // Koniec drugiej funkcji zwrotnej.
     ); // Koniec funkcji fadeIn.
   } // Koniec pierwszej funkcji zwrotnej.
); // Koniec funkcji animate.
```

Uwaga: Powyższy przykład można znaleźć w pliku *multiple-callbacks.html* umieszczonym w przykładach dołączonych do książki, w katalogu *R06*.

Jeśli do tego samego elementu strony chcemy dodać kolejne animacje, nie musimy stosować funkcji zwrotnych. Przykładowo załóżmy, że chcemy wsunąć obrazek na stronę, a następnie stopniowo go wygasić. W takim przypadku wystarczy użyć funkcji `animate()`, by przesunąć obrazek w odpowiednie miejsce, a następnie wywołać funkcję `fadeOut()`, by go wygasić. Można to zrobić, używając następującego fragmentu kodu:

```
$('#photo').animate(
   {
      left: '+=400px',
   },
   1000
); // Koniec funkcji animate.
$('#photo').fadeOut(3000);
```

W tym przypadku, choć przeglądarka wykona ten kod natychmiast, jQuery umieści oba efekty w kolejce, dzięki czemu najpierw zostanie wykonana funkcja `animate()`, a dopiero potem funkcja `fadeOut()`. Korzystając z techniki tworzenia sekwencji wywołań, charakterystycznej dla biblioteki jQuery (opisanej na stronie 156), powyższy kod można by zapisać w następującej postaci:

```
$('#photo').animate(
   {
      left: '+=400px',
   },
   1000).fadeOut(3000);
```

Gdybyśmy chcieli najpierw rozjaśnić obrazek, następnie go wygasić i, w końcu, ponownie rozjaśnić, korzystając przy tym z techniki tworzenia sekwencji wywołań, moglibyśmy to zrobić w następujący sposób:

```
$('#photo').fadeIn(1000).fadeOut(2000).fadeIn(250);
```

Uwaga: Więcej informacji na temat sposobu działania mechanizmu kolejkowania efektów wizualnych w bibliotece jQuery można znaleźć na stronie *http://api.jquery.com/jQuery.queue/*.

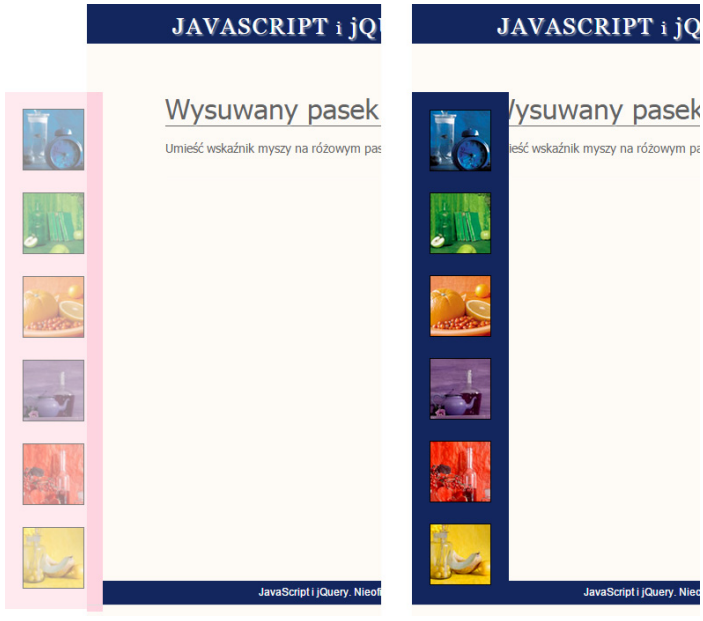
Dodatkową funkcją, która może się przydać podczas tworzenia kolejek efektów wizualnych, jest `delay()`. Powoduje ona odczekanie określonego okresu czasu (wyrażonego w milisekundach) przed rozpoczęciem kolejnego efektu zapisanego w kolejce. Załóżmy, że chcemy stopniowo wyświetlić obrazek, odczekać 10 sekund, a następnie go wygasić. Korzystając z funkcji `delay()`, możemy to zrobić w następujący sposób:

```
$('#photo').fadeIn(1000).delay(10000).fadeOut(250);
```

Przykład — animowany pasek ze zdjęciami

W tym przykładzie użyjemy funkcji `animate()` do przesunięcia znacznika `<div>`, który początkowo jest ukryty poza lewą krawędzią strony. Znacznik ten jest pozycjonowany w sposób bezwzględny (więcej informacji na temat tego sposobu pozycjonowania można znaleźć w ramce na stronie 215), a zatem większa część jego obszaru pozostaje niewidoczna — ukryta za lewą krawędzią strony i okna przeglądarki (co widać po lewej stronie rysunku 6.3). Kiedy użytkownik umieści wskaźnik myszy w widocznym obszarze znacznika `<div>`, zostanie on wysunięty w prawo, tak by stał

się w całości widoczny (co widać z prawej strony rysunku 6.3). Aby cały efekt był bardziej zabawny, zastosujemy dodatkowo wtyczkę pozwalającą na animowanie koloru tła oraz kilka różnych sposobów określania tempa animacji.



Rysunek 6.3. *Bardzo ciekawe efekty można uzyskać, chowając różne elementy strony poza krawędziami okna przeglądarki (tak jak pokazany z lewej strony rysunku element div, którego większa część jest niewidoczna). Korzystając z funkcji animate(), można przesuwać takie elementy, tak by były widoczne w całości (prawa część rysunku)*

Uwaga: Informacje na temat pobierania przykładów dołączonych do tej książki można znaleźć na stronie 46.

Podstawowe zadania, jakie należy wykonać, są raczej proste. Oto one.

1. **Pobranie znacznika `<div>`.**

   Pamiętasz zapewne, że bardzo wiele programów używających biblioteki jQuery zaczyna działanie od pobrania odpowiednich elementów strony — w tym przypadku będzie to znacznik `<div>`, który użytkownik ma wskazać myszą.

2. **Dołączenie procedury obsługi zdarzeń `hover`.**

   Zdarzenie `hover` (opisane na stronie 192) jest specjalną funkcją jQuery, a nie faktycznym zdarzeniem języka JavaScript. Pozawala ono na wykonywanie pierwszego zbioru czynności, w momencie gdy użytkownik wskaże element myszą, oraz drugiego, gdy użytkownik usunie wskaźnik myszy z obszaru elementu (w rzeczywistości zdarzenie to jest kombinacją zdarzeń `mouseEnter` oraz `mouseLeave`).

3. **Dodanie wywołania funkcji `animate()` w procedurze obsługi zdarzenia `mouseEnter`.**

   Kiedy użytkownik wskaże znacznik `<div>` myszą, przesuniesz go w prawo, tak że pojawi się w całości, wysuwając się zza lewej krawędzi okna przeglądarki. Dodatkowo dodasz animację koloru tła znacznika.

4. **Dodanie kolejnego wywołania funkcji `animate()` w ramach obsługi zdarzenia `mouseLeave`.**

   Kiedy użytkownik usunie wskaźnik myszy z obszaru znacznika, przesuniesz go z powrotem w początkowe położenie i zmienisz kolor tła na oryginalny.

Tworzenie kodu

1. **W edytorze tekstów otwórz plik *animate.html* umieszczony w katalogu *R06*.**

   Przygotowany plik już zawiera odwołanie do biblioteki jQuery oraz wywołanie funkcji `$(document).ready()` (patrz strona 190). Ponieważ jednak mamy zamiar animować kolor tła elementu oraz wykorzystać kilka interesujących metod określania tempa animacji, konieczne jest także dołączenie dwóch dodatkowych wtyczek jQuery, takich jak *color* oraz *easing*.

2. **Kliknij pusty wiersz umieszczony poniżej pierwszego znacznika `<script>` i dodaj wyróżniony pogrubieniem kod przedstawiony poniżej:**

```
<script src="../_js/jquery.min.js"></script>
<script src="../_js/jquery-ui.min.js"></script>
```

   jQuery UI jest *wtyczką* biblioteki jQuery. W świecie jQuery wtyczki są zwykłymi zewnętrznymi plikami JavaScript rozszerzającymi możliwości biblioteki, które często pozwalają dodawać do tworzonych witryn złożone efekty lub możliwości funkcjonalne bez konieczności samodzielnego pisania rozbudowanego kodu. Kolejnym zadaniem będzie wybranie odpowiedniego znacznika `<div>` i wywołanie funkcji `hover()`.

3. **Kliknij pusty wiersz umieszczony wewnątrz funkcji `$(document).ready()` i wpisz w nim `$('#dashboard').hover(); // Koniec funkcji hover.`, by kod wyglądał tak, jak przedstawiony poniżej**:

```
$(document).ready(function() {
  $('#dashboard').hover(); // Koniec funkcji hover.
}); // Koniec funkcji ready.
```

   Wywołanie `$('#dashboard')` powoduje pobranie znacznika `<div>` (o identyfikatorze `dashboard`). Funkcja `hover()` wymaga podania dwóch argumentów — dwóch funkcji anonimowych (patrz strona 193) — opisujących, co należy zrobić w momencie umieszczenia wskaźnika myszy w obszarze elementu oraz w momencie jego usunięcia. Zamiast wpisywać cały kod za jednym razem, utworzymy go etapami. Najpierw dodamy samą funkcję `hover()`, a następnie dwie, początkowo puste funkcje anonimowe. Takie rozwiązanie jest bardzo przydatne, bo kiedy nie zachowamy należytej uwagi, ogromna liczba zagnieżdżonych nawiasów, nawiasów klamrowych, przecinków i średników może być przytłaczająca.

4. **Kliknij pomiędzy nawiasami w wywołaniu funkcji `hover()` i dodaj dwie puste funkcje anonimowe:**

```
$(document).ready(function() {
  $('#dashboard').hover(
    function() {

    },
    function() {
```

```
    }
  ); // Koniec funkcji hover.
}); // Koniec funkcji ready.
```

Kiedy już dokończysz pisanie tego wywołania funkcji `.hover()`, będzie ono wyglądało na naprawdę skomplikowane (patrz krok 12.). Jednak w rzeczywistości jest to jedynie funkcja przyjmująca dwa argumenty będące funkcjami. Dobrym rozwiązaniem jest rozpoczęcie od wstawienia dwóch pustych, anonimowych funkcji, dzięki czemu jeszcze będziesz miał pewność, że struktura kodu jest prawidłowa.

Wskazówka: Dobrym pomysłem jest możliwie jak najczęstsze testowanie kodu, by upewnić się, że nie zawiera żadnych prostych błędów typograficznych. W kroku 4., w pierwszej funkcji anonimowej można wpisać na przykład `console.log('mouseOver')`, a w drugiej — `console.log('mouseLeave')`, a następnie wyświetlić stronę w przeglądarce. Aby zobaczyć efekty, będziesz musiał otworzyć konsolę JavaScript przeglądarki; w tym celu w Internet Explorerze naciśnij klawisz *F12*; w Chrome naciśnij kombinację klawiszy *Ctrl+Shift+J* (w systemie Windows) lub ⌘*+Option+J* (w systemie Mac OS), w przeglądarce Firefox naciśnij kombinację klawiszy *Ctrl+Shift+K* (w systemie Windows) lub ⌘*+Option+K* (w systemie Mac OS), a w przeglądarce Safari naciśnij kombinację klawiszy ⌘*+Option+C*. Kiedy umieścisz wskaźnik myszy w obszarze elementu `div`, w oknie konsoli powinien się pojawić komunikat `mouseOver`, a kiedy usuniesz wskaźnik myszy z elementu — komunikat `mouseLeave`. Jeśli w oknie konsoli nie zostaną wyświetlone żadne okienka informacyjne, będzie to oznaczało, że gdzieś popełniłeś błąd. W takim przypadku ponownie sprawdź kod strony bądź też wykonaj opisane na stronie 51 czynności pozwalające na odszukanie przyczyny błędu przy użyciu konsoli JavaScript.

5. **Wewnątrz pierwszej funkcji anonimowej wpisz: `$(this).animate(); // Koniec funkcji animate..`**

   Zgodnie z informacjami podanymi na stronie 189, wyrażenie `$(this)` używane wewnątrz procedur obsługi zdarzeń odwołuje się do elementu, z którym procedura została skojarzona. W tym przypadku wyrażenie to odwołuje się do znacznika `<div>` o identyfikatorze `dashboard`. Innymi słowy, przesuwanie wskaźnika myszy w obszarze elementu także będzie powodować jego animację.

6. **Dodaj literał obiektowy określający właściwości CSS, które chcesz animować:**

```
$(document).ready(function() {
  $('#dashboard').hover(
    function() {
      $(this).animate(
        {
          left: '0',
          backgroundColor: 'rgb(27,45,94)'
        }
      ); // Koniec funkcji animate.
    },
    function() {

    }
  ); // Koniec funkcji hover.
}); // Koniec funkcji ready.
```

   Pierwszym argumentem wywołania funkcji `animate()` jest literał obiektowy (patrz strona 165) określający animowane właściwości CSS. W naszym przypadku aktualna wartość właściwości `left` znacznika `<div>` wynosi `-92px`, co oznacza, że jego znacząca część jest ukryta poza lewą krawędzią okna przeglądarki. Jeśli w ramach animacji zmienimy wartość tej właściwości na `0`, w efekcie

przesuniemy element w prawo i w całości wyświetlimy go na stronie. Podobnie, dzięki zastosowaniu wtyczki *color*, możemy płynnie zmienić kolor tła tego znacznika z różowego na niebieski. Kolejną czynnością będzie określenie czasu trwania animacji.

7. **Za zamykającym nawiasem klamrowym } wpisz przecinek, naciśnij klawisz *Enter* i wpisz `500`.**

   Przecinek oznacza koniec pierwszego argumentu przekazywanego w wywołaniu funkcji `animate()`, natomiast liczba `500` określa czas trwania animacji (wyrażony w milisekundach). Teraz możesz podać metodę określania tempa animacji.

8. **Za cyfrą `500` wpisz przecinek, naciśnij klawisz *Enter* i wpisz `'easeInSine'`, tak by kod wyglądał tak samo jak przedstawiony poniżej:**

```
$(document).ready(function() {
  $('#dashboard').hover(
   function() {
     $(this).animate(
       {
         left: '0',
         backgroundColor: 'rgb(27,45,94)'
       },
       500,
       'easeInSine'
     ); // Koniec funkcji animate.
   },
   function() {

   }
  ); // Koniec funkcji hover.
}); // Koniec funkcji ready.
```

   Ostatni argument wywołania funkcji `animate()` — w naszym przypadku ma on postać `'easeInSine'` — nakazuje zastosować metodę określania tempa animacji, która sprawia, że animacja zaczyna się dosyć wolno, a następnie przyspiesza.

9. **Zapisz plik, wyświetl stronę w przeglądarce i wskaż znacznik `<div>` myszą**.

   Znacznik powinien przesunąć się w prawo i w całości pojawić na stronie. Jeśli tak się nie stanie, spróbuj określić źródło problemów, posługując się technikami opisanymi na stronie 51. Oczywiście, kiedy usuniesz wskaźnik myszy z obszaru elementu, nic się nie stanie. Wciąż musisz jeszcze uzupełnić kod drugiej funkcji anonimowej.

10. **W drugiej funkcji anonimowej umieść następujący fragment kodu:**

```
$(this).animate(
    {
       left: '-92px',
       backgroundColor: 'rgb(255,211,224)'
    },
    1500,
    'easeOutBounce'
); // Koniec funkcji animate.
```

   Powyższy kod odwraca zmiany wprowadzone przez pierwszą animację — przesuwa element w lewo, poza krawędź okna przeglądarki i ponownie zmienia jego tło na różowe. W tym przypadku animacja trwa nieco dłużej — półtorej, a nie pół sekundy — oraz używamy innej metody określania tempa.

11. **Zapisz plik. Wyświetl go w przeglądarce, wskaż znacznik `<div>` myszą, a następnie usuń jej wskaźnik z obszaru elementu.**

    Jak się przekonasz, znacznik `<div>` jest przesuwany na stronę, a następnie z niej wysuwany. Jeśli jednak spróbujesz kilkakrotnie i szybko umieścić wskaźnik myszy w jego obszarze, a następnie go usunąć, zauważysz pewne dziwne zachowanie: znacznik będzie wsuwany na stronę oraz z niej wysuwany na długo po tym, jak przestałeś poruszać wskaźnikiem myszy. Przyczyną tego problemu jest sposób, w jaki jQuery kolejkuje wykonywane animacje. Zgodnie z informacjami podanymi na stronie 223, wszelkie animacje operujące na pewnym elemencie trafiają do specjalnej, powiązanej z nim kolejki. Jeśli na przykład chcemy stopniowo wyświetlić element, a następnie go wygasić, jQuery wykona każdy z tych efektów w kolejności, jeden po drugim.

    Problem, jaki mogłeś zaobserwować, polegał na tym, że wraz z każdym umieszczeniem wskaźnika myszy w obszarze znacznika oraz jego usunięciem z tego obszaru do kolejki była dodawana następna animacja. A zatem wielokrotne, szybkie powtarzanie tych czynności doprowadziło do utworzenia długiej listy efektów — wsunięcia znacznika na stronę, wysunięcia go poza nią, ponownego wsunięcia i tak dalej — które biblioteka jQuery miała wykonać. Rozwiązaniem tego problemu jest przerwanie wykonywania wszelkich animacji znacznika przed rozpoczęciem kolejnej. Innymi słowy, jeśli umieścimy wskaźnik myszy w obszarze znacznika, który aktualnie jest w trakcie animacji, animację tę należy przerwać, a następnie rozpocząć kolejną — zgodnie ze sposobem obsługi zdarzenia `mouseEnter`. Na szczęście biblioteka jQuery udostępnia funkcję, która pozwala na takie przerwanie aktualnie wykonywanej animacji; jest nią `stop()`.

12. **Dodaj wywołanie funkcji `.stop()` pomiędzy `$(this)` oraz `.animate`, wewnątrz obu funkcji anonimowych. Uzupełniony, końcowy kod przykładu powinien wyglądać następująco**:

```
$(document).ready(function() {
  $('#dashboard').hover(
   function() {
     $(this).stop().animate(
       {
         left: '0',
         backgroundColor: 'rgb(27,45,94)'
       }
       500,
       'easeInSine'
     ); // Koniec funkcji animate.
   },
   function() {
     $(this).stop().animate(
       {
         left: '-92px',
         backgroundColor: 'rgb(255,211,224)'
       },
       1500,
       'easeOutBounce'
     ); // Koniec funkcji animate.
   }
  ); // Koniec funkcji hover.
}); // Koniec funkcji ready.
```

Wywołanie funkcji `stop()` powoduje zakończenie wszelkich animacji wykonywanych na elemencie `<div>` przed rozpoczęciem kolejnej i nie dopuszcza do umieszczenia w kolejce większej liczby animacji.

Zapisz stronę i spróbuj wyświetlić ją w przeglądarce. Pełną wersję tego przykładu możesz znaleźć w pliku *complete_animate.html*, w katalogu *R06*.

jQuery i przejścia oraz animacje CSS3

Jeśli jesteś na bieżąco z najnowszymi i najwspanialszymi technikami stosowania kaskadowych arkuszy stylów, to być może zastanawiasz się nad tym, po co w ogóle bawić się w tworzenie animacji przy użyciu biblioteki jQuery? W końcu przejścia (ang. *transitions*) CSS pozwalają na wykorzystanie samej technologii CSS do tworzenia animacji pomiędzy dwoma stylami; zapewniają one tym samym możliwość opracowania bardzo złożonych efektów wizualnych (patrz rysunek 6.4).

Rysunek 6.4. *Rachel Nabors, rysowniczka, specjalistka do spraw języka JavaScript i twórczyni animacji połączyła jQuery oraz animacje CSS3, aby stworzyć kompletną, animowaną przygodę dostępną na stronie http://codepen.io/rachelnabors/full/lqswg. Korzysta ona z jQuery, by wykonywać animacje utworzone przy użyciu CSS. Niestety, ten ultranowoczesny przykład nie działa we wszystkich przeglądarkach*

Nowe możliwości animacji opracowywanych przy użyciu CSS są niesamowite, jednak nie wszystkie przeglądarki je obsługują. Konkretnie chodzi o dwie, wciąż popularne wersje przeglądarki Internet Explorer — 8. i 9. — które nie obsługują ani animacji, ani przejść CSS. A zatem, jeśli chcesz uwzględnić te przeglądarki podczas tworzenia swojej witryny, konieczne będzie zastosowanie innego sposobu tworzenia animacji. W takim przypadku opisane wcześniej w tym rozdziale animacje budowane przy użyciu jQuery są najlepszym sposobem opracowania strony działającej we wszystkich przeglądarkach.

Uwaga: Przeglądarki nie potrafią animować wszystkich dostępnych właściwości CSS. Przykładowo nie ma możliwości animowania właściwości `font-family` i tworzenia tekstu, który przechodzi od jednego kroju czcionki do innego. Jednak w przejściach oraz animacjach i tak można używać wielu właściwości CSS. Ich pełna lista jest dostępna na stronie *https://developer.moz lla.org/en-US/docs/Web/CSS/CSS_animated_properties*.

Jeśli jednak chcesz korzystać z przejść oraz animacji CSS, biblioteka jQuery i tak może się okazać bardzo pomocna. W odróżnieniu od języka JavaScript, CSS nie udostępnia żadnych zdarzeń. Istnieje, co prawda, pseudoklasa `:hover`, która pozwala zastosować wybraną klasę po wygenerowaniu zdarzenia `mouseEnter` oraz zmienić ją na inną po wygenerowaniu zdarzenia `mouseLeave`. Z kolei pseudoklasa `:active` może posłużyć do symulacji kliknięcia. Mimo to, bardzo wiele zdarzeń, takich jak podwójne kliknięcie, przewinięcie strony czy też naciśnięcie klawiszy, nie ma żadnych odpowiedników w CSS. Oznacza to, że nie można zastosować samych kaskadowych arkuszy stylów, by rozpocząć animacje, kiedy użytkownik wpisze coś w polu tekstowym lub dwukrotnie kliknie przycisk. Co więcej, CSS nie umożliwia uruchomienia animacji w jednym elemencie strony, w odpowiedzi na akcję wykonaną przez użytkownika na innym elemencie; na przykład kliknięcie przycisku „Pokaż ustawienia" u góry strony nie spowoduje wyświetlenia elementu `div` umieszczonego w innym miejscu.

Uwaga: W tej książce nie znajdziesz szczegółowych informacji na temat przejść oraz animacji CSS. Jeśli chcesz dowiedzieć się czegoś więcej na ich temat, warto sięgnąć po książkę *CSS3. Nieoficjalny podręcznik. Wydanie III*. Szybkie wprowadzenie do tworzenia przejść CSS można znaleźć na stronie *http://www.css3files.com/transition/*. Podobne wprowadzenie do tworzenia animacji jest dostępne na stronie *http://www.css3files.com/animation/*.

jQuery i przejścia CSS

Przejścia powodują płynną modyfikację wartości właściwości CSS. Najprościej można to zrobić poprzez zastosowanie w elemencie nowego stylu, a następnie animowanie wprowadzonej zmiany. Możesz na przykład zmienić wygląd przycisku określony przez klasę `.button`, która ustawia kolor tła na niebieski. Używając pseudoklasy — `.button:hover` — możesz zmienić kolor tła przycisku na żółty. Teraz poprzez zastosowanie przejścia operującego na klasie `.button` możesz zażądać, by przeglądarka animowała zmianę koloru tła z niebieskiego na żółty w momencie umieszczenia wskaźnika myszy w obszarze przycisku, oraz z żółtego na niebieski, kiedy użytkownik usunie wskaźnik myszy z przycisku.

Możesz dodać przejście, by płynnie zmienić jedną wartość właściwości CSS na inną. Załóżmy przykładowo, że chcesz powoli wygasić wszystkie obrazki na stronie, używając do tego przejścia CSS. Zacznij od określenia początkowego stylu obrazków:

```
img {
  opacity: 1;
}
```

Właściwość `opacity` kontroluje poziom nieprzezroczystości elementu. Wartość `1` oznacza, że dany element jest całkowicie widoczny (nieprzezroczysty), a wartość `0`, że jest zupełnie przezroczysty. Następnie możesz utworzyć klasę, która nadaje właściwości `opacity` wartość `0`:

```
img.faded {
  opacity: 0;
}
```

Aby dodać animację powodującą płynne przejście pomiędzy tymi dwoma stylami, musisz skorzystać z właściwości CSS o nazwie `transition`. Należy ją dodać do stylu początkowego (czyli tego, który określi domyślny wygląd elementów na stronie). W tym przypadku jest to styl określający postać wszystkich elementów `img`. Oprócz tego, aby upewnić się, że przejście będzie działało we wszystkich przeglądarkach, które na to pozwalają, do właściwości `transition` należy dodać odpowiednie prefiksy; co pokazano na poniższym przykładzie:

```
img {
  opacity: 1;
  -webkit-transition: opacity 1s;
  -moz-transition: opacity 1s;
  -o-transition: opacity 1s;
  transition: opacity 1s;
}
```

W tym przykładzie określasz, że chcesz animować zmiany wartości właściwości `opacity`. Co więcej, określasz, że chcesz, by animacja ta trwała jedną sekundę — informuje o tym fragment `1s` umieszczony w przedstawionych regułach.

Uwaga: Kiedy twórcy przeglądarek dodają do nich nowe, innowacyjne możliwości, zazwyczaj zaczynają od poprzedzenia nazwy właściwości specjalnym prefiksem odpowiadającym nazwie przeglądarki, na przykład `-webkit-transition`. Dzięki tym prefiksom twórcy przeglądarek mogą w bezpieczny sposób przetestować nowe możliwości, aż do momentu gdy dodawane właściwości będą działały całkowicie poprawnie. Kiedy już wszyscy uzgodnią, jak ma działać nowa właściwość CSS, prefiksy zostają usunięte w nowszych wersjach przeglądarek i można stosować wyłącznie standardową nazwę danej właściwości CSS, na przykład zamiast `-webkit-transition` można używać właściwości `transition`.

Kiedy po dodaniu klasy `faded` zastosujesz ją w obrazku, zostanie wykonana animacja, która w ciągu 1 sekundy sprawi, że całkowicie widoczne obrazki staną się zupełnie przezroczyste. Innymi słowy, efekt będzie przypominał zastosowanie funkcji jQuery `fadeOut()` tworzącej animację o czasie trwania jednej sekundy. W tym przypadku najważniejsze jest dodanie do obrazków na stronie klasy `faded`. I właśnie do tego celu może się przydać jQuery. Jeśli zechcesz zastosować wybrany styl, kiedy użytkownik kliknie obrazek, wystarczy użyć funkcji `click()` w sposób przedstawiony na poniższym przykładzie:

```
$('img').click(function() {
  $(this).addClass('faded');
}
```

Kiedy użytkownik kliknie obrazek, jQuery doda do niego klasę, a przeglądarka zajmie się najtrudniejszym zadaniem, czyli animacją zmiany właściwości `opacity` (patrz rysunek 6.5). Gdybyś chciał płynnie wyświetlić obrazek po jego ponownym kliknięciu, zamiast funkcji `addClass()` wystarczyłoby zastosować funkcję `toggleClass()`:

```
$('img').click(function() {
  $(this).toggleClass('faded');
}
```

Rysunek 6.5. *Biblioteka jQuery i przejścia CSS doskonale ze sobą współpracują. Dzięki użyciu procedur obsługi zdarzeń określanych przy użyciu jQuery można uruchamiać przejścia. W tym przykładzie kliknięcie obrazka powoduje jego płynne ukrycie, ponowne kliknięcie pustego obszaru powoduje wyświetlenie obrazka. Przedstawiony kod nie działa w przeglądarce IE9 i wcześniejszych — nie obsługują one przejść CSS. Kompletną wersję tego przykładu znajdziesz w pliku jquery-trigger-css-animation.html, w katalogu R06*

Funkcja `toggleClass()` dodaje klasę, jeśli nie jest używana w elemencie, a jeśli jest, to ją usuwa. Ponieważ w CSS nie istnieje selektor odpowiadający kliknięciu elementu, zatem zastosowanie jQuery stanowi prosty sposób uruchamiania przejść CSS.

Uwaga: Przypisanie wartości 0 właściwości `opacity` elementu nie powoduje usunięcia go ze strony. Element wciąż istnieje, zajmuje miejsce na stronie (oraz w DOM), przy czym jest niewidoczny. I to właśnie dlatego zastosowanie funkcji `toggleClass()` w powyższym przykładzie daje efekt naprzemiennego znikania i pojawiania się obrazków. Wciąż można kliknąć niewidoczny obrazek, by usunąć z niego klasę `faded`. Gdybyśmy jednak faktycznie ukryli element, na przykład przy użyciu stylu `display: hidden`, zostałby on usunięty ze strony i nie można by go było ponownie kliknąć.

jQuery i animacje CSS

Animacje CSS zapewniają znacznie większą kontrolę niż proste przejścia. W ich przypadku definiuje się *klatki kluczowe* (ang. *keyframes*), które ustalają wartości właściwości CSS na określonych etapach animacji. Można na przykład utworzyć animację, która będzie kilkakrotnie zmieniać kolor przycisku — z niebieskiego na czerwony, następnie na pomarańczowy i w końcu na zielony – bądź przesuwać element `<div>` na górę strony, następnie na sam dół, w lewo, a później w prawo. Innymi słowy, w odróżnieniu od przejść, które pozwalają na określenie jedynie początkowego i końcowego stanu animacji, animacje CSS dają możliwość określenia serii etapów pośrednich, przez które animacja będzie kolejno przechodzić.

Poziom złożoności animacji CSS może bardzo szybko rosnąć, jednak istnieje wiele zasobów zawierających informacje na ich temat. Aby ułatwić Ci rozpoczęcie ich stosowania, w tym rozdziale przedstawiony zostanie przykład użycia jQuery do wyzwalania animacji. Załóżmy, że chcesz, by po kliknięciu przycisku wybrany element `<div>` zmienił kolor i został poszerzony. (Możesz na przykład zastosować takie rozwiązanie do wyróżnienia i wyświetlenia tekstu ukrytego w tym elemencie).

Pierwszym krokiem jest utworzenie animacji. Do tego celu należy użyć dyrektywy `@keyframes`:

```
@keyframes growProgressBar{
  0% {
    width: 0%;
    background-color: red;
  }
  50% {
    background-color: yellow;
  }
  100% {
    width:88%;
    background-color: green;
  }
}
```

Taki kod może wyglądać obco, jednak jego znaczenie jest proste: tworzy animację o podanej nazwie — w tym przypadku jest to `growProgressBar` — oraz serię klatek kluczowych. Każda klatka określa wartości jednej lub kilku właściwości CSS, które będą modyfikowane podczas trwania animacji. Przykładowo pierwsza z klatek przedstawionych w powyższym przykładzie — `0%` — odpowiada momentowi rozpoczynania animacji i określa, że szerokość elementu ma wynosić `0%`, a jego tło ma być czerwone.

Uwaga: Aby powyższy kod działał w przeglądarkach Chrome i Safari, konieczne jest zastosowanie prefiksu `–@webkit-keyframes`. Poza tym, ten kod nie będzie działał w przeglądarce Internet Explorer 9 i starszych.

Podczas przechodzenia do kolejnych klatek kluczowych tło elementu będzie zmieniać kolor z czerwonego na żółty, a następnie na zielony. Wartość procentowa użyta do zdefiniowania poszczególnych klatek kluczowych określa, w którym momencie animacji wartości właściwości CSS mają osiągać żądane wartości. Załóżmy, że powyższa animacja ma trwać 10 sekund (czas trwania animacji jest określany osobno, o czym dowiesz się już niebawem). A zatem, w momencie odpowiadającym 0% z 10 sekund kolor tła ma być czerwony, a szerokość elementu ma wynosić `0%`. W momencie odpowiadającym 50% z 10 sekund, czyli po 5 sekundach od rozpoczęcia animacji kolor tła ma być żółty. I w końcu, kiedy minie 100% czasu trwania animacji, czyli całe 10 sekund, tło elementu ma być zielone i zajmować 88% szerokości.

Ponieważ w powyższym przykładzie szerokość elementu została określona tylko dla pierwszej i ostatniej klatki kluczowej, będzie się ona zmieniać od `0%` do `88%` podczas całego czasu odtwarzania animacji.

Po określeniu klatek kluczowych można dodać animację do dowolnej liczby elementów. Załóżmy na przykład, że na stronie umieściłeś następujący fragment kodu HTML: `<div class= progressBar >`. Poniższy przykład pokazuje, w jaki sposób możesz dodać do tego elementu zdefiniowaną wcześniej animację:

```
.progressBar {
  animation-name: growProgressBar;
  animation-duration: 10s;
  animation-fill-mode: forwards;
}
```

Powyższy kod zastosuje animację w wybranym elemencie i określa, że ma ona trwać przez 10 sekund. Ostatni wiersz kodu — `animation-fill-mode: forwards;` — zapewnia, że gdy animacja zostanie zakończona, element zachowa wartości właściwości określone w ostatniej klatce kluczowej. (Gdyby został on pominięty, element powróciłby do stanu sprzed rozpoczęcia animacji).

Jednak powyższa animacja zostałaby rozpoczęta bezpośrednio po wczytaniu strony. A Twoim celem było rozpoczęcie jej, kiedy użytkownik kliknie przycisk. Animację można „wstrzymać" (czyli zatrzymać, jeszcze zanim w ogóle została rozpoczęta), używając kolejnej właściwości, czyli `animation-play-state`. Aby odtwarzanie animacji nie rozpoczynało się natychmiast po wczytaniu strony, do stylu elementu należy dodać tę właściwość i przypisać jej wartość `paused`:

```
.progressBar {
  animation-name: growProgressBar;
  animation-duration: 10s;
  animation-fill-mode: forwards;
  animation-play-state: paused;
}
```

Najprostszym zadaniem jest zastosowanie jQuery do rozpoczęcia animacji. W tym celu będziesz musiał jedynie zmienić wartość właściwości `animation-play-state` z `paused` na `running`. Bardzo łatwo zrobisz, używając funkcji `css()`. Możesz na przykład utworzyć przycisk o identyfikatorze `start`, którego kliknięcie rozpocznie animację. A tak mógłby wyglądać odpowiedni kod jQuery:

```
$('#start').click(function() {
  $('.progressBar').css('animation-play-state', 'running');
}
```

Gdybyś umieścił na stronie także drugi przycisk „Zatrzymaj", na przykład o identyfikatorze `pause`, poniższy kod pozwoliłby na zatrzymywanie animacji:

```
$('#pause').click(function() {
  $('.progressBar').css('animation-play-state', 'paused');
}
```

Na szczęście jQuery zwraca uwagę na prefiksy przeglądarek i potrafi z nich prawidłowo korzystać. Podczas ustawiania wartości właściwości CSS wymagającej takiego prefiksu, jQuery ustawi także wartość tej samej właściwości w każdym z istniejących prefiksów. Dziękujemy jQuery!

Innym rozwiązaniem jest tworzenie klatek kluczowych oraz odrębnego stylu, definiującego wartości wszystkich właściwości związanych z animacją. Mógłby on wyglądać następująco:

```
.animateDiv {
  animation-name: growProgessBar;
  animation-duration: 10s;
  animation-fill-mode: forwards;
}
```

W momencie wczytywania strony animowany element nie będzie należał do klasy `.animateDiv`, co oznacza, że nie będzie animowany — i właśnie o to chodziło. Potem możesz użyć jQuery, by dodać tę klasę do elementu. Gdy tylko to zrobisz, przeglądarka zacznie animację. Takie rozwiązanie pozwala uniknąć stosowania właściwości `animation-play-state`:

```
$('#start').click(function() {
  $('.progressBar').addClass('animateDiv');
}
```

W przykładach dołączonych do książki znajdziesz oba rozwiązania; znajdują się one w plikach *jquery-trigger-css-animation1.html* oraz *jquery-trigger-css-animation2.html*.

Trzeba wspomnieć, że stosowanie animacji CSS ma także jedną wadę — Internet Explorer 9 i starsze wersje tej przeglądarki w ogóle ich nie obsługują. Poza tym, kontrola przebiegu animacji CSS nie jest równie łatwa, jak animacji tworzonych przy użyciu biblioteki jQuery.

Uwaga: Najprawdopodobniej programiści będą stosować animacje CSS w połączeniu z odpowiednim kodem JavaScript. W3C oraz twórcy przeglądarek pracują nad wieloma różnymi sposobami kontrolowania animacji CSS przy użyciu JavaScriptu, dodając do języka nowe zdarzenia pozwalające na śledzenie postępu odtwarzania animacji. Jeśli chciałbyś wykonać jakiś kod jQuery po zakończeniu animacji CSS, sposób rozwiązania takiego problemu znajdziesz na stronie *http://blog.teamtreehouse.com/using-jquery-to-detect-when-css3-animations-and-transitions-end*.

ROZDZIAŁ 7

Popularne zastosowania jQuery

Podczas projektowania stron WWW bezustannie pracujemy, korzystając z grupy godnych zaufania elementów. Obrazy mogą poprawiać projekt strony i wyróżniać jej wybrane elementy. Odnośniki tworzą podstawę istnienia WWW, pozwalając użytkownikom na przeskakiwanie od jednej informacji do drugiej, my natomiast możemy kontrolować, jak one działają — czy są otwierane w tym samym oknie przeglądarki, czy w nowym. A kiedy na stronie będzie wiele odnośników, warto wiedzieć, w jaki sposób można je zgromadzić i przedstawić w formie świetnego paska nawigacyjnego. Biblioteka jQuery umożliwia poprawienie zarówno wyglądu tych wszystkich elementów, jak i możliwości interakcji z nimi. W tym rozdziale dowiesz się, jak skorzystać z jQuery, by używane obrazki, odnośniki, okna i paski nawigacyjne działały lepiej niż kiedykolwiek wcześniej.

Zamiana rysunków

Prawdopodobnie najczęściej używanym efektem opartym na języku JavaScript jest *podmienianie* rysunków (ang. *rollover effect*). Efekt polega na zastąpieniu jednego obrazka innym po umieszczeniu nad grafiką wskaźnika myszy. Ta prosta technika od czasu pojawienia się JavaScriptu jest używana do tworzenia interaktywnych pasków nawigacyjnych, których przyciski zmieniają wygląd po umieszczeniu na nich wskaźnika myszy.

Jednak od kilku lat coraz większa liczba projektantów używa do uzyskania tego efektu stylów CSS. Jeśli też do nich należysz, warto, byś wiedział, jak zastępować rysunki za pomocą języka JavaScript, aby tworzyć pokazy slajdów, galerie fotografii i inne interaktywne efekty graficzne na stronach WWW.

Zmienianie atrybutu src rysunków

Każdy rysunek widoczny na stronie WWW ma atrybut `src` (to skrót od ang. *source*, czyli źródło), który zawiera ścieżkę do pliku graficznego, czyli wskazuje obrazek zapisany na serwerze. Jeśli zmienisz tę właściwość i wskażesz inny plik, przeglądarka wyświetli nowy rysunek. Załóżmy, że na stronie znajduje się obrazek o identyfikatorze `photo`. Przy użyciu biblioteki jQuery można dynamicznie zmienić wartość atrybutu `src` obrazka.

Załóżmy więc, że na stronie mamy obrazek o identyfikatorze `photo`. Kod HTML tworzący taki obrazek mógłby mieć następującą postać:

```
<img src="images/image.jpg" width="100" height="100" id="photo" />
```

Aby podmienić obrazek wyświetlany w tym elemencie, wystarczy skorzystać z funkcji `attr()` (opisanej na stronie 166) i przy jej użyciu zmienić wartość atrybutu `src` tak, by wskazywał plik nowego obrazka:

```
$('#photo').attr('src','images/newImage.jpg');
```

Uwaga: Przy zmianie właściwości `src` rysunku w kodzie JavaScript ścieżkę do pliku graficznego należy podać względem strony, a *nie* pliku z kodem JavaScript. Bywa to skomplikowane przy używaniu zewnętrznych plików JavaScript (patrz strona 49) zapisanych w innym katalogu. Po napotkaniu wcześniejszej instrukcji przeglądarka spróbuje pobrać plik *newImage.jpg* z katalogu *images* utworzonego w tym samym folderze, co dana strona. Ta metoda działa dobrze nawet wtedy, gdy kod znajduje się w pliku zewnętrznym umieszczonym w innym miejscu witryny. Dlatego zwykle w takich plikach łatwiej używać ścieżek podawanych względem katalogu głównego (omówienie różnych rodzajów odsyłaczy znajdziesz w ramce na stronie 45).

Modyfikacja atrybutu `src` nie ma wpływu na pozostałe atrybuty znacznika `<img>`. Jeśli na przykład w kodzie HTML ustawiono atrybut `alt`, w nowym rysunku będzie miał on taką samą wartość jak w pierwotnym obrazku. Ponadto jeśli w kodzie określono atrybuty `width` i `height`, po zmianie właściwości `src` nowy rysunek będzie zajmował ten sam obszar, co poprzedni obrazek. Jeśli grafiki mają różne wymiary, nowy rysunek zostanie zniekształcony.

Przy zastępowaniu obrazków w pasku nawigacyjnym oba rysunki mają zwykle ten sam rozmiar i atrybut `alt`, dlatego użycie właściwości z pierwotnej wersji obrazka jest dopuszczalne. Jednak efektu zniekształcenia obrazka można bardzo łatwo uniknąć — wystarczy pominąć określanie właściwości `width` i `height` w kodzie HTML. Wtedy przy zamianie rysunków przeglądarka użyje wymiarów nowego pliku.

Inne rozwiązanie polega na pobraniu nowego rysunku, sprawdzeniu jego wymiarów i zmianie atrybutów `src`, `width`, `height` i `alt` znacznika `<img>`:

```
1 var newPhoto = new Image();
2 newPhoto.src = 'images/newImage.jpg';
3 var photo = $('#photo');
4 photo.attr('src',newPhoto.src);
5 photo.attr('width',newPhoto.width);
6 photo.attr('height',newPhoto.height);
```

Uwaga: Numery wierszy nie są częścią kodu, dlatego nie należy ich przepisywać. Liczby te ułatwiają czytanie opisu kodu.

Istotą tej techniki jest wiersz 1., który tworzy nowy obiekt rysunku (typu `Image`). Dla przeglądarki kod `new Image()` oznacza: „Przeglądarko, skrypt zaraz doda do strony nowy rysunek, więc się przygotuj". Następny wiersz nakazuje przeglądarce pobranie nowego obrazka. Wiersz 3. pobiera referencję do rysunku widocznego na stronie, a w wierszach od 4. do 6. skrypt zastępuje pierwotny obrazek nowym oraz dopasowuje właściwości `width` i `height` do wymiarów nowej grafiki.

Wskazówka: Funkcja `attr()` biblioteki jQuery pozwala zmienić jednocześnie kilka atrybutów. Wystarczy przekazać do niej literał obiektowy (patrz strona 165), zawierający nazwy atrybutów i ich nowe wartości. Wcześniejszy kod oparty na bibliotece jQuery można zapisać także w bardziej zwięzłej formie:

```
var newPhoto = new Image();
newPhoto.src = 'images/newImage.jpg';
$('#photo').attr({
  src: newPhoto.src,
  width: newPhoto.width,
  height: newPhoto.height
});
```

Zazwyczaj ta technika podmieniania obrazków jest wykorzystywana wraz z obsługą zdarzeń. Można na przykład zmieniać wyświetlany obrazek, kiedy użytkownik wskaże go myszą. Takie rozwiązanie jest powszechnie stosowane podczas tworzenia pasków nawigacyjnych. Obrazki można też podmieniać w odpowiedzi na dowolne zdarzenia, nowy obrazek może się pojawiać za każdym razem, gdy zostanie kliknięta strzałka prezentowana na stronie, tak jak się dzieje w pokazie slajdów.

Podmiana obrazków przy użyciu jQuery

Istnieje także inny sposób podmieniania obrazków, który nie wymaga modyfikowania atrybutu `src`, ani zawracania sobie głowy modyfikowaniem ich poszczególnych atrybutów. Zgodnie z informacjami podanymi na stronie 157, jQuery doskonale nadaje się do wprowadzania szybkich zmian w kodzie HTML strony. Można jej używać do dodawania, usuwania i zmiany kodu HTML strony. Chyba najprostszym sposobem zamiany jednego obrazka na inny jest zastąpienie całego dotychczasowego znacznika `<img>` nowym znacznikiem `<img>`, co można bardzo łatwo zrobić, używając metody jQuery o nazwie `replaceWith()`.

Załóżmy na przykład, że na stronie umieszczony jest następujący obrazek:

```
<img src="sad.png" alt="Smutna buźka" height="50" width="50" id="swap">
```

Taki znacznik można zamienić na inny przy użyciu następującego fragmentu kodu:

```
$('#swap').replaceWith('<img src="happy.png" alt="Wesoła buźka"
height="100" width="150" id="swap">');
```

Metoda `replaceWith()` zastępuje aktualnie wybrany element dowolnym kodem HTML podanym w jej wywołaniu. Za jej pomocą można podać inne wartości atrybutów `src`, `alt`, `width` oraz `height` w jednym łańcuchu, przekazywanym w wywołaniu metody; na przykład `<img src= happy.png  alt= Wesoła buźka  height= 100  width= 150  id= swap >`. Jednak zazwyczaj łatwiejszym rozwiązaniem jest modyfikowanie każdej z tych wartości niezależnie, co pokazano w poprzednim punkcie rozdziału.

Uwaga: Metoda `replaceWith()` zwraca kod HTML, który został zastąpiony nowym. Innymi słowy, można zachować zastępowany kod HTML. Jeśli na przykład chcemy zmienić obrazek, lecz zachować jego początkową wersję do późniejszego użytku, możemy to zrobić w następujący sposób:

```
var oldImage = $('#swap').replaceWith('<img src="happy.png" alt="Wesoła
➥buźka" height="100" width="150" id="swap">');
```

Teraz zmienna `oldImage` zawiera kod HTML, który został zastąpiony. Jeśli będziemy bazować na powyższym przykładzie, zmienna ta będzie zawierać następujący fragment kodu:

```
<img src="sad.png" alt="Smutna buźka" height="50" width="50" id="swap">
```

Zmiennej oldImage będzie można użyć ponownie później — na przykład, by ponownie wyświetlić początkowy obrazek.

Wstępne wczytywanie rysunków

Zastępowanie rysunków za pomocą omówionych wcześniej technik ma jedną wadę. Kiedy skrypt zmieni w atrybucie `src` ścieżkę do pliku graficznego, przeglądarka musi pobrać nowy obrazek. Jeśli program zacznie pobierać plik dopiero po najechaniu wskaźnikiem myszy na rysunek, przed pojawieniem się nowego obrazka wystąpi niepożądane opóźnienie. Jeżeli zdarzy się to w pasku nawigacyjnym, efekt zastępowania będzie niezwykle spowolniony, a czas reakcji — za długi.

Aby uniknąć opóźnienia, można wstępnie pobrać wszystkie rysunki wyświetlane w odpowiedzi na zdarzenia. Kiedy na przykład użytkownik umieści wskaźnik myszy nad przyciskiem w pasku nawigacyjnym, efekt zastępowania powinien działać błyskawicznie. *Wstępne pobieranie* (ang. *preload*) oznacza nakazanie przeglądarce wczytania obrazka, *zanim* skrypt będzie chciał go wyświetlić. Pobrany plik jest zapisywany w pamięci podręcznej przeglądarki, dlatego w odpowiednim momencie będzie można wczytać rysunek z dysku twardego komputera użytkownika, zamiast ponownie pobierać go z serwera.

Wstępne wczytywanie wymaga utworzenia nowego obiektu rysunku i ustawienia jego właściwości `src`. Wiesz już, jak to zrobić:

```
var newPhoto = new Image();
newPhoto.src = 'images/newImage.jpg';
```

Aby ta technika zadziałała, trzeba wywołać powyższy kod przed zastąpieniem rysunku widocznego na stronie. Jedna z metod wstępnego pobrania grafiki polega na utworzeniu na początku skryptu tablicy (patrz strona 77) zawierającej ścieżki do wszystkich wczytywanych w ten sposób plików. Następnie należy przejść po elementach tej listy i utworzyć na podstawie każdego z nich nowy obiekt rysunku:

```
1  var preloadImages = ['images/roll.png',
2                       'images/flower.png',
3                       'images/cat.jpg'];
4  for (var i = 0; i < preloadImages.length; i++) {
5    new Image().src = preloadImages[i];
6  }
```

Wiersze od 1. do 3. to pojedyncza instrukcja języka JavaScript, która tworzy tablicę o nazwie `preloadImages`, zawierającą trzy wartości — ścieżki do wstępnie wczytywanych rysunków. Na stronie 78 dowiedziałeś się, że kod tablicy jest bardziej czytelny, jeśli każdy element znajduje się w odrębnym wierszu. Wiersze od 4. do 6.

zawierają prostą pętlę `for` (patrz strona 112), której ciało jest wykonywane jeden raz dla każdego elementu tablicy `preloadImages`. Instrukcja umieszczona w wierszu 5. tworzy nowy obiekt obrazka i zapisuje w jego właściwości `src` ścieżkę dostępu do obrazka podaną z tablicy `preloadImages` — i to jest cała magia, która sprawia, że przeglądarka wczyta podane obrazki.

Ten sam efekt można uzyskać, stosując bibliotekę jQuery, przy czym w tym przypadku będzie można uniknąć użycia wywołania `new Image()`:

```
1  var preloadImages = ['images/roll.png',
2                       'images/flower.png',
3                       'images/cat.jpg'];
4  for (var i = 0; i < preloadImages.length;i++) {
5    $('<img>').attr('src',preloadImages[i]);
6  }
```

W wierszu 5. korzystamy z jQuery do utworzenia nowego elementu `<img>`. Zastosowana tu technika jest czymś nowym i może być trudna do zrozumienia. Przekazując w wywołaniu jQuery znacznik `<img>` (włącznie z nawiasami kątowymi), tworzymy nowy element HTML. Zazwyczaj w wywołaniu jQuery znaczniki kątowe są pomijane, na przykład `$('img')`, co każe jQuery odszukać wszystkie znaczniki `<img>` istniejące na stronie. A zatem, jak widać, jQuery potrafi nie tylko odnajdywać i pobierać elementy strony, lecz także *tworzyć* nowe.

Dalsza cześć wywołania — `.attr('src',preloadImages[i])` — stanowi wywołanie funkcji `attr()` (przedstawionej na stronie 166). Powoduje ono zapisanie we właściwości `src` nowego elementu ścieżki dostępu do pliku obrazka, co zmusza przeglądarkę do jego wczytania.

Obie z przedstawionych tu technik wstępnego wczytywania obrazków, zarówno ta, która nie wymaga biblioteki jQuery, jak i druga, która z niej korzysta, działają doskonale, zatem możesz użyć tej z nich, która w Twoim przypadku jest bardziej sensowna.

Efekt rollover z użyciem obrazków

Efekt rollover z użyciem obrazków to efekt wizualny polegający po prostu na zastąpieniu jednego obrazka drugim (co opisano na stronie 239), uruchamiany po umieszczeniu wskaźnika myszy nad grafiką. Aby uzyskać ten efekt, należy przypisać kod zastępujący obrazki do zdarzenia `mouseover`. Załóżmy, że na stronie znajduje się rysunek o identyfikatorze `photo`. Kiedy użytkownik umieści wskaźnik myszy nad tym obrazkiem, skrypt ma wyświetlić nowy plik. Przy użyciu biblioteki jQuery można osiągnąć ten efekt w następujący sposób:

```
1  <script src="js/jquery.min.js"></script>
2  <script>
3  $(document).ready(function() {
4    var newPhoto = new Image();
5    newPhoto.src = 'images/newImage.jpg';
6    $('#photo').mouseover(function() {
7      $(this).attr('src', newPhoto.src);
8    }); // Koniec funkcji mouseover.
9  }); // Koniec funkcji ready.
10 </script>
```

Wiersz 3. sprawia, że funkcja anonimowa zostanie wykonana dopiero po wczytaniu całej strony, dzięki czemu umieszczony wewnątrz niej kod będzie mógł odwołać się do bieżącej wersji obrazka. Wiersze 4. i 5. powodują wczytanie obrazka mającego zastąpić ten, który był wyświetlony. Pozostałe wiersze przypisują do rysunku zdarzenie `mousover` z funkcją, która zmienia atrybut `src` obrazka na ścieżkę do nowej fotografii.

W omawianym efekcie przesunięcie wskaźnika myszy poza rysunek powoduje zwykle przywrócenie pierwotnego obrazka. Dlatego trzeba dołączyć zdarzenie `mouseout`, aby ponownie wyświetlić pierwszy rysunek. Na stronie 192 dowiedziałeś się, że jQuery udostępnia zdarzenie `hover()`, które łączy zdarzenia `mouseover` i `mouseout`:

```
1  <script src="js/jquery.min.js"></script>
2  <script>
3  $(document).ready(function() {
4    var newPhoto=new Image();
5    newPhoto.src='images/newImage.jpg';
6    var oldSrc=$('#photo').attr('src');
7    $('#photo').hover(
8      function() {
9        $(this).attr('src', newPhoto.src);
10     },
11     function() {
12      $(this).attr('src', oldSrc);
13   }); // Koniec funkcji hover.
14 }); // Koniec funkcji ready.
15 </script>
```

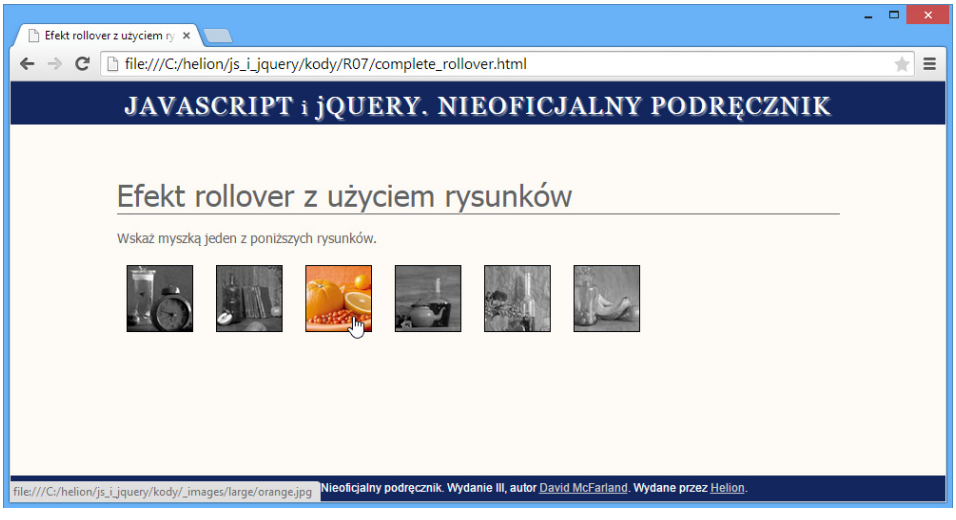
Funkcja `hover()` przyjmuje dwa argumenty. Pierwszym z nich jest funkcja anonimowa, która informuje przeglądarkę o tym, co ma zrobić, kiedy użytkownik umieści wskaźnik myszy nad rysunkiem. Drugi argument to funkcja uruchamiana po przeniesieniu wskaźnika myszy w inne miejsce strony. Nowy kod tworzy zmienną `oldSrc`, która przechowuje pierwotną wartość atrybutu `src`, czyli ścieżkę do pliku widocznego po wczytaniu strony.

Efekt ten można uruchamiać nie tylko za pomocą obrazków. Funkcję `hover()` można dołączyć do dowolnego znacznika — odnośnika, elementu formularza, a nawet akapitu. Oznacza to, że każdy znacznik może powodować zmianę rysunku w dowolnej części strony. Na przykład umieszczenie wskaźnika myszy nad znacznikiem `<h1>` może powodować zastąpienie rysunku nowym obrazkiem. Załóżmy, że docelowy plik jest taki sam jak we wcześniejszym przykładzie. W poprzednim skrypcie należy wprowadzić zmiany wyróżnione pogrubieniem:

```
1  <script src="js/jquery.min.js"></script>
2  <script>
3  $(document).ready(function() {
4    var newPhoto=new Image();
5    newPhoto.src='images/newImage.jpg';
6    var oldSrc=$('#photo').attr('src');
7    $('h1').hover(
8      function() {
9        $('#photo').attr('src', newPhoto.src);
10     },
11    function() {
12      $('#photo').attr('src', oldSrc);
13    }); // Koniec funkcji hover.
14  }); // Koniec funkcji ready.
15  </script>
```

Przykład — dodawanie efektu rollover z użyciem rysunków

W tym przykładzie dodasz efekt `rollover` do zbioru obrazków (patrz rysunek 7.1). Skrypt ma wstępnie wczytywać pliki potrzebne w efekcie, aby zlikwidować opóźnienie między umieszczeniem wskaźnika myszy nad rysunkiem a wyświetleniem nowego obrazka. Ponadto poznasz nową, bardziej wydajną technikę wstępnego pobierania rysunków i obsługi efektu `rollover`.



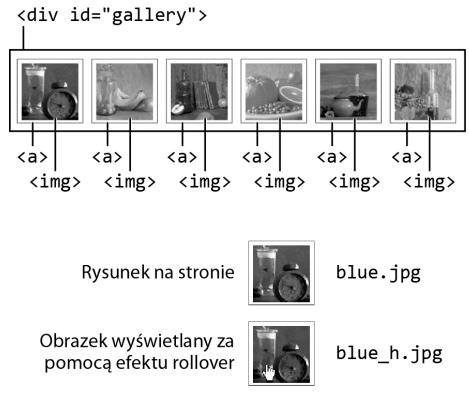
Rysunek 7.1. *Efekt rollover pozwala zwiększyć atrakcyjność wizualną paska nawigacyjnego, odnośnika, a nawet samych zdjęć*

Omówienie zadania

Plik *rollover.html* z katalogu *R07* zawiera zbiór sześciu zdjęć (patrz rysunek 7.2). Każde z nich jest częścią odnośnika, który prowadzi do większej wersji fotografii, a wszystkie zdjęcia znajdują się wewnątrz znacznika `<div>` o identyfikatorze `gallery`. Skrypt ma wykonywać dwie operacje.

- Wstępnie wczytywać używane w efekcie pliki, powiązane ze zdjęciami umieszczonymi w znaczniku `<div>`.
- Dołączać funkcję `hover()` do każdego rysunku ze znacznika `<div>`. Funkcja `hover()` ma zamieniać rysunki po umieszczeniu nad obrazkiem wskaźnika myszy, a następnie przywracać pierwotną wersję zdjęcia po przeniesieniu wskaźnika w inne miejsce strony.

Na podstawie opisu można zauważyć, że obie operacje są powiązane z rysunkami w elemencie `<div>`, dlatego jedną z możliwości jest pobranie obrazków z tego znacznika, a następnie przejście po nich w pętli, wstępne wczytanie odpowiadających im nowych rysunków i dołączenie funkcji `hover()`.



Rysunek 7.2. *Podstawowa struktura kodu HTML użytego w tym przykładzie składa się ze znacznika , który zawiera zbiór odnośników z rysunkami. Aby ułatwić wyświetlanie nowych zdjęć, ich nazwy utworzono na podstawie nazw pierwotnie widocznych obrazków*

Uwaga: Więcej informacji o pobieraniu przykładowych plików znajdziesz na stronie 46.

Tworzenie kodu

1. **Otwórz w edytorze tekstu plik *rollover.html* z katalogu *R07*.**

   Do strony dołączony jest już plik jQuery i funkcja `$(document).ready()` (patrz strona 190). Pierwszy krok polega na pobraniu wszystkich rysunków ze znacznika `<div>` i dodaniu pętli za pomocą funkcji `each()` biblioteki jQuery (patrz strona 167).

2. **Kliknij pusty wiersz w funkcji `$(document).ready()` i wpisz kod `$('#gallery img').each(function() {`.**

   Selektor `#gallery img` pobiera wszystkie znaczniki `<img>` ze znacznika o identyfikatorze `gallery`. Funkcja `each()` biblioteki jQuery umożliwia łatwe przejście w pętli po kolekcji elementów i wykonanie na nich serii operacji. Funkcja ta przyjmuje jako argument funkcję anonimową (patrz strona 168). W tym momencie warto zamknąć dodaną funkcję anonimową. Dobrym rozwiązaniem jest dodanie zamykającego nawiasu kończącego treść funkcji anonimowej (a także każdej innej), zanim przystąpimy do pisania umieszczonego wewnątrz niej kodu. Dlatego właśnie tę czynność wykonamy w kolejnym kroku.

3. **Wciśnij dwukrotnie klawisz *Enter* i wpisz kod `}); // Koniec funkcji each.`, aby zamknąć funkcję anonimową, wywołanie funkcji `each()` i instrukcję języka JavaScript. Kod powinien wyglądać następująco:**

```
1 <script src="../_js/jquery.min.js"></script>
2 <script>
3 $(document).ready(function() {
4   $('#gallery img').each(function() {
5
6   }); // Koniec funkcji each.
7 }); // Koniec funkcji ready.
```

   Na tym etapie skrypt przechodzi w pętli po wszystkich rysunkach z galerii, jednak nie wykonuje na nich żadnych operacji. Pierwsze zadanie polega na pobraniu właściwości `src` rysunków i zapisaniu ich w zmiennej, której skrypt użyje w dalszej części kodu.

Uwaga: Komentarze języka JavaScript — `// Koniec funkcji each.` i `// Koniec funkcji ready.` — nie są niezbędne, jednak ułatwiają ustalenie, której części skryptu dotyczy dany wiersz.

4. **Kliknij pusty wiersz (wiersz 5. w kodzie w kroku 3.) i wpisz:**

```
var imgFile = $(this).attr('src');
```

Na stronie 169 dowiedziałeś się, że konstrukcja `$(this)` wskazuje element aktualnie przetwarzany w pętli. Oznacza to, że odpowiada ona po kolei każdemu elementowi `<img>`. Funkcja `attr()` biblioteki jQuery (patrz strona 167) pobiera określony atrybut HTML (w tym skrypcie jest to atrybut `src` rysunku) i zapisuje go w zmiennej `imgFile`. Właściwość `src` pierwszego obrazka ma wartość `_images/small/blue.jpg` i jest ścieżką do pliku graficznego widocznego na stronie.

Tej samej właściwości `src` należy użyć do wstępnego pobrania rysunków.

5. **Wciśnij klawisz *Enter*, aby dodać pusty wiersz, a następnie wpisz trzy poniższe instrukcje:**

```
var preloadImage = new Image();
var imgExt = /(\.\w{3,4}$)/;
preloadImage.src = imgFile.replace(imgExt,'_h$1');
```

W celu wstępnego pobrania rysunku trzeba najpierw utworzyć obiekt `Image`. W tym przypadku skrypt tworzy zmienną `preloadImage` i zapisuje w niej taki obiekt. Następnie kod wstępnie wczytuje rysunek przez przypisanie wartości do właściwości `src` obiektu `Image`.

Jeden ze sposobów na wstępne wczytanie rysunków (opisany na stronie 242) polega na opracowaniu tablicy obrazków, przejściu po nich w pętli, utworzeniu dla każdego z nich obiektu `Image` i przypisaniu wartości do właściwości `src` takiego obiektu. Jednak takie rozwiązanie jest pracochłonne, gdyż może wymagać znajomości dokładnej ścieżki dostępu do każdego z używanych obrazków i podania jej w tablicy.

W tym skrypcie użyjesz bardziej pomysłowego (i mniej pracochłonnego) rozwiązania. Aby je zastosować, trzeba zapisać nowy rysunek w tej samej lokalizacji, co pierwotny, i nazwać obie wersje obrazka w podobny sposób. W tym przykładzie każdy rysunek na stronie jest zastępowany obrazkiem, który w nazwie ma przyrostek „_h". Na przykład rysunek *blue.jpg* jest zamieniany na obrazek *blue_h.jpg*. Oba pliki znajdują się w tym samym katalogu, dlatego prowadzi do nich ta sama ścieżka.

A oto pomysłowe rozwiązanie: zamiast ręcznie wprowadzać wartość właściwości `src`, aby wstępnie wczytać rysunek (na przykład `preloadImage.src='_images/small/blue_h.jpg'`), można zapisać ją za pomocą kodu JavaScript. Skrypt ustala potrzebną wartość na podstawie właściwości `src` pierwotnego rysunku. Innymi słowy, jeśli znamy ścieżkę dostępu do obrazka wyświetlonego na stronie, to wystarczy do niej, bezpośrednio przed rozszerzeniem, dodać znak podkreślenia i literkę *h*. A zatem *_images/small/blue.jpg* zostanie przekształcone na *_images/small/blue_h.jpg*, a *_images/small/oranges.jpg* na *_images/ small/ oranges_h.jpg*.

I właśnie tę operację wykonują dwa kolejne wiersze kodu. Pierwszy z nich — `var imgExt = /(\.\w{3,4}$)/;` — tworzy *wyrażenie regularne*. Wyrażenia regularne (poznasz je na stronie 571) są wzorcami znaków, których można poszukiwać w innych łańcuchach; przykładem takiego wzorca mogą być trzy cyfry umieszczone jedna za drugą. Wyrażenia regularne bywają trudne i złożone, jednak to, które zostało użyte w tym przykładzie, odpowiada znakowi kropki poprzedzającej trzy dowolne litery umieszczone na samym końcu łańcucha. Przykładowo wzorzec ten będzie odpowiadał łańcuchowi *.jpeg* w łańcuchu */images/small/blue.jpeg* oraz łańcuchowi *.png* w */images/orange.png*.

W następnym wierszu — `preloadImage.src = imgFile.replace(imgExt, ↪'_h$1');` — użyto metody `replace()` (patrz strona 585) do zastąpienia znalezionego tekstu innym fragmentem. Skrypt zmienia na przykład człon ".jpg" w ścieżce na „_h.jpg", czyli zastępuje nazwę *images/small/blue.jpg* ścieżką *images/small/blue_h.jpg*. To rozwiązanie jest dość skomplikowane, ponieważ wymaga użycia podwzorca wyrażenia regularnego (pełny opis tej techniki znajdziesz w ramce na stronie 586), dlatego na razie nie musisz się przejmować, jeśli w pełni nie rozumiesz, jak ono działa.

Po wstępnym wczytaniu nowych obrazków można przypisać do rysunków zdarzenie `hover()`.

6. **Wciśnij klawisz *Enter*, a następnie dodaj kod z wierszy od 9. do 11.:**

```
1 <script src="../_js/jquery.min.js"></script>
2 <script>
3 $(document).ready(function() {
4   $('#gallery img').each(function() {
5     var imgFile = $(this).attr('src');
6     var preloadImage = new Image();
7     var imgExt =  /(\.\w{3,4}$)/;
8     preloadImage.src = imgFile.replace(imgExt,'_h$1');
9     $(this).hover(
10
11    ); // Koniec funkcji hover.
12  }); // Koniec funkcji each.
13 }); // Koniec funkcji ready.
```

Funkcja `hover()` biblioteki jQuery umożliwia szybkie dodanie do elementu zdarzeń `mouseover` i `mouseout`. Należy przekazać do niej dwie funkcje. Pierwsza jest uruchamiana w momencie umieszczenia wskaźnika myszy nad elementem — tu zmienia rysunek na jego nową wersję. Drugą skrypt wywołuje, kiedy wskaźnik myszy znajdzie się w innym miejscu strony. Tu funkcja ta ponownie wyświetla pierwotny obrazek.

7. **W pustym wierszu (wiersz 10. w kodzie w kroku 6.) dodaj poniższy fragment:**

```
function() {
  $(this).attr('src', preloadImage.src);
},
```

Jest to pierwsza funkcja. Jej zadanie polega na zmianie właściwości `src` pierwotnego rysunku na właściwość `src` nowego obrazka. Przecinek na końcu tego fragmentu jest niezbędny, ponieważ funkcja jest pierwszym argumentem wywołania `hover()`, a poszczególne argumenty trzeba oddzielać przecinkami.

8. **Teraz dodaj drugą funkcję (wiersze od 13. do 15.). Gotowy skrypt powinien wyglądać następująco:**

```
1 <script src="../_js/jquery.min.js"></script>
2 <script>
3 $(document).ready(function() {
4   $('#gallery img').each(function() {
5     var imgFile = $(this).attr('src');
6     var preloadImage = new Image();
7     var imgExt =  /(\.\w{3,4}$)/;
8     preloadImage.src = imgFile.replace(imgExt,'_h$1');
9     $(this).hover(
10      function() {
11        $(this).attr('src', preloadImage.src);
12      },
13      function() {
14        $(this).attr('src', imgFile);
15      }
16    ); // Koniec funkcji hover.
17  }); // Koniec funkcji each.
18 }); // Koniec funkcji ready.
```

To druga funkcja. Jej zadaniem jest przywrócenie atrybutu `src` z pierwotnego rysunku. W wierszu 5. skrypt zapisuje ścieżkę do tego obrazka w zmiennej `imgFile`. Nowa funkcja w wierszu 14. używa tej zmiennej do przywrócenia pierwotnej wartości atrybutu `src`. Zapisz stronę, wyświetl ją w przeglądarce i umieść wskaźnik myszy nad czarno-białymi zdjęciami, aby zobaczyć ich kolorowe wersje.

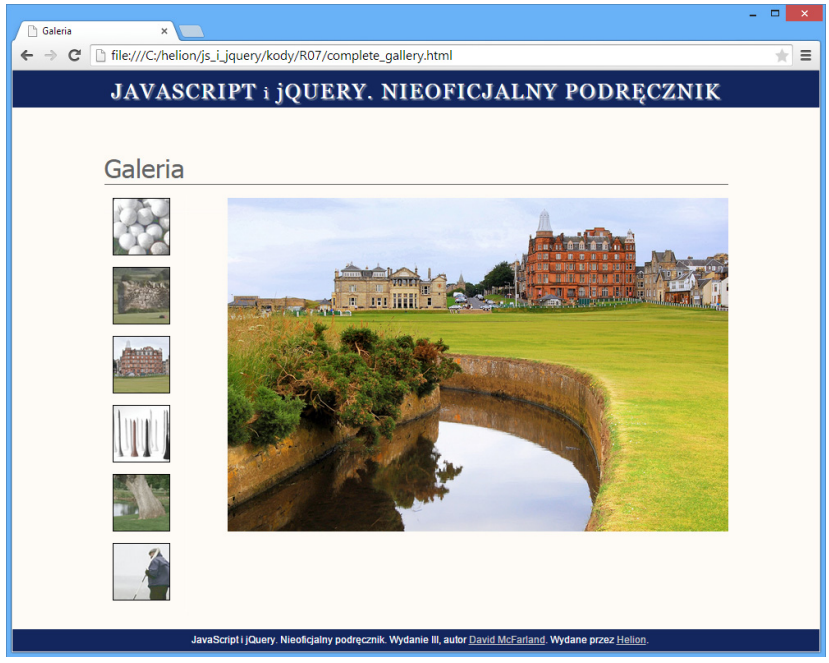
Uwaga: Taki sam efekt podmiany obrazków można osiągnąć przy użyciu wyłącznie arkuszy stylów CSS. Informacje, jak to zrobić, można znaleźć w artykule *http://kyleschaeffer.com/development/pure-css-image-hover/*. Mimo to, warto rozumieć, w jaki sposób można zmienić jeden obrazek na drugi za pomocą kodu JavaScript. Możliwości stylów CSS ograniczają się do kilku stanów, takich jak `:hover` lub `:active`, dlatego też wykorzystanie kodu JavaScript zapewnia możliwość rozpoczynania efektu w reakcji na inne zdarzenia, takie jak dwukrotne kliknięcie bądź naciśnięcie jakiegoś klawisza. Oprócz tego może się zdarzyć, że efekt będzie musiał być uruchamiany w odpowiedzi na zdarzenie skierowane do innego elementu strony niż podmieniany obrazek. Przykładowo na stronie może być umieszczony przycisk „Podmień obrazki", którego kliknięcie ma spowodować zmianę wszystkich obrazków widocznych na niej obrazków.

Przykład — galeria fotografii z efektami wizualnymi

W tym przykładzie przekształcisz wcześniejszy program w jednostronicową galerię fotografii. Skrypt ma wyświetlać na stronie większy rysunek w odpowiedzi na kliknięcie przez użytkownika miniatury (patrz rysunek 7.3). Ponadto użyjesz kilku efektów z biblioteki jQuery, aby uatrakcyjnić przejścia między dużymi zdjęciami.

Omówienie zadania

Galerie działają w prosty sposób — kliknięcie miniatury powoduje wyświetlenie większego zdjęcia. Jednak w tym przykładzie zobaczysz, jak za pomocą efektów stopniowego wyświetlania i ukrywania rysunków uatrakcyjnić prezentację.



Rysunek 7.3. *Gotowa strona z galerią fotografii. Kliknięcie miniatury powoduje stopniowe ukrycie bieżącego zdjęcia i wyświetlenie nowego. Plik complete_gallery.html z katalogu R07 to gotowa wersja tego przykładu*

Użyjesz też innej ważnej techniki — *„dyskretnego" kodu JavaScript*. To podejście umożliwia wyświetlanie większych wersji zdjęć także na komputerach z wyłączoną obsługą języka JavaScript. Aby uzyskać ten efekt, każdą miniaturę należy umieścić w odnośniku prowadzącym do pliku z dużą fotografią (patrz rysunek 7.4). Jeśli przeglądarka nie obsługuje języka JavaScript, kliknięcie odsyłacza spowoduje opuszczenie strony i wczytanie większej wersji zdjęcia. Nie jest to zbyt atrakcyjny sposób prezentacji, ponieważ użytkownik musi opuścić galerię, ale zdjęcia będą dostępne. Jeśli przeglądarka obsługuje język JavaScript, kliknięcie odnośnika spowoduje stopniowe wyświetlenie dużej fotografii na stronie.

Rysunek 7.4. *Podstawowa struktura galerii fotografii. Wszystkie miniatury są zawarte w odnośnikach, które wskazują na większe wersje zdjęć. Kliknięcie odsyłacza powoduje wczytywanie dużego rysunku w znaczniku o identyfikatorze photo*

Efekt ma zachodzić w odpowiedzi na kliknięcie odnośnika, dlatego skrypt musi po wystąpieniu zdarzenia `click` wykonać następujące operacje.

- **Zablokować domyślne działanie odnośnika**. Standardowo kliknięcie odnośnika powoduje przejście do nowej strony. W tym przykładzie użycie odsyłacza obejmującego miniaturę doprowadzi do opuszczenia strony i wyświetlenia większego zdjęcia. Ponieważ to kod JavaScript ma wyświetlać fotografie, należy dodać instrukcje blokujące przejście do nowej strony.
- **Pobrać wartość atrybutu `href` odnośnika**. Odsyłacz prowadzi do większego zdjęcia, dlatego wartość atrybutu `href` jest jednocześnie ścieżką do dużej fotografii.
- **Utworzyć na stronie nowy znacznik rysunku**. Ten znacznik powinien zawierać ścieżkę pobraną z atrybutu `href`.
- **Stopniowo ukrywać stare zdjęcie, a jednocześnie wyświetlać nowe**. Skrypt ma usuwać widoczną fotografię w czasie wyświetlania większej wersji kliknietej miniatury.

Trzeba uwzględnić także kilka innych drobiazgów, ale te cztery kroki opisują podstawowy proces.

Tworzenie kodu

W tym przykładzie użyjesz utworzonej wcześniej strony, jednak z nieco zmienionym układem. Umieszczono na niej nowy zbiór miniatur, wyświetlonych w lewej kolumnie, a na stronie znalazł się znacznik `<div>` o identyfikatorze `photo` (patrz rysunek 7.4).

Uwaga: Informacje o pobieraniu przykładowych plików znajdziesz na stronie 46.

1. **Otwórz w edytorze kodu plik *gallery.html* z katalogu *R07*.**

   Ten plik zawiera kod z poprzedniego przykładu oraz nowy znacznik `<div>` przeznaczony na większą wersję zdjęcia z miniatury. Ponieważ proces wyświetlania fotografii jest uruchamiany przez kliknięcie jednego z odnośników obejmujących miniatury, pierwszy krok polega na pobraniu odsyłaczy i dodaniu do każdego z nich zdarzenia `click`.

2. **Znajdź w kodzie JavaScript komentarz z tekstem „Poniżej wstaw nowy kod JavaScript." i dodaj pod nim następujący kod:**

```
$('#gallery a').click(function(evt) {

}); // Koniec funkcji click.
```

   Selektor `#gallery a` pobiera wszystkie odnośniki znajdujące się w znaczniku o identyfikatorze `gallery`. Polecenie `.click` to funkcja biblioteki jQuery, do której należy przekazać metodę obsługującą zdarzenie (jeśli chcesz przypomnieć sobie informacje o zdarzeniach, zajrzyj na stronę 182). Skrypt przekazuje do zdarzenia `click` funkcję anonimową. Na stronie 179 dowiedziałeś się, że do funkcji uruchamianych w odpowiedzi na wystąpienie zdarzenia automatycznie przekazywany jest obiekt zdarzenia. Tu do przechowywania tego obiektu

służy zmienna `evt`. Użyjesz jej w następnym kroku, aby zablokować przejście do następnej strony po kliknięciu odnośnika.

3. **Między dwoma wierszami kodu dodanymi w kroku 2. wpisz instrukcję `evt.preventDefault();`.**

   Standardowo kliknięcie odnośnika powoduje wczytanie określonego w nim dokumentu (strony, pliku graficznego, dokumentu PDF i tak dalej). Tu odsyłacz ma umożliwiać wyświetlenie większej wersji zdjęcia przez przeglądarki bez obsługi języka JavaScript. Aby zapobiec przejściu do nowej strony w przeglądarkach używających JavaScriptu, należy uruchomić funkcję `preventDefault()` obiektu zdarzenia (patrz strona 195).

   Następnie trzeba pobrać wartość atrybutu `href` odnośnika.

4. **Wciśnij klawisz *Enter*, aby utworzyć nowy, pusty wiersz. Wpisz w nim instrukcję wyróżnioną na poniższym przykładzie pogrubioną czcionką:**

```
$('#gallery a').click(function(evt) {
  evt.preventDefault();
  var imgPath = $(this).attr('href');
}); // Koniec funkcji click.
```

   Konstrukcja `$(this)` wskazuje tu na kliknięty element, czyli odnośnik. Jego atrybut `href` zawiera adres strony lub zasobu, do którego prowadzi dany odsyłacz. Tu w odnośniku znajduje się ścieżka do większego zdjęcia. Jest to ważna informacja, ponieważ można jej użyć do dodania znacznika rysunku wyświetlającego ten plik. Jednak zanim to zrobisz, musisz pobrać referencję do dużego zdjęcia aktualnie widocznego na stronie. Trzeba przecież ustalić, który element skrypt ma stopniowo usuwać ze strony.

Wskazówka: Zauważ, że każdy wiersz kodu w zdarzeniu `click()` z punktu 4. ma wcięcie. Nie jest to wymagane, ale poprawia czytelność kodu, co opisano w ramce na stronie 67. Wielu programistów dodaje wcięcia o szerokości dwóch odstępów lub jednej tabulacji.

5. **Wciśnij klawisz *Enter* i wpisz:**

```
var oldImage = $('#photo img');
```

   Zmienna `oldImage` przechowuje element pobrany przez bibliotekę jQuery. Jest to znacznik `<img>`, zawarty w znaczniku `<div>` o identyfikatorze `photo` (patrz rysunek 7.4). Teraz należy utworzyć znacznik, w którym znajdzie się nowe zdjęcie.

6. **Ponownie wciśnij klawisz *Enter*, a następnie dodaj do skryptu następujący kod:**

```
var newImage = $('<img src="' + imgPath +'">');
```

   Jest to dość złożony wiersz. JQuery umożliwia pobieranie elementów umieszczonych w kodzie HTML strony, na przykład wyrażenie `$('img')` zwraca wszystkie rysunki ze strony. Ponadto za pomocą obiektów jQuery można dodawać do strony *nowe* elementy. Na przykład wyrażenie `$('<p>Witaj</p>')` tworzy nowy znacznik akapitu, zawierający słowo „Witaj". Dodany wiersz tworzy nowy znacznik `<img>` i zapisuje go w zmiennej `newImage`.

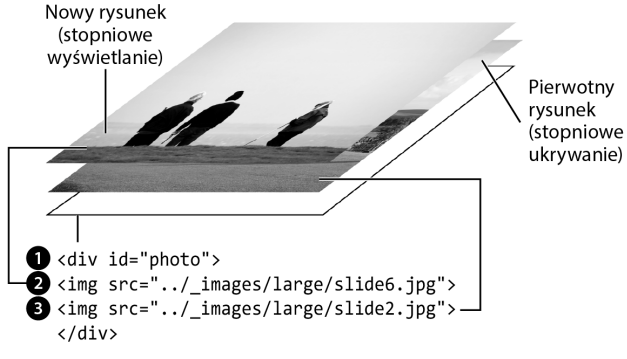
   Ponieważ obiekty jQuery przyjmują jako argument łańcuch znaków (na przykład `'<p>Witaj</p>'`), nowy wiersz *łączy* kilka łańcuchów w jeden. Pierw-

szy fragment tekstu (w apostrofach) to <img src= . Druga część znajduje się w zmiennej imgPath (utworzyłeś ją w kroku 4.) i zawiera ścieżkę do pliku graficznego (na przykład ../_images/large/slide1.jpg). Trzeci łańcuch (także w apostrofach) to >. Połączone fragmenty tworzą znacznik HTML typu <img src= ../_images/large/slide1.jpg >. Kiedy skrypt przekaże ten argument do obiektu jQuery — $('<img src= ../_images/large/slide2.jpg >') — przeglądarka utworzy nowy element strony. Na razie nie znajduje się on na stronie, ale przeglądarka może go dodać w dowolnym momencie.

7. **Dodaj kod z wierszy od 6. do 8. Gotowy fragment powinien wyglądać następująco:**

```
1 $('#gallery a').click(function(evt) {
2   evt.preventDefault();
3   var imgPath = $(this).attr('href');
4   var oldImage = $('#photo img');
5   var newImage = $('<img src="' + imgPath + '">');
6   newImage.hide();
7   $('#photo').prepend(newImage);
8   newImage.fadeIn(1000);
9 }); // Koniec funkcji click.
```

W wierszu 6. skrypt ukrywa nowy rysunek (zapisany w zmiennej newImage) przy użyciu funkcji hide() (patrz strona 212). Ten krok jest niezbędny, ponieważ jeśli po prostu dodasz znacznik rysunku utworzony w wierszu 5., rysunek zostanie wyświetlony natychmiast, bez atrakcyjnego efektu. Dlatego najpierw należy ukryć rysunek, a następnie dodać go do strony w znaczniku o identyfikatorze photo (wiersz 7.). Funkcja prepend() (patrz strona 159) dodaje kod HTML na początku znacznika. Na tym etapie znacznik zawiera dwa zdjęcia. Na rysunku 7.5 pokazano, że są one umieszczone jedno nad drugim. Fotografia górna jest niewidoczna, ale funkcja fadeIn() w wierszu 8. stopniowo wyświetla ją w ciągu 1000 milisekund (1 sekundy).



Rysunek 7.5. *Jeśli dwa zdjęcia mają znajdować się w tym samym miejscu, to aby stopniowo wyświetlić jedno i ukryć inne, trzeba użyć pomysłowego kodu CSS. Pozycjonowanie bezwzględne umożliwia umieszczenie elementu nad stroną, a nawet nad innym obiektem. Tu oba rysunki znajdują się w tym samym znaczniku , co sprawia, że jedno zdjęcie jest widoczne nad drugim. Arkusz stylów gallery.css z katalogu R07 dodawany w sekcji nagłówkowej strony zawiera wszystkie style potrzebne do uzyskania tego efektu (koniecznie przyjrzyj się stylowi #photo img). Oprócz tego, znacznik zawierający obrazki musi zostać umiejscowiony w sposób względny (atrybut CSS position o wartości relative), aby poszczególne zdjęcia były rozmieszczane względem niego*

Teraz trzeba stopniowo ukryć pierwotny rysunek.

8. **Wciśnij klawisz *Enter*, a następnie dodaj trzy poniższe wiersze kodu:**

```
oldImage.fadeOut(1000,function(){
  $(this).remove();
}); // koniec funkcji fadeOut.
```

W kroku 5. utworzyłeś zmienną `oldImage` i zapisałeś w niej referencję do pierwotnego rysunku. Skrypt ma go stopniowo ukryć, dlatego należy wywołać funkcję `fadeOut()`. Przyjmuje ona dwa argumenty. Pierwszy określa czas trwania efektu (1000 milisekund, czyli 1 sekundę), a drugi to funkcja zwrotna (patrz strona 223 w rozdziale 6.). Usuwa ona znacznik `<img>` danego rysunku, a skrypt wywołuje ją *po* stopniowym ukrywaniu zdjęcia.

Uwaga: Funkcję `remove()` opisano na stronie 162. Usuwa ona znacznik z modelu DOM, co powoduje skrócenie kodu HTML w pamięci przeglądarki i zwolnienie zasobów komputera. Jeśli pominiesz tę operację, przy każdym kliknięciu miniatury skrypt doda nowy znacznik `<img>` (krok 7.) i tylko ukryje poprzedni znacznik — nie usunie go. Dlatego na stronie znajdzie się mnóstwo niepotrzebnych, ukrytych znaczników `<img>`, co wydłuży czas reakcji przeglądarki.

Teraz trzeba wykonać ostatnią operację — wczytać pierwszy rysunek. Na razie znacznik `<div>` przeznaczony na zdjęcia jest pusty. Możesz dodać ręcznie dowolny znacznik `<img>`, aby po wczytaniu strony widoczna była większa wersja na przykład pierwszego miniaturowego zdjęcia. Jednak nie musisz tego robić — w końcu od takich zadań jest JavaScript!

9. **Dodaj wiersz na końcu funkcji `click()` (wiersz 13.). Gotowy kod powinien wyglądać następująco:**

```
1  $('#gallery a').click(function(evt) {
2    evt.preventDefault();
3    var imgPath = $(this).attr('href');
4    var oldImage = $('#photo img');
5    var newImage = $('<img src="' + imgPath + '">');
6    newImage.hide();
7    $('#photo').prepend(newImage);
8    newImage.fadeIn(1000);
9    oldImage.fadeOut(1000,function(){
10     $(this).remove();
11   }); // Koniec funkcji fadeOut.
12 }); // Koniec funkcji click.
13 $('#gallery a:first').click();
```

Ostatnia instrukcja ma dwie części. Pierwsza z nich to selektor `#gallery a:first`, który pobiera tylko pierwszy odnośnik ze znacznika `<div>` o identyfikatorze `gallery`. Na końcu znajduje się funkcja `click()`. Do tej pory używałeś funkcji `click()` biblioteki jQuery do określania funkcji uruchamianej w momencie wystąpienia zdarzenia. Jeśli jednak nie przekażesz do funkcji `click()` żadnego zdarzenia, jQuery po prostu zgłosi je, co spowoduje wywołanie wcześniej zdefiniowanych metod obsługi zdarzeń. Dlatego ten wiersz zgłasza zdarzenie `click` dla pierwszego odnośnika, co prowadzi do uruchomienia funkcji z wierszy od 1. do 11. Powoduje to wyświetlenie większej wersji pierwszego miniaturowego zdjęcia po wczytaniu strony.

Zapisz stronę i wyświetl ją w przeglądarce. Nie tylko miniatury będą zmieniać kolor po umieszczeniu nad nimi wskaźnika myszy, ale też kliknięcie małych obrazków spowoduje stopniowe wyświetlenie ich dużych wersji. Jeśli kod nie wykonuje tych operacji, jego działającą wersję znajdziesz w pliku *complete_gallery.html*.

Kontrola działania odnośników

Odnośniki to istota sieci WWW. Bez błyskawicznego dostępu do informacji przez odsyłacze łączące strony i witryny rozwój sieci WWW byłby niemożliwy — w ogóle nie powstałaby *sieć*. Ponieważ odnośniki to jeden z najczęściej używanych i najważniejszych elementów języka HTML, naturalne jest, że istnieje wiele technik usprawniania ich działania opartych na języku JavaScript. W tym rozdziale poznasz podstawowe sposoby używania tego języka do obsługi odsyłaczy i metody otwierania odnośników w nowych oknach.

Z pewnością wiesz już wiele o odnośnikach. W końcu są one istotą sieci WWW, a skromny znacznik `<a>` jest jednym z pierwszych elementów języka HTML, jakie poznaje każdy projektant stron internetowych. Za pomocą kodu JavaScript można zmienić prosty odnośnik w bramę do interaktywnego świata, jednak trzeba najpierw nauczyć się kontrolować odsyłacze za pomocą tego języka. Gdy zapoznasz się z podstawami, w dalszej części rozdziału zobaczysz kilka praktycznych technik zarządzania odnośnikami przy użyciu kodu JavaScript.

Pobieranie odnośników w kodzie JavaScript

Aby można było manipulować odnośnikiem, trzeba go najpierw pobrać. Możesz wskazać wszystkie odsyłacze ze strony, tylko jeden z nich lub grupę powiązanych odnośników, które na przykład znajdują się w tej samej części strony lub prowadzą do innych witryn.

Biblioteka jQuery umożliwia pobieranie elementów dokumentu na wiele sposobów. Przykładowo wyrażenie `$('a')` tworzy kolekcję wszystkich odnośników ze strony. Ponadto jQuery umożliwia doprecyzowanie wyrażenia i szybkie pobranie wszystkich odsyłaczy z określonego obszaru strony. Jeśli chcesz zapisać na przykład wszystkie odnośniki z listy wypunktowanej o identyfikatorze `mainNav`, możesz użyć kodu `$('#mainNav a')`. Ponadto selektory atrybutów (patrz strona 152) umożliwiają pobranie odnośników, których atrybut `href` (ścieżka do pliku, do którego prowadzi dany odsyłacz) ma wartość pasującą do wzorca. Można w ten sposób pobrać odnośniki prowadzące do innych witryn lub plików PDF (przykład zastosowania tej techniki znajdziesz na stronie 153).

Po pobraniu odnośników za pomocą jQuery można nimi manipulować przy użyciu funkcji tej biblioteki. Możesz na przykład wywołać funkcję `each()` (patrz strona 167), aby w pętli dodać do każdego odsyłacza klasę, używając do tego celu funkcji `addClass()` (patrz strona 162), a nawet zdarzenie (patrz strona 182). W dalszej części rozdziału zobaczysz wiele operacji, które można wykonywać na odnośnikach.

Określanie lokalizacji docelowej

Po pobraniu odnośników trzeba czasem sprawdzić, dokąd prowadzą. W galerii fotografii utworzonej na stronie 249 każdy odsyłacz wskazywał na większy rysunek. Kod JavaScript pobierał ścieżkę do odpowiedniego pliku graficznego i wyświetlał go,

a dokładniej — pobierał wartość atrybutu `href` odnośnika i używał go do utworzenia na stronie nowego znacznika `<img>`. Można też sprawdzić wartość atrybutu `href` i — jeśli odnośnik prowadzi do innej strony — wyświetlić ją „nad" bieżącą, zamiast przechodzić pod nowy adres.

Funkcja `attr()` biblioteki jQuery (patrz strona 166) zapewnia łatwy dostęp do atrybutu `href`. Załóżmy, że odnośnik prowadzący do strony głównej witryny ma określony identyfikator. Ścieżkę zapisaną w tym odsyłaczu można pobrać w następujący sposób:

```
var homePath = $('#homeLink').attr('href');
```

Informacje te są przydatne w wielu sytuacjach. Czasem programista chce umieścić obok odnośnika pełny adres URL, jeśli odsyłacz prowadzi poza witrynę. Może to być odnośnik z tekstem „Dowiedz się więcej o chrząszczach", prowadzący pod adres *http://www.barkbeetles.org* . Warto zmienić ten tekst na „Dowiedz się więcej o chrząszczach (*www.barkbeetles.org*)", aby użytkownicy po wydrukowaniu strony wiedzieli, dokąd prowadzi dany odsyłacz.

Można łatwo osiągnąć pożądany efekt za pomocą poniższego kodu JavaScript:

```
1 $('a[href^=http://]').each(function() {
2   var href = $(this).attr('href');
3   href = href.replace('http://','');
4   $(this).after(' (' + href + ')');
5 });
```

Uwaga: Numery wierszy nie są częścią kodu, dlatego nie przepisuj ich. Podano je tylko w celu ułatwienia opisu kodu wiersz po wierszu.

Wiersz 1. pobiera wszystkie odnośniki do zewnętrznych stron, a następnie uruchamia funkcję `each()` (patrz strona 167), która wykonuje podaną dalej funkcję dla wszystkich znalezionych odsyłaczy (czyli pobiera i przetwarza każdy z nich w „pętli"). Ta następna funkcja znajduje się w wierszach od 2. do 4. Wiersz 2. pobiera wartość atrybutu `href` odnośnika (na przykład *http://www.barkbeetles.org*). Wiersz 3. jest opcjonalny. Jego zadaniem jest uproszczenie adresu URL przez usunięcie członu „http://" (zmienna `href` zawiera po tej operacji adres typu *www.barkbeetles.org*; więcej informacji na temat metody `replace()` języka JavaScript można znaleźć na stronie 585). Wiersz 4. dodaje do tekstu odnośnika zawartość zmiennej `href` w nawiasach — `(www.barkbeetles.org)`. Wiersz 5. kończy funkcję.

To proste rozwiązanie można nieco rozwinąć na przykład po to, by u dołu strony utworzyć bibliografię zawierającą listę wszystkich odnośników umieszczonych w tekście prezentowanego artykułu. W takim przypadku zamiast dodawania adresów po każdym odnośniku można adresy te umieścić u dołu strony, w osobnym elemencie `div`. Wypróbuj ten kod!

Blokowanie domyślnego działania odnośników

Kiedy dodajesz zdarzenie `click` do odnośnika, zwykle nie chcesz, aby jego kliknięcie powodowało opuszczenie bieżącej strony i przejście do lokalizacji docelowej.

W galerii zdjęć ze strony 249 w odpowiedzi na kliknięcie odsyłacza na stronie z miniaturami strona wczytywała większy rysunek. Domyślnie takie kliknięcie powoduje opuszczenie strony i wyświetlenie dużego zdjęcia w pustym oknie. Jednak w galerii użytkownik powinien pozostać na tej samej stronie, a skrypt ma wczytać większy obrazek.

Domyślne działanie odnośników można zablokować na kilka sposobów, na przykład przez zwrócenie wartości `false` lub wywołanie funkcji `preventDefault()` biblioteki jQuery (patrz strona 195). Załóżmy, że dany odnośnik prowadzi do strony logowania. Aby zwiększyć interaktywność witryny, można po kliknięciu tego odsyłacza użyć kodu JavaScript do wyświetlenia formularza logowania. Jeśli przeglądarka obsługuje język JavaScript, po kliknięciu odnośnika na stronie pojawi się formularz. Jeżeli użytkownik wyłączył obsługę tego języka, odsyłacz zabierze go do strony logowania.

Aby uzyskać ten efekt, trzeba wykonać kilka operacji.

1. **Pobrać odnośnik do strony logowania.**

   Na początku tego podrozdziału znajdziesz możliwe rozwiązania tego problemu.

2. **Dołączyć zdarzenie `click`.**

   Aby wykonać to zadanie, użyj funkcji `click()` biblioteki jQuery. Funkcja ta przyjmuje inną funkcję jako argument. W tej drugiej funkcji należy umieścić operacje wykonywane po kliknięciu odnośnika. Tu będzie ona zawierać tylko dwie instrukcje.

3. **Wyświetlić formularz logowania.**

   Formularz logowania bezpośrednio po wczytaniu strony powinien być ukryty. Można go zawrzeć w znaczniku `<div>` i umieścić za pomocą pozycjonowania bezwzględnego bezpośrednio pod odnośnikiem. Aby wyświetlić formularz, użyj funkcji `show()` lub jednego z innych efektów biblioteki jQuery (patrz strona 212).

4. **Zablokować domyślne działanie odnośnika!**

   Ten krok jest najważniejszy. Jeśli nie zablokujesz odnośnika, przeglądarka opuści bieżącą stronę i przejdzie do strony logowania.

Poniższy fragment zatrzymuje działanie odnośnika przez zwrócenie wartości `false`. Użyty odsyłacz ma identyfikator `showForm`, a znacznik `<div>` w formularzu logowania — identyfikator `loginForm`:

```
1 $('#showForm').click(function(){
2   $('#loginForm').fadeIn('slow');
3   return false;
4 });
```

Wiersz 1. odpowiada etapom 1. i 2. opisanym na powyższej liście. Wiersz 2. wyświetla ukryty formularz. Wiersz 3. to informacja dla przeglądarki: „Zatrzymaj się! Nie przechodź pod adres odnośnika". Wiersz `return false;` trzeba umieścić na końcu funkcji, ponieważ interpreter wyjdzie z niej, kiedy napotka instrukcję `return`.

Można też użyć funkcji `preventDefault()` biblioteki jQuery:

```
1 $('#showForm').click(function(evt){
2   $('#loginForm').fadeIn('slow');
3   evt.preventDefault();
4 });
```

Skrypt ten działa podobnie jak poprzedni fragment. Podstawowa różnica polega na tym, że funkcja przypisana do zdarzenia `click` przyjmuje argument `evt`, który reprezentuje zdarzenie (obiekt zdarzenia opisano na stronie 194). Zdarzenia mają specyficzne funkcje i właściwości. Funkcja `preventDefault()` blokuje domyślne działanie powiązane z danym zdarzeniem (w przypadku kliknięcia odnośnika takim działaniem jest wczytanie nowej strony).

Otwieranie zewnętrznych odnośników w nowym oknie

Właściciele witryn, które istnieją dzięki dużej liczbie użytkowników, chcą zatrzymać odwiedzających na własnych stronach. Magazyny internetowe są utrzymywane z dochodów z reklam, dlatego nie powinny odsyłać użytkowników poza witrynę, jeśli można tego uniknąć. Sklepy internetowe nie chcą tracić kupujących w wyniku kliknięcia odnośnika prowadzącego poza witrynę. Projektant stron WWW umieszczający na stronie odnośniki do ukończonych projektów nie chce, aby potencjalny klient opuszczał jego witrynę w celu przyjrzenia się jednej z gotowych stron.

Dlatego w wielu witrynach strony zewnętrzne są otwierane w nowym oknie. Kiedy użytkownik zakończy przeglądanie tej strony i zamknie okno, pierwotna witryna wciąż będzie dostępna. Język HTML od dawna umożliwia uzyskanie tego efektu za pomocą atrybutu `target` odnośnika. Jeśli przypiszesz do tego atrybutu wartość `_blank`, przeglądarka wykryje, że ma otworzyć odnośnik w nowym oknie (lub na nowej karcie).

Uwaga: Eksperci z dziedziny użyteczności strony WWW nie są zgodni, czy otwieranie nowych okien to dobre, czy złe rozwiązanie. Zobacz na przykład artykuł *http://www.nngroup.com/articles/the-top-ten-web-design-mistakes-of-1999/*.

Samodzielne, ręczne dodawanie atrybutu `target= _blank` do wszystkich odnośników wskazujących strony spoza naszej witryny może być czasochłonne, a co gorsza, łatwo o takiej modyfikacji zapomnieć. Na szczęście za pomocą języka JavaScript i biblioteki jQuery można w szybki oraz łatwy sposób sprawić, aby przeglądarka otwierała odnośniki do stron zewnętrznych (lub dowolnych innych lokalizacji) w nowym oknie lub na nowej karcie. Ogólny proces jest całkiem prosty.

1. **Pobierz odnośniki, które chcesz otwierać w nowym oknie.**

   W tym rozdziale użyjesz do tego selektora biblioteki jQuery (patrz strona 148).

2. **Dodaj do odnośnika atrybut `target` o wartości `_blank`.**

   Możesz się zastanawiać: „Przecież to nieprawidłowy kod HTML! Nie mogę tego zrobić". Po pierwsze, jest on niepoprawny tylko w wersjach Strict języków HTML 4.01 i XHTML 1.0, dlatego można go stosować w dokumentach innych typów, takich jak HTML5. Po drugie, strona przejdzie walidację, ponieważ walidatory kodu HTML (na przykład *http://validator.w3.org*) sprawdzają tylko kod umieszczony w pliku ze stroną, a nie fragmenty dodawane za pomocą języka

JavaScript. Po trzecie, każda przeglądarka obsługuje atrybut `target`, dlatego możesz mieć pewność, że strona otworzy się w nowym oknie niezależnie od użytej wersji języka.

Za pomocą biblioteki jQuery można zrealizować dwa powyższe cele w jednym wierszu kodu:

```
$('a[href^="http://"]').attr('target','_blank');
```

Selektor jQuery — `$('a[href^= http:// ]')` — to selektor atrybutów (patrz strona 152) pobierający znaczniki `<a>`, w których atrybut `href` zaczyna się od członu „http://" (na przykład *http://www.yahoo.com*). Następnie skrypt wywołuje funkcję `attr()` biblioteki jQuery (patrz strona 166) w celu przypisania wartości `_blank` do atrybutu `target`. I to już wszystko!

Jeśli przypuszczasz, że na stronie pojawią się także adresy do bezpiecznych stron WWW, rozpoczynające się od *https://*, powinieneś użyć następującego wywołania:

```
$('a[href^="http://"], a[href^="https://"]').attr('target','_blank');
```

Korzysta ono z selektora grupowego i pozwala na wybieranie adresów URL rozpoczynających się od *http://* lub *https://*.

I w końcu, jeśli używasz ścieżek bezwzględnych także do wskazywania plików we własnej witrynie, musisz wykonać jeszcze jedną operację. W witrynie o adresie *www.nazwa_witryny.com*, w której odnośniki do innych stron i plików z tej witryny mają postać *http://www.nazwa_witryna.com/strona.html*, także te strony będą wyświetlane w nowym oknie. Jeśli chcesz ustrzec nieszczęsnych użytkowników przed otwieraniem każdej strony w nowym oknie, musisz użyć następującego kodu:

```
var myURL = location.protocol + '//' + location.hostname;
$('a[href^="http://"], a[href^="https://"]')
➥.not('[href^="'+myURL+'"]').attr('target','_blank');
```

Uwaga: Symbol ➥ na początku wiersza informuje, że jest on kontynuacją poprzedniego. *Ponieważ naprawdę długie wiersze kodu JavaScript nie mieszczą się na stronach książki, podzielono je na dwie części.*

Ten fragment najpierw zapisuje adres URL witryny w zmiennej `myURL`. Ten adres można uzyskać dzięki obiektowi okna przeglądarki. Za pomocą przeglądarki można wykryć protokół użyty do otwarcia strony — http lub (w przypadku stron zabezpieczonych) https. Wartość ta jest zapisana we właściwości `protocol` obiektu `location`. Z kolei nazwę witryny, na przykład *www.sawmac.com*, można pobrać z właściwości `hostname`. Dlatego fragment `location.protocol + '//' + location.hostname` generuje łańcuch znaków typu *http://www.sawmac.com*. Oczywiście nazwa witryny zależy od tego, skąd pochodzi dana strona z kodem JavaScript. Jeśli umieścisz powyższe instrukcje na stronie z witryny *http://www.nazwa_witryny.com*, właściwość `location.hostname` będzie miała wartość `www.nazwa_witryny.com`.

Drugi wiersz kodu rozpoczyna się od selektora jQuery, który pobiera wszystkie odnośniki rozpoczynające się od członu „http://". Funkcja `not()` usuwa odnośniki, które rozpoczynają się od adresu URL danej witryny, na przykład odnośniki wskazujące na stronę *http://www.sawmac.com*. Funkcja `not()` jest przydatna przy usuwaniu niektórych elementów z kolekcji pobranych za pomocą jQuery. Więcej informacji na jej temat zawiera strona *http://api.jquery.com/not*.

Aby użyć omawianego kodu na stronie, wystarczy dołączyć plik biblioteki jQuery, dodać funkcję `$(document).ready()` (patrz strona 190) i wstawić opisany wcześniej fragment:

```
<script src="js/jquery.min.js"></script>
<script>
$(document).ready(function() {
  var myURL = location.protocol + '//' + location.hostname;
  $('a[href^="http://"], a[href^="https://"]').not('[href^="'+myURL+'"]').
  ➥attr('target','_blank');
});
</script>
```

Inne rozwiązanie wymaga utworzenia zewnętrznego pliku JavaScript (patrz strona 49). Należy umieścić w nim funkcję, która wyświetla zewnętrzne odnośniki w nowym oknie. Ten plik trzeba dołączyć do strony, a następnie wywołać na niej dodaną funkcję.

Możesz na przykład przygotować plik *open_external.js* i umieścić w nim poniższy kod:

```
function openExt() {
  var myURL = location.protocol + '//' + location.hostname;
  $('a[href^="http://"] , a[href^="https://"]')
  ➥.not('[href^="'+myURL+'"]').attr('target','_blank');
}
```

Następnie do każdej strony, na której chcesz zastosować tę funkcję, dodaj poniższy kod:

```
<script src="js/jquery.min.js"></script>
<script src="js/open_external.js"></script>
<script>
$(document).ready(function() {
  openExt();
  // Pozostały kod JavaScript strony.
});
</script>
```

Zaletą pliku zewnętrznego jest to, że jeśli używasz funkcji na setkach stron, w przyszłości będziesz mógł łatwo uatrakcyjnić skrypt. Możesz na przykład tak zmodyfikować funkcję `openExt()`, aby otwierała odnośniki zewnętrzne w ramce na aktualnej stronie. Dlatego zewnętrzny plik *.js* ułatwia zachowanie spójnego działania skryptów w całej witrynie.

Tworzenie nowych okien

Przeglądarka umożliwia otwarcie nowego okna i ustawienie wielu jego właściwości, na przykład szerokości, wysokości, pozycji na ekranie. Możesz nawet określić, czy okno ma mieć paski przewijania, menu i pasek adresu. Do wyświetlania okien służy metoda `open()`, której podstawowa składnia wygląda następująco:

```
open(URL, nazwa, właściwości)
```

Metoda ta przyjmuje trzy argumenty. Pierwszy to adres URL strony otwieranej w nowym oknie. Jest to ta sama wartość, która pojawia się w atrybucie `href` odnośników (na przykład `http://www.google.com`, `/pages/map.html` lub `../../portfolio.html`). Drugi argument to nazwa okna, którą możesz ustalić dowolnie,

przestrzegając przy tym zasad nazywania zmiennych (patrz strona 63). Jako trzeci argument należy przekazać łańcuch znaków z ustawieniami nowego okna (na przykład wysokością i szerokością).

Ponadto przy otwieraniu nowego okna zwykle warto zapisać referencję do niego w zmiennej. Aby otworzyć stronę główną Google w oknie o wymiarach 200×200 pikseli, użyj następującego kodu:

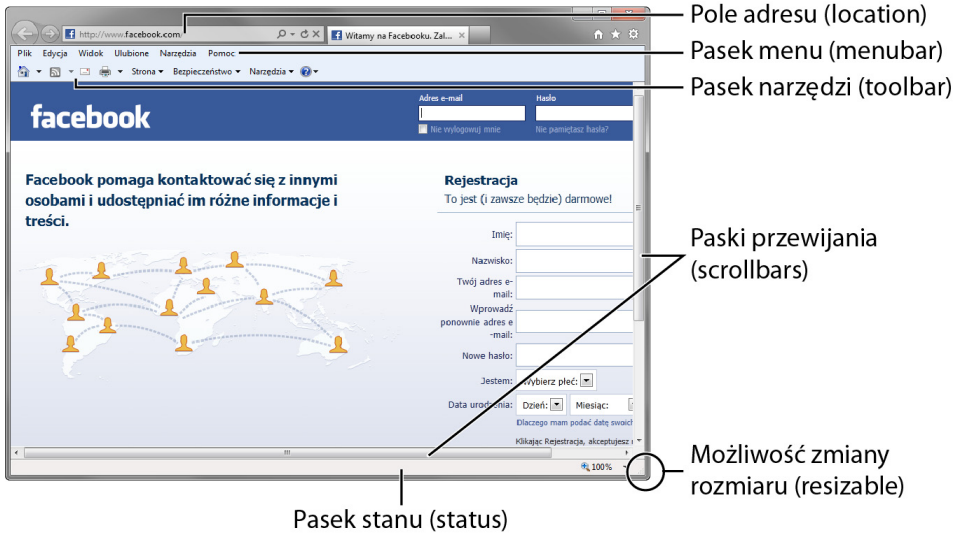
```
var newWin = open('http://www.google.com/',
➥'theWin','height=200,width=200');
```

Ten kod otwiera nowe okno i zapisuje referencję do niego w zmiennej `newWin`. W podpunkcie „Używanie referencji do okien" dowiesz się (patrz strona 263), jak użyć takiej referencji do kontrolowania nowego okna.

Uwaga: Nazwa nowego okna (tu jest to `theWin`) nie ma większego znaczenia. Jednak jeśli spróbujesz otworzyć następne okno przy użyciu tej nazwy, przeglądarka nie utworzy go, lecz wczyta stronę zażądaną w metodzie `open()` we wcześniej zapisanym oknie o danej nazwie.

Właściwości okien

Okno przeglądarki zawiera wiele elementów: paski przewijania, uchwyty do zmiany rozmiaru, paski narzędzi i tak dalej (patrz rysunek 7.6). Ponadto okna mają szerokość, wysokość i pozycję na ekranie. Większość tych właściwości można ustawić



Rysunek 7.6. *Różne właściwości okna przeglądarki, na przykład paski przewijania, paski narzędzi i uchwyty do zmiany rozmiaru, są określane wspólną nazwą — „chromowanie". W każdej przeglądarce obsługiwane są one nieco inaczej, a programiści aplikacji sieciowych mają małą kontrolę nad ich działaniem i wyglądem. Nie ma jednak żadnego powodu do rozpaczy — przy tworzeniu nowego okna za pomocą języka JavaScript można wyłączyć niektóre mechanizmy*

przy tworzeniu nowego okna. W tym celu należy przygotować łańcuch znaków z listą właściwości rozdzielonych przecinkami i przekazać go jako trzeci argument metody `open()`. Aby ustawić szerokość i wysokość nowego okna oraz dodać pasek adresu, trzeba użyć następującego kodu:

```
var winProps = 'width=400,height=300,location=yes';
var newWin = open('about.html','aWin',winProps);
```

Właściwości określające rozmiar i pozycję są podawane w pikselach, natomiast pozostałe ustawienia przyjmują wartość `yes` (włącza daną właściwość) lub `no` (wyłącza wybraną opcję). Jeśli nie podasz wartości właściwości typu `yes-no`, takiej jak `toolbar` lub `location`, przeglądarka wyłączy ją. Na przykład brak wartości `location` spowoduje ukrycie pola adresu, które standardowo pojawia się w górnej części okna. Jedynie właściwości `height`, `width`, `left`, `top` oraz `toolbar` działają spójnie we wszystkich przeglądarkach. Zgodnie z informacjami podanymi na poniższej liście, istnieją przeglądarki, które całkowicie ignorują niektóre z właściwości, dlatego, wyświetlając okienka przy użyciu JavaScriptu, koniecznie należy przetestować ich działanie we wszystkich możliwych przeglądarkach.

- Właściwość `height` określa wysokość okna w pikselach. Nie można użyć wartości procentowych ani żadnych innych jednostek. Jeśli nie ustawisz tej właściwości, przeglądarka użyje wysokości obecnie wyświetlanego okna.
- Właściwość `width` określa szerokość okna. Podobnie jak w ustawieniu `height`, obsługiwane są tylko wartości w pikselach, a brak tej właściwości powoduje użycie szerokości obecnie wyświetlanego okna.
- Właściwość `left` określa odległość w pikselach od lewej krawędzi monitora.
- Właściwość `top` określa odległość w pikselach od górnej krawędzi monitora.
- Właściwość `scrollbars` informuje, czy przy prawej i dolnej krawędzi mają pojawić się paski przewijania, jeśli strona jest większa od okna. Aby całkowicie ukryć paski przewijania, ustaw tę właściwość na `no`. Nie można określić, który pasek przewijania ma być widoczny — to ustawienie typu „wszystko albo nic”; poza tym niektóre przeglądarki, takie jak Chrome oraz Safari, w ogóle nie pozwalają ukrywać pasków przewijania.
- Właściwość `status` kontroluje wygląd paska stanu widocznego w dolnej części okna. Przeglądarki Firefox i Internet Explorer standardowo nie umożliwiają ukrycia go, dlatego pasek stanu jest w nich zawsze widoczny.
- Właściwość `toolbar` określa, czy widoczny jest pasek narzędzi z przyciskami nawigacyjnymi, kartami i innymi kontrolkami dostępnymi w danej przeglądarce. W Safari paski narzędzi i adresu stanowią całość. Włączenie jednego z nich powoduje wyświetlenie przycisków paska narzędzi i pola adresu.
- Właściwość `location` określa, czy dostępne jest pole adresu. Znajduje się w nim adres URL strony, a użytkownik może wpisać nową ścieżkę, aby przejść do innej lokalizacji. Przeglądarki Opera, Internet Explorer i Firefox nie umożliwiają całkowitego ukrycia adresu. Jeśli nie włączysz właściwości `location`, adres URL pojawi się w pasku tytułu. To ograniczenie ma zapobiec szkodliwemu wykorzystaniu języka JavaScript do takich operacji jak otwieranie nowego okna i przenoszenie użytkownika do innej witryny, która wygląda tak samo jak pierwotna. Jeśli włączysz tę opcję w przeglądarce Safari, oprócz pola adresu pojawi się także pasek narzędzi.

- Właściwość `menubar` ma zastosowanie w przeglądarkach, które udostępniają menu w górnej części okna (na przykład standardowe menu *Plik* i *Edycja* znane z większości programów). To ustawienie działa tylko w systemie Windows. W komputerach Mac menu znajduje się w górnej części ekranu, a nie w poszczególnych oknach. Właściwość ta nie ma zastosowania także w Internet Explorerze 7 oraz kolejnych wersjach, które domyślnie nie wyświetlają paska menu.

Uwaga: Przykłady zadziwiających skryptów wykorzystujących metodę window.open() można znaleźć na stronach: *http://experiments.instrum3nt.com/markmahoney/ball/* oraz *http://www.thewildernessdowntown.com*.

Używanie referencji do okien

Po otwarciu nowego okna można je kontrolować za pomocą powiązanej z nim referencji. Załóżmy, że otworzyłeś okno za pomocą poniższego kodu:

```
var newWin = open('products.html','theWin','width=300,height=300');
```

Ten wiersz zapisuje w zmiennej `newWin` referencję do nowego okna. Następnie można wywołać dla tej zmiennej wszystkie metody okna przeglądarki, aby je kontrolować. Jeśli zechcesz zamknąć okno, użyj metody `close()`:

```
newWin.close();
```

Przeglądarki obsługują wiele metod obiektu okna. Poniżej opisano kilka funkcji najczęściej używanych do kontrolowania samego okna.

- Metoda `close()` zamyka podane okno. Samo polecenie `close()` zamyka bieżące okno, jednak można użyć także referencji, na przykład `newWin.close()`. Metodę tę może wywoływać dowolne zdarzenie, na przykład kliknięcie myszą przycisku z etykietą „Zamknij okno".

Uwaga: Jeśli używasz jednego z opisanych poleceń bez podania obiektu, przeglądarka uruchomi je dla okna, w którym działa skrypt. Wywołanie `close();` w programie zamyka okno ze stroną, na której wywołano skrypt. Jednak jeśli otworzyłeś nowe okno i zapisałeś referencję do niego (na przykład w zmiennej utworzonej przy jego otwieraniu, takiej jak `newWin`), możesz zamknąć okno z poziomu strony, na której je utworzono, używając tej referencji — `newWin.close()`.

- Metoda `blur()` powoduje przeniesienie aktywności poza dane okno, czyli ukrycie go za innymi otwartymi oknami. Jest to sposób na schowanie wyświetlonego okna, a reklamodawcy wykorzystują go do tworzenia reklam „pop under". Kiedy użytkownik zamknie wszystkie okna, będzie na niego czekać irytująca reklama.
- Metoda `focus()` (przeciwieństwo funkcji `blur()`) powoduje wyświetlenie okna przed pozostałymi.
- Metoda `moveBy()` pozwala przenieść okno o podaną liczbę pikseli w prawo i w dół. Przyjmuje ona dwa argumenty. Pierwszy określa w pikselach przesunięcie w prawo, a drugi — przesunięcie w dół. Na przykład wywołanie `newWin.moveBy ↪(200,300);` przenosi okno zapisane w zmiennej `newWin` o 200 pikseli w prawo i 300 pikseli w dół ekranu. W wywołaniu metody można także przekazywać wartości ujemne, a zatem, aby przesunąć okno o 100 pikseli w górę oraz 300 pikseli w lewo, należałoby użyć następującego wywołania:

```
newWin.moveBy(-100,-300);
```

- Metoda `moveTo()` przenosi okno w miejsce ekranu określone przez docelowe współrzędne jego lewego górnego rogu. To polecenie odpowiada ustawieniu właściwości `left` i `top` (patrz strona 262) przy otwieraniu nowego okna. Aby na przykład umieścić okno w lewym górnym rogu monitora, użyj wywołania `moveTo(0,0)`.

Uwaga: Działające przykłady skryptów wykorzystujących wiele tych metod można znaleźć na stronie All Is Not Lost: *http://www.allisnotlo.st/*.

- Metoda `resizeBy()` zmienia szerokość i wysokość okna. Przyjmuje dwa argumenty. Pierwszy określa w pikselach zmianę szerokości, a drugi — wysokości okna. Na przykład polecenie `resizeBy(100,200);` sprawia, że okno będzie o 100 pikseli szersze i 200 pikseli wyższe. Liczby ujemne pozwalają zmniejszyć okno.
- Metoda `resizeTo()` zmienia wymiary okna zgodnie z podaną szerokością i wysokością. Na przykład wywołanie `resizeTo(200,400);` powoduje, że okno będzie miało 200 pikseli szerokości i 400 pikseli wysokości.
- Metoda `scrollBy()` przewija dokument w oknie o określoną liczbę pikseli w prawo i w dół. Na przykład polecenie `scrollBy(100,200);` przesuwa dokument o 200 pikseli w dół i 100 pikseli w prawo. Jeśli przewinięcie jest niemożliwe (dokument w całości mieści się w oknie lub został przewinięty do końca), wywołanie funkcji nie przynosi żadnych skutków.
- Metoda `scrollTo()` przewija dokument w oknie do określonej pozycji. Na przykład instrukcja `scrollTo(100,200);` powoduje wyświetlenie dokumentu przesuniętego o 200 pikseli w dół i 100 pikseli w prawo. Jeśli przewinięcie jest niemożliwe (dokument w całości mieści się w oknie lub został przewinięty do końca), wywołanie funkcji nie przynosi żadnych skutków.

Wskazówka: Wtyczka ScrollTo biblioteki jQuery umożliwia wygodne kontrolowanie procesu przewijania dokumentu za pomocą kodu JavaScript. Więcej informacji o tej wtyczce znajdziesz na stronie *https://github.com/flesler/jquery scrollTo*.

Zdarzenia, które mogą otwierać nowe okna

W krótkiej historii sieci WWW okna wyskakujące zyskały złą sławę. Niestety, w wielu witrynach nadużywa się metody `open()` do wyświetlania niepożądanych nowych okien zaskoczonym użytkownikom. Obecnie większość przeglądarek umożliwia zablokowanie okien wyskakujących, dlatego jeśli nawet dodasz kod JavaScript, który otwiera takie okno bezpośrednio po wczytaniu strony lub zamknięciu okna, przeglądarka nie pozwoli na jego wyświetlenie. Odwiedzający albo zobaczy informację o zablokowaniu okna, albo w ogóle się nie dowie o próbie jego wyświetlenia.

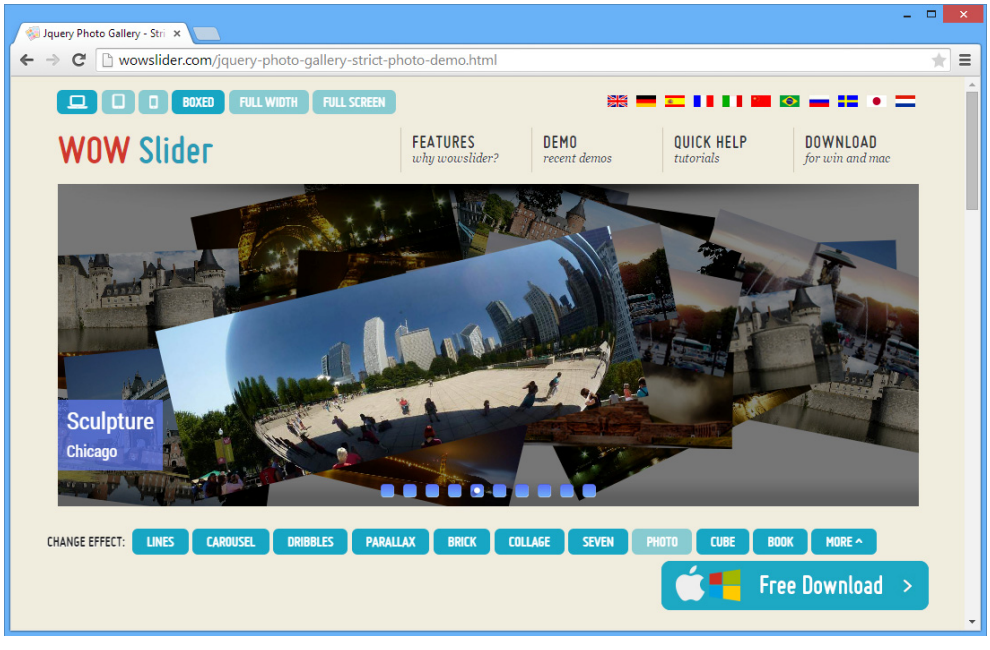
Większość zdarzeń, na przykład `mouseover`, `mouseout` i `keypress`, w wielu przeglądarkach nie może prowadzić do otwarcia nowego okna. Jedyny niezawodny sposób otwarcia okna za pomocą kodu JavaScript polega na wykonaniu tej operacji po kliknięciu odnośnika lub przesłaniu formularza. Dlatego trzeba dołączyć zdarzenie `click` do dowolnego elementu HTML (nie musi to być odnośnik) i otworzyć nowe okno. Załóżmy, że chcesz, aby niektóre odsyłacze otwierały stronę w nowym, kwa-

dratowym oknie o boku 300 pikseli. Okno to ma mieć paski przewijania i umożliwiać zmianę rozmiaru. Pozostałe elementy „chromowania", takie jak pasek narzędzi, mają być niedostępne. Do każdego specjalnego odnośnika należy dodać klasę, na przykład `popup`, a następnie umieścić na stronie poniższy kod oparty na bibliotece jQuery:

```
$('.popup').click(function() {
  var winProps='height=300,width=300,resizable=yes,scrollbars=yes';
  var newWin=open($(this).attr('href'),'aWin',winProps);
}
```

Przedstawienie wtyczek jQuery

Kiedy dysponujesz znajomością języka JavaScript i biblioteki jQuery, wiesz wszystko, czego Ci potrzeba, by w kilku wierszach kodu dodawać do stron użyteczne możliwości interakcji z użytkownikiem. Być może poszukujesz jednak jeszcze ciekawszych i bardziej złożonych dodatków do tworzenia interfejsu użytkownika stron. A co byś powiedział na *slider*? To bardzo popularny element stron, pozwalający na wyświetlanie sekwencji zdjęć jedno po drugim i prezentujący je na dużym obszarze strony (patrz rysunek 7.7). Takie slidery mogą prezentować zdjęcia, klipy wideo i zwyczajne znaczniki `<div>` z treścią HTML. Użytkownicy mogą zmieniać prezentowane slajdy, klikając przyciski; mogą też kliknąć sam slajd, by przejść na inną stronę witryny.



Rysunek 7.7. *Wow Slider (dostępny na stronie http://wowslider.com/) jest wtyczką jQuery o bogatych możliwościach, pozwalającą na bardzo łatwe tworzenie animowanych slajdów. Udostępnia wiele opcji prezentacji slajdów oraz umożliwia tworzenie interesujących i pięknych efektów przejść pomiędzy poszczególnymi slajdami*

Tworzenie sliderów (nazywanych także czasami „karuzelami") może być trudne i złożone. Na szczęście jedną z ogromnych zalet biblioteki jQuery jest obszerny ekosystem *wtyczek*. Wtyczka jQuery to plik JavaScript, działający w połączeniu z biblioteką. Takie wtyczki wykonują przeróżne interesujące rzeczy, na przykład mogą tworzyć na stronie slidery, pomagać w weryfikacji danych wpisywanych przez użytkowników w formularzach i tak dalej. Biblioteka jQuery UI, którą poznasz w trzeciej części tej książki, jest bardzo dużą wtyczką jQuery pozwalającą na tworzenie kontrolek przydatnych podczas pisania dużych aplikacji internetowych. Jednak przeważająca większość wtyczek jQuery to niewielkie pliki JavaScript, które w bardzo dobry sposób realizują stosunkowo proste zadania.

Wow Slider przedstawiony na rysunku 7.7 jest wtyczką komercyjną (co oznacza, że należy go kupić, aby mógł znaleźć się na witrynie firmowej). Jednak bardzo wiele wtyczek jest dostępnych bezpłatnie i w formie kodu otwartego, co oznacza, że nie tylko można ich używać bezpłatnie, lecz co ważniejsze, można otworzyć źródłowy plik JavaScript i zobaczyć, jak jest napisany. Analiza kodu pisanego przez innych programistów jest wspaniałym sposobem poznawania metod tworzenia wtyczek, zapewniającym także możliwość poprawiania ich poprzez wprowadzanie odpowiednich zmian w kodzie.

W tej książce poznasz kilka przydatnych wtyczek jQuery, jednak na internecie można ich znaleźć dosłownie tysiące. Doskonałym miejscem do rozpoczęcia spotkania z wtyczkami jQuery jest ich katalog, dostępny na stronie *http://plugins.jquery.com/*. Kolejne wtyczki można znaleźć, przeglądając zasoby takich serwisów jak *Sitepoint.com* lub *WebDesignerDepot.com* bądź po prostu szukając ich w sieci. Przykładowo wpisanie w wyszukiwarce frazy *google maps jquery plugin* spowoduje wyświetlenie wielu, najprawdopodobniej całkiem przydatnych wtyczek do umieszczania na stronach map firmy Google.

Czego szukać we wtyczce jQuery?

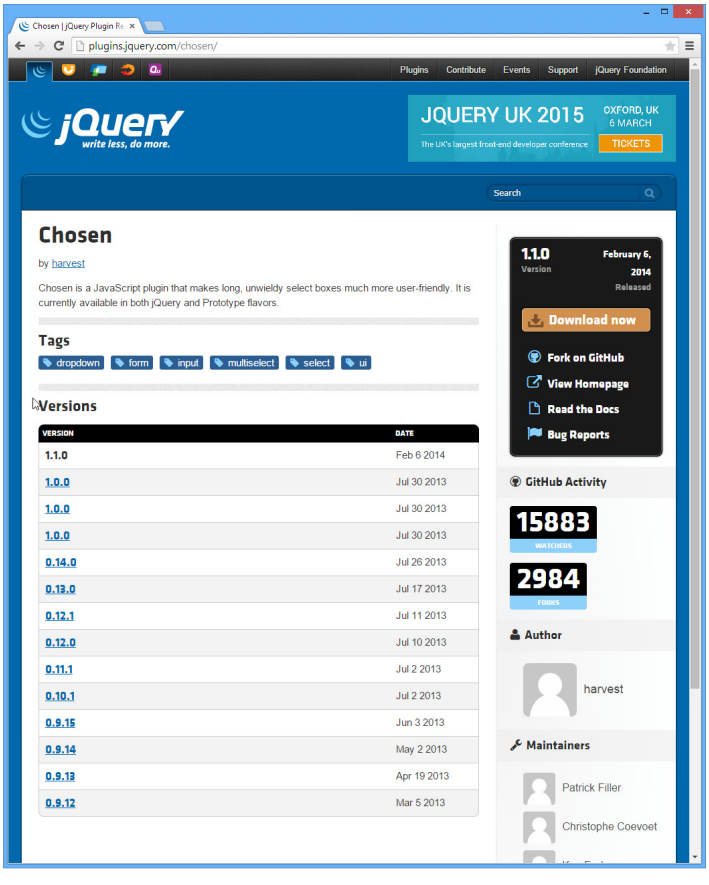
Zanim skopiujesz pierwszą wtyczkę jQuery, jaką znajdziesz, i spróbujesz jej użyć, powinieneś się zastanowić, czy rzeczywiście jej potrzebujesz. Na pewno znajdziesz bardzo dużo wtyczek, które robią wiele interesujących i fajnych rzeczy, i łatwo można ulec pokusie, by z nich skorzystać, bo wyglądają fantastycznie i dają odjazdowe możliwości. Problem polega na tym, że bez trudu można dodać do witryny bardzo wiele wtyczek, a to zmusi użytkowników do ich pobierania. To z kolei doprowadzi do wydłużenia czasu pobierania strony, a co więcej, potencjalnie może także przyczynić się do spowolnienia działania przeglądarek użytkowników, które, wyświetlając stronę, będą musiały trudzić się i wykonywać wiele programów JavaScript.

Co więcej, dodając do witryny wtyczkę, stajemy się zależni od jej twórcy. Jeśli w udostępnionej wersji wtyczki znajduje się niewykryty błąd, to Ty bądź twórca wtyczki będziecie musieli go poprawić. Zawsze można też zrezygnować ze stosowania takiej wtyczki.

Dlatego warto ograniczyć się do stosowania tylko tych wtyczek, które są naprawdę potrzebne, a także poszukiwać istniejących od dawna i popularnych. Jeśli wtyczka została dodana do katalogu dziś rano i ma numer wersji 0.0.0.1, lepiej

dwa razy się zastanów, zanim jej użyjesz. Stopień „dojrzałości" wtyczki można poznać po numerze wersji — numer 0.0.1 będzie sugerował, że wtyczka powstała bardzo niedawno, natomiast numer 4.1.10, że była już wielokrotnie usprawniana i rozbudowywana.

Warto także zwracać uwagę na datę wydania wtyczki. Witryna jQuery udostępnia informacje na temat każdej wtyczki dostępnej w jej katalogu. Przykładowo wtyczka Chosen (której listę wersji przedstawiono na rysunku 7.8) została opublikowana na witrynie jQuery 5 marca 2013 roku. Serwisy, takie jak GitHub (*http://github.com*), udostępniające bardzo wiele projektów o otwartym kodzie źródłowym, prezentują także datę utworzenia każdego z nich. Koniecznie należy także sprawdzić, kiedy wtyczka była ostatnio aktualizowana. Jeśli historia wtyczki pokazuje, że była wielokrotnie aktualizowana, poprawiana i rozwijana, będzie to sygnał, że istnieje już jakiś czas i najprawdopodobniej w przypadku wystąpienia problemów zostanie poprawiona przez twórców. Jeśli jednak wtyczka była ostatni raz aktualizowana cztery lata temu, lepiej trzymać się od niej z daleka — najprawdopodobniej nie będzie działać z najnowszą wersją biblioteki jQuery i nie ma żadnych wątpliwości, że nie przetestowano jej w najnowszych wersjach przeglądarek.



Rysunek 7.8. *Witryna jQuery zawiera własną listę dostępnych bezpłatnie wtyczek. Można ją znaleźć na stronie http://plugins.jquery.com/. Każda wtyczka ma swoją własną stronę, na której prezentowana jest historia jej rozwoju (data dodania oraz numery i daty publikacji jej kolejnych wersji). Można także zobaczyć, ile osób śledzi rozwój danej wtyczki (są oni określani jako „obserwujący" — „watchers"). Im więcej osób śledzi daną wtyczkę, tym bardziej jest popularna. Przykładowo aż 15883 osoby obserwują wtyczkę Chosen!*

Wtyczek można także szukać w wyszukiwarkach internetowych, takich jak Google lub Bing, a następnie przeczytać opinie i dyskusje na forach dyskusyjnych ich twórców. Jeśli znajdziesz wiele wpisów typu „Nie jestem w stanie uruchomić wtyczki XYZ w WordPressie" lub „Pomocy! Wtyczka XYZ kompletnie popsuła funkcjonalność mojej strony", to także i Ty możesz napotkać problemy.

Podstawy stosowania wtyczek jQuery

Choć poszczególne wtyczki jQuery różnią się pod względem złożoności i jakości, jednak sposób ich przygotowywania do użycia jest zazwyczaj podobny. Pobranie wtyczki sprowadza się do skopiowania na komputer jej pliku — zwykłego pliku *.js*. Oprócz niego zazwyczaj dostarczany jest także plik CSS, określający wizualną postać kodu HTML, który wtyczka dodaje do strony. Przykładowo niezwykle użyteczny kalendarz dostępny w bibliotece jQuery UI (patrz strona 375), dodający do pól formularzy HTML wyskakujący kalendarz, który niezwykle ułatwia wybieranie dat, jest formatowany przy użyciu dołączonego, zewnętrznego arkusza stylów. Do wtyczek często dodawane są także pliki graficzne, zawierające dodatkowe wizualne elementy stosowane w kodzie HTML.

Oto podstawowy proces korzystania z wtyczek.

1. **Pobierz pliki wtyczki.**

   Możesz je znaleźć na witrynie wtyczki, a coraz częściej także w serwisie GitHub. Wraz z samą wtyczką często pobierane są pliki używane podczas jej tworzenia, takie jak strona demonstracyjna, specjalne testy oraz pliki dodatkowe. Na razie będziesz potrzebował wyłącznie plików *.js*, *.css* oraz ewentualnie także plików graficznych.

2. **Przenieś pliki do katalogu witryny.**

   Operację możesz wykonać na kilka sposobów. Możesz przenieść plik *.js* wtyczki do tego samego katalogu, w którym przechowujesz wszystkie pozostałe pliki JavaScript, a plik *.css* do katalogu z pozostałymi arkuszami stylów. Jednak zazwyczaj najlepszym rozwiązaniem jest umieszczenie pliku *.css* oraz wszystkich plików graficznych wtyczki w jednym katalogu, gdyż zapewne arkusz stylów wtyczki będzie się do nich odwoływał.

   Inne rozwiązanie, które znacząco ułatwia zarządzanie wtyczkami (i ich ewentualne usuwanie), polega na umieszczeniu wszystkich komponentów danej wtyczki w jednym katalogu, o nazwie odpowiadającej nazwie wtyczki, a następnie umieszczeniu go w katalogu zawierającym inne pliki JavaScript lub w odrębnym katalogu przeznaczonym na wtyczki (o nazwie na przykład *plugins*). Załóżmy przykładowo, że chciałbyś używać wtyczki o nazwie *Super Plugin*, która składa się z plików *jquery.super-plugin.min.js*, *super-plugin.css* oraz katalogu o nazwie *images* zawierającego kilka dodatkowych plików. W takim przypadku mógłbyś utworzyć katalog o nazwie *super_plugin* i umieścić w nim wszystkie pliki wtyczki. Następnie ten katalog *super_plugin* mógłbyś umieścić na witrynie (na przykład w katalogu o nazwie *scripts* lub *plugins* — przeznaczonym wyłącznie do przechowywania wtyczek).

3. **Dołącz arkusz CSS wtyczki do strony WWW.**

   Ten kod umieścisz w sekcji `<head>` strony, bezpośrednio poniżej odwołań do innych używanych arkuszy stylów:

   ```
   <link href="css/site.css" rel="stylesheet">
   <link href="plugins/super_plugin/super-plugin.css" rel="stylesheet">
   ```

4. **Dołącz plik JavaScript wtyczki do strony WWW.**

   Wtyczki jQuery wymagają samej biblioteki jQuery, a zatem to właśnie ją najpierw należy dołączyć do strony; dopiero potem możesz dodać plik JavaScript wtyczki:

   ```
   <script src="js/jquery.min.js"></script>
   <script src="plugins/super_plugin/jquery.super-
   plugin.min.js"></script>
   ```

5. **Zmodyfikuj kod HTML strony.**

   Owszem, to polecenie jest dosyć niejasne. Jednak każda wtyczka jest inna i odmienne są zasady jej używania. W większości przypadków, aby uruchomić wtyczkę, musisz otworzyć plik HTML strony i dodać do niego odpowiedni kod. Zadanie to może być trywialne i sprowadzać się do dodania kilku klas do już istniejących elementów strony. Takie klasy będą działać jak „uchwyty" dla wtyczki, wskazujące jej, które elementy strony powinna pobrać i zmodyfikować.

   W innych przypadkach konieczne będzie dodanie do strony kodu HTML o ściśle określonej strukturze. Przykładem takiej wtyczki jest widżet akordeonu (patrz strona 363) wchodzący w skład biblioteki jQuery UI. Wymaga on zastosowania kodu składającego się z „kontenera" oraz serii par zawierających nagłówek oraz znacznik `<div>`; oto przykład takiego kodu:

   ```
   <div id="accordion">
     <h3>Pierwszy nagłówek</h3>
     <div>Pierwszy panel treści</div>
     <h3>Drugi nagłówek</h3>
     <div>Drugi panel treści</div>
   </div>
   ```

   Wtyczka w jakiś sposób zmieni sposób prezentacji kodu HTML: wyświetli go w formie „harmonijki", której sekcje użytkownik strony może rozwijać lub zwijać, w formie grupy ruchomych slajdów (takich jak tworzone przez wtyczkę Wow Slider przedstawioną na rysunku 7.6) bądź też pokaże go jako etykietki ekranowe.

6. **Wywołaj funkcję wtyczki.**

   Także w tym przypadku wszystko zależy od konkretnej wtyczki. Jednak wiele z nich stosuje tę samą konwencję: najpierw należy użyć jQuery do wybrania elementu, a następnie wywołać funkcję wtyczki. Do wybrania elementu zazwyczaj używa się jego identyfikatora lub klasy dodanej do kodu HTML. W przypadku komponentu akordeonu jQuery UI operacja ta sprowadza się do pobrania znacznika zawierającego pozostałe elementy kontenera i wywołania funkcji `accordion()`:

   ```
   $('#accordion').accordion();
   ```

   Owszem, całkiem często wszystko sprowadza się do jednego wiersza kodu. Większość wtyczek zapewnia także możliwość modyfikowania sposobu działania poprzez przekazanie odpowiednich instrukcji, które zazwyczaj zapisywane

są w formie literału obiektowego. Komponent kalendarza biblioteki jQuery UI można modyfikować na wiele różnych sposobów i to wyłącznie za pomocą przekazania do wtyczki odpowiednich informacji. Aby na przykład wyświetlić w kalendarzu trzy miesiące, a poniżej nich przycisk do wyboru bieżącej daty, wystarczy użyć literału obiektowego o następującej postaci:

```
$('#date').datepicker({
  numberOfMonths: 3,
  showButtonPanel: true
});
```

Każda wtyczka jest inna, jednak sposób korzystania z większości jest zgodny z podanym powyżej ogólnym opisem. W następnym podrozdziale zdobędziesz nieco doświadczenia w stosowaniu pewnej wtyczki.

Responsywne menu nawigacyjne

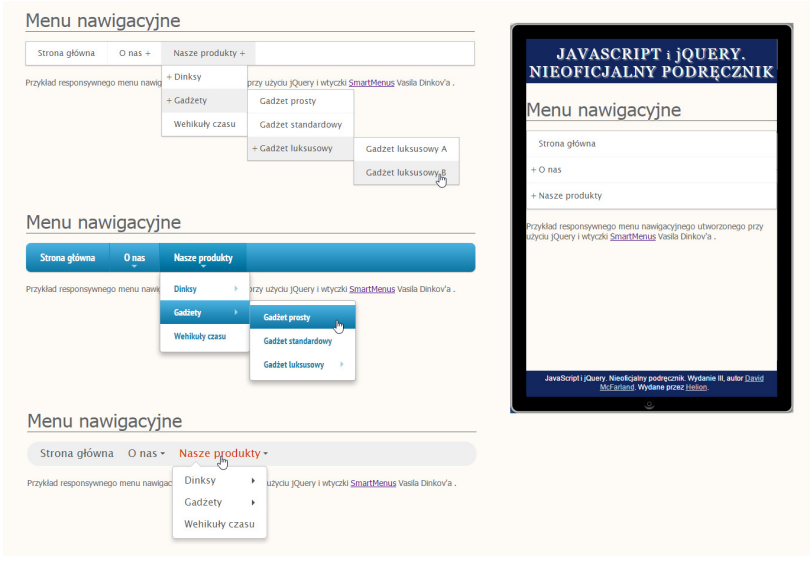
Wraz z powiększaniem się witryny coraz trudniej zapewnić dostęp do wszystkich jej sekcji bez przytłaczania użytkowników odnośnikami. Aby ułatwić nawigację, wielu projektantów stron WWW używa systemów menu rozwijanych, które ukrywają odnośniki, jeśli te nie są potrzebne (patrz rysunek 7.9). Choć można je utworzyć za pomocą języka CSS, to jednak takie rozwiązania nie są idealne. Po pierwsze, menu tego rodzaju są bardzo wrażliwe. Jeśli kursor choć na ułamek sekundy znajdzie się poza listą opcji, menu zniknie. Po drugie, kod CSS takich rozwiązań jest bardzo złożony, więc każdy, kto nie jest mistrzem CSS, będzie preferował rozwiązanie korzystające ze skryptów JavaScript.

Na szczęście niewielka ilość kodu JavaScript pozwala utworzyć animowane menu, które działa płynnie we wszystkich przeglądarkach. W tym podrozdziale została przedstawiona wtyczka jQuery (jQuery SmartMenus), która upraszcza proces tworzenia rozwijanych menu. Dodatkowo menu tworzone przez tę wtyczkę jest *responsywne* — czyli automatycznie dostosowuje się do szerokości okna przeglądarki, a w przypadku oglądania na telefonach (patrz rysunek 7.9 po prawej stronie) jest zmieniane na listę.

Takie menu są oparte na kodzie HTML i CSS w dużo większym stopniu niż techniki języka JavaScript opisane do tej pory. Kod HTML posłuży do utworzenia zagnieżdżonej grupy odnośników, a CSS — do upodobnienia odsyłaczy do paska nawigacyjnego oraz rozmieszczania i ukrywania menu podrzędnych. Następnie trzeba dodać kod JavaScript obsługujący animowane wyświetlanie menu po umieszczeniu wskaźnika myszy nad przyciskami paska nawigacji.

Kod HTML

Kod HTML menu nawigacyjnego to prosta lista wypunktowana, utworzona za pomocą znacznika `<ul>`. Każdy znacznik `<li>` najwyższego poziomu reprezentuje jeden z głównych przycisków menu. Aby utworzyć menu podrzędne, należy dodać znacznik `<ul>` do znacznika `<li>`, do którego ma należeć dane menu.



Rysunek 7.9. *Poruszanie się po witrynie o wielu stronach i sekcjach bywa trudne. Pasek nawigacyjny z menu rozwijanymi to elegancki sposób na uproszczenie prezentacji odnośników do różnych stron witryny (z lewej). Wtyczka jQuery o nazwie SmartMenus (http://www.smartmenus.org) ułatwia dodawanie wielu opcji nawigacyjnych i udostępnia kilka gotowych tematów graficznych: prosty (z lewej, u góry), niebieski (z lewej, po środku) czy też przejrzysty (z lewej u dołu). Podczas wyświetlania na telefonie menu jest przekształcane do postaci listy odnośników*

Kod HTML menu widocznego na rysunku 7.9 wygląda następująco:

```
<ul id="navigation" >
  <li><a href="#">Strona główna</a></li>
  <li><a href="#">O nas</a>
    <ul>
      <li><a href="#">Nasza historia</a></li>
      <li><a href="#">Dojazd</a></li>
      <li><a href="#">Godziny pracy</a></li>
    </ul>
  </li>
  <li><a href="#">Nasze produkty</a>
    <ul>
      <li><a href="#">Gizmo</a>
        <ul>
          <li><a href="#">Gizmo - wersja podstawowa</a></li>
          <li><a href="#">Gizmo - wersja standard</a></li>
          <li><a href="#">Gizmo - wersja Deluxe</a></li>
        </ul>
      </li>
      <li><a href="#">Gadżety</a>
        <ul>
          <li><a href="#">Gadżety drewniane</a></li>
          <li><a href="#">Gadżety szklane</a></li>
          <li><a href="#">Gadżety zaawansowane</a>
            <ul>
              <li><a href="#">Gadżety elektroniczne</a></li>
              <li><a href="#">Gadżety mechaniczne</a></li>
            </ul>
          </li>
        </ul>
```

```
        </li>
        <li><a href="#">Maszyna czasu</a></li>
      </ul>
    </li>
  </ul>
```

Uwaga: By uprościć prezentowany przykład, zastąpiono w nim faktyczne adresy URL, podawane w atrybutach `href` odnośników, znakami #, na przykład `<a href="#">`. W rzeczywistym menu nawigacyjnym odnośniki te wskazywałyby faktyczne strony WWW, na przykład `<a href="produkty/gadzety/podstawowe.html">`.

Trzy najważniejsze przyciski nawigacyjne to *Strona główna*, *O nas* oraz *Nasze produkty*. Przycisk *O nas* udostępnia kolejne menu, reprezentowane przez zagnieżdżoną listę zawierającą opcje *Nasza historia*, *Dojazd* oraz *Godziny pracy*. Także przycisk *Nasze produkty* udostępnia menu zawierające opcje *Gizmo*, *Gadżety* oraz *Maszyna czasu*. Zarówno w opcji *Gizmo*, jak i w opcji *Gadżety* dodane zostały kolejne poziomy menu (reprezentowane przez następne dwie listy zagnieżdżone); co więcej, także w opcji *Gadżety zaawansowane*, dostępnej w menu *Gadżety*, istnieją kolejne opcje (reprezentowane przez jeszcze jedną listę zagnieżdżoną). Lista zagnieżdżona to kolejna lista z większym wcięciem. Ten kod HTML ma postać:

- Strona główna
- O nas
  - Nasza historia
  - Dojazd
  - Godziny pracy
- Nasze produkty
  - Gizmo
    - Gizmo — wersja podstawowa
    - Gizmo — wersja standard
    - Gizmo — wersja Deluxe
  - Gadżety
    - Gadżety drewniane
    - Gadżety szklane
    - Gadżety zaawansowane
      - Gadżety elektroniczne
      - Gadżety mechaniczne
  - Maszyna czasu

Jak zwykle, lista zagnieżdżona znajduje się w znaczniku `<li>` nadrzędnego elementu. Przykładowo znacznik `<ul>` zawierający opcje *Gizmo*, *Gadżety* oraz *Maszyna czasu* znajduje się w znaczniku `<li>` elementu *Nasze produkty* (jeśli chcesz przypomnieć sobie informacje o listach HTML, odwiedź stronę *http://www.htmldog.com/guides/html/beginner/lists/*).

Wskazówka: Odnośniki najwyższego poziomu (czyli *Strona główna*, *O nas* i *Nasze produkty*) zawsze muszą prowadzić do stron z odnośnikami do stron podrzędnych danej sekcji (na przykład *Nasza historia* i *Dojazd* dla przycisku *O nas*). Zapewnia to dostęp do odnośników niższego poziomu także w przeglądarkach z wyłączoną obsługą języka JavaScript.

Kod CSS

Wtyczka jQuery SmartMenus napisana przez Vasila Dinkova (*http://www.smartmenus.org/*) wykonuje przeważającą większość czynności związanych z takim rozmieszczeniem elementów list, byśmy nie musieli zawracać sobie głowy tworzeniem arkusza CSS, który wyświetlałby przyciski obok siebie oraz generował rozwijane menu z opcjami podrzędnymi. Główny plik CSS ma nazwę *sm-core.css* i odpowiada za określenie podstawowego położenia przycisków nawigacyjnych.

Można także utworzyć własny arkusz stylów, by dostosować wygląd przycisków, bądź skorzystać z jednego z dostępnych, predefiniowanych tematów graficznych. Każdy taki temat jest dostarczany w formie odrębnego pliku CSS: *sm-simple.css*, *sm-clean.css* i *sm-blue.css*. Sposób użycia tych tematów oraz dostosowywania wyglądu menu do własnych potrzeb poznasz, czytając przykład, który zaczyna się na stronie 277.

Kod JavaScript

Sposób wykorzystania kodu JavaScript do wyświetlania menu jest prosty. Kiedy użytkownik wskaże myszą element listy, a ten zawiera listę zagnieżdżoną (czyli menu podrzędne), kod JavaScript wyświetli tę listę. Kiedy wskaźnik myszy zostanie usunięty z obszaru elementu, umieszczona w tym elemencie lista jest ukrywana.

Jest jednak kilka drobiazgów, które komplikują rozwiązanie. Na przykład menu rozwijane, które błyskawicznie znikają w momencie przeniesienia wskaźnika myszy poza opcje, wymagają precyzji w korzystaniu z myszy. Łatwo przesunąć jej wskaźnik w inne miejsce przy korzystaniu z takiego menu. Jeśli opcje nagle znikną, użytkownik musi ponownie najechać wskaźnikiem na przycisk, aby otworzyć listę odnośników. Jeśli menu rozwijane ma kilka poziomów, bardzo łatwo na jednym z nich wyjechać wskaźnikiem poza ich obszar i utracić dostęp do opcji.

Aby rozwiązać ten problem, zwykle w skryptach do obsługi menu nawigacyjnego używa się mechanizmu zegara, który opóźnia ukrywanie menu rozwijanych. To rozwiązanie nie wymaga dużej precyzji w ruchach myszą i sprawia, że menu rozwijane są mniej wrażliwe.

Przykład

Znasz już podstawy tworzenia menu nawigacyjnych. W tym praktycznym przykładzie użyjesz kodów CSS i JavaScript do przekształcenia prostej listy HTML z opcjami ze strony 272 na pasek nawigacyjny.

Uwaga: Informacje o pobieraniu przykładowych plików znajdziesz na stronie 46.

1. **Otwórz w edytorze tekstu plik *menu.html* z katalogu *R07*.**

   Plik zawiera wypunktowaną listę odnośników, którą przekształcisz na pasek nawigacyjny. Pierwszym krokiem do utworzenia naszego sprytnego menu jest dołączenie do strony pliku CSS, który w tym przypadku będzie już gotowy.

2. **Kliknij pusty wiersz pod znacznikiem `<link href="../_css/site.css" rel="stylesheet">` i dodaj następujący kod:**

   ```
   <link href="smartmenus/sm-core-css.css" rel="stylesheet" >
   <link href="smartmenus/sm-simple.css" rel="stylesheet" >
   ```

   Pierwszy znacznik `<link>` dołącza do strony podstawowy arkusz stylów — jest on wymagany, gdyż określa podstawowe sposoby formatowania menu, stosowane niezależnie od wybranego tematu lub ustawień związanych z czcionkami. Zawiera style dla „prostego" tematu graficznego przedstawionego na rysunku 7.9.

   Teraz musisz wprowadzić niewielkie zmiany w kodzie HTML.

3. **Odszukaj znacznik `<ul>` określający początek paska nawigacyjnego (znajduje się on bezpośrednio poniżej znacznika `<h1>`) i dodaj do niego atrybut `class` z dwiema wartościami, tak by wyglądał jak przedstawiony poniżej:**

   ```
   <ul class="sm sm-simple">
   ```

   Klasa o nazwie `sm` jest wymagana przez wszystkie menu tworzone przy użyciu wtyczki SmartMenus. Wtyczka potrzebuje jej, by wykonać swoje magiczne sztuczki i zastosować w menu podstawowe style. Druga klasa — `sm-simple` — określa, który z dostępnych tematów graficznych ma być używany. Jeśli w kroku 2. dołączyłeś jeden z dodatkowych arkuszy stylów — *sm-clean.css* lub *sm-blue.css* — możesz tu podać odpowiednią nazwą klasy. Innymi słowy, klasę `sm-simple` będziesz mógł zastąpić klasami `sm-clear` lub `sm-blue`.

   Teraz nadszedł czas, by zająć się kodem JavaScript.

4. **Kliknij pusty wiersz poniżej kodu `<script src="../_js/jquery.min.js"></script>` i wpisz:**

   ```
   <script src="smartmenus/jquery.smartmenus.min.js"></script>
   ```

   Powyższy znacznik spowoduje wczytanie wtyczki SmartMenus. Pamiętaj, że wtyczka ta wymaga biblioteki jQuery, dlatego musisz ją dołączyć do strony poniżej znacznika `<script>` dołączającego samą bibliotekę. Zwróć też uwagę, że plik JavaScript wtyczki oraz dwa pliki CSS są umieszczone w katalogu *smartmenus*. Zgodnie z informacjami podanymi na stronie 268, przechowywanie wszystkich plików związanych z konkretną wtyczką w odrębnym katalogu, którego nazwa odpowiada nazwie wtyczki, jest bardzo dobrym pomysłem. Dzięki temu będziesz mógł łatwo odnaleźć wszystkie pliki wtyczki, kiedy będziesz chciał ją na przykład zaktualizować (gdy pojawi się jej nowa wersja) lub usunąć (kiedy wtyczka Cię znudzi lub przestanie działać).

   Teraz będziesz musiał zająć się napisaniem kodu. Przykładowy plik zawiera już parę znaczników `<script>` wraz z umieszczonym wewnątrz nich wywołaniem funkcji `$(document).ready()`.

5. **Kliknij pusty wiersz wewnątrz funkcji `$(document).ready()` i wpisz wiersz kodu zaznaczony pogrubioną czcionką:**

```
$(document).ready(function() {
  $('.sm').smartmenus();
}); // Koniec funkcji ready.
```

W celu uaktywnienia menu najpierw musisz użyć jQuery, aby pobrać znacznik `<ul>` zawierający główny pasek nawigacyjny — w tym przypadku znacznik ten należy do klasy `sm`, zatem można go pobrać przy użyciu wywołania `$('.sm')`. Wywołanie `.smartmenus()` przygotowuje menu za pomocą wtyczki SmartMenus. Tak, to naprawdę jest takie proste!

6. **Zapisz stronę i wyświetl ją w przeglądarce. Spróbuj umieszczać wskaźnik myszy nad różnymi opcjami i zmieniać wielkość okna przeglądarki, tak by było jak najwęższe.**

Możesz zauważyć, że teraz wskazanie jednego przycisku w głównym menu powoduje wyświetlenie menu rozwijanego w dół, a wskazanie jednej z opcji dostępnych w tym menu powoduje wyświetlenie kolejnego poziomu menu umieszczonego z boku. Świetna sprawa.

Jednak możesz zrobić jeszcze więcej. Przekazując do wtyczki literał obiektowy (patrz strona 165), możesz modyfikować sposób jej działania. Aktualnie menu podrzędne stopniowo stają się widoczne, kiedy wskażesz wybraną opcję myszą, a kiedy usuniesz wskaźnik — stopniowo zanikają. Jednak dodając odpowiedni kod, możesz sprawić, że menu podrzędne będą się wysuwały. Kod, który trzeba w tym celu napisać, jest dosyć złożony, zatem dodasz go poniżej krok po kroku.

7. **Zmodyfikuj dodany wcześniej kod, dopisując do niego parę nawiasów klamrowych (wyróżnionych poniżej pogrubioną czcionką):**

```
$(document).ready(function() {
  $('.sm').smartmenus({ });
}); // Koniec funkcji ready.
```

Dodana para nawiasów klamrowych powoduje przekazanie do funkcji literału obiektowego. Jak się dowiedziałeś na stronie 165, literały obiektowe są sposobem przekazywania do funkcji par nazwa – wartość. Przykładowo poniżej przedstawiony został prosty literał obiektowy zawierający imię autora tej książki:

```
{ name : 'Dave' }
```

W tym przykładzie `name` jest nazwą (nazywaną także czasami kluczem), natomiast `'Dave'` jest wartością. Teraz nadszedł czas, by zrobić trochę miejsca na wartości, które niebawem dodasz.

8. **Wewnątrz pary nawiasów klamrowych definiujących literał obiektowy wpisz nowy wiersz, a następnie dodaj komentarz na końcu następnego wiersza:**

```
$(document).ready(function() {
  $('.sm').smartmenus({

  }); // Koniec funkcji smartmenus.
}); // Koniec funkcji ready.
```

Komentarz wskaże miejsce, w którym kończy się funkcja `smartmenus()`. W następnym kroku dodasz pierwszą parę nazwa – wartość.

9. **Dodaj kod zapisany na poniższym przykładzie w wierszach od 3. do 5.:**

```
1  $(document).ready(function() {
2    $('.sm').smartmenus({
3      showFunction: function($ul, complete) {
4        $ul.slideDown(250, complete);
5      }
6    }); // Koniec funkcji smartmenus.
7  }); // Koniec funkcji ready.
```

W tym przypadku nazwą (albo *kluczem*) jest `showFunction`. Wtyczka SmartMenus udostępnia wiele takich ustawień o predefiniowanych wartościach, a `showFunction` jest jedną z nich. Jeśli właściwość o takiej nazwie zostanie przekazana do wtyczki, funkcja stanowiąca jej wartość zostanie zastosowana do wyświetlania menu podrzędnych. W wyróżnionym fragmencie kodu dzieje się całkiem sporo, powinieneś jednak zauważyć, że w wierszu 4. wywoływana jest funkcja `slideDown()`. (Informacje na temat efektów animacji tworzonych przez jQuery można znaleźć na stronie 216). To wywołanie możesz jednak zastąpić dowolnym innym efektem animacji udostępnianym przez jQuery i opisanym na stronie 211 — takim jak `.hide()` lub `.fadeIn()` — możesz także opracować swoją własną animację, używając funkcji `.animate()` opisanej na stronie 220.

Liczba podana w wywołaniu funkcji `slideDown()` określa czas trwania animacji — w tym przypadku będzie to 250 milisekund. Możesz ją zmniejszyć, by animacja była odtwarzana szybciej, lub zwiększyć — by trwała dłużej.

Teraz zadbasz o to, by menu podrzędne były wysuwane ze strony, gdy użytkownik usunie wskaźnik myszy z elementu, którego wskazanie je wyświetliło.

10. **Zmodyfikuj kod, by wyglądał tak, jak na poniższym przykładzie (zmiany zostały wyróżnione pogrubioną czcionką):**

```
1  $(document).ready(function() {
2    $('.sm').smartmenus({
3      showFunction: function($ul, complete) {
4        $ul.slideDown(250, complete);
5      },
6      hideFunction: function($ul, complete) {
7        $ul.slideUp(250, complete);
8      }
9    }); // Koniec funkcji smartmenus.
10 }); // Koniec funkcji ready.
```

Nie zapomnij o znaku przecinka za zamykającym nawiasem klamrowym w wierszu 5. Służy on do oddzielenia par literału obiektowego. W tym kroku dodajesz do literału drugą opcję wtyczki SmartMenus: `hideFunction`.

Uwaga: Wtyczka SmartMenus udostępnia wiele ustawień kontrolujących sposób jej działania. Więcej informacji na ich temat można znaleźć na stronie wtyczki: *http://www smartmenus.org/docs/*.

11. **Zapisz stronę i wyświetl ją w przeglądarce.**

Teraz powinieneś już dysponować w pełni funkcjonalnym paskiem nawigacyjnym. Sprawdź style zapisane w pliku CSS, by przekonać się, w jaki sposób są formatowane przyciski; spróbuj zmieniać różne właściwości CSS, żeby dostosować wygląd menu. Pełną wersję tego przykładu możesz znaleźć w pliku *complete_menu.html* w katalogu R07.

Aby przekonać się, jak wyglądają inne tematy graficzne wtyczki SmartMenus, zmień odwołanie do pliku *sm-simple.css*, a następnie użyj odpowiedniej klasy w znaczniku `<ul>` zamiast klasy `sm-simple`. Aby na przykład użyć niebieskiego tematu, zmień znacznik `<link>` na następujący:

```
<link href="smartmenus/sm-blue.css" rel="stylesheet">
```

Zmodyfikuj też odpowiednio znacznik `<ul>`:

```
<ul class="sm sm-blue">
```

Pełną wersję tego przykładu znajdziesz w kodach dołączonych do książki, w pliku *complete_menu.html* umieszczonym w katalogu *R07*. Oprócz niego znajdziesz tam także analogiczne przykłady używające dwóch pozostałych tematów graficznych wtyczki SmartMenus, są to odpowiednio pliki: *complete_menu_blue.html* oraz *complete_menu_clean.html*.

Dostosowywanie wyglądu wtyczki SmartMenus

Najprostszym sposobem modyfikowania wyglądu menu nawigacyjnych tworzonych przy użyciu wtyczki SmartMenus jest wprowadzanie zmian w pliku CSS. Arkusz stylów dostarczany wraz z wtyczką zawiera wiele bardzo przydatnych komentarzy, które doskonale wyjaśniają przeznaczenie poszczególnych stylów. Przykładowo w pliku *sm-simple.css* umieszczony jest selektor o postaci `sm-simple, .sm-simple ul`. Ta reguła określa wygląd elementu zawierającego cały pasek nawigacyjny (czyli zewnętrznego znacznika `<ul>`) oraz każdego z menu podrzędnych (czyli zagnieżdżonych znaczników `<ul>`).

Można także tworzyć swoje własne style. Poniżej opisano, jak to zrobić.

1. **Zrób kopię arkusza stylów, który najbardziej Ci się podoba, nadając mu przy tym inną nazwę.**

   Przykładowo skopiuj plik *sm-simple.css* i nadaj mu nazwę *sm-mymenu.css*.

2. **Zmień prefiks arkusza stylów (taki jak `.sm-simple`) na własny.**

   Zmień na przykład `.sm-simple` na `.sm-mymenu` (bądź też na dowolną inną nazwę). Jeśli skorzystasz z edytora dysponującego mechanizmem wyszukiwania i zamiany, operacja ta będzie szybka i prosta. Pamiętaj jednak, że ta zmiana jest opcjonalna; równie dobrze możesz pozostawić starą nazwę (taką jak `.sm-simple`) już używaną w pliku.

3. **Wprowadź zmiany w regułach stylów.**

   W stylach możesz wprowadzić całkowicie dowolne zmiany. Styl `.sm-simple` określa postać każdego z przycisków, natomiast styl `.sm-simple a span.sub-arrow` ustala wygląd strzałki, wyświetlanej przy opcji zawierającej menu podrzędne.

4. **Dołącz swój nowy arkusz do strony.**

   Ta czynność wymaga zmiany adresu w odwołaniu do arkusza stylów:

   ```
   <link rel="stylesheet" href="smartmenus/sm-mymenu.css">
   ```

5. **Jeśli zmieniłeś prefiks w arkuszu stylów — na przykład, zmieniłeś `.sm-simple` na `.sm-mymenu` — podaj tę nazwę w atrybucie `class` znacznika `<ul>`, na przykład:**

```
<ul class="sm sm-mymenu">
```

Wtyczka SmartMenus jest bardzo łatwa w użyciu i sprawia, że dodawanie do stron responsywnych menu będzie Ci zajmować jedynie kilka chwil.

UWAGA NA WTYCZKI!

Inne wtyczki jQuery usprawniające nawigację

Wtyczka jQuery o nazwie SmartMenus jest prosta i efektywna; istnieje też bardzo wiele innych wtyczek tej biblioteki, które umożliwiają tworzenie znacznie bardziej zaawansowanych mechanizmów nawigacyjnych.

- Wtyczka jPanel (*http://jpanelmenu.com/*) tworzy menu przypominające boczne panele — takie które wsuwa się z boku na stronę po kliknięciu przycisku. Taki sposób nawigacji jest wykorzystywany w serwisie Facebook, na stronach Google oraz w wielu aplikacjach przeznaczonych na smartfony.
- Wtyczka Multi-level Push Menu (*http://multi-level-push-menu.make.rs/*) jest kolejnym systemem nawigacyjnym prezentowanym z boku strony. Zapewnia ona możliwość tworzenia wielu poziomów nawigacji, reprezentowanych przez niewielkie karty, które użytkownik może otwierać i zamykać. Jest ona doskonałym sposobem organizowania i zapewniania dostępu do bardzo dużej kolekcji odnośników.

Jeśli żadna z tych wtyczek nie przypadnie Ci do gustu, możesz sprawdzić listę 15 innych responsywnych wtyczek nawigacyjnych dostępnych na stronie *http://speckyboy.com/2013/08/01/15-responsive-navigation-jquery-plugins/*.

8

ROZDZIAŁ

Wzbogacanie formularzy

Od czasu powstania sieci WWW formularze umożliwiają pobieranie informacji od użytkowników witryn. Tymi danymi mogą być adresy e-mail, pod które należy przesyłać biuletyny, informacje związane z wysyłką kupionych towarów lub opinie internautów na temat witryny. Formularze wymagają od użytkowników *myślenia* — czytania etykiet, wpisywania danych, wybierania opcji i tak dalej. Ponieważ funkcjonowanie niektórych witryn w pełni opiera się na formularzach (Amazon nie utrzymałby się długo na rynku, gdyby użytkownicy nie mogli zamawiać książek za pomocą tej techniki), projektanci stron WWW muszą wiedzieć, jak ułatwić korzystanie z tego mechanizmu. Na szczęście możliwość zwiększania interaktywności elementów za pomocą języka JavaScript pomaga tworzyć formularze łatwe w użyciu i pobierające od internautów bardziej precyzyjne dane.

Wprowadzenie do formularzy

Język HTML udostępnia różne znaczniki do tworzenia formularzy podobnych do tego z rysunku 8.1. Najważniejszy jest znacznik `<form>`, który wyznacza początek (otwierający znacznik `<form>`) i koniec formularza (zamykający znacznik `</form>`). W tym elemencie należy określić metodę używaną do wysyłania danych (`post` lub `get`) i miejsce, gdzie strona ma przesyłać dane formularza.

Do tworzenia kontrolek formularza — przycisków, pól tekstowych i list rozwijanych — służą znaczniki `<input>`, `<textarea>` oraz `<select>`. Większość elementów formularza to znaczniki `<input>`. Służą one do tworzenia między innymi pól tekstowych, pól na hasło, przycisków opcji, pól wyboru i przycisków wysyłania formularza. Elementy te różnią się wartością atrybutu `type`. Aby na przykład utworzyć pole tekstowe, należy użyć znacznika `<input>` i przypisać do atrybutu `type` wartość `text`:

```
<input name="user" type="text">
```

Rysunek 8.1. *Prosty formularz może zawierać wiele różnych kontrolek, w tym pola tekstowe, przyciski opcji, pola wyboru, listy opcji i tak dalej. Informacje na temat wszystkich dostępnych pól formularzy HTML i artykułów o tym, jak z nich korzystać, znajdziesz na stronie https://developer.mozilla.org/en-US/docs/Web/Guide/HTML/Forms*

Oto kod HTML formularza widocznego na rysunku 8.1. Znacznik `<form>` i elementy **formularza** wyróżniono pogrubieniem:

```
<form action="process.php" method="post" name="signup" id="signup">
  <div>
    <label for="username" class="label">Imię i nazwisko</label>
    <input name="username" type="text" id="username" size="36">
  </div>
  <div><span class="label">Hobby</span>
    <input type="checkbox" name="hobby" id="helisking"
    ↪value="heliskiing">
    <label for="heliskiing">Heliskiing</label>
    <input type="checkbox" name="hobby" id="pickle" value="pickle">
    <label for="pickle">Jedzenie korniszonów</label>
    <input type="checkbox" name="hobby" id="walnut" value="walnut">
    <label for="walnut">Produkcja masła orzechowego</label>
  </div>
  <div>
    <label for="planet" class="label">Planeta urodzenia</label>
    <select name="planet" id="planet">
      <option>Ziemia</option>
      <option>Mars</option>
      <option>Alpha Centauri</option>
      <option>Nigdy o niej nie słyszałeś</option>
    </select>
  </div>
  <div class="labelBlock">Czy chciałbyś otrzymywać od nas irytujące listy
  ↪elektroniczne?
  </div>
  <div class="indent">
    <input type="radio" name="spam" id="yes" value="yes"
    ↪checked="checked">
    <label for="yes">Tak</label>
    <input type="radio" name="spam" id="definitely" value="definitely">
```

```
    <label for="definitely">Zdecydowanie</label>
    <input type="radio" name="spam" id="choice" value="choice">
    <label for="choice">A czy mam jakiś wybór?</label>
  </div>
  <div>
    <input type="submit" name="submit" id="submit" value="Wyślij">
  </div>
</form>
```

Uwaga: `<label>` to następny znacznik często używany w formularzach, który jednak nie tworzy kontrolek (takich jak przyciski), ale umożliwia dodanie etykiety z opisem przeznaczenia danego elementu.

Pobieranie elementów formularzy

Wielokrotnie widziałeś już, że użycie elementów strony wymaga uprzedniego ich pobrania. Aby ustalić, jaką wartość zawiera pole formularza, trzeba je najpierw znaleźć. Także jeśli zechcesz ukryć lub wyświetlić elementy *formularza*, musisz je zidentyfikować przy użyciu kodu JavaScript.

Jak już zapewne wiesz, jQuery pozwala pobierać elementy strony przy użyciu niemal wszystkich możliwych selektorów CSS. Najłatwiejszy sposób na pobranie jednego elementu *formularza* polega na przypisaniu znacznikowi identyfikatora:

```
<input name="user" type="text" id="user">
```

W takim przypadku do pobrania elementu można użyć poniższego wywołania jQuery:

```
var userField = $('#user');
```

Po pobraniu pola należy wykonać na nim określone operacje. Załóżmy, że chcesz ustalić wartość pola, aby sprawdzić, jaki tekst wpisał użytkownik. Jeśli identyfikator pola to `user`, za pomocą jQuery można pobrać wartość elementu w następujący sposób:

```
var fieldValue = $('#user').val();
```

Uwaga: Opis funkcji `val()` biblioteki jQuery znajdziesz na stronie 283.

Co jednak zrobić, aby pobrać z *formularza* wszystkie elementy określonego typu? Przyjmijmy, że chcesz dodać obsługę zdarzenia `click` do każdego przycisku opcji na stronie.

Jednak znacznik `<input>` służy do tworzenia przycisków opcji, pól tekstowych, pól z hasłem, pól wyboru, przycisków do przesyłania danych, przycisków do czyszczenia formularzy oraz pól ukrytych. Dlatego też nie możemy wyszukać wszystkich znaczników `<input>` — musimy mieć możliwość odszukania znaczników `<input>` określonego typu.

Można w tym celu użyć selektora atrybutu CSS w poniższy sposób:

```
$('input[type="radio"]');
```

Na szczęście jQuery udostępnia jeszcze prostszy sposób pobierania różnych rodzajów pól formularzy (patrz tabela 8.1). Przy użyciu jednego z selektorów formularzy udostępnianych przez tę bibliotekę można łatwo zidentyfikować wszystkie pola konkretnego typu i manipulować nimi. Załóżmy, że kiedy użytkownik przesyła formularz, strona ma sprawdzić, czy wszystkie pola tekstowe zawierają wartość. Musimy zatem pobrać wszystkie pola tekstowe, a następnie upewnić się, że każde z nich zawiera jakąś wartość. Gdy korzystamy z jQuery, pierwszą część zadania możemy wykonać w następujący sposób:

```
$(':text')
```

Tabela 8.1. *JQuery udostępnia wiele selektorów, które ułatwiają manipulowanie różnymi polami formularzy*

Selektor	Przykład	Opis działania
`:input`	`$(':input')`	Pobiera pola wejściowe, pola tekstowe, znaczniki `<select>` i przyciski, czyli wszystkie elementy formularzy.
`:text`	`$(':text')`	Pobiera wszystkie pola tekstowe.
`:password`	`$(':password')`	Pobiera wszystkie pola z hasłem.
`:radio`	`$(':radio')`	Pobiera wszystkie przyciski opcji.
`:checkbox`	`$(':checkbox')`	Pobiera wszystkie pola wyboru.
`:submit`	`$(':submit')`	Pobiera wszystkie przyciski do przesyłania danych.
`:image`	`$(':image')`	Pobiera wszystkie przyciski z rysunkami.
`:reset`	`$(':reset')`	Pobiera wszystkie przyciski do czyszczenia formularza.
`:button`	`$(':button')`	Pobiera wszystkie pola typu `button`.
`:file`	`$(':file')`	Pobiera wszystkie pola do przesyłania plików.
`:hidden`	`$(':hidden')`	Pobiera wszystkie pola ukryte.

Następnie wystarczy przejść w pętli po pobranych elementach za pomocą funkcji `.each()` (patrz strona 168), aby upewnić się, że każde pole ma określoną wartość. (Dużo więcej informacji o walidacji pól formularzy znajdziesz na stronie 299).

Można połączyć selektory formularzy z innymi selektorami. Załóżmy, że na stronie znajdują się dwa formularze, a chcesz pobrać pola tekstowe z tylko jednego z nich. Jeśli ten formularz ma identyfikator `signup`, można znaleźć jego pola tekstowe tylko w poniższy sposób:

```
$('#signup :text')
```

Ponadto jQuery udostępnia bardzo przydatne filtry, które wyszukują pola formularza mające określony stan.

- Filtr `:checked` pobiera wszystkie włączone (zaznaczone) pola wyboru i przyciski opcji. Jeśli chcesz znaleźć takie kontrolki, użyj następującego kodu:

  ```
  $(':checked')
  ```

  Co jeszcze lepsze, przy użyciu tego filtra możesz znaleźć zaznaczony przycisk opcji z danej grupy. Przyjmijmy, że dysponujesz grupą przycisków opcji „Wybierz metodę przesyłki", zawierającą różne wartości (takie jak UPS, USPS czy

też FedEx) i chcesz poznać wartość przycisku zaznaczonego przez użytkownika. Grupa powiązanych ze sobą przycisków opcji ma atrybut `name` o tej samej wartości, na przykład `shipping`. W celu pobrania wartości zaznaczonego przycisku opcji możesz skorzystać z selektora atrybutów jQuery (patrz strona 152) oraz filtra `:checked`.

```
var checkedValue = $('input[name= shipping ]:checked').val();
```

Selektor `$('input[name= shipping ]')` pobiera wszystkie elementy `input` o nazwie `shipping`, jednak dodanie filtra `:checked` — `$('input[name= ➥ shipping ]:checked')` — sprawia, że zostanie zwrócony tylko ten z nich, który w danej chwili jest zaznaczony. Z kolei funkcja `val()` zwraca wartość przypisaną do danego pola, na przykład USPS.

- Filtr `:selected` pobiera wszystkie zaznaczone elementy `option` z listy rozwijanej, co umożliwia znalezienie opcji wybranych przez użytkownika w znaczniku `<select>`. Załóżmy, że znacznik `<select>` o identyfikatorze `state` zawiera listę 50 stanów USA. Aby znaleźć stan zaznaczony przez internautę, można użyć następującego kodu:

  ```
  var selectedState=$('#state :selected').val();
  ```

  Zauważ, że — inaczej niż w filtrze `:checked` — między identyfikatorem a filtrem znajduje się odstęp (`'#state :selected'`). Dzieje się tak, ponieważ ten filtr pobiera znaczniki `<option>`, a nie `<select>`. Po polsku ostatnia instrukcja jQuery oznacza: „Znajdź wszystkie zaznaczone opcje ze znacznika `<select>` o identyfikatorze `state`". Odstęp sprawia, że wyrażenie działa jak selektor potomków CSS — najpierw wyszukuje znacznik o określonym identyfikatorze, a następnie pobiera z niego zaznaczone elementy.

Uwaga: Listy rozwijane `<select>` mogą umożliwiać zaznaczanie wielu opcji. Oznacza to, że filtr `:selected` zwraca czasem więcej niż jeden element.

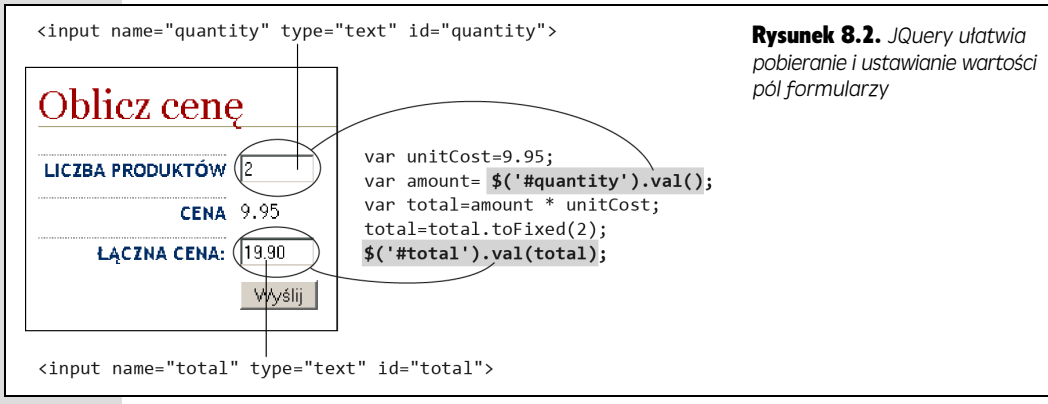
Pobieranie i ustawianie wartości elementów formularzy

Czasem skrypt musi sprawdzić wartość elementu *formularza*. Jest to potrzebne na przykład do określenia, czy użytkownik wpisał adres e-mail w polu tekstowym, lub do obliczenia ceny zakupionych produktów. W innych sytuacjach program ma *ustawić* wartość elementu *formularza*. Formularz zamówienia często służy do pobierania informacji związanych z płatnością i wysyłką. Przydatne jest wtedy pole wyboru z etykietą „Takie same jak przy płatności", które pozwala automatycznie dodać dane na temat wysyłki na podstawie zawartości pól z informacjami o płatności.

JQuery udostępnia prostą funkcję do wykonywania obu wspomnianych zadań. Jest to funkcja `val()`, która umożliwia zarówno ustawianie, jak i pobieranie wartości pól formularzy. Jeśli wywołasz ją bez argumentów, wczyta zawartość danego pola. Jeżeli podasz argument, funkcja użyje go do zmiany wartości pola. Załóżmy, że pole do pobierania adresu e-mail ma identyfikator `email`. Poniższy kod pobiera wartość tego pola:

```
var fieldValue = $('#email').val();
```

Aby zmienić wartość pola, wystarczy przekazać argument do funkcji `val()`. Przyjmijmy, że strona zawiera formularz do zamawiania towarów, a skrypt ma automatycznie obliczać cenę zakupów na podstawie liczby produktów podanej przez użytkownika (patrz rysunek 8.2). Należy *pobrać* wpisaną liczbę, pomnożyć ją przez cenę produktu, a następnie *ustawić* wartość pola z ceną.



Rysunek 8.2. *JQuery ułatwia pobieranie i ustawianie wartości pól formularzy*

Kod do pobierania liczby produktów i ustawiania łącznej ceny w formularzu z rysunku 8.2 wygląda następująco:

```
1  var unitCost=9.95;
2  var amount=$('#quantity').val(); // Pobieranie wartości
3  var total=amount * unitCost;
4  total=total.toFixed(2);
5  $('#total').val(total); // Ustawianie wartości.
```

Wiersz 1. kodu tworzy zmienną, która przechowuje cenę produktu, 2. wiersz tworzy nową zmienną i pobiera liczbę wprowadzoną przez użytkownika w polu o identyfikatorze `quantity`. Wiersz 3. oblicza łączną cenę zakupów, mnożąc liczbę produktów przez cenę jednostkową, a wiersz 4. formatuje wynik przez dodanie dwóch miejsc po kropce (opis działania metody `toFixed()` znajdziesz na stronie 590). Wiersz 5. przypisuje łączną cenę zakupów do pola o identyfikatorze `total`. Na stronie 287 dowiesz się, jak uruchomić ten kod za pomocą zdarzeń.

Sprawdzanie stanu przycisków opcji i pól wyboru

Choć funkcja `val()` pomaga pobrać wartość dowolnego elementu *formularza*, w niektórych polach skrypt powinien ją uwzględniać tylko wtedy, jeśli użytkownik zaznaczył dany element. Przyciski opcji i pola wyboru wymagają, aby internauta wybrał określoną wartość. Na stronie 282 zobaczyłeś, jak użyć filtra `:checked` do znalezienia zaznaczonych przycisków opcji i pól wyboru, jednak po ich pobraniu potrzebny jest sposób na określenie stanu danego elementu.

Atrybut `checked` języka HTML określa, czy dany element jest włączony. Jeśli chcesz, aby pole było domyślnie zaznaczane, użyj tego atrybutu w następujący sposób:

```
<input type="checkbox" name="news" id="news" checked="checked" />
```

A tak ten sam kod wygląda w języku HTML5:

```
<input type="checkbox" name="news" id="news" checked />
```

Choć w pierwszym przykładzie — z `checked= checked` — wygląda na to, że `checked` jest atrybutem HTML, jednak w rzeczywistości jest to właściwość DOM (patrz strona 145). Okazuje się, że jest to właściwość elementu pola wyboru, której wartość może zmieniać się dynamicznie, gdy użytkownik zaznaczy dane pole lub usunie z niego zaznaczenie. Takie dynamiczne właściwości ma także wiele innych pól formularzy; na przykład pola tekstowe udostępniają właściwość o nazwie `disabled`, która określa, czy w polu można coś wpisywać (`disabled` ma wartość `false`), czy nie (`disabled` ma wartość `true`).

Wartości właściwości DOM można odczytywać przy użyciu metody `prop()` jQuery w sposób przedstawiony na poniższym przykładzie:

```
if ($('#news').prop('checked')) {
  // Pole jest zaznaczone.
} else {
  // Pole nie jest zaznaczone.
}
```

Kod `$('#news').prop('checked')` zwróci wartość `true`, jeśli pole jest zaznaczone. W przeciwnym razie zwróci wartość `false`. Dlatego powyższa prosta instrukcja warunkowa umożliwia wykonanie jednego zbioru operacji, jeśli pole jest włączone, i innego fragmentu kodu, jeżeli użytkownik nie zaznaczył tego pola. Aby przypomnieć sobie informacje o instrukcjach warunkowych, zajrzyj na stronę 93.

Właściwość `checked` jest dostępna też w przyciskach opcji. Także w ich przypadku stan atrybutu `checked` można sprawdzić, używając funkcji `prop()`.

Zdarzenia związane z formularzami

W rozdziale 5. dowiedziałeś się, że zdarzenia umożliwiają tworzenie interaktywnych stron, reagujących na działania użytkowników. *Formularze* i ich elementy obsługują wiele różnych zdarzeń. Przy ich użyciu można sprawić, aby formularze inteligentnie reagowały na działania internautów.

Zdarzenie submit

Kiedy użytkownik prześle formularz przez wciśnięcie przycisku przesyłania lub klawisza *Enter* albo *Return* w czasie wpisywania danych w polu tekstowym, przeglądarka zgłosi zdarzenie `submit`. Można je wykorzystać do uruchamiania skryptów w momencie wysyłania formularza. Pozwala to przeprowadzić walidację pól w celu sprawdzenia, czy zawierają prawidłowe dane. Kiedy użytkownik spróbuje przesłać formularz, skrypt sprawdzi pola, a jeśli wykryje problem, zablokuje wysyłanie danych i poinformuje internautę o błędach. Jeżeli wpisane informacje będą prawidłowe, formularz zostanie przesłany w standardowy sposób.

Aby uruchomić funkcję w momencie zgłoszenia zdarzenia `submit` formularza, należy najpierw pobrać ten formularz, a następnie użyć funkcji `submit()` biblioteki jQuery. Załóżmy, że chcesz sprawdzić, czy pole z imieniem i nazwiskiem użytkownika, widoczne na rysunku 8.1, zawiera w momencie przesyłania formularza dane. W tym

celu należy dodać do formularza zdarzenie `submit` i sprawdzić wartość wspomnianego pola przed przesłaniem danych. Jeśli pole jest puste, trzeba poinformować o tym użytkownika i wstrzymać przesyłanie formularza. W przeciwnym razie należy wysłać dane.

W kodzie HTML formularza przedstawionym na stronie 280 możesz zobaczyć, że formularz ma identyfikator `signup`, a identyfikator pola `name` to `username`. Dlatego można użyć biblioteki jQuery do przeprowadzenia walidacji w następujący sposób:

```
1 $(document).ready(function() {
2   $('#signup').submit(function() {
3     if ($('#username').val() == '') {
4       alert('Podaj nazwę użytkownika.');
5       return false;
6     }
7   }); // Koniec funkcji submit
8 }); // Koniec funkcji ready
```

Wiersz 1. tworzy funkcję `$(document).ready()`, niezbędną, jeśli skrypt ma zostać uruchomiony dopiero po wczytaniu kodu HTML strony (patrz strona 190). Wiersz 2. dołącza funkcję do zdarzenia `submit` formularza. Wiersze od 3. do 6. to kod do obsługi walidacji. Wiersz 3. sprawdza, czy wartość pola to pusty łańcuch znaków (`''`). Jeśli tak jest, oznacza to, że użytkownik nie wprowadził danych. Jeśli pole nie zawiera tekstu, skrypt wyświetla okno dialogowe z informacją o popełnionym błędzie.

Wiersz 5. jest bardzo istotny, ponieważ wstrzymuje przesyłanie formularza. Jeśli pominiesz ten krok, strona prześle formularz nawet wtedy, jeśli nie zawiera on nazwy użytkownika. Wiersz 6. kończy instrukcję warunkową, a wiersz 7. zamyka funkcję `submit()`.

Uwaga: Możesz zatrzymać przesyłanie formularza także za pomocą funkcji `preventDefault()` obiektu zdarzenia (patrz strona 195).

```
$('form').submit(function(evt) {
  // Zapobiegamy przesłaniu formularza
  evt.preventDefault();
});
```

Zdarzenie `submit` można powiązać tylko z formularzami. Trzeba pobrać formularz i dołączyć do niego to zdarzenie. Aby wskazać formularz, można użyć identyfikatora podanego w znaczniku `<form>` w kodzie HTML lub — jeśli strona zawiera tylko jeden formularz — użyć prostego selektora elementu:

```
$('form').submit(function() {
  // Kod uruchamiany przy przesyłaniu formularza.
});
```

Zdarzenie focus

Kiedy użytkownik kliknie pole tekstowe lub przejdzie do niego za pomocą klawisza *Tab*, dane pole zostanie *aktywowane*. Przeglądarka zgłosi wtedy powiązane z tym procesem zdarzenie `focus`, aby poinformować, że kursor znajduje się w danym polu. Można założyć, że na tym elemencie skoncentrowana jest uwaga internauty.

Zdarzenie to jest używane stosunkowo rzadko, jednak niektórzy projektanci stron WWW stosują je do usuwania tekstu znajdującego się już w polu. Załóżmy, że formularz zawiera następujący kod HTML:

```
<input name="username" type="text" id="username"
➥value="Podaj nazwę użytkownika.">
```

Ten kod tworzy w formularzu pole tekstowe ze zdaniem „Podaj nazwę użytkownika.". Technika ta pozwala wyświetlić informacje pomocne przy wypełnianiu pola. Następnie zamiast zmuszać użytkownika wypełniającego formularz do samodzielnego wykasowania tekstu, można usunąć go w momencie aktywowania pola:

```
1 $('#username').focus(function() {
2   var field = $(this);
3   if (field.val()==field.attr('defaultValue')) {
4     field.val('');
5   }
6 });
```

Wiersz 1. pobiera pole (jego identyfikator to `username`) i przypisuje funkcję do zdarzenia `focus`. Wiersz 2. tworzy zmienną `field` i zapisuje w niej referencję do elementu pobranego za pomocą jQuery. Jak opisano na stronie 169, konstrukcja `$(this)` wskazuje na element aktualnie przetwarzany w funkcji jQuery. Tu jest to pole formularza.

Wiersz 4. usuwa zawartość pola przez przypisanie do niego pustego łańcucha znaków (dwóch apostrofów). Jednak nie należy usuwać tekstu przy każdym aktywowaniu pola. Załóżmy, że użytkownik otworzył formularz i kliknął pole. Przy pierwszym takim zdarzeniu skrypt powinien usunąć tekst „Podaj nazwę użytkownika". Jednak jeśli internauta wprowadził już dane, przeszedł poza pole, a następnie wrócił do niego, nie należy kasować wpisanej nazwy. Zapobiega temu instrukcja warunkowa z wiersza 3.

Pola tekstowe mają właściwość o nazwie `defaultValue`, która zawiera tekst widoczny w polu po wczytaniu strony. Nawet jeśli skrypt usunie ten tekst, przeglądarka zapamięta domyślną wartość pola. Instrukcja warunkowa sprawdza, czy bieżąca zawartość pola (`field.val()`) jest taka sama jak wartość początkowa (`field.prop('defaultValue')`). Jeśli tak jest, interpreter usuwa tekst z pola.

Oto opis całego procesu. Kiedy przeglądarka wczyta kod HTML przykładowej strony, pole tekstowe zawiera zdanie „Podaj nazwę użytkownika" (jest to wartość atrybutu `defaultValue` pola). Kiedy użytkownik po raz pierwszy aktywuje pole, instrukcja warunkowa sprawdza, czy jego bieżąca zawartość jest taka sama jak domyślna, czyli czy tekst „Podaj nazwę użytkownika" jest taki sam jak „Podaj nazwę użytkownika". Warunek ten jest spełniony, dlatego skrypt usuwa zawartość pola.

Załóżmy jednak, że użytkownik wpisał nazwę **`helloKitty`**, przeszedł do innego pola, a następnie zauważył, że wprowadził błędny tekst. Kiedy ponownie kliknie pole tekstowe, aby poprawić pomyłkę, przeglądarka po raz wtóry zgłosi zdarzenie `focus` i uruchomi powiązaną z nim funkcję. Tym razem program sprawdzi, czy tekst „helloKitty" jest taki sam jak „Podaj nazwę użytkownika". Jest to nieprawda, dlatego skrypt nie usunie zawartości pola i umożliwi poprawienie błędu.

Uwaga: Formularze HTML5 udostępniają także atrybut `placeholder`, pozwalający na określenie tymczasowego komunikatu, który będzie wyświetlany w polu tekstowym. Tekst ten zostanie usunięty, kiedy użytkownik zacznie coś wpisywać w polu:

```
<input name="username" type="text" id="username" placeholder="Podaj
➥nazwę użytkownika">
```

Rozwiązanie to jest znacznie łatwiejsze niż stosowanie przedstawionej wcześniej sztuczki wykorzystującej bibliotekę jQuery; jednak atrybut `placeholder` nie działa w przeglądarkach Internet Explorer 9 i wcześniejszych.

Zdarzenie blur

Po opuszczeniu pola w wyniku kliknięcia poza nim lub wciśnięcia klawisza *Tab* przeglądarka zgłasza zdarzenie `blur`. Jest ono powszechnie używane do obsługi pól tekstowych i obszarów tekstowych w celu uruchamiania walidacji po wyjściu poza te elementy. Załóżmy, że na stronie znajduje się długi formularz z licznymi pytaniami, a wiele z nich wymaga podania wartości określonego typu (na przykład adresu e-mail, liczby, kodu pocztowego i tak dalej). Jeśli użytkownik popełni kilka pomyłek przy wypełnianiu tych pól, a następnie wciśnie przycisk przesyłania danych, zobaczy długą listę błędów. Zamiast wyświetlać wszystkie te informacje jednocześnie, można sprawdzać poszczególne pola w trakcie ich wypełniania. W ten sposób jeśli internauta zrobi błąd, natychmiast się o tym dowie i będzie mógł go naprawić.

Poniższy kod HTML tworzy pole do pobierania liczby produktów zamawianych przez klienta:

```
<input name="quantity" type="text" id="quantity">
```

Warto się upewnić, że pole zawiera tylko liczby (1, 2, 3 i tak dalej), a nie tekst, na przykład „Jeden", „Dwa" albo „Trzy". Można to sprawdzić po opuszczeniu pola przez użytkownika:

```
1 $('#quantity').blur(function() {
2   var fieldValue=$(this).val();
3   if (isNaN(fieldValue)) {
4     alert('Musisz wpisać liczbę');
5   }
6 });
```

Wiersz 1. przypisuje funkcję do zdarzenia `blur`. Wiersz 2. pobiera wartość pola i zapisuje ją w zmiennej `fieldValue`. Wiersz 3. sprawdza przy użyciu metody `isNaN()` (patrz strona 589), czy wartość jest liczbą. Jeśli tak nie jest, skrypt uruchamia wiersz 4., który wyświetla okno dialogowe.

Gdybyśmy dysponowali formularzem zawierającym wiele pól, w których należy wpisać wartości liczbowe, to w każdym z nich moglibyśmy podać tę samą nazwę klasy — na przykład `class= numOnly` — a następnie sprawdzać je, używając poniższego kodu:

```
1  $('.numOnly').blur(function() {
2    var fieldValue=$(this).val();
3    if (isNaN(fieldValue)) {
4      alert('Musisz wpisać liczbę');
5    }
6  });
```

Takie rozwiązanie pozwala sprawdzić zawartość wszystkich pól tekstowych wymagających podania wartości liczbowej i to jedynie w pięciu wierszach kodu!

Zdarzenie click

Zdarzenie `click` jest uruchamiane w odpowiedzi na kliknięcie dowolnego elementu formularza. Jest ono szczególnie przydatne przy obsłudze przycisków opcji i pól wyboru, ponieważ można powiązać z nim funkcje, które modyfikują formularz na podstawie przycisków wybranych przez użytkownika. Załóżmy, że formularz zamówienia zawiera odrębne obszary na informacje związane z płatnością i wysyłką. Aby zaoszczędzić pracy klientom, którzy chcą użyć w obu obszarach tych samych danych, można udostępnić pole wyboru z tekstem „Takie same jak przy płatności". Kiedy użytkownik je zaznaczy, skrypt powinien ukryć obszar na informacje o wysyłce, co upraszcza formularz i poprawia jego czytelność. Przykład zastosowania tej techniki zobaczysz na stronie 298.

Podobnie jak przy innych zdarzeniach, do przypisywania funkcji do zdarzenia `click` pola formularza można użyć funkcji biblioteki jQuery. Tu jest to funkcja `click()`:

```
$(':radio').click(function() {
  // Funkcja uruchamiana dla kliknięty przycisków opcji.
});
```

Uwaga: Zdarzenie `click` także można powiązać z polami tekstowymi, jednak działa ono inaczej niż zdarzenie `focus`. To ostatnie jest zgłaszane przy kliknięciu pola tekstowego *lub* przejściu do niego za pomocą klawisza *Tab*, natomiast zdarzenie `click` zachodzi tylko przy kliknięciu.

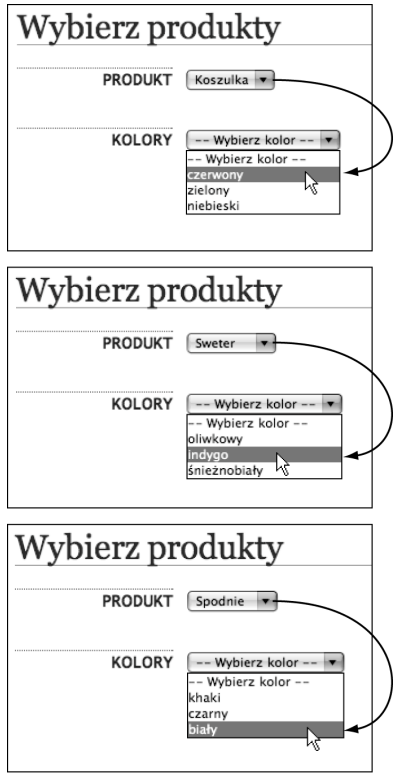
Zdarzenie change

Zdarzenie `change` jest związane z listami rozwijanymi formularza (takimi jak lista „Planeta urodzenia" na rysunku 8.3). Kiedy użytkownik zaznaczy opcję, przeglądarka zgłasza zdarzenie `change`. Można użyć go do uruchomienia walidacji. Wielu projektantów stron WWW często umieszcza jako pierwszą opcję instrukcję, na przykład „Wybierz państwo". Aby się upewnić, że użytkownik przypadkowo nie wybrał tej opcji, można sprawdzić zaznaczoną odpowiedź za każdym razem, kiedy dana osoba ją zmieni.

Można też tak zaprogramować formularz, aby zmieniał się w zależności od zaznaczonych opcji. Przykładowo wybór opcji na jednej liście rozwijanej może uruchamiać funkcję, która zmienia odpowiedzi dostępne na drugiej liście. Na rysunku 8.3 przedstawiono formularz z dwiema listami rozwijanymi. Zaznaczenie opcji w górnej zmienia listę kolorów dostępnych w dolnej.

Aby dodać zdarzenie `change` do listy rozwijanej, należy użyć funkcji `change()` biblioteki jQuery. Załóżmy, że lista o identyfikatorze `country` zawiera nazwy państw. Po każdej zmianie opcji skrypt ma sprawdzać, czy użytkownik nie zaznaczył tekstu instrukcji — „Wybierz państwo". Można to zrobić w następujący sposób:

```
  $('#country').change(function() {
if ($(this).val()=='Wybierz państwo') {
  alert('Wybierz nazwę państwa z listy rozwijanej.');
 }
}
```



Rysunek 8.3. *Zdarzenie change list rozwijanych formularzy pozwala wykonać ciekawe operacje po zaznaczeniu opcji przez użytkownika. Tu wybranie odpowiedzi z pierwszej listy dynamicznie zmienia opcje dostępne na drugiej liście. Kiedy zaznaczysz produkt na górnej liście, na dolnej pojawią się kolory, w których dostępny jest dany produkt*

Inteligentne formularze

Korzystanie z formularzy wymaga od internautów sporo wysiłku. Trzeba wypełnić pola tekstowe, zaznaczyć opcje, zaznaczyć pola wyboru i tak dalej. Jeśli chcesz, aby użytkownicy wypełniali formularze, należy je jak najbardziej uprościć. Na szczęście JavaScript pozwala znacznie ułatwić korzystanie z formularzy. Można między innymi ukryć pola formularza, jeśli nie są potrzebne, wyłączyć pola, których nie można użyć w danym kontekście, lub obliczyć łączną cenę na podstawie wybranych opcji. JavaScript oferuje niezliczone możliwości w zakresie zwiększania użyteczności formularzy.

Aktywowanie pierwszego pola formularza

Zwykle aby zacząć wypełnianie formularza, należy kliknąć pierwsze pole tekstowe i rozpocząć wpisywanie danych. Czy na stronie z formularzem logowania trzeba zmuszać użytkownika do przenoszenia kursora nad pole i klikania go? Czy nie lepiej od razu umieścić kursor we właściwym elemencie, gotowym do natychmiastowego pobrania danych uwierzytelniających? Przy użyciu języka JavaScript można to zrobić w bardzo łatwy sposób.

Umożliwia to słowo `focus`. Nie tylko oznacza ono zdarzenie, na które może reagować kod JavaScript, ale też jest poleceniem wydawanym w celu umieszczenia kursora w danym polu tekstowym. Wystarczy pobrać to pole, a następnie uruchomić funkcję `focus()` biblioteki jQuery. Załóżmy, że po wyświetleniu strony kursor ma się znajdować w polu `name` formularza widocznego na rysunku 8.1. W kodzie HTML tego formularza (patrz strona 280) pole to ma identyfikator `username`. Dlatego aby w kodzie JavaScript aktywować to pole, czyli umieścić w nim kursor, należy użyć poniższej instrukcji:

```
$(document).ready(function() {
  $('#username').focus();
});
```

W tym fragmencie pole tekstowe ma identyfikator `username`. Można jednak przygotować także uniwersalny skrypt, który zawsze aktywuje pierwsze pole tekstowe formularza bez konieczności podawania identyfikatora:

```
$(document).ready(function() {
  $(':text:first').focus();
});
```

Jak wiesz ze strony 282, jQuery udostępnia wygodne polecenie pobierające wszystkie pola tekstowe — `$(':text')`. Dodatkowo, dodając `:first` do dowolnego selektora, można pobrać pierwszy odnaleziony element; a zatem wywołanie w postaci `$(':text:first')` pobiera pierwsze pole tekstowe na stronie. Dodanie wywołania `.focus()` powoduje umieszczenie w tym polu kursora, który będzie tam cierpliwie czekał, aż użytkownik zacznie coś wpisywać.

Jeśli na stronie znajduje się kilka formularzy (na przykład „Wyszukaj na stronie" i „Zarejestruj się, aby otrzymywać biuletyn"), możesz doprecyzować selektor przez wskazanie formularza, którego pole skrypt ma aktywować. Załóżmy, że chcesz umieścić kursor w pierwszym polu tekstowym formularza rejestracji, jednak pierwsze takie pole na stronie znajduje się w formularzu wyszukiwania. Aby aktywować odpowiednie pole, dodaj do właściwego formularza identyfikator (na przykład `singup`), a następnie użyj następującego kodu:

```
$(document).ready(function() {
  $('#signup :text:first').focus();
});
```

Nowy selektor, `$('#signup :text:first')`, pobiera jedynie pierwsze pole tekstowe formularza `signup`.

Wyłączanie i włączanie pól

Pola tekstowe są zwykle przeznaczone do wypełniania. W końcu jaki jest pożytek z pola, którego nie można uzupełnić? Jednak czasem strona powinna *uniemożliwiać* użytkownikom wypełnienie pola tekstowego albo zaznaczenie pola wyboru lub opcji listy rozwijanej. Załóżmy, że dane pole można uzupełnić tylko wtedy, jeśli użytkownik zaznaczył poprzedni element. Na przykład amerykański formularz podatkowy 1040 obejmuje pole z numerem ubezpieczenia współmałżonka. Mogą je wypełnić tylko osoby żonate lub zamężne.

Aby uniemożliwić korzystanie z pola formularza, którego nie należy uzupełniać, trzeba je *wyłączyć* w kodzie JavaScript. Operacja ta powoduje, że nie można zaznaczyć elementu (przyciski opcji i pola wyboru), wpisać w nim tekstu (pola tekstowe), zaznaczyć go (lista rozwijana) ani kliknąć (przyciski przesyłania).

Aby wyłączyć pole formularza, wystarczy ustawić właściwość `disabled` na `true`, na przykład w celu wyłączenia wszystkich pól `input` należy użyć następującego kodu:

```
$(':input').prop('disabled', true);
```

Zwykle pola wyłącza się w odpowiedzi na określone zdarzenie. Przykładowo w formularzu 1040 można wyłączyć pole na numer ubezpieczenia współmałżonka, jeśli podatnik poinformuje, że jest samotny. Załóżmy, że do podania tej informacji służy przycisk o identyfikatorze `single`, a pole na numer ubezpieczenia ma identyfikator `spouseSSN`. Do wyłączenia tego pola można użyć następującego kodu JavaScript:

```
$('#single').click(function() {
  $('#spouseSSN').prop('disabled', true);
});
```

Oczywiście warto też mieć możliwość ponownego włączenia pola. Aby to zrobić, wystarczy przypisać do właściwości `disabled` wartość `false`. Poniższy kod włącza wszystkie pola formularza:

```
$(':input').prop('disabled', false);
```

CZĘSTO ZADAWANE PYTANIA

Blokowanie wielokrotnego przesyłania danych

Czasem te same informacje z formularza są przesyłane kilkakrotnie. Jak temu zapobiec?

Serwery WWW nie zawsze działają szybko, podobnie jak sam internet. Często pojawia się opóźnienie między wciśnięciem przycisku przesyłania danych a wyświetleniem strony z tekstem „Informacje zostały wysłane". Czasem to opóźnienie jest dość długie, a niecierpliwi internauci klikają wtedy przycisk przesyłania po raz drugi (a nawet trzeci i czwarty), podejrzewając, że nie zadziałał przy pierwszej próbie.

To zjawisko może prowadzić do dwukrotnego przesłania tych samych danych. W sklepach internetowych może to także oznaczać kilkakrotne obciążenie karty kredytowej użytkownika! Na szczęście za pomocą kodu JavaScript można w łatwy sposób wyłączyć przycisk przesyłania po rozpoczęciu procesu wysyłania danych. Aby klient nie mógł ponownie kliknąć przycisku, należy użyć właściwości `disabled` tego elementu.

Załóżmy, że identyfikator formularza to `formID`, a przycisk przesyłania ma identyfikator `submit`. Należy dodać do formularza funkcję `submit()` i wyłączyć w niej przycisk:

```
$('#formID').submit(function() {
  $('#submit').prop('disabled',true);
});
```

Jeśli na stronie znajduje się tylko jeden formularz, nie trzeba nawet używać identyfikatorów:

```
$('form').submit(function() {
  $('input[type=submit]').
    prop('disabled',true);
});
```

Ponadto za pomocą atrybutu `value` można zmienić komunikat na przycisku przesyłania. Początkowo ten tekst to zwykle „Wyślij", jednak po rozpoczęciu przesyłania można go zmienić na przykład na „Przesyłanie w toku":

```
$('#formID').submit(function() {
  var subButton =
$(this).find(':submit');
  subButton.prop('disabled',true);
  subButton.val('Przesyłanie w toku');
});
```

Pamiętaj o umieszczeniu tego kodu w funkcji `$(document).ready(function()` (patrz strona 190).

Uwaga: Przy wyłączaniu pól formularza pamiętaj, aby używać wartości logicznych `true` i `false` (patrz strona 63), a nie łańcuchów znaków `'true'` i `'false'`. Poniższe wywołanie jest błędne:

```
$(':input').prop('disabled', 'false');
```

Prawidłowy jest następujący zapis:

```
$(':input').prop('disabled', false);
```

Wróćmy do formularza podatkowego. Jeśli użytkownik wybierze opcję „żonaty" lub „zamężna" (w kodzie może mieć ona identyfikator `married`), skrypt powinien włączyć pole na numer ubezpieczenia współmałżonka:

```
$('#married').click(function() {
  $('#spouseSSN').prop('disabled', false);
});
```

Technika ta została wykorzystana w przykładzie przedstawionym na stronie 296.

Ukrywanie i wyświetlanie opcji formularza

Można także, oprócz wyłączania pól, w inny sposób zapobiec niepotrzebnemu wypełnianiu formularza — ukrywając zbędne elementy. I tak w formularzu podatkowym warto ukryć pole z numerem ubezpieczenia współmałżonka, jeśli użytkownik zaznaczył opcję „samotny". Jeżeli podatnik wybrał opcję „zamężna" lub „żonaty", należy wyświetlić odpowiednie pole. Można to zrobić za pomocą poniższego kodu:

```
$('#single').click(function() {
  $('#spouseSSN').hide();
});
$('#married').click(function() {
  $('#spouseSSN').show();
});
```

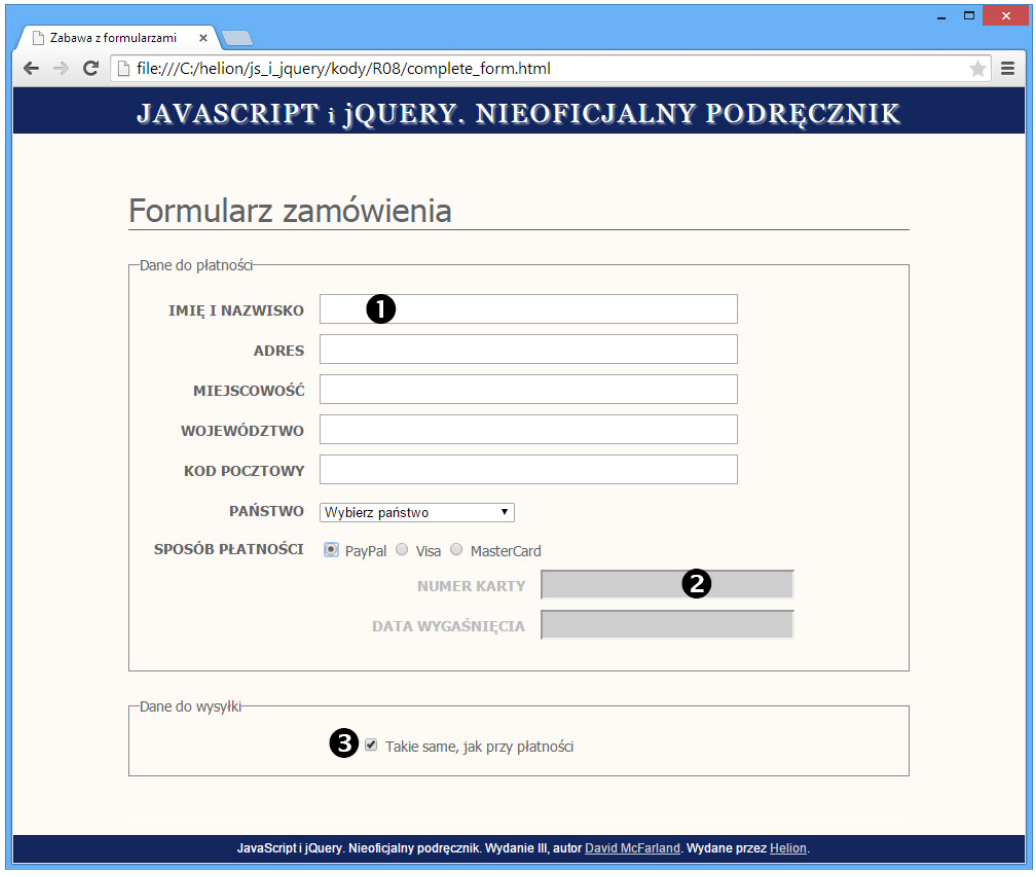
Wskazówka: Funkcje `hide()` i `show()` biblioteki jQuery, a także inne funkcje przeznaczone do wyświetlania i ukrywania elementów opisano na stronie 212.

Z perspektywy użyteczności ukrycie pola ma istotną zaletę w porównaniu z jego wyłączeniem, ponieważ upraszcza układ formularza. W końcu wyłączone pola są wciąż widoczne i mogą przyciągać (a dokładniej — rozpraszać) uwagę użytkownika.

Często warto ukryć lub wyświetlić więcej niż jedno pole formularza. Prawdopodobnie zechcesz schować etykietę danego pola i cały powiązany z nim tekst. Jedno z podejść polega na umieszczeniu wszystkich ukrywanych znaczników (pól, etykiet i innych fragmentów kodu HTML) w znaczniku `<div>`, określeniu identyfikatora tego znacznika i schowaniu go. W następnym przykładzie zobaczysz, jak zastosować to rozwiązanie.

Przykład — proste wzbogacanie formularza

W tym przykładzie dodasz trzy usprawnienia z obszaru użyteczności do prostego formularza zamówień, obejmującego pola na informacje związane z płatnością i wysyłką. Po pierwsze, skrypt ma automatycznie umieszczać kursor w pierwszym polu formularza po wczytaniu strony. Po drugie, program ma wyłączać i włączać pola formularza na podstawie wyborów dokonanych przez użytkownika. Po trzecie, kod ma ukrywać całe sekcje formularza, jeśli nie są potrzebne (patrz rysunek 8.4).



Rysunek 8.4. *Za pomocą kodu JavaScript można zwiększyć użyteczność formularzy i dodać do nich interaktywne mechanizmy, na przykład ukrywanie niepotrzebnych elementów i wyłączanie pól, których użytkownik nie powinien wypełniać*

Uwaga: Informacje o pobieraniu przykładowych plików znajdziesz na stronie 46.

Aktywowanie pola

Pierwsze pole przykładowego formularza pobiera imię i nazwisko osoby składającej zamówienie (patrz rysunek 8.4). Aby ułatwić wypełnianie formularza, można po wczytaniu strony automatycznie umieścić kursor w tym polu.

1. **Otwórz w edytorze tekstu plik *form.html* z katalogu *R08*.**

   Do strony dołączono już plik biblioteki jQuery i funkcję `$(document).ready()` (patrz strona 190). Strona zawiera też formularz z dwiema sekcjami. Jedna służy do pobierania informacji o płatności, a w drugiej użytkownik może wpisać dane potrzebne przy wysyłce. Przed przejściem do następnego punktu przyjrzyj się tej stronie.

   Pierwszy krok — a przy tym jedyny w tej części przykładu — polega na aktywowaniu odpowiedniego pola.

2. **Kliknij pusty wiersz pod funkcją `$(document).ready()` i wpisz `$(':text:first').focus();`, aby kod wyglądał następująco:**

```
$(document).ready(function() {
  $(':text:first').focus();
}); // Koniec funkcji ready
```

   Powyższa instrukcja wybiera pierwsze pole tekstowe na stronie, a następnie wywołuje dla niego funkcję `focus()` — w efekcie przeglądarka umieści w nim kursor.

   Zapisz plik i wyświetl go w przeglądarce.

   Po wczytaniu strony w pierwszym polu znajdzie się migający kursor, co oznacza, że pole to jest aktywne i można natychmiast rozpocząć wpisywanie w nim danych.

Wyłączanie pól formularza

Poprzedni punkt był jedynie rozgrzewką. W tej części przykładu wyłączysz i włączysz dwa pola formularza w odpowiedzi na zaznaczenie określonych opcji. Jeśli wyświetlisz formularz w przeglądarce (lub spojrzysz na rysunek 8.4), zobaczysz, że na końcu sekcji z informacjami o płatności znajdują się trzy przyciski opcji do wyboru sposobu zapłaty: *PayPal*, *Visa* i *MasterCard*. Ponadto poniżej widoczne są dwa pola — na numer karty i datę jej ważności. Te dwie informacje dotyczą tylko kart kredytowych, a nie płatności za pomocą serwisu PayPal, dlatego jeśli użytkownik wybierze przycisk *PayPal*, należy wyłączyć oba pola.

Kod HTML tej sekcji strony wygląda następująco (pola formularza wyróżniono pogrubieniem):

```
<div><span class="label">Sposób płatności</span>
  <input type="radio" name="payment" id="paypal" value="paypal">
  <label for="paypal">PayPal</label>
  <input type="radio" name="payment" id="visa" value="visa">
  <label for="visa">Visa</label>
  <input type="radio" name="payment" id="mastercard" value="mastercard">
  <label for="mastercard">MasterCard</label>
</div>
<div id="creditCard" class="indent">
```

```
10  <div>
11    <label for="cardNumber" class="label">Numer karty</label>
12    <input type="text" name="cardNumber" id="cardNumber">
13  </div>
14  <div>
15    <label for="expiration" class="label">Data wygaśnięcia</label>
16    <input type="text" name="expiration" id="expiration">
17  </div>
18 </div>
```

3. **Wróć do pliku *form.html* w edytorze tekstu.**

   Następne fragmenty należy dodać do kodu przygotowanego w poprzedniej sekcji. Najpierw przypisz funkcję do zdarzenia `click` związanego z przyciskiem opcji *PayPal*.

4. **Dodaj do skryptu z początkowej części strony kod wyróżniony pogrubieniem:**

```
$(document).ready(function() {
  $(':text:first').focus();
  $('#paypal').click(function() {

  }) // Koniec funkcji click
}); // Koniec funkcji ready
```

   Przycisk opcji *PayPal* ma identyfikator `paypal` (wiersz 2. we wcześniejszym kodzie HTML), dlatego aby pobrać ten element, wystarczy użyć wyrażenia `$('#paypal')`. Pozostała część kodu przypisuje funkcję anonimową do zdarzenia `click` (jeśli ten fragment jest niezrozumiały, zajrzyj na stronę 182, do omówienia procesu przypisywania funkcji do zdarzeń). Kliknięcie przycisku opcji *PayPal* nie tylko spowoduje wybranie go (to domyślne działanie przeglądarki), ale też uruchomienie funkcji, którą wkrótce utworzysz.

   Następnie należy wyłączyć pola na numer karty kredytowej i datę jej wygaśnięcia, ponieważ nie są związane z opcją *PayPal*.

5. **Do funkcji anonimowej dodanej w poprzednim kroku dopisz nowy fragment (wiersz 4. w poniższym kodzie):**

```
1 $(document).ready(function() {
2   $(':text:first').focus();
3   $('#paypal').click(function() {
4     $('#creditCard input').prop('disabled', true);
5   }); // Koniec funkcji click
6 }); // Koniec funkcji ready
```

   Choć skrypt ma wyłączać dwa pola formularza, można to zrobić za pomocą jednego wiersza kodu. Oba te pola znajdują się w znaczniku `<div>` o identyfikatorze `creditCard` (wiersz 9. we wcześniejszym kodzie HTML). Dlatego selektor jQuery `$('#creditCard input')` oznacza: „Pobierz wszystkie znaczniki `<input>` z elementu o identyfikatorze `creditCard`". To elastyczne podejście gwarantuje pobranie wszystkich pól `input`, dlatego jeśli dodasz nowe pole, na przykład CVV, kod automatycznie je uwzględni (kod CVV to trzycyfrowa liczba na odwrocie karty kredytowej, którą trzeba czasem podać w formularzu, aby zwiększyć bezpieczeństwo transakcji internetowych).

   Aby wyłączyć pola, wystarczy ustawić atrybut `disabled` na `true` (patrz strona 291). Jednak instrukcja ta nie ma wpływu na etykiety („Numer karty" i „Data ważności"). Choć skrypt wyłącza pola, tekst etykiet wciąż jest czarny i pogrubiony, co jest mylące dla użytkowników, którzy mogą sądzić, że pola można wypełnić.

Aby podkreślić stan etykiet, należy zmienić kolor tekstu na jasnoszary. Przy okazji można użyć szarego koloru jako tła pól, aby *wyglądały* na wyłączone.

6. **Dodaj do skryptu kod wyróżniony pogrubieniem:**

```
1 $(document).ready(function() {
2   $(':text:first').focus();
3   $('#paypal').click(function() {
4     $('#creditCard input').prop('disabled', true)
5     ➥.css('backgroundColor','#CCC');
6     $('#creditCard label').css('color','#BBB');
7   }); // koniec funkcji click
8 }); // Koniec funkcji ready
```

Uwaga: Symbol ➥ na początku wiersza informuje, że jest on kontynuacją poprzedniego. Ponieważ *naprawdę* długie wiersze kodu JavaScript nie mieszczą się na stronach książki, podzielono je na dwie części. Niemniej jednak, zgodnie z informacjami podanymi na stronie 67, język JavaScript dosyć wyrozumiale podchodzi do zagadnień znaków nowego wiersza i odstępów, a zatem jest całkowicie dopuszczalne, by jedną instrukcję JavaScriptu zapisać w kilku wierszach kodu (czasami poprawia to przejrzystość kodu), jak pokazano na poniższym przykładzie:

```
$('#creditCard input').prop('disabled',true)
                      .css('backgroundColor','#CCC');
```

Warto zwrócić uwagę, że niektórzy programiści w takim kodzie zapisywanym w kilku wierszach dodają jeszcze odpowiednie wcięcia — w tym przypadku dodano je przed wywołaniem funkcji `.css()`, dzięki czemu jest ona umieszczona dokładnie pod wywołaniem funkcji `.prop()`.

Skrypt najpierw wywołuje funkcję `css()` biblioteki jQuery, aby zmienić kolor tła pól tekstowych (zauważ, że instrukcja ta jest częścią wiersza 4., gdyż operuje na tym samym elemencie, co funkcja `prop()`). Następnie program zmienia przy użyciu tej samej funkcji kolor czcionki znaczników `<label>` w odpowiednim znaczniku `<div>` (funkcję `css()` opisano na stronie 166).

Jeśli na tym etapie wyświetlisz stronę w przeglądarce, zobaczysz, że kliknięcie przycisku *PayPal* wyłącza pola na numer karty kredytowej i datę jej ważności oraz zmienia kolor etykiet. Jednak jeśli później wybierzesz przycisk *Visa* lub *MasterCard*, pola wciąż będą niedostępne! Zaznaczenie jednego z tych przycisków powinno powodować ponowne włączenie pól.

7. **Pod funkcją `click()` dodaj nowy, pusty wiersz (nowy kod wstawisz między wierszami 7. a 8. z punktu 6.), a następnie wpisz poniższy fragment:**

```
$('#visa, #mastercard').click(function() {
  $('#creditCard input').prop('disabled', false)
  ➥.css('backgroundColor','');
  $('#creditCard label').css('color','');
}); // koniec funkcji click
```

Selektor `$('#visa, #mastercard')` pobiera oba przyciski opcji (patrz wiersze 4. i 6. kodu HTML ze strony 295). Zauważ, że w celu usunięcia koloru tła i tekstu ustawionego po kliknięciu przycisku *PayPal* wystarczy przekazać jako barwę pusty łańcuch znaków — `$('#creditCard label').css('color','');`. Spowoduje to użycie koloru zdefiniowanego w arkuszu stylów.

Kod przykładu jest już prawie gotowy. W ostatnim punkcie całkowicie ukryjesz część strony na podstawie zaznaczonej opcji.

Ukrywanie pól formularza

W tym przykładzie, podobnie jak w wielu formularzach zamówień, na stronie znajdują się odrębne pola na pobieranie informacji związanych z płatnością i wysyłką. Często są to te same dane, dlatego nie ma powodu, aby bez potrzeby zmuszać użytkowników do wypełniania obu grup pól. W wielu formularzach dostępne jest pole wyboru z tekstem typu „Takie same jak przy płatności", które określa, że oba zbiory informacji są identyczne. Prawdopodobnie jeszcze lepszym (a na pewno ciekawszym) rozwiązaniem jest całkowite ukrycie pól związanych z wysyłką, jeśli nie są potrzebne. Ten efekt można uzyskać za pomocą języka JavaScript.

1. **Otwórz w edytorze tekstu plik *form.html*.**

   Teraz wzbogacisz kod napisany w dwóch poprzednich częściach przykładu. Najpierw przypisz funkcję do zdarzenia `click` pola wyboru z etykietą „Takie same jak przy płatności". Kod HTML tego pola wygląda następująco:

   ```
   <input type="checkbox" name="hideShip" id="hideShip">
   ```

2. **Dodaj poniższy fragment po kodzie utworzonym w kroku 4. na stronie 296, ale przed ostatnim wierszem skryptu — `}); // Koniec funkcji ready`.**

   ```
   $('#hideShip').click(function(){

   }); // koniec funkcji click
   ```

   Ponieważ `hideShip` to identyfikator pola wyboru, powyższy kod pobiera ten element i dodaje funkcję do związanego z nim zdarzenia `click`. Tu funkcja w odpowiedzi na zaznaczenie pola wyboru ma ukrywać nie jeden znacznik, ale całą grupę pól. Aby było to łatwiejsze, pola na informacje o wysyłce umieszczono w znaczniku `<div>` o identyfikatorze `shipping`. Aby ukryć wszystkie te pola, wystarczy schować wspomniany znacznik `<div>`.

   Jednak skrypt ma ukrywać pola na dane tylko po zaznaczeniu pola wyboru. Jeśli użytkownik kliknie to pole wyboru po raz drugi, aby usunąć zaznaczenie, należy ponownie wyświetlić znacznik `<div>` i zawarte w nim elementy. Dlatego najpierw trzeba sprawdzić, czy pole wyboru jest już zaznaczone.

3. **Dodaj kod wyróżniony pogrubieniem:**

   ```
   $('#hideShip').click(function(){
     if ($(this).prop('checked')) {

     }
   }); // koniec funkcji click
   ```

   Prosta instrukcja warunkowa (patrz strona 93) ułatwia sprawdzenie stanu pola wyboru i ukrycie lub wyświetlenie pól formularza. Konstrukcja `$(this)` wskazuje kliknięty obiekt, którym jest tu pole wyboru. Właściwość `checked` tego elementu pozwala sprawdzić, czy użytkownik zaznaczył pole, czy nie. Jeśli pole jest zaznaczone, atrybut ma wartość `true`. W przeciwnym razie atrybut ten ma wartość `false`. Aby dokończyć kod, trzeba dodać fragment ukrywający i wyświetlający pola formularza.

4. **Dodaj do skryptu kod wyróżniony pogrubieniem (wiersze od 16. do 18.). Gotowy program powinien wyglądać następująco:**

```
<script>
$(document).ready(function() {
  $(':text:first').focus();
  $('#paypal').click(function() {
    $('#creditCard input').prop('disabled', true)
      ↪.css('backgroundColor','#CCC');
   $('#creditCard label').css('color','#BBB');
  });  // koniec funkcji click
  $('#visa, #mastercard').click(function() {
    $('#creditCard input').prop('disabled', false)
      ↪.css('backgroundColor','');
    $('#creditCard label').css('color','');
  });  // koniec funkcji click
  $('#hideShip').click(function() {
    if ($(this).prop('checked')) {
      $('#shipping').slideUp('fast');
    } else {
      $('#shipping').slideDown('fast');
    }
  });  // koniec funkcji click
}); // Koniec funkcji ready
</script>
```

Wyrażenie `$('#shipping')` pobiera znacznik `<div>` z polami formularza, natomiast funkcje `slideUp()` i `slideDown()` (patrz strona 216) ukrywają i wyświetlają ten element przez wysunięcie go ze strony lub wsunięcie na nią. Możesz także wypróbować inne efekty udostępniane przez jQuery, takie jak `fadeIn()` czy `fadeOut()`, bądź nawet utworzyć swoją własną animację, korzystając z funkcji `animate()` opisanej na stronie 220.

Gotową wersję przykładu, *complete_form.html*, znajdziesz w katalogu *R08*. Jeśli Twoja wersja nie działa, porównaj kod z gotowym rozwiązaniem i przypomnij sobie etapy rozwiązywania problemów, które opisane zostały na stronie 51.

Walidacja formularzy

Frustrujące jest przeglądanie informacji zwrotnych przesłanych za pomocą formularza przez użytkownika, który nie podał nazwiska, adresu e-mail ani żadnych innych kluczowych informacji. Dlatego w niektórych formularzach warto wymagać podania niektórych danych.

Na przykład formularz do rejestracji osób, które chcą otrzymywać biuletyn, nie będzie zbyt przydatny, jeśli użytkownik nie poda adresu e-mail. Także jeśli sklep ma wysłać klientowi katalog lub produkt, warto się upewnić, że formularz zawiera dane adresowe.

Ponadto przy pobieraniu danych za pomocą formularzy dobrze jest sprawdzić, czy informacje mają odpowiedni format, na przykład wartość liczbową dla liczby przesyłanych produktów lub URL dla adresu. Upewnianie się, że użytkownik wprowadził poprawne dane, to proces *walidacji formularza*. Przy użyciu języka JavaScript można wykryć wszystkie usterki przed przesłaniem błędnych informacji przez użytkownika.

Walidacja wymaga sprawdzenia danych pól formularza pod kątem obecności wymaganych informacji i formatu podanych wartości. Proces ten zachodzi zwykle po wykryciu zdarzenia `submit` formularza, które jest zgłaszane w momencie kliknięcia przycisku przesyłania lub wciśnięcia klawisza *Enter*, kiedy kursor znajduje się w polu tekstowym. Jeśli dane są prawidłowe, informacje z formularza trafiają w standardowy sposób na serwer. Jednak po wykryciu problemów skrypt wstrzymuje proces przesyłania i wyświetla błędy na stronie, zwykle obok niepoprawnie wypełnionych pól (patrz rysunek 8.5).

Rysunek 8.5. *Jeśli użytkownik w trakcie rejestrowania konta w serwisie Google nie wypełni prawidłowo formularza, zobaczy liczne komunikaty o błędach (zakreślone)*

Sprawdzanie, czy pole tekstowe zawiera dane, jest proste. Na stronie 283 dowiedziałeś się, że wystarczy uzyskać dostęp do właściwości `value` (na przykład za pomocą funkcji `val()` biblioteki jQuery). Jeśli jej wartość to pusty łańcuch znaków, pole nie zawiera danych. Trudniej jednak sprawdzić inne elementy, na przykład pola wyboru, przyciski opcji i listy rozwijane. Ponadto do sprawdzania, czy użytkownik wpisał dane określonego *rodzaju*, na przykład adres e-mail, kod pocztowy, liczbę, datę i tak dalej, potrzebny jest skomplikowany kod JavaScript. Na szczęście nie musisz samodzielnie przygotowywać takiego mechanizmu. W sieci dostępnych jest wiele skryptów do walidacji formularzy, a jednym z najlepszych jest wtyczka biblioteki jQuery.

Uwaga: HTML5 posiada wiele wbudowanych funkcji służących do walidacji formularzy, które pozwalają uniknąć konieczności stosowania kodu JavaScript. Niestety, choć wiele najnowszych przeglądarek obsługuje te możliwości, nie dotyczy to przeglądarek Internet Explorer 9 oraz wcześniejszych. W czasie powstawania tej książki możliwości te nie były dostępne także w przeglądarce Safari dla systemów iOS oraz Android, natomiast wersja Safari dla komputerów stacjonarnych miała z nimi pewne problemy.

Wtyczka Validation

Validation (*http://jqueryvalidation.org/*) to rozbudowana, ale łatwa w użyciu wtyczka biblioteki jQuery, utworzona przez Jörna Zaefferera. Narzędzie to sprawdza, czy wszystkie wymagane pola formularza są wypełnione i czy spełniają określone warunki. Na przykład pole na liczbę produktów musi zawierać wartość liczbową, a w polu na e-mail musi znaleźć się adres. Jeśli użytkownik popełni błąd, wtyczka wyświetli komunikat z jego opisem.

Oto proces korzystania z wtyczki Validation.

1. **Pobieranie i dołączanie pliku *jquery.js* do strony zawierającej sprawdzany formularz.**

   Więcej informacji o pobieraniu biblioteki jQuery znajdziesz w podrozdziale „Jak zdobyć jQuery?" (patrz strona 135). Wtyczka Validation korzysta z tej biblioteki, dlatego najpierw trzeba dołączyć plik *jquery.js*.

2. **Pobieranie i dołączanie wtyczki Validation.**

   Wtyczkę tę znajdziesz pod adresem *http://jqueryvalidation.org/*. Dostępny pakiet obejmuje wiele dodatkowych elementów, w tym wersje demonstracyjne, testy i tak dalej. Niezbędny jest jednak tylko plik *jquery.validate.min.js*. (Wtyczkę — plik *jquery.validate.min.js* — znajdziesz też w katalogu *jquery_validate* umieszczonym w katalogu *R08*; patrz przykład na stronie 311.) Jest to zwyczajny zewnętrzny plik JavaScript, aby więc go dołączyć, zastosuj się do instrukcji ze strony 49.

3. **Dołączanie reguł walidacji.**

   Reguły walidacji to instrukcje, które informują na przykład o tym, że dane pole jest wymagane, a w innym ma znaleźć się adres e-mail. Na tym etapie należy określić, które pola wymagają walidacji i czego ma ona dotyczyć. Zasady walidacji można dodać na kilka sposobów. Prosta metoda polega na użyciu samego kodu HTML (patrz strona 302), a bardziej elastyczny, ale też skomplikowany sposób wymaga zastosowania kodu JavaScript (patrz strona 305).

4. **Dodawanie komunikatów o błędach.**

   Ten krok jest opcjonalny. Wtyczka Validation udostępnia wbudowany zestaw komunikatów o błędach, na przykład „This field is required", „Please enter a valid date" i „Please enter a valid number". Te podstawowe wiadomości są wystarczające, jednak czasem warto dostosować formularz, aby komunikaty udostępniały bardziej szczegółowe instrukcje związane z poszczególnymi polami (na przykład „Wpisz nazwę użytkownika" lub „Podaj datę urodzenia").

Są dwie metody dodawania komunikatów o błędach. Prosty sposób opisano na stronie 304, a bardziej elastyczną metodę poznasz na stronie 309.

Uwaga: Na stronie 317 zobaczysz, jak kontrolować styl i rozmieszczenie komunikatów o błędach.

5. **Wywoływanie funkcji `validate()` dla formularza.**

   Wtyczka udostępnia funkcję `validate()`, która wykonuje wszystkie potrzebne operacje. Aby ją wywołać, należy najpierw pobrać formularz za pomocą jQuery, a następnie uruchomić dla niego tę funkcję. Załóżmy, że formularz ma identyfikator `signup`. Oto potrzebny kod HTML:

   ```
   <form action="process.php" method="post" name="signup" id="signup">
   ```

   Najprostszy kod uruchamiający walidację wygląda następująco:

   ```
   $('#signup').validate();
   ```

   Funkcja `validate()` przyjmuje wiele różnych informacji, które wpływają na działanie wtyczki. Na przykład choć można określić reguły walidacji i komunikaty o błędach w kodzie HTML formularza (zobacz następny punkt rozdziału), można też podać je w funkcji `validate()` (metodę tę poznasz na stronie 305).

   Cały kod JavaScript potrzebny do przeprowadzenia podstawowej walidacji formularza (z uwzględnieniem dwóch wcześniej opisanych kroków) może być bardzo prosty:

   ```
   <script src="js/jquery.min.js"></script>
   <script src="js/jquery.validate.min.js"></script>
   <script>
   $(document).ready(function() {
     $('#signup').validate();
   }); // koniec funkcji ready
   </script>
   ```

Wskazówka: Pamiętaj, aby zawsze umieszczać skrypt w funkcji `document.ready()` biblioteki jQuery. Gwarantuje to uruchomienie programu dopiero po wczytaniu kodu HTML strony (patrz strona 190).

Podstawowa walidacja

Aby użyć wtyczki Validation, wystarczy dołączyć jej plik JavaScript, dodać kilka atrybutów `class` i `title` do elementów sprawdzanego formularza oraz wywołać dla niego metodę `validate()`. Jej wywołanie to najprostszy, a w wielu formularzach także wystarczający sposób na przeprowadzenie walidacji. Jednak jeśli chcesz kontrolować miejsce wyświetlania komunikatów o błędach, zastosować do pola więcej niż jedną regułę lub określić minimalną albo maksymalną liczbę znaków w polu tekstowym, musisz zastosować metodę zaawansowaną, opisaną na stronie 305.

Aby włączyć walidację, wykonaj instrukcje podane w poprzednim punkcie (dołącz pliki biblioteki jQuery oraz wtyczki Validation i tak dalej). Ponadto należy podać w kodzie HTML pól formularza reguły i komunikaty o błędach.

Dodawanie reguł walidacji

Najprostszy sposób na walidację pola za pomocą wtyczki Validation polega na przypisaniu do elementu *formularza* nazw klas opisanych w tabeli 8.2. Wtyczka pobiera wszystkie elementy *formularza* i sprawdza dla każdego z nich, czy nazwa klasy nie odpowiada jednej z technik walidacji. Jeśli tak jest, wtyczka stosuje do danego pola odpowiednią regułę.

Tabela 8.2. *Wtyczka Validation udostępnia standardowe mechanizmy potrzebne przy walidacji*

Reguła walidacji	Opis
`required`	Dane pole nie zostanie przesłane, jeśli użytkownik go nie wypełni, nie zaznaczy lub nie wybierze.
`date`	Informacje muszą mieć format MM/DD/RRRR, na przykład 10/30/2014 to poprawny zapis, natomiast 10-30-2014 — już nie.
`url`	Tekst musi być pełnym, poprawnym adresem internetowym, na przykład *http://www.chia-vet.com*. Częściowe adresy URL, na przykład *www.chia-vet.com* lub *chia-vet.com*, są uznawane za nieprawidłowe.
`email`	Dane muszą mieć format adresu e-mail: *bob@chia-vet.com*. Ta klasa nie powoduje sprawdzania, czy adres jest prawdziwy, dlatego użytkownik może wpisać tekst *nikt@nigdzie.com*, a dane przejdą walidację.
`number`	Dane muszą być liczbą, na przykład 32, 102.50, a nawet -145.5555, jednak nie można używać żadnych innych symboli. Zapis $45.00 i 100,000 jest nieprawidłowy.
`digits`	Dane muszą być dodatnią liczbą ca kowitą. 1, 20 i 12333 to prawidłowe wartości, natomiast 10.33 i -12 nie przejdą walidacji.
`creditcard`	Użytkownik musi wpisać numer karty kredytowej we właściwym formacie.

Załóżmy, że do pobierania nazwy użytkownika służy pole tekstowe. Jego kod HTML wygląda następująco:

```
<input name="name" type="text">
```

Aby poinformować wtyczkę, że to pole jest wymagane (formularza nie można przesłać, jeśli użytkownik nie wprowadzi danych w tym polu), należy dodać do znacznika klasę `required`. W tym celu trzeba użyć atrybutu `class`:

```
<input name="name" type="text" class="required">
```

Dodanie klasy w ten sposób nie ma nic wspólnego ze stylami CSS, choć zwykle klasy dodaje się do znaczników po to, by uzyskać możliwość formatowania ich przy użyciu arkuszy stylów. Tu jednak nazwa klasy informuje wtyczkę o rodzaju walidacji, którą należy przeprowadzić dla danego pola.

Uwaga: Walidacja z wykorzystaniem języka JavaScript jest doskonałym sposobem na zwrócenie uwagi użytkownika, który przez przypadek pominął jakieś pole lub wpisał w nim nieprawidłowe informacje; jednak z drugiej strony, nie stanowi dobrego sposobu ochrony przed celowo przesyłanymi niebezpiecznymi danymi. Tego typu zabezpieczenia tworzone przy użyciu JavaScriptu można łatwo ominąć, dlatego też, jeśli chcemy mieć absolutną pewność, że dane otrzymywane od użytkowników będą prawidłowe, konieczne jest zaimplementowanie mechanizmów walidacji także po stronie serwera.

Wymuszanie podania danych w polu to prawdopodobnie najczęstsze zadanie wykonywane przy walidacji. Często warto także sprawdzić, czy wpisane informacje mają właściwy format. Jeśli użytkownik ma określić, ile produktów chce kupić, powinien podać liczbę. Aby sprawdzić, czy wpisano dane **i** podano je we właściwym formacie, należy użyć klasy `required` oraz jednej z pozostałych klas wymienionych w tabeli 8.2.

Formularz może zawierać pole na datę urodzenia. Załóżmy, że ta informacja jest nie tylko wymagana, ale ponadto użytkownik musi ją podać w formie daty. Kod HTML takiego pola powinien wyglądać następująco:

```
<input name="dob" type="text" class="required date">
```

Zauważ, że nazwy klas (`required` i `date`) są rozdzielone odstępem.

Jeśli pominiesz klasę `required` i użyjesz tylko jednego z pozostałych sposobów walidacji (na przykład `class= date `), pole będzie opcjonalne, jednak jeżeli użytkownik wprowadzi dane, musi to zrobić w odpowiednim formacie (tu jest to data).

Wskazówka: Jeśli informacje w polu mają mieć określony format, pamiętaj o dodaniu do formularza instrukcji, aby użytkownik wiedział, w jaki sposób ma wpisać dane. Przy polu przeznaczonym na datę może to być wiadomość typu: „Wpisz datę w formacie MM/DD/RRRR, na przykład 01/25/2015".

Dodawanie komunikatów o błędach

Wtyczka Validation udostępnia uniwersalne komunikaty o błędach, pasujące do wykrywanych problemów. Jeśli wymagane pole jest puste, wtyczka wyświetli komunikat „This field is required" (czyli „To pole jest wymagane"). Jeżeli użytkownik musi wpisać datę, pojawi się wiadomość „Please enter a valid date" (czyli „Wpisz poprawną datę"). Można jednak zastąpić podstawowe komunikaty własnymi.

Najprostsza metoda polega na dodaniu do pola atrybutu `title` i zapisaniu w nim komunikatu. Załóżmy, że użyłeś klasy `required`, aby utworzyć wymagane pole:

```
<input name="name" type="text" class="required">
```

Aby podać własny komunikat, wystarczy dodać atrybut `title`:

```
<input name="name" type="text" class="required"
↪title="Podaj nazwę użytkownika.">
```

Projektanci stron WWW zwykle używają atrybutu `title`, aby zwiększyć dostępność pól formularza przez podanie instrukcji wyświetlanych po najechaniu kursorem nad pole lub przy odczytywaniu zawartości ekranu przez przeznaczone do tego narzędzia. Jednak przy korzystaniu z wtyczki Validation w atrybucie `title` należy umieścić wyświetlany komunikat o błędzie. Wtyczka wyszukuje ten atrybut we wszystkich sprawdzanych polach. Jeśli go znajdzie, używa jego wartości jako tekstu komunikatu o błędzie.

Jeśli używasz więcej niż jednej metody walidacji, powinieneś podać tytuł dostosowany do obu problemów. Jeśli na przykład pole jest wymagane i musi zawierać datę, komunikat „To pole jest wymagane" nie ma sensu, ponieważ użytkownik mógł wprowadzić datę w złym formacie. Oto przykład informacji dostosowanej do obu błędów — pustego pola i niewłaściwego formatu:

```
<input name="dob" type="text" class="required date"
↪title="Podaj datę w formacie 01/28/2014.">
```

Dodawanie reguł walidacji i komunikatów o błędach za pomocą nazw klas oraz tytułów jest proste i działa doskonale. Jednak czasem programista ma większe potrzeby. Wtyczka Validation udostępnia w tym celu drugą, bardziej zaawansowaną metodę dodawania walidacji do formularza. Możliwe, że chcesz, aby skrypt wyświetlał różne komunikaty w zależności od rodzaju błędu — jeden, kiedy użytkownik pozostawi pole puste, i drugi, kiedy informacje mają nieodpowiedni format. Nie można uzyskać tego efektu za pomocą podstawowych metod walidacji omówionych w tym punkcie. Na szczęście wtyczka Validation oferuje też inną, bardziej rozbudowaną technikę, która umożliwia precyzyjne zarządzanie regułami walidacji.

Zaawansowanej metody trzeba użyć między innymi do zagwarantowania minimalnej liczby znaków wprowadzonych w polu. Na przykład przy tworzeniu hasła warto się upewnić, że ma ono przynajmniej sześć znaków.

Zaawansowana walidacja

Wtyczka Validation udostępnia także drugi sposób dodawania walidacji do formularza. Ta technika nie wymaga zmiany kodu HTML pól. Ponadto wtyczka obsługuje wiele dodatkowych opcji do sterowania działaniem wtyczki. Aby je ustawić, należy przekazać do funkcji `validate()` literał obiektowy (patrz strona 165) z odrębnymi obiektami opisującymi każdą opcję. Aby na przykład określić regułę walidacji, należy przekazać obiekt z jej kodem. Najpierw trzeba wpisać otwierający nawias klamrowy po pierwszym nawiasie funkcji walidacyjnej, a następnie zamykający nawias klamrowy przed końcowym nawiasem tej funkcji:

```
$('idOfForm').validate({
  // Tu opcje.
});  // Koniec funkcji validate
```

Nawiasy klamrowe reprezentują literał obiektowy, w którym znajdą się ustawienia opcji. Korzystanie z wtyczki Validation w ten sposób może być nieco skomplikowane, a najlepszy sposób na zrozumienie jej działania to przyjrzenie się prostemu przykładowi. Ilustruje go rysunek 8.6.

Rysunek 8.6. *Nawet w tak prostym formularzu można użyć zaawansowanych opcji wtyczki Validation, aby uzyskać dodatkową kontrolę nad walidacją*

Wskazówka: W tym samym formularzu można połączyć technikę prostej walidacji (patrz strona 302) i podejście zaawansowane. Do pól, które mają tylko jedną regułę walidacji i jeden komunikat o błędzie, można użyć prostej metody, ponieważ jest szybka. Do przeprowadzenia bardziej skomplikowanej walidacji należy użyć techniki zaawansowanej. W przykładzie ze strony 311 zastosowano oba podejścia do walidacji jednego formularza.

Kod HTML formularza z rysunku 8.6 wygląda następująco:

```
<form action="process.php" method="post" id="signup">
  <div>
    <label for="name">Nazwa użytkownika</label>
```

```
    <input name="name" type="text">
  </div>

  <div>
    <label for="email">Adres e-mail</label>
    <input name="email" type="text">
  </div>
  <div>
    <input type="submit" name="submit" value="Wyślij">
  </div>
</form>
```

Ten formularz zawiera dwa pola tekstowe (wyróżnione pogrubieniem) — jedno na nazwę użytkownika i drugie na adres e-mail. W tym punkcie zobaczysz, jak za pomocą zaawansowanych reguł przeprowadzić walidację obu elementów, aby się upewnić, że oba pola są wypełnione, a adres e-mail ma ponadto właściwy format.

Uwaga: Pełną listę opcji wtyczki Validation znajdziesz na stronie *http://jqueryvalidation.org/*.

Zaawansowane reguły

Zaawansowany sposób określania reguł walidacji polega na przekazaniu obiektu z nazwami pól formularza i stosowanymi do nich zasadami. Podstawowa struktura tego obiektu wygląda następująco:

```
rules: {
  nazwa_pola: 'rodzaj_walidacji'
}
```

Nazwa obiektu to `rules`, a zawiera on pola i stosowane do nich sposoby walidacji. Cały obiekt należy następnie przekazać do funkcji `validate()`. Aby wymusić wypełnienie pola na nazwę użytkownika z rysunku 8.6, należy wywołać dla formularza funkcję `validate()` i przekazać do niej odpowiedni obiekt `rules`:

```
$('#signup').validate({
  rules: {
    name: 'required'
  }
}); // Koniec funkcji validate
```

Nazwa pola to `name`, a reguła określa, że pole to jest wymagane. Aby zastosować kilka zasad walidacji, należy utworzyć dla danego pola nowy obiekt. Jeśli chcesz rozwinąć reguły walidacji formularza z rysunku 8.6, możesz dodać zasadę, zgodnie z którą adres e-mail jest wymagany, a ponadto musi mieć właściwy format:

```
1 $('#signup').validate({
2   rules: {
3     name: 'required',
4     email: {
5       required:true,
6       email:true
7     }
8   }
9 }); // Koniec funkcji validate
```

Uwaga: Zgodnie z regułami tworzenia literałów obiektowych języka JavaScript każdą parę nazwa – wartość, oprócz ostatniej, należy zakończyć przecinkiem. W wierszu 3. w powyższym kodzie po regule `name: 'required'` trzeba dodać przecinek, ponieważ następuje po niej następna zasada (dla pola `email`). Jeśli chcesz przypomnieć sobie działanie literałów obiektowych, zajrzyj na stronę 165.

Wiersze od 4. do 7. (wyróżnione pogrubieniem) określają reguły dla pola `email`. Nazwa pola to `email` (patrz kod HMTL na stronie 306), para `required:true` sprawia, że pole to jest wymagane, a para `email:true` gwarantuje, iż dane będą miały format adresu e-mail.

Możesz użyć w ten sposób dowolnych technik walidacji opisanych w tabeli 8.2. Załóżmy, że dodałeś do przykładowego formularza pole `birthday`. Aby zagwarantować, że użytkownik wpisze w nie datę, można wydłużyć listę reguł:

```
$('#signup').validate({
  rules: {
    name: 'required',
    email: {
      required:true,
      email:true
    },
    birthday: 'date'
  }
}); // Koniec funkcji validate
```

Jeśli także pole `birthday` ma być wymagane (reguła `required`), należy wprowadzić następujące zmiany:

```
$('#signup').validate({
  rules: {
    name: 'required',
    email: {
      required:true,
      email:true
    },
    birthday: {
      date:true,
      required:true
    }
  }
}); // Koniec funkcji validate
```

Jak już wspomniano, jedną z najwartościowszych technik zaawansowanej walidacji jest określanie minimalnej i maksymalnej długości wprowadzanych danych. W formularzu na skargi warto ograniczyć długość komentarza do 200 znaków, aby klienci zwięźle wyrażali swe opinie, zamiast pisać długie elaboraty. Dostępne są też reguły określające, że podana liczba ma mieć wartość z określonego przedziału. Jeśli na przykład nie oczekujesz, by informacje w formularzu podawały mumie lub wampiry, możesz odrzucać rok urodzenia wcześniejszy niż 1900.

- Opcja `minlength`. Pole musi zawierać *przynajmniej* określoną liczbę znaków. Aby zagwarantować, że w polu znajdzie się co najmniej sześć znaków, użyj następującej reguły:

  ```
  minlength:6
  ```

- Opcja `maxlength`. Pole może zawierać *co najwyżej* określoną liczbę znaków. Aby zagwarantować, że w polu znajdzie się nie więcej niż 100 znaków, użyj następującej reguły:

  ```
  maxlength:100
  ```

- Opcja `rangelength` łączy reguły `minlength` i `maxlength`. Przy jej użyciu można określić minimalną i maksymalną liczbę znaków. Poniższa zasada określa, że pole musi zawierać przynajmniej 6, ale nie więcej niż 100 znaków:

  ```
  rangelength:[6,100]
  ```

- Opcja `min`. Wymaga, aby pole zawierało liczbę równą lub większą od określonej. Następna reguła oznacza, że w polu musi znaleźć się liczba 10 lub większa:

  ```
  min:10
  ```

  Jeśli użytkownik wpisze `8`, pole nie przejdzie walidacji, ponieważ wartość ta jest mniejsza od 10. Także jeżeli wprowadzone zostanie słowo (na przykład `osiem`), walidacja zakończy się niepowodzeniem i skrypt wyświetli komunikat o błędzie.

- Opcja `max`. Działa podobnie jak `min`, ale określa największą wartość, jaką można wpisać w polu. Aby się upewnić, że w polu znajdzie się liczba nie większa od 1000, użyj poniższej reguły:

  ```
  max:1000
  ```

- Opcja `range`. Łączy opcje `min` i `max`, co pozwala określić najmniejszą i największą liczbę, którą można wprowadzić w danym polu. Aby zagwarantować, że w polu znajdzie się liczba z przedziału od 10 do 1000, użyj następującej reguły:

  ```
  range:[10,1000]
  ```

- Opcja `equalTo`. Wymaga, aby dane pole miało taką samą wartość jak inny element. W formularzach rejestracji użytkownik często musi dwukrotnie wpisać hasło. Zmniejsza to prawdopodobieństwo popełnienia literówki przy pierwszym wprowadzaniu danych. Aby użyć tej opcji, trzeba podać łańcuch znaków z selektorem jQuery. Załóżmy, że pierwsze pole z hasłem ma identyfikator `password`. Aby się upewnić, że zawartość pola weryfikacji hasła pasuje do tekstu z pierwszego pola, należy użyć następującej reguły:

  ```
  equalTo: '#password'
  ```

Zaawansowane reguły walidacji można łączyć ze sobą przez dodawanie ich do pojedynczych pól. Oto przykład. Załóżmy, że formularz zawiera dwa pola — jedno przeznaczone na hasło i drugie do jego potwierdzania. Kod HTML tych dwóch pól wygląda następująco:

```
<input name="password" type="password" id="password">
<input name="confirm_password" type="password" id="confirm_password">
```

Uwaga: Do wtyczki jQuery Validation można także dołączyć drugą — *additional-methods.js* — która zawiera dodatkowe reguły walidacji, takie jak minimalna liczba słów, numery identyfikacyjne samochodów w USA, holenderskie numery kont bankowych oraz wiele innych, tajemniczych (lecz potencjalnie przydatnych) reguł. Nie istnieje żadna dokumentacja tych reguł, więc informacji na ich temat trzeba szukać bezpośrednio w kodzie pliku *additional-methods.js*. Jeśli uda się znaleźć interesujące, należy zapisać kopię pliku — na przykład *moje-reguly.js* — a następnie usunąć z niego wszystkie te reguły, które nie są potrzebne. Dzięki temu znacząco przyspieszymy wczytywanie tego pliku.

Oba pola są wymagane, a hasło musi mieć przynajmniej 8, lecz nie więcej niż 16 znaków. Trzeba też sprawdzić, czy wartości obu pól są takie same. Jeśli identyfikator formularza to `signup`, można przeprowadzić walidację obu pól za pomocą poniższego kodu:

```
$('#signup').validate({
  rules: {
    password: {
      required:true,
      rangelength:[8,16]
    },
```

```
      confirm_password: {
        equalTo:'#password'
      }
    }
  }); // Koniec funkcji validate
```

Zaawansowane komunikaty o błędach

Na stronie 304 dowiedziałeś się, że można łatwo dodać do pola komunikat o błędzie przez podanie atrybutu `title` z tekstem takiej wiadomości. Jednak to podejście nie umożliwia wyodrębnienia komunikatów o błędach powiązanych z konkretnymi problemami. Załóżmy, że pole jest wymagane i musi zawierać liczbę. Warto wyświetlać różne komunikaty dla każdego błędu: „To pole jest wymagane" i „Wpisz liczbę". Za pomocą atrybutu `title` nie można uzyskać tego efektu. Jedynym rozwiązaniem jest przekazanie do funkcji `validate()` obiektu JavaScript zawierającego inny komunikat o błędzie — ten, który chcemy wyświetlać.

Ten proces przypomina tworzenie zaawansowanych reguł opisane w poprzednim punkcie. Podstawowa struktura obiektu `messages` wygląda następująco:

```
messages: {
  nazwa_pola: {
    typ_walidacji: 'Komunikat o błędzie'
  }
}
```

PORADNIA DLA ZAAWANSOWANYCH

Walidacja z wykorzystaniem serwera

Choć walidacja za pomocą języka JavaScript doskonale nadaje się do szybkiego sprawdzania wprowadzonych danych, czasem do określenia poprawności pola niezbędny jest serwer. Przyjmijmy, że formularz rejestracyjny pozwala utworzyć nazwę użytkownika używaną na forum. Dwie osoby nie mogą korzystać z tej samej nazwy, dlatego przed przesłaniem formularza warto poinformować użytkownika o tym, czy dana nazwa nie jest już zajęta. Wymaga to pobrania danych z serwera.

Wtyczka Validation obsługuje zaawansowaną metodę walidacji *zdalnej*, która umożliwia komunikację z serwerem. Technika ta pozwala przekazać nazwę pola i wpisaną w nim wartość na serwer, który wykonuje aplikacje napisane w odpowiednim języku, takim jak PHP, Ruby, C#, Java lub JavaScript. Strona na serwerze może pobrać informacje i na przykład sprawdzić, czy dana nazwa jest dostępna, a następnie przekazać do formularza wartość `true` (walidacja zakończyła się powodzeniem) lub `false` (nazwa nie przeszła walidacji).

Załóżmy, że pole `username` jest wymagane i nie może zawierać nazwy używanej już w witrynie. Aby utworzyć regułę dla tego pola (za pomocą techniki zaawansowanej, którą opisano na stronie 306), należy dodać poniższy fragment do obiektu `rules`:

```
username : {
  required: true,
  remote: 'check_username.php'
}
```

Opcja `remote` przyjmuje łańcuch znaków ze ścieżką do strony na serwerze. Tu nazwa tej strony to *check_username.php*. Kiedy wtyczka Validation przystąpi do walidacji danego pola, prześle jego nazwę (`username`) i wprowadzoną wartość do strony *check_username.php*, która sprawdzi, czy podana nazwa użytkownika jest dostępna. Jeśli tak, strona PHP zwróci wartość `'true'`. Jeżeli nazwa jest już zajęta, strona zwróci wartość `'false'`, a pole nie przejdzie walidacji.

Działanie tego mechanizmu jest możliwe dzięki Ajaksowi (technologię tę poznasz w części IV). Aby zobaczyć działający przykład zastosowania tej techniki, odwiedź stronę *http://jqueryvalidation.org/files/demo/captcha/*.

W tym fragmencie należy zastąpić fragment *nazwa_pola* nazwą sprawdzanego pola, a zamiast tekstu *typ_walidacji* trzeba podać jedną z metod walidacji. Aby połączyć dodane wcześniej metody walidacji pól na hasło z komunikatami specyficznymi dla błędów, dodaj kod wyróżniony pogrubieniem:

```
$('#signup').validate({
  rules: {
    password: {
      required:true,
      rangelength:[8,16]
    },
    confirm_password: {
      equalTo:'#password'
    }
  }, // Koniec obiektu rules.
  messages: {
    password: {
      required: "Wpisz hasło, którego chcesz używać.",
      rangelength: "Hasło musi mieć od 8 do 16 znaków."
    },
    confirm_password: {
      equalTo: "Hasła nie pasują do siebie."
    }
  } // Koniec obiektu messages.
}); // Koniec funkcji validate
```

Wskazówka: Jak widać, zastosowanie tej zaawansowanej metody może wymagać tworzenia wielu literałów obiektowych, a znaczna liczba używanych przy tym nawiasów klamrowych — { } — może pogarszać przejrzystość kodu i utrudniać jego zrozumienie. Dobrym rozwiązaniem, z którego można skorzystać podczas stosowania zaawansowanych metod walidacji przy użyciu wtyczki Validation, jest uważne pisanie kodu, zrezygnowanie z pośpiechu i częste testowanie. Jeśli walidacja nie działa prawidłowo, najprawdopodobniej do naszego kodu wkradła się jakaś literówka; w takim przypadku trzeba ją poprawić przed rozpoczęciem pisania kolejnej reguły. Kiedy napiszemy już wszystkie reguły, które będą prawidłowo działać, możemy przystąpić do dodawania literałów obiektowych z komunikatami o błędach. Także i je warto pisać powoli, dodawać komunikaty jeden po drugim i często przeprowadzać testy. Warto także przeglądać konsolę błędów przeglądarki (patrz strona 51) i sprawdzać, czy na niej nie zostały wyświetlone informacje o błędach.

Określanie stylu komunikatów o błędach

Kiedy wtyczka Validation znajdzie nieprawidłowe pole, wykonuje dwie operacje. Najpierw przypisuje do tego pola klasę, a następnie dołącza do niego znacznik `<label>` z komunikatem o błędzie. Oto kod HTML pola na adres e-mail:

```
<input name="email" type="text" class="required">
```

Jeśli dodasz do strony wtyczkę Validation, a użytkownik spróbuje przesłać formularz bez wypełnienia pola `email`, wtyczka zablokuje proces wysyłania i zmieni kod HTML tego pola przez dodanie nowego znacznika. Zmodyfikowany kod HTML będzie wyglądał następująco:

```
<input name="email" type="text" class="required error">
<label for="email" generated="true" class="error">
To pole jest wymagane.</label>
```

Oznacza to, że wtyczka doda do pola formularza klasę `error`, a także wstawi znacznik `<label>` tej samej klasy, zawierający komunikat o błędzie.

Aby zmienić wygląd komunikatu o błędzie, wystarczy dodać do arkusza odpowiedni styl. Aby na przykład czcionka tekstu wiadomości była pogrubiona i czerwona, należy umieścić w arkuszu poniższy styl:

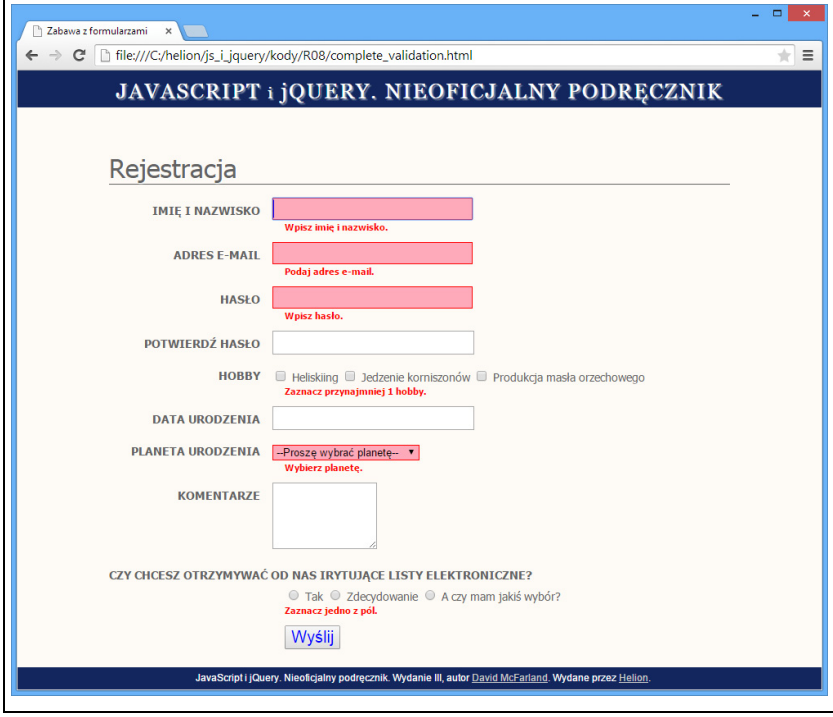
```
label.error {
  color: #F00;
  font-weight: bold;
}
```

Ponieważ wtyczka Validation dodaje klasę `error` także do nieprawidłowych pól formularza, można utworzyć styl, który określa wygląd również tych elementów. Następny styl dodaje czerwone obramowanie wokół pól z błędami:

```
input.error, select.error, textarea.error {
  border: 1px red solid;
}
```

Przykład zastosowania walidacji

W tym przykładzie dodasz do formularza opcje walidacji prostej i zaawansowanej (patrz rysunek 8.7).



Rysunek 8.7. *Nie pozwalaj użytkownikom na przesyłanie niepełnych danych! Dzięki wtyczce Validation biblioteki jQuery możesz mieć pewność, że uzyskasz potrzebne informacje*

Uwaga: Informacje o pobieraniu przykładowych plików znajdziesz na stronie 46.

Prosta walidacja

W tym przykładzie najpierw zastosujesz prostą metodę walidacji za pomocą wtyczki Validation, opisaną na stronie 301. Następnie użyjesz bardziej złożonych technik opartych na metodzie zaawansowanej, omówionej na stronie 305. Zobaczysz, że można swobodnie łączyć oba podejścia w jednym formularzu.

1. **Otwórz w edytorze tekstu plik *validation.html* z katalogu *R08*.**

   Ten plik zawiera formularz z wieloma elementami — polami tekstowymi, polami wyboru, przyciskami opcji i listami rozwijanymi. Dodasz do niego mechanizm walidacji, najpierw jednak należy dołączyć do strony wtyczkę Validation.

2. **W pustym wierszu bezpośrednio pod znacznikiem `<script>`, który dołącza do strony plik *jquery.js*, dodaj następujący kod:**

   ```
   <script src="jquery_validate/jquery.validate.min.js">
   </script>
   ```

   Wtyczka Validation znajduje się w podkatalogu *jquery_validate* w tym samym katalogu, w którym zostały umieszczone przykłady do tego rozdziału.

   Na stronie znajduje się już dodatkowy znacznik `<script>` z funkcją `ready()` biblioteki jQuery. Teraz wystarczy wywołać dla formularza funkcję `validate()`.

3. **W pustym wierszu pod kodem `$(document).ready(function()` wpisz poniższy fragment:**

   ```
   $('#signup').validate();
   ```

   Słowo `signup` w selektorze to identyfikator formularza:

   ```
   <form action="process.html" method="post" name="signup" id="signup">
   ```

   Obiekt jQuery `$('#signup')` pobiera formularz, a funkcja `validate()` wiąże go z wtyczką Validation. Jednak aby uruchomić walidację, należy określić jej reguły. Pole `name` powinno być wymagane i mieć niestandardowy komunikat o błędzie.

4. **Znajdź kod HTML pola `name` — `<input name="name" type="text" id="name">` — i dodaj do niego atrybuty `class` i `title`. Znacznik powinien wyglądać następująco (zmiany wyróżniono pogrubieniem):**

   ```
   <input name="name" type="text" id="name"
   ↪class="required" title="Wpisz imię i nazwisko.">
   ```

   Para `class= required` informuje wtyczkę Validate o tym, że pole to jest wymagane, natomiast atrybut `title` określa komunikat o błędzie, który użytkownik zobaczy, jeśli nie wypełni pola.

5. **Zapisz stronę, otwórz ją w przeglądarce i kliknij przycisk *Wyślij*.**

   Ponieważ pole `name` nie jest wypełnione, obok niego pojawi się komunikat o błędzie (zakreślony na rysunku 8.8).

Rysunek 8.8. *Na razie nie przejmuj się wyglądem komunikatu o błędzie. Na stronie 319 zobaczysz, jak go sformatować*

Gratulacje! Właśnie dodałeś do formularza walidację za pomocą prostej metody opisanej na stronie 302. Następnie należy dodać regułę walidacji dla pola z datą urodzenia.

Uwaga: Jeśli zamiast komunikatu o błędzie zobaczysz stronę z nagłówkiem „Formularz przetworzono", skrypt nie uruchomił walidacji, a formularz został przesłany. Prześledź ponownie kroki od 1. do 4. i upewnij się, że nie zrobiłeś literówki.

6. **Znajdź kod HTML pola z datą urodzenia — `<input name="dob" type="text" id="dob">` — i dodaj do niego atrybuty `class` oraz `title`. Znacznik powinien wyglądać następująco (zmiany wyróżniono pogrubieniem):**

```
<input name="dob" type="text" id="dob" class="date"
↪title="Podaj datę urodzenia w formacie 01/19/2000.">
```

Ponieważ nie dodałeś klasy `required`, pole to jest opcjonalne. Jednak jeśli użytkownik je wypełni, para `class= date` poinformuje wtyczkę o tym, że dane muszą mieć format daty. Atrybut `title` ponownie służy do określenia komunikatu o błędzie, wyświetlanego, jeśli pole zawiera nieprawidłowy tekst. Zapisz stronę i wypróbuj ją w przeglądarce. Wpisz w polu na datę urodzenia dowolny tekst (na przykład **`kjsdf`**) i spróbuj przesłać formularz.

Uwaga: Jeśli naprawdę chcesz, aby internauta musiał wprowadzić datę urodzenia *i* użyć właściwego formatu, dodaj słowo `required` do atrybutu `class`. Pamiętaj o tym, aby oddzielić klasy `date` i `required` spacją:

```
class="date required"
```

Tej samej techniki można użyć do walidacji list rozwijanych (znacznik `<select>`).

7. **Znajdź kod HTML otwierającego znacznika `select` — `<select name="planet" id="planet">` — i dodaj atrybuty `class` oraz `title`, aby znacznik wyglądał tak, jak poniżej (zmiany wyróżniono pogrubieniem):**

```
<select name="planet" id="planet" class=" required"
↪title="Wybierz planetę.">
```

Walidację można dodać do list rozwijanych w taki sam sposób, jak robi się to w przypadku pól tekstowych. Wystarczy podać atrybuty `class` i `title`.

Teraz wypróbujesz zaawansowaną technikę walidacji.

Walidacja zaawansowana

Na stronie 305 dowiedziałeś się, że prosta walidacja nie umożliwia wykonania niektórych operacji, na przykład przypisania różnych komunikatów o błędach do poszczególnych problemów lub określenia liczby wprowadzanych znaków. Dlatego czasem do utworzenia wiadomości i reguł walidacji trzeba użyć zaawansowanej techniki wtyczki Validate.

Zacznij od dodania dwóch reguł walidacji i dwóch różnych komunikatów o błędach do pola `email`.

1. **Znajdź w kodzie JavaScript w początkowej części pliku wywołanie `$('#signup').validate();` i zmodyfikuj je w następujący sposób:**

```
$('#signup')validate({

});  // Koniec funkcji validate
```

Trzeba wpisać otwierający i zamykający nawias klamrowy między nawiasami funkcji `validate()`, dodać pusty wiersz między nowymi nawiasami klamrowymi i dołączyć komentarz języka JavaScript (wskazuje on koniec funkcji `validate()`). Wkrótce dodasz do skryptu wiele nowych nawiasów klamrowych i zwykłych, dlatego zapamiętanie, które z nich są powiązane z konkretnymi instrukcjami, może być trudne. Ten komentarz ułatwia zrozumienie kodu, ale — podobnie jak wszystkie komentarze — nie jest niezbędny.

Następnie należy przygotować strukturę reguł walidacji.

2. **W pustym wierszu dodanym w poprzednim kroku (między nawiasami klamrowymi) wpisz:**

```
rules: {

} // Koniec obiektu rules.
```

Aby kod był bardziej czytelny, warto dodać dwa odstępy przed słowem `rules` i znakiem `}`. Wcięcia pomagają zauważyć, że wyróżnione tak wiersze są częścią funkcji `validate()`.

Ten kod tworzy pusty obiekt, który możesz zapełnić nazwami pól i metod walidacji. Ponadto komentarz języka JavaScript wskazuje koniec obiektu `rules`. Następnie należy dodać reguły dla pola `email`.

3. **Zmodyfikuj wywołanie funkcji `validate()`, aby wyglądało następująco (zmiany wyróżniono pogrubieniem):**

```
$('#signup')validate({
  rules: {
    email: {
      required: true,
      email: true
    }
  } // Koniec obiektu rules.
}); // Koniec funkcji validate.
```

Nowy fragment to następny literał obiektowy języka JavaScript. Pierwsza część, `email`, to nazwa pola z kodu HTML, które chcesz sprawdzać. Dwa następne wiersze określają metody walidacji. Pole jest wymagane (czyli użytkownicy muszą je wypełnić w celu przesłania formularza), a dane muszą mieć format adresu e-mail. „Testuj wcześnie i często" — to zdanie powinno być mottem każdego programisty. Zanim przejdziemy dalej, sprawdź, czy skrypt działa prawidłowo.

4. **Zapisz plik, wyświetl stronę w przeglądarce i spróbuj przesłać formularz.**

Powinieneś zobaczyć domyślny komunikat błędu generowany przez wtyczkę, kiedy brakuje pola wymaganego — „This field is required" (To pole jest wymagane). Kliknij pole i wpisz w nim kilka liter. Wyświetlony komunikat zmieni się — teraz będzie miał następującą postać: „Please enter a valid email address" (Proszę podać prawidłowy adres email; jest to standardowy komunikat generowany przez wtyczkę, gdy użytkownik wpisze tekst niebędący prawidłowym

adresem poczty elektronicznej do pola, które ma zawierać taki adres). Jeśli na stronie nie zostaną pokazane żadne komunikaty o błędach, wyświetl kod strony i porównaj go z podanym w poprzednim punkcie.

Teraz dodasz do pola niestandardowe komunikaty o błędach.

5. **Ponownie przejdź do edytora i wpisz przecinek po zamykającym nawiasie obiektu `rules` (ale przed komentarzem `// Koniec obiektu rules.`), a następnie dodaj poniższy kod:**

```
messages: {

} // Koniec obiektu messages.
```

Ten kod to następny literał obiektowy języka JavaScript — `messages`. Należy w nim podać komunikaty o błędach dołączane do pól formularza. Końcowy komentarz (`// Koniec obiektu messages.`) jest opcjonalny. Teraz trzeba dodać tekst komunikatów o błędach związanych z polem `email`.

6. **Zmodyfikuj wywołanie funkcji `validate()`, aby wyglądało następująco (nowe fragmenty wyróżniono pogrubieniem):**

```
1  $('#signup')validate({
2    rules: {
3      email: {
4        required: true,
5        email: true
6      }
7    }, // Koniec obiektu rules.
8    messages: {
9      email: {
10       required: "Podaj adres e-mail.",
11       email: "To nie jest prawidłowy adres e-mail."
12     }
13   } // Koniec obiektu messages.
14 }); // Koniec funkcji validate.
```

Zapisz stronę i ponownie wyświetl ją w przeglądarce. Spróbuj przesłać formularz bez wypełniania pola z adresem e-mail. Powinien pojawić się komunikat „Podaj adres e-mail.". Teraz wpisz w tym polu dowolny tekst, na przykład „witaj", i spróbuj przesłać formularz. Tym razem powinieneś zobaczyć wiadomość „To nie jest prawidłowy adres e-mail.".

Jeśli zamiast komunikatów o błędach zobaczysz tekst „Formularz przetworzono", w kodzie JavaScript musiał pojawić się błąd. Prawdopodobnie zabrakło przecinka po obiekcie `rules` (wiersz 7.) lub na liście komunikatów dla pola `email` w obiekcie `messages` (wiersz 10.).

Teraz należy dodać reguły walidacji dla dwóch pól na hasło.

7. **Zmodyfikuj obiekt `rules`, aby wyglądał następująco (zmiany wyróżniono pogrubieniem):**

```
1    rules: {
2      email: {
3        required: true,
4        email: true
5      },
6      password: {
7        required: true,
8        rangelength:[8,16]
```

```
    },
    confirm_password: {
      equalTo:'#password'
    }
  }, // Koniec obiektu rules.
```

Nie zapomnij o dodaniu przecinka w wierszu 5. Jest niezbędny do oddzielenia reguł dla pola `email` od zasad dla pola `password`.

Pierwszy zestaw reguł dotyczy pierwszego pola z hasłem. Jest ono wymagane i musi mieć przynajmniej 8, ale nie więcej niż 16 znaków. Druga zasada określa, że zawartość pola z potwierdzeniem hasła musi być taka sama jak pierwszego pola (szczegółowy opis tej reguły znajdziesz na stronie 308).

Wskazówka: W tym przykładzie po każdym kroku warto zapisać i przetestować stronę. Jeśli walidacja przestanie działać, będziesz wiedział, w którym miejscu popełniłeś błąd.

Do nowych reguł trzeba jeszcze przypisać komunikaty o błędach.

8. **Zmodyfikuj obiekt `messages`, aby wyglądał następująco (zmiany wyróżniono pogrubieniem):**

```
  messages: {
    email: {
      required: "Podaj adres e-mail.",
      email: "To nie jest prawidłowy adres e-mail."
    },
    password: {
      required: 'Wpisz hasło.',
      rangelength: 'Hasło musi mieć od 8 do 16 znaków.'
    },
    confirm_password: {
      equalTo: 'Podane hasła nie pasują do siebie.'
    }
  } // Koniec obiektu messages.
```

Nie zapomnij o przecinku w wierszu 5.

Dodawanie reguł i komunikatów o błędach nie powinno już sprawiać Ci problemów. Teraz należy uruchomić walidację pól wyboru i przycisków opcji.

Walidacja pól wyboru i przycisków opcji

Pola wyboru i przyciski opcji zwykle występują w grupach, a proces dodawania walidacji do kilku powiązanych elementów związany jest ze skomplikowanym wyszukiwaniem wszystkich znaczników z grupy. Na szczęście wtyczka Validation automatycznie wykonuje wszystkie skomplikowane operacje i umożliwia szybkie dodanie walidacji do wymienionych pól formularza.

1. **Znajdź kod HTML pierwszego pola wyboru — `<input name="hobby" type="checkbox" id="heliskiing" value="heliskiing">` — i dodaj do niego atrybuty `class` oraz `title` (zmiany wyróżniono pogrubieniem):**

```
<input name="hobby" type="checkbox" id="heliskiing"
value="heliskiing" class="required"
title="Zaznacz przynajmniej jedno hobby.">
```

Zastosowano tu prostą metodę walidacji, omówioną na stronie 302. Możesz też użyć techniki zaawansowanej, aby dołączyć reguły i komunikaty o błędach w funkcji `validate()`, jednak jeśli potrzebna jest tylko jedna zasada i wiadomość, podstawowe rozwiązanie jest prostsze i mniej narażone na błędy.

Tu wszystkie trzy pola wyboru mają tę samą nazwę, dlatego wtyczka Validation traktuje je jak grupę. Oznacza to, że reguła obowiązuje we *wszystkich trzech polach*, choć atrybuty `class` i `title` dodałeś tylko do jednego z nich. Ustawienia te sprawiają, że użytkownik musi wybrać przynajmniej jedno pole przed przesłaniem formularza.

Teraz należy zrobić to samo dla przycisków opcji z dolnej części formularza.

2. **Znajdź kod HTML pierwszego przycisku opcji — `<input type="radio" name="spam" id="yes" value="yes">` — i dodaj do niego atrybuty `class` oraz `title` (zmiany wyróżniono pogrubieniem):**

```
<input type="radio" name="spam" value="yes"
class="required" title="Zaznacz jedno z pól.">
```

Grupa powiązanych przycisków opcji zawsze ma tę samą nazwę (tu jest to `spam`), dlatego choć dodałeś regułę i komunikat o błędzie do tylko jednej kontrolki tego typu, będą one obowiązywać we wszystkich trzech. Ponieważ pole jest wymagane, użytkownik musi wybrać jeden z trzech przycisków opcji, aby wysłać formularz.

3. **Zapisz plik, wyświetl go w przeglądarce i kliknij przycisk *Wyślij*.**

Zwróć uwagę na dziwne zjawisko. Komunikaty o błędach dla pól wyboru i przycisków opcji pojawiają się bezpośrednio po pierwszych kontrolkach, zakreślonych na rysunku 8.9. Co gorsza, wiadomość znajduje się między polem formularza a jego etykietą (na przykład między polem wyboru a napisem „Heliskiing").

Wtyczka Validation umieszcza komunikat o błędzie bezpośrednio po polu formularza, do którego zastosowano regułę walidacji. Zwykle jest to właściwe rozwiązanie. Jeśli komunikat znajduje się po polu tekstowym lub menu (jak we wcześniejszych częściach przykładu), wygląda dobrze. Jednak tu wiadomość należy wyświetlić w innym miejscu, najlepiej pod wszystkimi polami wyboru lub przyciskami opcji.

Na szczęście wtyczka Validation umożliwia kontrolowanie rozmieszczenia komunikatów o błędach. Służy do tego następny literał obiektowy języka JavaScript przekazywany do funkcji `validate()`.

4. **Znajdź dodany wcześniej skrypt walidacji i wpisz przecinek po zamykającym nawiasie klamrowym obiektu `messages` (ale przed komentarzem `// Koniec obiektu messages.`). Wstaw pusty wiersz po obiekcie `messages` i wpisz poniższy kod:**

```
errorPlacement: function(error, element) {
  if ( element.is(":radio") || element.is(":checkbox")) {
    error.appendTo( element.parent());
  } else {
    error.insertAfter(element);
  }
} // Koniec obiektu errorPlacement.
```

Rysunek 8.9. *Wtyczka Validation wyświetla komunikaty o błędach w niewłaściwym miejscu. W przypadku pól wyboru i przycisków opcji wygląda to fatalnie. Aby umieścić wiadomość w innym miejscu, trzeba przekazać funkcji validate() odpowiednie instrukcje*

Wtyczka Validation przyjmuje opcjonalny obiekt `errorPlacement`, zawierający funkcję anonimową (patrz strona 168), która określa lokalizację komunikatu o błędzie. Każdy błąd jest przesyłany przez tę funkcję, gdy zatem chcesz zmienić położenie jedynie wybranych komunikatów, konieczne będzie dodanie logiki warunkowej umożliwiającej identyfikację tych elementów formularza, dla których położenie komunikatów ma zostać zmienione. Przyjmuje ona wiadomość i nieprawidłowy element *formularza*, co pozwala użyć instrukcji warunkowej (patrz strona 93) do sprawdzenia, czy dany znacznik jest przyciskiem opcji lub polem wyboru. Jeśli tak jest, skrypt umieszcza komunikat o błędzie na końcu elementu zawierającego nieprawidłowy znacznik. Na tej stronie grupa pól wyboru znajduje się w znaczniku `<div>`, podobnie jak przyciski opcji. Dlatego można użyć funkcji `appendTo()` biblioteki jQuery (patrz strona 157), aby dodać komunikat o błędzie bezpośrednio przed zamykającym znacznikiem `</div>`.

Kod JavaScript formularza jest już gotowy. Oto kompletny skrypt (wraz z funkcją `$(document).ready()`):

```
$(document).ready(function() {
  $('#signup').validate({
    rules: {
      email: {
        required: true,
        email: true
      },
      password: {
        required: true,
        rangelength:[8,16]
      },
      confirm_password: {equalTo:'#password'},
    }, // Koniec obiektu rules.
    messages: {
      email: {
        required: "Podaj adres e-mail.",
        email: "To nie jest prawidłowy adres e-mail."
      },
      password: {
        required: 'Wpisz hasło.',
        rangelength: 'Hasło musi mieć od 8 do 16 znaków.'
      },
      confirm_password: {
        equalTo: 'Podane hasła nie pasują do siebie.'
      }
    }, // Koniec obiektu messages.
    errorPlacement: function(error, element) {
      if ( element.is(":radio") || element.is(":checkbox")) {
        error.appendTo( element.parent());
      } else {
        error.insertAfter(element);
      }
    } // Koniec obiektu errorPlacement.
  }); // Koniec funkcji validate
}); // Koniec funkcji ready
```

Formatowanie komunikatów o błędach

Na stronie funkcjonuje już walidacja formularza, jednak komunikaty o błędach nie wyglądają zbyt atrakcyjnie. Nie tylko są porozrzucane na stronie, ale ponadto nie wyróżniają się w wystarczającym stopniu. Będą prezentować się dużo lepiej z pogrubioną, czerwoną czcionką, umieszczone pod nieprawidłowymi polami formularza. Wszystkie te modyfikacje można wprowadzić przy użyciu prostego arkusza stylów.

1. **Na początku pliku *validation.html* kliknij pusty wiersz umieszczony pomiędzy otwierającym znacznikiem `<style>` a zamykającym znacznikiem `</style>`.**

   Strona zawiera pusty arkusz stylów, w którym umieścisz swój kod CSS. Podczas tworzenia rzeczywistej witryny taki kod zostałby zapewne umieszczony w zewnętrznym pliku CSS — bądź to w głównym arkuszu stylów używanym także przez inne strony, bądź w specjalnym, wykorzystywanym tylko przez formularze (na przykład, w pliku *forms.css*). Jednak w tym przykładzie, dla zachowania prostoty, style zostaną umieszczone bezpośrednio na stronie.

2. **Dodaj do pliku poniższy kod CSS:**

```
#signup label.error {
  font-size: 0.8em;
  color: #F00;
  font-weight: bold;
  display: block;
  margin-left: 215px;
}
```

Selektor CSS `#signup label.error` wskazuje wszystkie znaczniki `<label>` klasy `error` umieszczone w elemencie o identyfikatorze `signup`. Tu jest to identyfikator formularza, a wtyczka Validation umieszcza komunikaty o błędach w znaczniku `<label>` i dodaje do nich klasę `error` (patrz strona 310). Oznacza to, że ten styl CSS formatuje tylko wiadomości o błędach we wspomnianym formularzu.

Użyte właściwości CSS są całkiem proste. Najpierw styl modyfikuje czcionkę: zmniejsza rozmiar do 0,8 em, zmienia kolor na czerwony i pogrubia. Instrukcja `display: block` informuje przeglądarkę, że ma traktować dany znacznik `<label>` jak element blokowy. Oznacza to, że zamiast umieszczać komunikat o błędzie *obok* pola, przeglądarka potraktuje go jak niezależny akapit ze znakami przełamania wiersza na początku i na końcu. Ponadto trzeba dodać lewy margines, aby wiadomość pojawiała się równo z polami formularza (które mają wcięcie 215 pikseli względem lewej krawędzi głównego obszaru strony).

Aby w jeszcze wyraźniejszy sposób wyróżnić pola, w których podczas weryfikacji danych natrafiono na problemy, możesz utworzyć reguły CSS modyfikujące wygląd konkretnych pól formularza.

3. **Dodaj ostatnią regułę:**

```
#signup input.error, #signup select.error {
  background: #FFA9B8;
  border: 1px solid red;
}
```

Ta reguła powoduje wyróżnienie nieprawidłowych pól formularza przez dodanie do nich koloru tła i czerwonego obramowania wokół ich krawędzi.

To już wszystko. Zapisz plik i wyświetl stronę *validation.html* w przeglądarce, aby sprawdzić, jaki wpływ style CSS mają na komunikaty o błędach (aby zobaczyć zmiany, prawdopodobnie będziesz musiał wcisnąć w przeglądarce przycisk *Odśwież*).

Formularz powinien wyglądać tak, jak ten z rysunku 8.7. Jego gotową wersję znajdziesz w pliku *complete_validation.html* w katalogu *R08*.

III

CZĘŚĆ

Wprowadzenie do biblioteki jQuery UI

9
ROZDZIAŁ

Rozbudowa interfejsu użytkownika

Przekonałeś się, jak przy użyciu biblioteki jQuery i niewielkich fragmentów kodu JavaScript można rozszerzać możliwości formularzy, obrazów oraz odnośników. Nauczyłeś się dodawać do swoich stron animacje i tworzyć proste elementy interfejsu użytkownika, takie jak wysuwający się formularz do logowania (patrz strona 216) czy też animowany pasek z miniaturkami zdjęć (patrz strona 225). Być może teraz jesteś gotów na poznanie bardziej złożonych elementów interfejsu użytkownika, takich jak okna dialogowe, zestawy kart oraz etykietki ekranowe. Mógłbyś się nauczyć tworzenia takich rozwiązań od zera, jednak — zgodnie z tym, czego się dowiedziałeś na stronie 265 — istnieje wiele wtyczek jQuery, które już je implementują. jQuery UI jest wtyczką jQuery rozwiązującą bardzo wiele problemów związanych z obsługą interfejsu użytkownika, w której cały niezbędny kod został zgromadzony w jednym, łatwym w użyciu pakiecie.

Czym jest jQuery UI?

jQuery UI (*http://jqueryui.com/*) jest zaawansowaną wtyczką jQuery oraz jej siostrzanym projektem (patrz rysunek 9.1). Udostępnia obszerny zestaw efektów, sposobów interakcji z użytkownikiem oraz elementów interfejsu użytkownika (powszechnie nazywanych *widżetami*), które upraszczają proces tworzenia interaktywnych aplikacji internetowych. W dalszej części książki, w rozdziale 14., użyjesz jQuery UI oraz własnego kodu JavaScript do opracowania prostej (lecz użytecznej) aplikacji internetowej.

Rysunek 9.1. *Witryna jQuery UI pozwala na pobranie wtyczki i tworzenie własnych „tematów" graficznych określających postać elementów interfejsu użytkownika budowanych przy użyciu jQuery UI. Można na niej znaleźć także przykłady dostępnych widżetów, efektów animacji oraz sposobów interakcji, a dokumentacja umieszczona pod adresem http://api.jquery.com/ pozwala poznać zasady ich działania*

Wtyczka jQuery UI składa się z wielu różnych fragmentów, które można podzielić na trzy kategorie.

- **Widżety**. Widżet to fragment kodu JavaScript tworzący użyteczny element interfejsu użytkownika. Przykładowo widżet `Dialog` pozwala na tworzenie wyskakujących okien dialogowych — przypominają one niestandardowe okienka informacyjne (patrz strona 330), lecz zapewniają pełną kontrolę nad swoim wyglądem i działaniem. Można ich używać, by na przykład wyświetlić formularz do logowania bądź regulamin witryny. Takie okna dialogowe doskonale nadadzą się do wyświetlania ważnych informacji, za każdym razem gdy użytkownik wejdzie na witrynę, lub informacji o zdjęciu, kiedy użytkownik wskaże je myszą.

  Kolejnym przykładem może być widżet kalendarza, który zapewnia użytkownikom możliwość łatwego i wygodnego wybierania dat. Za jego pomocą można wyświetlić na stronie okienko z kalendarzem, w którym użytkownik klika datę. Z powodzeniem można by go zastosować podczas tworzenia formularza rezerwacji na stronie biura podróży („Wybierz początek okresu rezerwacji") bądź też jako sposób poruszania się po liście nadchodzących zdarzeń.

Wtyczka jQuery UI udostępnia wiele takich widżetów, a kilka z nich poznasz w tym rozdziale oraz w następnym.

- **Sposoby interakcji.** Wtyczka jQuery UI zawiera także kilka bardzo użytecznych narzędzi pozwalających użytkownikom na prowadzenie interakcji ze stroną. Pozwala na przykład tworzyć elementy, które można przeciągać. Wyobraź sobie aplikację sklepu internetowego, w której użytkownik może dosłownie przeciągnąć wybrany produkt do swojego koszyka, albo internetową wersję gry w warcaby, w której gracze, zamiast klikać, przeciągają pionki. Inny mechanizm jQuery UI pozwala na tworzenie elementów, których wielkość da się zmieniać —możesz na przykład wyświetlić okno dialogowe zawierające formularz do napisania wpisu na blogu. Użytkownik przeglądający taką stronę może przeciągnąć wierzchołek pola, by go powiększyć lub zmniejszyć. Innymi słowy, zwykły znacznik `<div>` może działać jak okno przeglądarki, dysponujące elementami do zmiany wielkości. Biblioteka jQuery UI oferuje kilka takich sposobów interakcji, które zostały opisane w rozdziale 12.
- **Efekty.** jQuery UI udostępnia także kilka sposobów animacji, takich jak stopniowe pojawienie się (ang: *fade in*, patrz strona 214), zaniknięcie (ang: *fade out*, patrz strona 214), zsunięcie (ang. *slide down*, patrz strona 216) oraz bardziej ogólną funkcję `animate()`. Jednak dostępnych sposobów animacji jest znacznie więcej — jQuery UI pozwala na animowanie zmian koloru, zmian stylów CSS i tak dalej. Informacje na ten temat zostały podane w dalszej części książki, od strony 461.

Dlaczego warto używać jQuery UI?

Można się zastanawiać, dlaczego warto używać akurat jQuery UI, skoro istnieją tysiące innych wtyczek jQuery. Nie tylko w jQuery UI można znaleźć wiele wymyślnych wtyczek pozwalających na tworzenie etykietek ekranowych, kart czy też okien dialogowych. Można znaleźć wtyczki oferujące dokładnie to samo, co jQuery UI, a nawet znacznie więcej. Więcej informacji na temat tych rozwiązań podano na stronie 327. Mimo to, istnieje kilka powodów przemawiających za tym, że jQuery UI jest świetnym rozwiązaniem. Oto one.

- Wtyczka ta jest rozwijana przez Fundację jQuery (*https://jquery.org*). Jest to organizacja niedochodowa, powołana w celu promowania rozwoju biblioteki jQuery, jQuery UI oraz kilku innych projektów. Innymi słowy, jQuery i jQuery UI są jak rodzeństwo, a zespoły odpowiedzialne za ich rozwój ściśle współpracują; jeśli zatem zmieni się biblioteka jQuery, zmiany są bardzo szybko uwzględniane także w jQuery UI.
- jQuery UI jest kompletnym pakietem. Jeśli byłoby trzeba, można by, element po elemencie, zebrać zestaw wtyczek powielających wszystkie możliwości jQuery UI. Jednak w takim przypadku musielibyśmy używać kilkunastu wtyczek napisanych przez różne osoby i wymagających kilkunastu różnych plików CSS oraz JavaScript. Zapanowanie nad tymi wszystkimi wtyczkami byłoby dosyć czasochłonnym wyzwaniem. Natomiast wtyczka jQuery UI składa się z jednego pliku JavaScript oraz dwóch plików CSS. Gdy pojawią się jakiekolwiek zmiany, aktualizacja tych trzech plików będzie szybka i łatwa.

UWAGA NA WTYCZKI

Co zamiast jQuery UI?

- Wtyczka jQuery UI nie jest jedyną biblioteką ułatwiającą tworzenie interfejsu użytkownika. Poniżej opisano kilka najpopularniejszych rozwiązań, które mogą być zamiennikami. **Kendo UI** (*http://www.telerik.com/kendo-ui*) jest kompletnym zestawem wtyczek przeznaczonych do tworzenia aplikacji internetowych (oraz mobilnych). Zawiera niektóre możliwości biblioteki jQuery UI, na przykład kalendarz do wybierania dat oraz etykiety ekranowe, lecz także wiele bardziej zaawansowanych rozwiązań, takich jak narzędzia do wizualizacji danych pozwalające na tworzenie różnego typu wykresów i diagramów. Podobnie jak jQuery, także Kendo UI udostępnia narzędzie do tworzenia tematów graficznych oraz obszerną dokumentację. Niestety, w odróżnieniu od jQuery UI, która jest darmowa, Kendo UI jest produktem komercyjnym, którego cena waha się od 399 do 999 USD.
- **Wijmo UI** (*http://wijmo.com/*) jest zestawem zaawansowanych widżetów do tworzenia interfejsów użytkownika. Bazuje na jQuery oraz jQuery UI i obejmuje ponad 40 widżetów, wśród których są wykresy, siatki, arkusze kalkulacyjne i tak dalej.

  Jest to doskonały zestaw wtyczek, dostarczający wszystko, czego aplikacje internetowe mogą potrzebować (a nawet więcej), i działa świetnie zarówno na klasycznych komputerach, jak i na urządzeniach mobilnych. Jednak ze względu na cenę, która waha się od 495 do 1195 USD *na programistę*, Wijmo UI jest narzędziem, na które mogą sobie pozwolić wyłącznie firmy z dużymi budżetami.
- **jQWidgets** (*http://www.jqwidgets.com/*) jest kolejnym zestawem wtyczek dysponującym własnym narzędziem do tworzenia tematów graficznych oraz obszernym zestawem widżetów, takich jak tabele danych, siatki, suwaki, okno do wyboru koloru i wiele innych. Podobnie jak pozostałe wymienione tu biblioteki, także i ta jest produktem komercyjnym, a cena zaczyna się od 199 USD.

- Zapewnia ona jednolity wygląd. Wszystkie widżety jQuery UI mają podobny, spójny wygląd. Panele kart są podobne do okien dialogowym oraz kalendarza, dzięki czemu nie trzeba spędzać długich godzin na dostosowywaniu wyglądu wielu różnych wtyczek, by sprawiały wrażenie, że faktycznie należą do jednej witryny. Co więcej, narzędzie *ThemeRoller* (opisane bardziej szczegółowo w rozdziale 11.) daje możliwość wybierania czcionek, kolorów oraz innych aspektów wyglądu widżetów jQuery UI. Za jego pomocą znacznie łatwiej można dostosować wygląd widżetów jQuery UI do schematu kolorów, czcionek i ogólnego wyglądu naszej istniejącej witryny.
- jQuery UI to projekt, który ma bardzo dobre wsparcie. Wiele wtyczek jQuery powstało jako efekt pracy i miłości jednego programisty, ewentualnie dwóch, a w najlepszym przypadku trzech. Jeśli stracą oni zainteresowanie projektem, znajdą nową pracę albo pójdą do klasztoru, ich wtyczki już nigdy nie będą aktualizowane, a jakiekolwiek błędy odnalezione w ich kodzie nie zostaną poprawione. Natomiast projektem jQuery UI zajmuje się bardzo wiele osób, dlatego można mieć pewność, że jeszcze bardzo długo będzie aktywnie rozwijany. (A konkretnie, jak wiele osób się nich zajmuje? Gdyby zajrzeć na listę osób zaangażowanych w prace nad projektem — dostępną na stronie *https://github.com/jquery/jquery-ui/blob/master/AUTHORS.txt* — można się przekonać, że jest to ponad 270 osób).

Stosowanie jQuery UI

Witrynę poświęconą bibliotece jQuery UI można znaleźć pod adresem *http://jqueryui.com/*. Na stronie głównej poczesne miejsce zajmuje ramka zawierająca odnośniki pozwalające na pobranie biblioteki (patrz rysunek 9.1). Nie zwracaj uwagi na odnośniki *Quick Download* — są one przeznaczone dla programistów, którzy chcą pracować na kodzie biblioteki lub szczegółowo go analizować. (Jeśli jednak klikniesz ten odnośnik, pobierzesz archiwum zawierające kilkanaście plików używanych podczas tworzenia jQuery UI, a kilka z nich służy do automatyzacji procesu przygotowywania biblioteki do użycia i są zupełnie nieprzydatne dla osób, które chcą jedynie korzystać z biblioteki na swojej stronie).

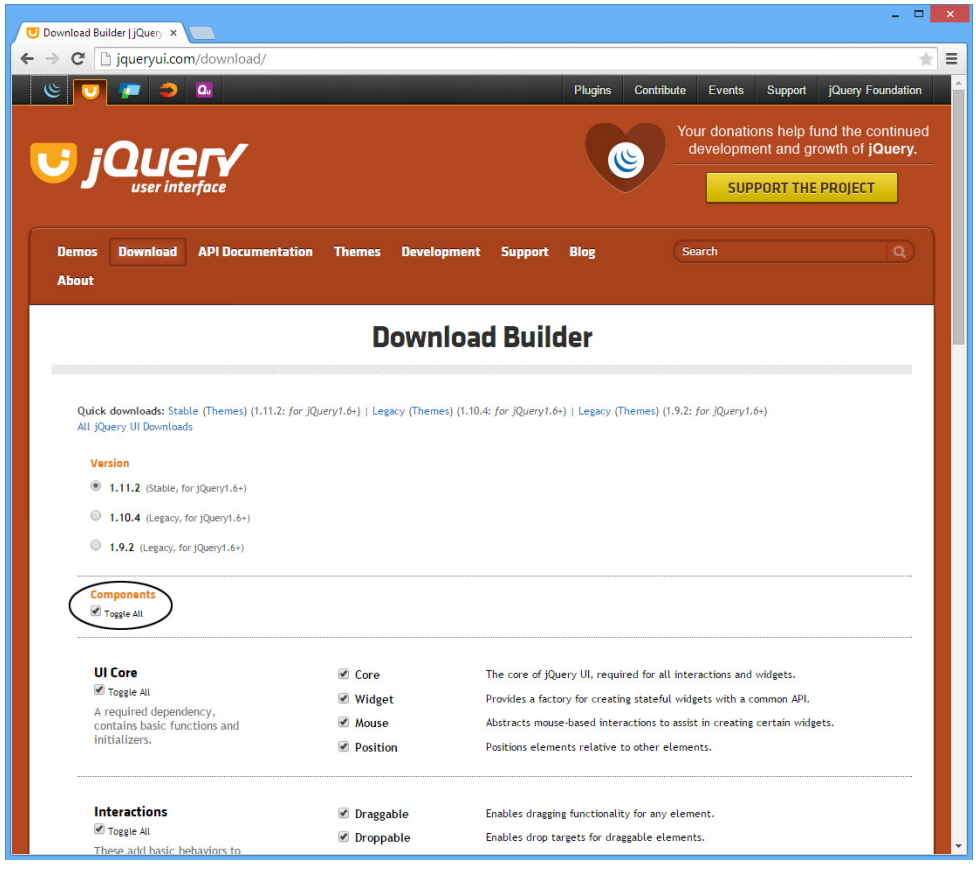
Kliknij zatem przycisk *Custom Download* lub opcję *Download* umieszczoną w pasku nawigacyjnym w górnej części strony. W obu przypadkach zostanie wyświetlona strona *Download Builder* (patrz rysunek 9.2) pozwalająca na wybranie komponentów, których chcesz używać, i odrzucenie tych, które nie będą potrzebne. Możesz na przykład dojść do wniosku, że widżety paska postępów, suwaka oraz pola do edycji wartości liczbowych (ang. *spinner*) nie będą potrzebne; wtedy wystarczy usunąć zaznaczenie z pól wyboru umieszczonych przy ich nazwach, a nie zostaną dołączone do pobieranej wersji biblioteki. Wybierając tylko te widżety, które będą naprawdę niezbędne, możesz uzyskać możliwie najmniejszą wersję biblioteki.

Poniżej listy dostępnych komponentów znajduje się sekcja o nazwie *Theme* (temat graficzny). Można w niej wybrać różne tematy graficzne, które mogą być stosowane w jQuery UI, a nawet przejść do narzędzia *ThemeRoller* pozwalającego na wybranie własnych kolorów, czcionek oraz określenie innych aspektów wyglądu jQuery UI (w tym celu wystarczy kliknąć odnośnik *design a custom theme*[1]). Więcej informacji o tematach graficznych oraz ich tworzeniu można znaleźć w rozdziale 11.

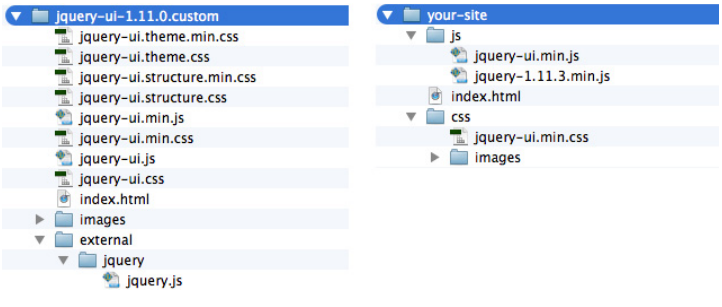
Aby pobrać pliki jQuery UI, wystarczy kliknąć przycisk *Download* umieszczony na samym dole strony *Download Builder*. Spowoduje to pobranie pliku ZIP, zawierającego katalog o nazwie, takiej jak *jquery-ui-1.11.1.custom*. Wewnątrz niego znajdują się dwa podkatalogi (przedstawione z lewej strony rysunku 9.3). Nas interesuje jedynie katalog *images*, zawierający obrazy używane przez jQuery UI. Katalog *external* można pominąć: zawiera on kopię biblioteki jQuery, która i tak zapewne już wcześniej została pobrana i dodana do tworzonej witryny.

Aby zastosować bibliotekę jQuery UI, potrzebny będzie plik JavaScript, zawierający cały kod wymagany do tworzenia widżetów, efektów i zapewniania możliwości interakcji z użytkownikiem. Dodatkowo potrzebny będzie plik CSS, który określi wygląd widżetów oraz efektów tworzonych przez jQuery UI. Jednak, jak widać na rysunku 9.3, takich plików CSS i JavaScript jest kilka, a zatem należy wiedzieć, które z nich wybrać.

[1] zaprojektuj niestandardowy temat — *przyp. tłum.*



Rysunek 9.2. *Strona do przygotowywania wersji biblioteki jQuery UI dostosowanej do potrzeb. Można w niej wybrać tylko te widżety, sposoby interakcji oraz efekty, które będą potrzebne. W tym celu wystarczy usunąć znaczniki pól wyboru przy nazwach tych elementów biblioteki, których nie planujemy używać. Jeśli chcemy wybrać tylko kilka z dostępnych elementów, można kliknąć pole wyboru Toggle All (przełącz wszystkie; zakreślone na rysunku), aby usunąć zaznaczenie wszystkich pól, a następnie zaznaczyć tylko te, które nas interesują. Niektóre widżety są zależne od innych elementów biblioteki — na szczęście strona jest na tyle inteligentna, że sama zaznaczy takie dodatkowe, niezbędne elementy. Jeśli na przykład usuniemy zaznaczenie wszystkich elementów, a następnie wybierzemy widżet Accordion, automatycznie zostaną zaznaczone także pola wyboru Core oraz Widget, gdyż są wymagane do działania wybranego widżetu*



Rysunek 9.3. *Archiwum pobrane z witryny jQuery UI zawiera wiele plików, a większość z nich nie będzie potrzebna (z lewej). Pliki JavaScript i CSS, w których nazwach nie występuje fragment „min”, nie są zminimalizowane: możesz z nich skorzystać, jeśli chcesz dowiedzieć się, jak działają kody CSS oraz JavaScript wtyczki*

Zgodnie z informacjami zamieszczonymi na stronie 139, wszystkie pliki biblioteki zawierające w nazwie litery „min", takie jak *jquery-ui.min.js*, to pliki „zminimalizowane", co oznacza, że w celu zoptymalizowania wielkości usunięto z nich wszystkie niepotrzebne znaki odstępów oraz przeprowadzono inne zmiany. Właśnie te zminimalizowane pliki najlepiej nadają się do wykorzystania na witrynie, gdyż zapewnią najkrótszy czas pobierania strony. Jednak proces minimalizacji sprawia, że zawartość pliku staje się nieczytelna, dlatego też wprowadzanie zmian w takich plikach jest niemożliwe. Zazwyczaj możliwość czytania i analizy kodu JavaScript biblioteki jQuery UI nie jest potrzebna, chyba że zależy nam na zrozumieniu sposobów jej działania. Oznacza to, że na witrynie warto używać pliku *jquery-ui.min.js*.

Nieco większym problemem może być wybór pliku CSS — w pobranym archiwum ZIP jest ich aż sześć! Nam potrzebny będzie plik *jquery-ui.min.css*, gdyż zawiera cały kod CSS niezbędny do działania jQuery UI. Optymalny sposób organizacji plików jQuery UI został przedstawiony na rysunku 9.3, po prawej stronie.

Uwaga: Dlaczego dostępnych jest wiele plików CSS? Oprócz *jquery-ui.min.css*, w pobranym archiwum umieszczone są także pliki *jquery-ui.theme.min.css* oraz *jquery-ui structure.min.css*. Plik, w którego nazwie występuje słowo „structure", zawiera kod CSS określający „strukturę" widżetów jQuery UI, czyli takie informacje jak rozmieszczenie elementów na stronie; z kolei plik „theme" zawiera wyłącznie informacje o używanych kolorach, czcionkach, ich wielkości, o wypełnieniach oraz innych wizualnych aspektach widżetów. Innymi słowy, aby uzyskać ten sam efekt, który daje użycie pliku *jquery-ui.min.css*, konieczne byłoby dołączenie do strony obu pozostałych arkuszy stylów. To za dużo zachodu.

Dodawanie jQuery UI do strony

Pamiętaj, że jQuery UI jest zwyczajną wtyczką biblioteki jQuery, zatem odnoszą się do niej podstawowe zasady stosowania wtyczek przedstawione na stronie 265. Oznacza to, że najpierw należy dołączyć arkusz stylów CSS, następnie plik biblioteki jQuery, wprowadzić niezbędne zmiany w kodzie HTML strony i w końcu wywołać odpowiednią funkcję wtyczki. Poniżej znajdziesz szczegółowy opis tych czynności, zamieszczony, by zebrać je i przedstawić w jednym miejscu.

1. **Pobierz wtyczkę jQuery UI zgodnie z informacjami podanymi w poprzedniej części rozdziału.**

   Kiedy już pobierzesz pliki na dysk, musisz przenieść zarówno je, jak i katalog — czyli wszystkie pliki niezbędne do działania biblioteki jQuery — do katalogu tworzonej witryny, co pokazano na rysunku 9.3. Najpierw umieść plik *jquery-ui.min.css* oraz katalog *images* w katalogu zawierającym arkusze stylów; następnie przenieś plik *jquery-ui.min.js* do katalogu z plikami JavaScript. Zawsze używaj plików zminimalizowanych, czyli tych, w których nazwach znajduje się fragment „min". Ich mniejszy rozmiar oznacza, że będą szybciej pobierane. Do korzystania z jQuery UI potrzebujesz jedynie plików *jquery-ui.min.js*, *jquery-ui.min.css* oraz katalogu *images*. Ten ostatni należy umieścić w tym samym katalogu, w którym wcześniej umieściłeś plik CSS (patrz rysunek 9.3, po prawej stronie).

2. **Dołącz arkusz stylów CSS do strony w następujący sposób:**

```
<link href="css/jquery-ui.min.css" rel="stylesheet">
```

   Co więcej, ten plik warto dołączyć do strony *przed* znacznikiem dołączającym arkusz stylów witryny:

```
<link href="css/jquery-ui.min.css" rel="stylesheet">
<link href="css/site.css" rel="stylesheet">
```

   Dzięki temu, gdybyś uznał za niezbędne wprowadzenie drobnych zmian w temacie graficznym jQuery, mógłbyś umieścić odpowiednie reguły CSS w arkuszu stylów witryny.

   Ogólnie rzecz biorąc, nie warto modyfikować arkusza stylów jQuery UI, gdyż może się okazać, że w przyszłości będzie trzeba go zastąpić nowszym — dostarczonym wraz z nowszą wersją wtyczki. (Więcej informacji na temat przygotowywania i stosowania tematów graficznych jQuery UI można znaleźć w rozdziale 11.).

3. **Do strony dołącz pliki biblioteki jQuery oraz jQuery UI.**

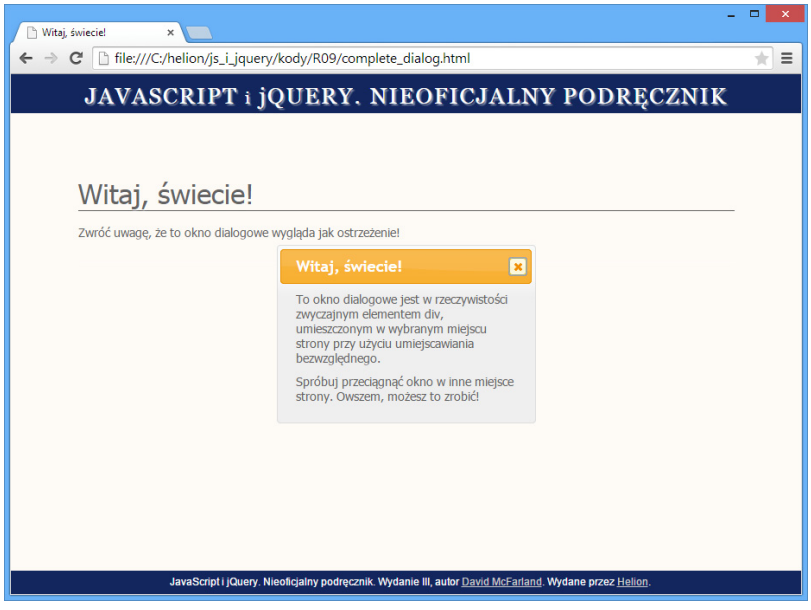
```
<script src="js/jquery-1.11.0.min.js"></script>
<script src="js/jquery-ui.min.js"></script>
```

   Wtyczka jQuery UI nie będzie działać bez biblioteki jQuery, dlatego koniecznie musisz się upewnić, że jej plik został dołączony przed plikiem JavaScript jQuery UI.

Kiedy już umieścisz wszystkie niezbędne pliki biblioteki jQuery UI na witrynie i dołączysz do jej stron, będziesz mógł skorzystać z jej możliwości. Ponieważ każdy efekt, sposób interakcji z użytkownikiem oraz widżet są inne, nie można podać jednego zestawu instrukcji opisującego sposób ich użycia. W dalszej części rozdziału zostały przedstawione niektóre spośród najczęściej używanych widżetów jQuery UI.

Wyświetlanie komunikatów przy użyciu okien dialogowych

Standardowe okno dialogowe przeglądarek WWW ma bardzo destrukcyjny charakter (patrz rysunek 1.3): jego wygląd w niczym nie przypomina projektu graficznego strony, a używanej w nim czcionki, jej wielkości oraz koloru nie można zmieniać. Co więcej, dodatkowo mogą być w nim prezentowane komunikaty, których nie chcemy wyświetlać; przykładowo przeglądarka Chrome dodaje komunikat „Zapobiegaj wyświetlaniu dodatkowych okien dialogowych na tej stronie". Na szczęście jQuery UI udostępnia widżet okna dialogowego pozwalający na budowanie własnych okien dialogowych, takich jak przedstawione na rysunku 9.4. Do takich okien dialogowych można dodawać tekst, formularze oraz obrazy, nadawać im wygląd odpowiadający projektowi witryny, a nawet określać czynności, które będą wykonywane podczas interakcji z użytkownikiem.



Rysunek 9.4. *Widżet Dialog jQuery UI pozwala na bardzo łatwe tworzenie okien dialogowych odpowiadających swoim wyglądem projektowi witryny*

Stosowanie okien dialogowych jQuery UI jest zaskakująco proste (podobnie zresztą, jak korzystanie z większości widżetów i możliwości tej biblioteki).

1. **Dołącz do strony pliki CSS i JavaScript jQuery UI, zgodnie z informacjami podanymi wcześniej — na stronie 329.**

   Od czegoś trzeba zacząć!

2. **Dodaj do treści strony znacznik `<div>` zawierający komunikat, który chcesz wyświetlić w oknie dialogowym, oraz atrybut `title` zawierający tytuł okna.**

   Przykładowo poniżej został zamieszczony fragment kodu HTML użyty do utworzenia okna dialogowego przedstawionego na rysunku 9.4.

```
<div id="hello" title="Witaj, świecie!">
  <p>To okno dialogowe jest w rzeczywistości zwyczajnym elementem
     div, umieszczonym w wybranym miejscu strony przy użyciu
     umiejscawiania bezwzględnego.</p>
  <p>Spróbuj przeciągnąć okno w inne miejsce strony.
    Owszem, możesz to zrobić!</p>
</div>
```

   Ponieważ musisz nakazać jQuery UI przekształcić ten znacznik `<div>` w okno dialogowe, zatem potrzebujesz jakiegoś sposobu, by je zidentyfikować. Dobrym rozwiązaniem jest dodanie identyfikatora, takiego jak `id= hello` .

Uwaga: Do tworzenia okien dialogowych wcale nie trzeba używać znaczników `<div>`. Doskonale do tego celu nada się każdy element blokowy, na przykład `<article>` lub `<p>`.

3. **Umieść na stronie wywołanie funkcji `$(document).ready()`:**

```
$(document).ready(function() {

}); // Koniec funkcji ready.
```

4. **Użyj jQuery do wybrania elementu `<div>`, a następnie wywołaj funkcję `dialog()`:**

```
$(document).ready(function() {
  $('#hello').dialog();
}); // Koniec funkcji ready.
```

   Zastosowałeś tu wywołanie `$('#hello')`, gdyż właśnie taki identyfikator nadałeś znacznikowi `<div>` w kroku 2., choć możesz także użyć dowolnego innego sposobu wybierania elementów udostępnianego przez jQuery (patrz strona 147), by pobrać odpowiedni element strony i przekształcić go w okno dialogowe.

Wykonanie powyższych czynności sprawi, że okno dialogowe zostanie wyświetlone natychmiast po wczytaniu strony. Takie rozwiązanie jest dobre, jeśli chcemy wyświetlić użytkownikowi jakiś pilny komunikat, na przykład „Witryna zostanie wyłączona w celach konserwacji dziś w godzinach od 3:00 do 4:00 rano", bądź też chcemy wyświetlić reklamę, zanim użytkownik będzie mógł przeczytać zawartość witryny. W następnej części rozdziału dowiesz się, w jaki sposób można początkowo ukryć okno dialogowe i wyświetlić je dopiero w odpowiedzi na jakieś zdarzenie. Jednak najpierw spróbuj utworzyć swoje pierwsze okno dialogowe.

Miniprzykład — tworzenie okna dialogowego

Teraz, kiedy już wiesz, jak działa widżet okna dialogowego, spróbuj go wypróbować w praktyce — dodaj do strony okno dialogowe, które będzie wyświetlane bezpośrednio po jej wczytaniu.

Uwaga: Informacje dotyczące sposobu pobierania przykładów można znaleźć na stronie 46.

1. **W edytorze tekstów otwórz plik *hello_world.html* umieszczony w katalogu *R09*.**

   Plik zawiera już znaczniki dołączające bibliotekę jQuery oraz szkielet wywołania funkcji `$(document).ready()` (patrz strona 190), jednak będziesz musiał dołączyć pliki CSS oraz JavaScript biblioteki jQuery UI.

2. **Do sekcji nagłówka strony dodaj poniższe wiersze kodu wyróżnione pogrubioną czcionką:**

```
<link href="../_css/jquery-ui.min.css" rel="stylesheet">
<link href="../_css/site.css" rel="stylesheet">
<script src="../_js/jquery.min.js"></script>
<script src="../_js/jquery-ui.min.js"></script>
```

   Zwróć uwagę, że arkusz stylów jQuery UI dołączyłeś *przed* arkuszem *site.css*, natomiast plik JavaScript *za* plikiem biblioteki jQuery. Teraz nadszedł czas, by dodać kod HTML, który stanie się oknem dialogowym.

3. **Odszukaj pusty wiersz umieszczony bezpośrednio poniżej komentarza `<!-- Tutaj dodaj kod HTML okna dialogowego. -->` i wpisz w nim:**

```
<div id="hello" title="Witaj, świecie!">
  <p>Okienko dialogowe jQuery UI.</p>
</div>
```

   A teraz przekształcisz ten znacznik `<div>` w okno dialogowe.

4. **W pustym wierszu w wywołaniu funkcji `$(document).ready()` wpisz:**

```
$(document).ready(function() {
  $('#hello').dialog();
}); // Koniec funkcji ready.
```

   To wywołanie pobiera znacznik `<div>` dodany w poprzednik kroku i zamienia go w okno dialogowe. To *naprawdę* jest takie proste!

5. **Zapisz stronę i wyświetl ją w przeglądarce.**

   Na stronie zostanie wyświetlone okno dialogowe. Możesz zmieniać jego położenie — wystarczy wskazać jego pomarańczowy pasek tytułu, wcisnąć lewy przycisk myszy i przeciągnąć okno w inne miejsce. Możesz nawet zmienić wielkość okna — wystarczy przeciągnąć dowolny z jego wierzchołków. Kiedy ponownie umieścisz wskaźnik myszy w obszarze paska tytułu, zauważysz, że wskaźnik zmieni kształt na czterokierunkową strzałkę. To nie jest standardowe zachowanie przeglądarki WWW — to jeden z wielu drobnych efektów tworzonych przez jQuery UI.

   Kompletną wersję tego przykładu znajdziesz w pliku *complete_hello_world.html*.

Określanie właściwości okna dialogowego

Okna dialogowe udostępniają wiele właściwości, takich jak wysokość, szerokość, sposób animacji wyświetlania i ukrywania okna, które możemy określać, przekazując w wywołaniu funkcji `dialog()` literał obiektowy z odpowiednimi parami nazwa – wartość. Zgodnie z informacjami podanymi na stronie 165, literały obiektowe to grupa par nazwa – wartość, zapisanych wewnątrz nawiasów klamrowych. Oto przykład takiego literału:

```
{
  name : 'Dave',
  awesomeAuthor : true
}
```

Literał obiektowy zawierający opcje, które wtyczka jest w stanie zrozumieć, jest przekazywany w wywołaniu funkcji `dialog()`. Załóżmy na przykład, że nie chcesz, by użytkownicy mogli przesuwać okno i zmieniać jego wielkość. W tym celu wystarczy, że przekażesz w wywołaniu funkcji `dialog()` literał obiektowy z dwiema opcjami, tak jak na poniższym przykładzie:

```
$('#hello').dialog({
  draggable : false,
  resizable : false
});
```

Widżet `Dialog` utworzono w taki sposób, aby uniemożliwiał przeciąganie okna dialogowego, jeśli właściwości `draggable` zostanie przypisana wartość `false`, oraz uniemożliwiał zmianę wielkości okna, jeśli właściwości `resizable` zostanie przypisana wartość `false`. Poniżej przedstawionych zostało kilka najbardziej przydatnych opcji okien dialogowych.

- **`draggable`.** Właściwości tej można przypisać wartość `false`, aby unieruchomić okno dialogowe i uniemożliwić użytkownikom jego przesuwanie. (Jeśli chcesz zapewnić użytkownikom możliwość przesuwania okna, nie musisz robić nic szczególnego — to standardowe zachowanie okien dialogowych jQuery UI).

- **`resizable`.** Właściwości należy przypisać wartość `false`, aby użytkownicy nie mogli zmieniać wielkości okna dialogowego. (Także w tym przypadku, jeśli chcesz, by użytkownicy mogli zmieniać wielkość okna, nie musisz nic robić — okna domyślnie zapewniają tę możliwość).
- **`height` oraz `width`.** Standardowo jQuery UI wyświetla okno dialogowe na tyle duże, by zmieściła się w nim cała prezentowana zawartość. Jednak to domyślne działanie można zmienić — wystarczy przekazać dokładne wymiary okna wyrażone w pikselach. Aby na przykład utworzyć okno dialogowe o szerokości 600 pikseli i 400 wysokości, wystarczyłoby przekazać następujące właściwości:

  ```
  width: 600,
  height: 400
  ```

  Określając wymiary okna, można używać wyłącznie pikseli (inne sposoby określania długości, takie jak wartości procentowe bądź jednostki *em*, nie są obsługiwane), przy czym do liczb nie należy dodawać liter `px`, które zazwyczaj są zapisywane za wymiarami podawanymi w regułach stylów. Jeśli określisz wymiary okna, lecz jego zawartość będzie na tyle duża, że się w nim cała nie zmieści, jQuery UI wyświetli wewnątrz okna pasek przewijania — użytkownik będzie mógł z niego skorzystać, by wyświetlić całą zawartość okna. (Tak zazwyczaj działają okna dialogowe zawierające regulaminy korzystania z witryny lub usługi, które trzeba w nieskończoność przewijać, by przeczytać wszystkie paragrafy).

  Nie trzeba podawać wartości obu tych właściwości. Może Ci na przykład zależeć, by okno miało odpowiednią szerokość, lecz jego wysokość nie będzie mieć znaczenia. W takim przypadku wystarczy przekazać samą właściwość `width`.
- **Minimalna szerokość i wysokość.** Określając wartości właściwości `minWidth` oraz `minHeight`, można nakazać utworzenie okna dialogowego o określonych wymiarach minimalnych. Aby na przykład okno dialogowe nie było węższe od 600 pikseli, ani niższe od 400 pikseli, należałoby zastosować następujące właściwości:

  ```
  minWidth: 600,
  minHeight: 400
  ```

  W przypadku określenia tych właściwości jQuery UI pozwoli, by jego wymiary były większe od podanych, ale nigdy nie dopuści, by były od nich mniejsze. Innymi słowy, jeśli zawartość będzie zbyt duża, by można ją wyświetlić w oknie o takich wymiarach, jQuery UI powiększy je, by wyświetlić całą zawartość.
- **Maksymalna szerokość i wysokość.** Można także nakazać jQuery UI utworzenie okna, którego wielkość nie będzie mogła przekroczyć zadanej szerokości i wysokości. Te maksymalne wymiary określane są odpowiednio przy użyciu właściwości `maxWidth` oraz `maxHeight`. Aby na przykład okno dialogowe mogło mieć co najwyżej 600 pikseli szerokości i 400 pikseli wysokości, należałoby użyć poniższych właściwości:

  ```
  maxWidth: 600,
  maxHeight: 400
  ```

Jeśli zawartość okna nie jest zbyt duża, to jQuery UI może je zmniejszyć poniżej tych wartości maksymalnych, jednak nigdy ich nie przekroczy. Jeśli zawartość jest na tyle duża, że nie można jej zmieścić w oknie o podanych wymiarach, zostanie w nim wyświetlony pasek przewijania, za pomocą którego użytkownik będzie mógł przesuwać zawartość w górę i w dół.

- **`modal`.** *Modalne* okna dialogowe są używane w celu przyciągnięcia uwagi użytkownika oraz uniemożliwienia mu wykonywania innych operacji niż związane z obsługą okna. Kiedy użytkownik wyświetli modalne okno dialogowe, nie będzie mógł kliknąć w żadnym innymi miejscu strony — cała powierzchnia strony z wyjątkiem okna zostanie przykryta ciemną, półprzezroczystą nakładką, przez co nawet trudno odczytać inne treści. Modalnych okien dialogowych można używać, jeśli chcemy, by użytkownik musiał przeczytać komunikat, a może nawet podjąć jakąś ważną decyzję (taką jak: „Czy na pewno chcesz usunąć z biblioteki wszystkie odcinki serialu *Dr Who*?"), zanim będzie mógł wykonywać jakiekolwiek inne czynności. Aby utworzyć modalne okno dialogowe, należy przypisać właściwości `modal` wartość `true`:

```
modal: true
```

Uwaga: Pełną listę dostępnych opcji oraz dodatkowe informacje na temat stosowania widżetu okna dialogowego można znaleźć na stronie *http://api.jqueryui.com/dialog/*.

- **`show` oraz `hide`.** Okno dialogowe zazwyczaj jest wyświetlane na ekranie, kiedy zostaje otworzone, i znika po zamknięciu. Ale czy takie działanie jest zabawne? Dlatego, dzięki właściwościom `show` oraz `hide`, można określać animacje, które będą odtwarzane odpowiednio w momencie wyświetlania oraz ukrywania okna dialogowego. Obie te właściwości mogą przyjmować wiele różnych wartości. Jeśli przypiszemy im wartość `true`, okno będzie jedynie dosyć szybko pojawiać się lub zanikać.

```
show: true,
hide: true
```

Można im jednak przypisać jakąś liczbę, określającą czas trwania animacji wyrażony w milisekundach. Załóżmy na przykład, że chcesz, by okno pojawiało naprawdę szybko — powiedzmy, w czasie 250 milisekund — lecz zanikało przez całe 2 sekundy. Możesz to zrobić, używając poniższych właściwości:

```
show: 250,
hide: 2000
```

Jednak możliwości nie ograniczają się do stopniowego pojawiania się lub zanikania. W obu tych właściwościach można także zapisać nazwę dowolnego efektu jQuery (patrz strona 211). Trzeba tylko pamiętać, by zapisać ją w apostrofach, na przykład: `'slideDown'`. A zatem gdybyś chciał, by oko dialogowe było wsuwane i wysuwane ze strony, powinieneś użyć następujących właściwości:

```
show: 'slideDown',
hide: 'slideUp'
```

Można także zastosować efekty jQuery UI (patrz strona 461), takie jak `'scale'` lub `'explode'`. A jakby tego wszystkiego było mało, to w każdej z tych właściwości można też zapisać *kolejny* literał obiektowy określający nazwę efektu, czas jego trwania, opóźnienie oraz tempo animacji (patrz strona 465). Załóżmy

na przykład, że chcesz, by po wydaniu polecenia zamknięcia okno odczekało 250 milisekund, a następnie znikało przez 1 sekundę przy użyciu efektu `'explode'` odtwarzanego za pomocą funkcji `'easeInQuad'` (lepiej nie próbuj tego robić w domu). W takim przypadku powinieneś skorzystać z właściwości `hide` o następującej postaci:

```
hide: {effect: 'explode', delay: 250, duration: 1000, easing: 'easeInQuad'}
```

- **`position`.** Standardowo okno dialogowe jest wyświetlane dokładnie pośrodku okna przeglądarki. Można je jednak wyświetlić w innym miejscu. Do określania położenia okna dialogowego służy właściwość `position`. Podobnie jak właściwościom `show` oraz `hide`, także jej można przypisywać wartości kilku różnych typów. W przypadku prostego określenia współrzędnych X i Y, wystarczy przekazać tablicę (patrz strona 77) zawierającą dwie liczby. Pierwsza z nich określa przesunięcie okna dialogowego od lewej krawędzi strony wyrażone w pikselach; natomiast druga — przesunięcie od górnej krawędzi strony. Załóżmy na przykład, że chciałbyś wyświetlić okno 100 pikseli od lewej krawędzi strony i tuż poniżej (dajmy na to 10 pikseli) od górnej krawędzi. W takim przypadku właściwość `position` powinna mieć następującą postać:

  ```
  position: [100,10]
  ```

  Położenie okna dialogowego można także określać przy użyciu słów kluczowych: `center`, `left`, `top`, `right` oraz `bottom`. Aby przykładowo wyświetlić okno dialogowe w prawym dolnym rogu okna przeglądarki, należałoby użyć następującej właściwości `position`:

  ```
  position: 'right bottom'
  ```

  Trzeba przy tym pamiętać, że pierwsze z podanych słów kluczowych musi określać położenie okna w poziomie (czyli może przyjmować wartości: `left`, `center` lub `right`), natomiast drugie — położenie w pionie (czyli może przyjmować wartości: `top`, `center` lub `bottom`). Oba słowa kluczowe należy oddzielić od siebie znakiem odstępu.

  I w końcu, ostatnią możliwością określenia położenia okna dialogowego jest zapisanie we właściwości `position` obiektu `position` jQuery UI. Ten bardzo użyteczny obiekt został dokładniej opisany w ramce na stronie 343.

Miniprzykład — przekazywanie opcji do okna dialogowego

W wywołaniu funkcji `dialog()` można przekazać dowolną kombinację opcji opisanych w poprzednim punkcie rozdziału. Ich działanie i sposób użycia możesz wypróbować, rozwijając przykład zamieszczony we wcześniejszej części rozdziału. W tym przykładzie wyświetlisz modalne okno dialogowe, dzięki czemu aż do momentu jego zamknięcia użytkownik nie będzie mógł nic zrobić na stronie; oprócz tego uniemożliwisz przeciąganie okna oraz zmianę jego wielkości. I jeszcze ostatnia sprawa: zadbasz o to, by okno znikło z wielkim hukiem.

1. **W edytorze tekstów ponownie wyświetl plik *hello_world.html*, nad którym pracowałeś w przykładzie na stronie 331.**

   Zacznij od przekazania pustego literału obiektowego w wywołaniu funkcji `dialog()`.

2. **Umieść kursor w wywołaniu funkcji `dialog()`, pomiędzy nawiasem otwierającym i zamykającym. Wpisz `{`, następnie dwukrotnie naciśnij klawisz *Enter* i wpisz `}`. Kod powinien wyglądać tak:**

```
$(document).ready(function() {
  $('#hello').dialog({

  });
}); // Koniec funkcji ready.
```

Teraz możesz się zająć dodawaniem par określających wartości właściwości.

3. **Wewnątrz literału obiektowego wpisz: `modal: true`:**

```
$(document).ready(function() {
  $('#hello').dialog({
    modal: true
  });
}); // Koniec funkcji ready.
```

Zapisz plik i otwórz go w przeglądarce. Cały obszar strony na zewnątrz okna dialogowego zostanie przykryty ciemną, półprzezroczystą warstwą w skośne paski. Aby strona ponownie była widoczna, musisz zamknąć okno dialogowe.

4. **W wierszu, który właśnie dodałeś, za słowem `true` wpisz przecinek. Naciśnij klawisz *Enter* i dopisz kolejne dwa wiersze (wyróżnione pogrubioną czcionką):**

```
$(document).ready(function() {
  $('#hello').dialog({
    modal: true,
    resizable: false,
    draggable: false
  });
}); // Koniec funkcji ready.
```

Pamiętaj, że pary nazwa – wartość muszą być oddzielane od siebie przecinkami. Każdy wiersz kodu zawierający taką parę, z wyjątkiem ostatniego, musi się kończyć przecinkiem. Dwa wiersze kodu, które oddałeś w tym kroku, uniemożliwiają użytkownikom odpowiednio zmienianie wielkości okna dialogowego oraz przeciąganie okna w inne miejsce. Teraz nadszedł czas, by zapewnić, że okno będzie zamykane w sposób atrakcyjny wizualnie.

5. **Na końcu ostatniego z wpisanych wierszy kodu dodaj przecinek, naciśnij klawisz *Enter*, po czym wpisz: `hide: 'explode'`:**

```
$(document).ready(function() {
  $('#hello').dialog({
    modal: true,
    resizable: false,
    draggable: false,
    hide: 'explode'
  });
}); // Koniec funkcji ready.
```

Właściwość dodana w tym kroku sprawi, że jQuery UI podczas zamykania okna dialogowego wykorzysta efekt o nazwie `'explode'`. Zapisz stronę i wyświetl ją w przeglądarce. Zwróć uwagę na to, co się dzieje podczas zamykania okna dialogowego. Spróbuj się pobawić, testując inne efekty udostępniane przez jQuery UI — w tym celu zastąp `'explode'` innymi wartościami, takimi jak `'bounce'`, `'blinds'` lub `'drop'`.

Kompletną wersję tej strony można znaleźć w przykładach dołączonych do książki, w pliku *complete_dialog_properties.html*, umieszczonym w katalogu *R09*.

Otwieranie okna dialogowego w odpowiedzi na zdarzenia

Okna dialogowe jQuery UI są łatwe w użyciu i stanowią doskonały zamiennik dla nudnych okien informacyjnych przeglądarki. Jednak najprawdopodobniej nie będziesz chciał, żeby okno było wyświetlane za każdym razem, gdy użytkownik wyświetli stronę. Okna dialogowe są znacznie bardziej przydane, gdy wyświetla się je w reakcji na czynności wykonywane przez użytkownika. Kiedy przykładowo użytkownik kliknie przycisk „Zapisz się, by otrzymywać naszą gazetkę", można wyświetlić okno dialogowe zawierające formularz rejestracyjny, zamiast przenosić go na inną stronę.

Okna dialogowe można wyświetlać w odpowiedzi na dowolne ze zdarzeń, które poznałeś na stronie 182: kliknięcia, naciśnięcia klawiszy, zmianę wielkości okna przeglądarki (choć takie rozwiązanie byłoby dość dziwne). Oprócz tego, zawsze można otwierać okna dialogowe z poziomu własnego kodu. Załóżmy na przykład, że utworzyłeś na swojej stronie quiz, który wymaga podania odpowiedzi w określonym czasie. Jeśli czas upłynie, zanim użytkownik skończy podawać odpowiedzi, możesz wyświetlić okno dialogowe z komunikatem: „Czas minął!".

W celu wyświetlenia okna dialogowego musisz zrobić kilka rzeczy. Przede wszystkim, musisz zażądać, by nie było ono otwierane bezpośrednio po utworzeniu — a normalnie właśnie tak się dzieje (patrz poprzedni miniprzykład). Poza tym, będziesz potrzebował czegoś, co później spowoduje wyświetlenie okna; w przeważające większości przypadków okna dialogowe są wyświetlane przy użyciu funkcji obsługujących zdarzenia (patrz strona 182).

Aby ukryć okno dialogowe bezpośrednio po jego utworzeniu, w opcjach przekazywanych w wywołaniu funkcji `dialog()` należy przekazać właściwość `autoOpen` o wartości `false`. Załóżmy, że na swojej stronie umieściłeś znacznik `<div>` o identyfikatorze `login`. Aby przekształcić ten znacznik w okno dialogowe jQuery UI, a jednocześnie sprawić, że początkowo będzie ono niewidoczne, musiałbyś użyć następującego kodu JavaScript:

```
$('#login').dialog({
  autoOpen: false
})
```

Po wczytaniu takiej strony jQuery UI przekształci znacznik `<div>` w okno dialogowe i jednocześnie je ukryje. Aby wyświetlić takie okno, należy przekazać do funkcji `dialog()` argument `'open'`. Przykładowo załóżmy, że na swojej stronie umieściłeś odnośnik „Zaloguj się" i chcesz, by jego kliknięcie powodowało wyświetlenie okna dialogowego. Załóżmy też, że odnośnik ten ma identyfikator `loginLink`. Poniższy kod JavaScript pobiera odnośnik i dodaje do niego funkcję obsługującą zdarzenia `click`, która powoduje wyświetlenie okna dialogowego:

```
$('#loginLink').click(function(evt) {
  evt.preventDefault();
  $('#login').dialog('open');
}); // Koniec funkcji click.
```

Drugi wiersz kodu — `evt.preventDefault()` — sprawia, że przeglądarka nie obsłuży kliknięcia odnośnika, czyli nie przejdzie na wskazywaną przez niego stronę (więcej informacji na temat metody `preventDefault()` można znaleźć na stronie 195).

Choć użytkownik zawsze może zamknąć okno dialogowe, klikając przycisk *Close* widoczny w jego prawym górnym rogu, jednak można to także zrobić w sposób programowy. Jeśli na przykład umieścisz na stronie okno dialogowe zawierające formularz, możesz chcieć, by było ono zamykane po przesłaniu formularza; dzięki temu użytkownik nie będzie musiał najpierw wysyłać formularza, a potem zamykać okna.

Uwaga: Jak obsługiwać wyświetlanie i ukrywanie okna dialogowego dowiesz się w następnym przykładzie.

Przyjmijmy teraz, że okno dialogowe, którego przykład znajduje się nieco wcześniej — znacznik `<div>` o identyfikatorze `login` — zawiera formularz. Chciałbyś, że okno było zamykane po przesłaniu formularza. Tak można to zrobić:

```
$('#login form').submit(function() {
  $('#login').dialog('close');
}); // Koniec funkcji submit.
```

Aby zamknąć okno dialogowe, wystarczy pobrać odpowiedni znacznik `<div>` i wywołać funkcję `dialog()`, przekazując do niej argument `'close'`. Bardzo proste!

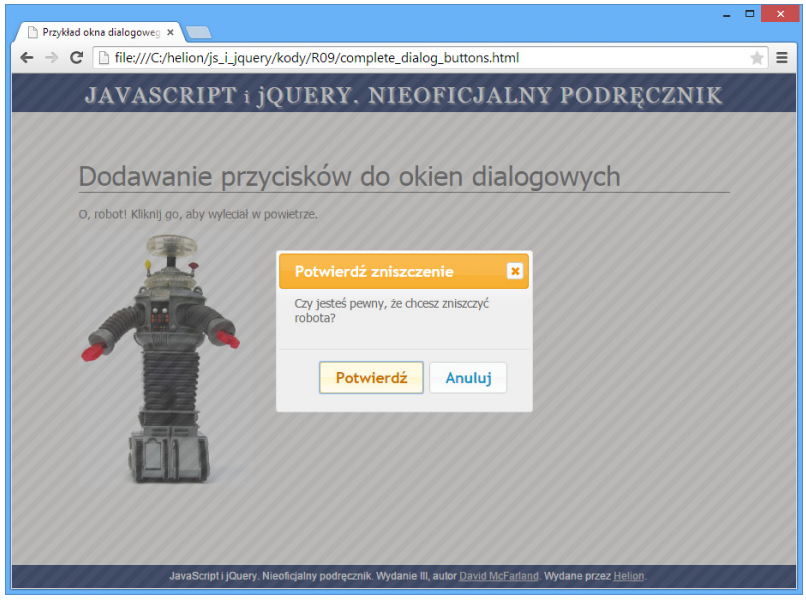
Uwaga: Przedstawiony tu przykład jest tylko częścią kompletnego rozwiązania. Po przesłaniu formularza musiałbyś wykonać jeszcze kilka innych operacji. Przesłanie formularza powoduje opuszczenie bieżącej strony WWW i przejście na następną, prezentującą efekty przetworzenia formularza. Dlatego też musiałbyś uniemożliwić przesłanie formularza i samodzielnie wysłać podane w nim dane na serwer, używając wyłącznie kodu JavaScript oraz technologii AJAX. Rozwiązania tego typu poznasz w rozdziale 13.

Dodawanie przycisków do okien dialogowych

Okna dialogowe nadają się nie tylko do wyświetlania komunikatów dla użytkowników witryny. Doskonale sprawdzają się także w sytuacjach, gdy trzeba pobrać od użytkowników jakieś dane. Załóżmy na przykład, że napisałeś aplikację, która pozwala użytkownikom na tworzenie własnej listy zadań do zrobienia (i faktycznie, w rozdziale 14. napiszesz taką aplikację). Jeśli użytkownik doda zadanie, lecz później zechce je usunąć, wystarczy, że kliknie przycisk w odpowiednim wierszu listy. Aby jednak mieć pewność, że użytkownik nie usunie żadnego zadania przez przypadek, mógłbyś wyświetlać okno dialogowe, które poprosi użytkownika o potwierdzenie decyzji (patrz rysunek 9.5).

Widżet `Dialog` jQuery UI pozwala na dodawanie przycisków do wszystkich tworzonych okien dialogowych. Co więcej, istnieje także możliwość wykonywania określonych fragmentów kodu, w zależności od tego, który przycisk zostanie kliknięty. Jeśli użytkownik kliknie przycisk *Usuń*, zadanie zostanie usunięte z listy, jeśli jednak kliknie przycisk *Anuluj*, nic się nie stanie.

W celu utworzenia przycisków do wywołania funkcji `dialog()` należy przekazać właściwość `buttons`, której wartością jest literał obiektowy zawierający nazwę każdego z przycisków oraz kod, który ma zostać wykonany po jego kliknięciu. W ramach przykładu załóżmy, że chcesz wyświetlić w oknie dialogowym przyciski *Potwierdź* oraz *Anuluj*. Można to zrobić w następujący sposób:



Rysunek 9.5. *Chcesz, aby okno dialogowe było interaktywne? Dodaj do niego przycisk! Taki przycisk, kiedy użytkownik go kliknie, może wykonywać zupełnie dowolne operacje. Przykładowo kliknięcie widocznego tu przycisku Potwierdź sprawi, że umieszczony na stronie obrazek robota „eksploduje"*

```
$('#dialog').dialog({
  buttons : {
    "Potwierdź" : function() {
      // Kod wykonywany po kliknięciu przycisku "Potwierdź".
    },
    "Anuluj" : function() {
      // Kod wykonywany po kliknięciu przycisku  "Anuluj".
    }
  } // Koniec właściwości buttons.
}); // Koniec funkcji dialog.
```

To całkiem spory kawałek kodu, w którym znajduje się wiele nawiasów klamrowych { i }. Pamiętaj jednak, że każdy z tych przycisków jest jedynie elementem literału obiektowego — właściwością składającą się z nazwy oraz funkcji. Przykładowo poniżej przedstawiony został kod tworzący przycisk *Potwierdź*:

```
"Potwierdź" : function() {
  // Kod wykonywany po kliknięciu przycisku "Potwierdź".
}
```

Jego pierwszy element — łańcuch znaków `Potwierdź` — to tekst, który zostanie wyświetlony na przycisku w oknie dialogowym. Drugim elementem jest funkcja (patrz strona 168) zawierająca kod, który zostanie wykonany po kliknięciu przycisku. Ten kod może wykonywać całkowicie dowolne operacje, zaczynając od usunięcia elementu strony, a kończąc na uruchomieniu przeglądarkowej gry napisanej w całości w JavaScripcie. W większości przypadków będziemy także chcieli zamknąć okno po kliknięciu przycisku. Zatem przyda się argument `'close'`. Aby po kliknięciu przycisku *Potwierdź* i zakończeniu wszelkich innych operacji, których wykonanie potwierdzono, okno dialogowe zostało zamknięte, należałoby dodać na końcu funkcji anonimowej wywołanie `$(this).dialog('close')`:

```
"Potwierdź" : function() {
  // Kod wykonywany po kliknięciu przycisku "Potwierdź".
  $(this).dialog('close');
}
```

Więcej informacji ma temat wywołania `$(this)` można znaleźć na stronie 169. W kontekście okna dialogowego `$(this)` odwołuje się do znacznika `<div>`, który został przekształcony w to okno. Dysponując tym odwołaniem, można na jego rzecz wywołać funkcję `dialog()`, aby zamknąć okno. Teraz nadszedł czas, abyś nabrał nieco doświadczenia w dodawaniu przycisków do okien dialogowych.

Miniprzykład — dodawanie przycisków do okien dialogowych

W tym przykładzie zapewnisz użytkownikowi możliwość usunięcia obrazka ze strony. Aby usunąć obrazek, wystarczy go kliknąć. Aby mieć pewność, że obrazek nie zostanie usunięty przypadkowo, dodasz do strony okno dialogowe, w którym użytkownik będzie musiał potwierdzić żądaną operację.

1. **W edytorze tekstów otwórz plik *dialog_buttons.html*.**

   W tym przykładzie do strony jest już dołączony arkusz stylów jQuery UI oraz oba pliki JavaScript; możesz zatem zacząć od razu od dodania kodu HTML okna dialogowego.

2. **Odszukaj pusty wiersz, umieszczony bezpośrednio pod komentarzem `<!-- Tutaj dodaj okno dialogowe. -->` i wpisz w nim:**

   ```
   <div id="confirm" title="Potwierdź zniszczenie">
     <p>Czy jesteś pewny, że chcesz zniszczyć robota?</p>
   </div>
   ```

   Teraz zmienisz ten element `<div>` w okno dialogowe.

3. **Odszukaj pusty wiersz poniżej wywołania funkcji `$(document).ready()` i wpisz w nim:**

   ```
   $('#confirm').dialog({

   });
   ```

   Ten kod zmienia wybrany element w okno dialogowe. Para nawiasów klamrowych { oraz } reprezentuje pusty literał obiektowy (patrz strona 165), w którym zaraz zapiszesz opcje okna dialogowego. Najpierw zadbasz o to, by okno było modalne. W ten sposób użytkownik nie będzie mógł zrobić na stronie nic innego, dopóki go nie zamknie.

4. **Wewnątrz literału obiektowego wpisz `modal: true`:**

   ```
   $('#confirm').dialog({
     modal: true
   });
   ```

   Dodatkowo chcesz, aby początkowo okno dialogowe było ukryte. Będzie ono wyświetlane dopiero po kliknięciu przez użytkownika obrazka przedstawiającego robota.

5. **Za słowem `true` dodaj przecinek, naciśnij klawisz *Enter*, a następnie wpisz kod wyróżniony poniżej pogrubioną czcionką:**

   ```
   $('#confirm').dialog({
     modal: true,
     autoOpen: false
   });
   ```

Poszczególne pary nazwa – wartość muszą być oddzielone od siebie przecinkami, dlatego nie zapomnij dodać przecinka za wartością `true`. Już za chwilkę zajmiesz się dodawaniem do okna dialogowego przycisków, jednak najpierw dodasz kod niezbędny do wyświetlenia tego okna w odpowiedzi na kliknięcie obrazka robota. Znacznik obrazka ma identyfikator `robot`, zatem bez trudu będziesz mógł się do niego odwołać i dodać obsługę zdarzenia `click`.

6. **Za wywołaniem funkcji `dialog()` dodaj trzy wiersze tekstu wyróżnione poniżej pogrubioną czcionką; po tej zmianie kod powinien wyglądać w następujący sposób:**

```
$(document).ready(function() {
  $('#confirm').dialog({
    modal: true,
    autoOpen: false
  });
  $('#robot').click(function() {

  }); // Koniec funkcji click.
}); // Koniec funkcji ready.
```

Ten kod pobiera obrazek robota i dodaje do niego procedurę obsługi zdarzeń (patrz strona 182). Teraz, aby wyświetlić okno dialogowe, wystarczy jedynie wybrać odpowiedni element `<div>` i wywołać funkcję `dialog()`, przekazując do niej argument `'open'`.

7. **W wywołaniu funkcji `click()` dopisz jeden wiersz kodu (wyróżniony pogrubioną czcionką):**

```
$(document).ready(function() {
  $('#confirm').dialog({
    modal: true,
    autoOpen: false
  });
  $('#robot').click(function() {
    $('#confirm').dialog('open');
  }); // Koniec funkcji click.
}); // Koniec funkcji ready.
```

Teraz, kiedy użytkownik kliknie obrazek robota, zostanie wyświetlone okno dialogowe.

8. **Zapisz plik i wyświetl go w przeglądarce. Kliknij obrazek robota, aby zobaczyć swoje nowe okno dialogowe.**

W oknie dialogowym nie ma jeszcze żadnych przycisków. Dodaj je, krok po kroku, w kilku kolejnych punktach; tylko tak dobrze zrozumiesz, jak działają.

9. **Wróć do edytora tekstów i pliku *dialog_buttons.html*. Wewnątrz wywołania funkcji `dialog()`, w ostatnim wierszu przekazywanych do niej opcji, za wartością `false`, wpisz przecinek, naciśnij klawisz *Enter*, po czym wpisz poniższy kod wyróżniony pogrubioną czcionką:**

```
$(document).ready(function() {
  $('#confirm').dialog({
    modal: true,
    autoOpen: false,
    buttons : {

    }
  });
  $('#robot').click(function() {
```

```
        $('#confirm').dialog('open');
      }); // Koniec funkcji click.
    }); // Koniec funkcji ready.
```

W ten sposób przekażesz do funkcji `dialog()` następną opcję — `buttons`. Jej wartością jest kolejny literał obiektowy zawierający informacje o przyciskach. Najpierw dodasz przycisk *Potwierdź*.

PORADNIA DLA ZAAWANSOWANYCH

Dokładne umiejscawianie za pomocą jQuery UI

Widżety okna dialogowego oraz etykietki ekranowej pozwalają na kontrolę położenia elementu. Służy do tego właściwość `position`. Właściwość ta jest obiektem (patrz strona 165) definiującym, gdzie zostanie umieszczone pude ko elementu w odniesieniu do innego elementu strony. Sposób tworzenia tego obiektu jest dosyć dziwny, jednak kiedy nabędziemy nieco doświadczenia, wyda się całkiem prosty. Obiekt `position` może zawierać kilka właściwości, lecz najczęściej stosowane są dwie z nich: `my` oraz `at`.

Aby na przykład umieścić okno dialogowe w prawym dolnym rogu okna przeglądarki, należałoby użyć następującego kodu:

```
$('#dialog').dialog({
  position: {
    my: 'right bottom',
    at: 'right bottom'
  }
}); // Koniec funkcji dialog.
```

Właściwość `my` odnosi się do okna dialogowego; natomiast właściwość `at` do okna przeglądarki. A zatem właściwość `my` określa, które miejsce okna dialogowego znajdzie się w miejscu wskazanym przez właściwość `at`. W tym przypadku obiekt `position` informuje, że prawy dolny wierzchołek (`my`) okna dialogowego ma się znaleźć w prawym, dolnym wierzchołku okna (`at`).

Podczas określania wartości tych właściwości stosowana jest składnia CSS (taka sama jak używana we właściwości CSS `background-position`). Pierwsze słowo kluczowe tej wartości określa położenie w poziomie i może nim być: `left`, `right` lub `center`. Drugie słowo określa położenie w pionie, a jego dopuszczalnymi wartościami są: `top`, `center` oraz `bottom`.Położenie okien dialogowych jest określane względem okna przeglądarki. W odróżnieniu do nich, etykietki ekranowe są rozmieszczane względem *elementu wyzwalającego*, czyli elementu powodującego ich wyświetlenie. Jednak określając położenie okna lub etykietki, można zastosować także trzecią właściwość — `of` — pozwalającą umieścić je względem innego elementu strony. Może się ona przydać podczas tworzenia prezentacji opisującej różne części interfejsu, w której użytkownik poznaje poszczególne elementy strony.

Moglibyśmy na przykład wyświetlić okno dialogowe koło przycisku do logowania, by pokazać użytkownikowi, gdzie może się zalogować.

We właściwości `of` zapisywany jest selektor (patrz strona 148) lub obiekt jQuery. Załóżmy, że przycisk do logowania ma identyfikator `login`, a okno dialogowe chcemy wyświetlić bezpośrednio poniżej niego; w takim przypadku możemy użyć następującego kodu:

```
$('#dialog').dialog({
  position: {
    my: 'center top',
    at: 'center bottom',
    of: '#login'
  }
}); // Koniec funkcji dialog.
```

Aby uzyskać dodatkową kontrolę nad położeniem okna dialogowego lub etykietki ekranowej, można także zastosować *przesunięcie*. Chcemy przykładowo, by okno dialogowe nachodziło na przycisk do logowania na wysokość 10 pikseli, można to uzyskać, stosując następujący kod:

```
$('#dialog').dialog({
  position: {
    my: 'center top-10',
    at: 'center bottom',
    of: '#login'
  }
}); // Koniec funkcji dialog.
```

Określając przesunięcie, można dodawać lub odejmować liczby (będą one traktowane jako wartości wyrażone w pikselach) bądź wartości procentowe (na przykład: `my: 'center top+25%'`). Trzeba zwrócić uwagę, by nie umieszczać znaków odstępu pomiędzy słowem kluczowym (np. `top`) a operatorem (+ lub -) oraz wartością (na przykład `25%`). Jeśli tego nie dopilnujemy, przesunięcie nie zostanie uwzględnione.

Więcej informacji na temat obiektu `position` można znaleźć na stronie *http://api.jqueryui.com/position/*.

10. **Wewnątrz obiektu `buttons` dodaj następujący kod:**

```
buttons : {
  "Potwierdź" : function() {

  }
} // Koniec właściwości buttons.
```

Słowo `Potwierdź` stanie się nazwą pierwszego przycisku. Kiedy użytkownik go kliknie, zostanie wywołana podana funkcja anonimowa. Zapisz stronę i wypróbuj jej działanie w przeglądarce: kiedy klikniesz obrazek robota, powinno się pojawić okno dialogowe zawierające jeden przycisk: *Potwierdź*. Niestety, kliknięcie tego przycisku nie daje żadnych rezultatów. Wróć do edytora.

11. **Wewnątrz funkcji anonimowej dodanej w poprzednim kroku wpisz poniższy wiersz kodu:**

```
buttons : {
  "Potwierdź" : function() {
    $('#robot').effect('explode');
  }
} // Koniec właściwości buttons.
```

Ten kod wybiera obrazek robota, a następnie odtwarza na nim efekt jQuery UI o nazwie `'explode'`, którego już użyłeś na stronie 337. (Więcej informacji na temat tego oraz wielu innych efektów jQuery UI można znaleźć na stronie 461).

Czas dodać kolejny przycisk.

12. **Za zamykającym nawiasem klamrowym funkcji przycisku *Potwierdź* wstaw przecinek, naciśnij klawisz *Enter*, po czym wpisz poniższy kod wyróżniony pogrubioną czcionką:**

```
buttons : {
  "Potwierdź" : function() {
    $('#robot').effect('explode');
  },
  "Anuluj" : function() {

  }
} // Koniec właściwości buttons.
```

Jeśli teraz zapiszesz stronę i wyświetlisz ją w przeglądarce, zobaczysz, że okno dialogowe ma już dwa przyciski. Kliknięcie przycisku *Anuluj* jeszcze nic nie robi i z resztą nie powinno, gdyż przycisk ten służy jedynie do odwołania akcji potwierdzanej przez pierwszy przycisk. Łatwo jednak zauważysz, że kliknięcie przycisku *Anuluj* nawet nie zamyka okna dialogowego. A to akurat powinno robić.

13. **Wewnątrz funkcji anonimowej przycisku *Anuluj* wpisz `$(this).dialog ↪('close');`:**

```
buttons : {
  "Potwierdź" : function() {
    $('#robot').effect('explode');
  },
  "Anuluj" : function() {
    $(this).dialog('close');
  }
} // Koniec właściwości buttons.
```

Ponieważ przyciski są tworzone w funkcji `dialog()`, która został wywołana na rzecz znacznika `<div>`, zatem wywołanie `$(this)` odwołuje się do samego okna dialogowego. A zatem, w tym przypadku wywołanie `$(this).dialog('close')` ma taki sam efekt, co wywołanie `$('#confirm').dialog('close');`.

Zapisz stronę i wyświetl ją w przeglądarce. Kliknij obrazek robota, by wyświetlić okno dialogowe, następnie kliknij przycisk *Anuluj* — okno zostanie zamknięte! Jeszcze raz kliknij obrazek robota, lecz tym razem w oknie dialogowym kliknij przycisk *Potwierdź*: obrazek robota „eksploduje" i zniknie. Niestety, okno dialogowe nie zniknie wraz z robotem. Na szczęście ten problem można łatwo rozwiązać.

14. **Dodaj wywołanie `$(this).dialog('close');` jako ostatni wiersz kodu funkcji anonimowej przycisku *Potwierdź*. Poniżej przedstawiona została końcowa postać kodu:**

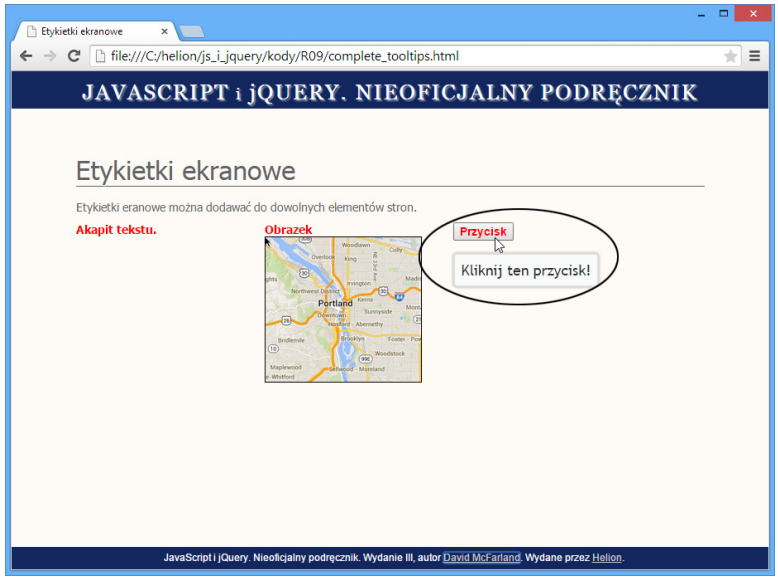
```
$(document).ready(function() {
  $('#confirm').dialog({
    modal: true,
    autoOpen: false,
    buttons : {
      "Potwierdź" : function() {
        $('#robot').effect('explode');
        $(this).dialog('close');
      },
      "Anuluj" : function() {
        $(this).dialog('close');
      }
    } // Koniec właściwości buttons.
  });
  $('#robot').click(function() {
    $('#confirm').dialog('open');
  }); // Koniec funkcji click.
}); // Koniec funkcji ready.
```

Zapisz stronę i wyświetl ją w przeglądarce. Dokończona strona (po wyświetleniu okna dialogowego) powinna wyglądać tak, jak na rysunku 9.5. Spróbuj kliknąć oba przyciski i sprawdź, co się stanie. Kompletną wersję strony zbudowanej w tym przykładzie można znaleźć w pliku *complete_dialog_buttons.html*, umieszczonym w katalogu *R09*.

Prezentowanie informacji w etykietkach ekranowych

Czasami może się zdarzyć, że będziemy musieli dostarczyć użytkownikom strony nieco więcej informacji. Załóżmy na przykład, że na swojej stronie prezentujesz wiersz ikon prowadzących do różnych serwisów społecznościowych, takich jak Twitter, Facebook, Reddit, Instagram i tak dalej. Ktoś, kto nie zna tych serwisów, zapewne nie będzie w stanie rozpoznać tych ikon i domyślić się, gdzie każda z nich prowadzi. Aby pomóc takim użytkownikom, mógłbyś dodać do każdej z ikon etykietkę ekranową — małe okienko wyświetlane po wskazaniu danej ikony myszą i prezentujące jej opis, na przykład: „Moja strona na Facebooku".

Biblioteka jQuery UI zapewnia możliwość bardzo szybkiego i prostego dodawania takich etykietek ekranowych do dowolnych elementów strony (patrz rysunek 9.6).



Rysunek 9.6. *Za pomocą etykietek ekranowych można przekazywać użytkownikom dodatkowe informacje. Pozwalają one tworzyć niewielkie okienka wyświetlane po wskazaniu elementu myszą (patrz zakreślony fragment rysunku). Etykietki są doskonałym sposobem na wyjaśnianie, co się stanie po kliknięciu ikony, prezentowanie podpisu pod rysunkiem lub nawet wyświetlanie zdjęć i innej zawartości HTML po wskazaniu wybranego elementu strony*

Można nawet dodawać bardziej rozbudowane etykiety, prezentujące większe fragmenty kodu HTML, zawierające nawet obrazki i odnośniki. Poniższa lista prezentuje podstawowe czynności związane z dodawaniem etykietek ekranowych.

1. **Dołącz do strony biblioteki jQuery i jQuery UI; czynności zostały opisane na stronie 329.**

   Niewiele mógłbyś zrobić bez niezbędnych plików CSS i JavaScript.

2. **W dowolnym elemencie strony, do którego chcesz dodać etykietkę ekranową, podaj atrybut `title`:**

   ```
   <a href="https://twitter.com/helionpl" title="Śledź nas na  Twitterze">
   <img src="twitter.png">
   </a>
   ```

   Niektóre przeglądarki same wyświetlają etykietki po dodaniu do elementu atrybutu `title`. Podobnie jak informacyjne okna dialogowe, także etykietki ekranowe wyświetlane przez przeglądarkę nie zapewniają możliwości zmiany wyglądu. Co więcej, nie są one dostępne we wszystkich przeglądarkach. Dlatego stosowanie etykietek jQuery UI zapewnia maksymalną kontrolę i efektywność.

3. **Wybierz odpowiedni element i wywołaj funkcję `tooltip()`:**

   ```
   $(document).ready(function() {
     $('[title]').tooltip();
   }); // Koniec funkcji ready.
   ```

   W tym przypadku zastosowany został prosty selektor atrybutu (patrz strona 152), pozwalający na wybranie wszystkich elementów, w których został podany atrybut `title`. Wywołanie funkcji `tooltip()` doda do każdego z tych elementów etykietkę ekranową. I to naprawdę już wszystko. jQuery UI zajmie się całą resztą i utworzy atrakcyjne etykietki dla każdego z wybranych elementów.

Jeśli trzeba, można bardziej precyzyjnie wybrać elementy, do których zostaną dodane etykietki. Jeśli na przykład nie chcesz, by były dodawane do wszystkich elementów zawierających atrybut `title`, mógłbyś dopisać do wybranych elementów jakąś klasę — na przykład: `tooltip` — a następnie wybrać je, używając odpowiedniego selektora:

```
$('.tooltip').tooltip();
```

Miniprzykład — szybkie dodawanie etykietek ekranowych

Biblioteka jQuery UI udostępnia najszybszy sposób dodawania etykietek ekranowych do wybranych elementów stron (ich użytkownicy na pewno Ci podziękują).

1. **W edytorze tekstów otwórz plik *tooltips.html*.**

   Do tej przykładowej strony zostały już dołączone niezbędne pliki CSS oraz JavaScript. Możesz zatem natychmiast przystąpić do działania i dodać atrybut `title` do wybranych elementów. Zawartość tych atrybutów będzie później prezentowana na etykietkach ekranowych.

2. **Odszukaj akapit `<p>Akapit tekstu.</p>` i dodaj do niego atrybut `title` o wartości `"Owszem, jestem akapitem."`:**

   ```
   <p title="Owszem, jestem akapitem.">Akapit tekstu.</p>
   ```

   Atrybut `title` można dodawać do dowolnych elementów umieszczanych w ciele strony. (Tworząc te etykietki, możesz się wykazać znacznie większą kreatywnością).

3. **Odszukaj obrazek `<img src="images/map.png">` i dodaj do niego atrybut `title` o wartości `"A ja jestem mapą!"`:**

   ```
   <img src="images/map.png" title="A ja jestem mapą!">
   ```

   Został jeszcze jeden znacznik i wszystkie zmiany w kodzie HTML będą wprowadzone.

4. **Odszukaj akapit z przyciskiem `<button >Przycisk</button>` i dodaj do znacznika `<button>` atrybut `title` o wartości `"Kliknij ten przycisk!"`:**

   ```
   <button title="Kliknij ten przycisk!">Przycisk</button>
   ```

   Teraz pozostało jedynie dodanie kodu jQuery.

5. **Wewnątrz funkcji `$(document).ready()` dodaj wywołanie `$('[title']).tooltip()`, tak by pełny kod wyglądał tak:**

   ```
   $(document).ready(function() {
     $('[title]').tooltip();
   }); // Koniec funkcji ready.
   ```

   Możesz wierzyć lub nie, lecz to już wszystko. Wywołanie `$('[title]')` wybiera wszystkie znaczniki, w których został podany atrybut `title`, a kolejne wywołanie — `.tooltip()` — dodaje do nich etykietę ekranową. Zapisz stronę, a następnie wyświetl ją w przeglądarce. Wskaż myszą akapit tekstu, obrazek oraz przycisk, aby sprawdzić, czy będą wyświetlane etykietki (patrz rysunek 9.6). Gotowa wersja przykładu jest dostępna w pliku *complete_tooltip.html*, w katalogu *R09*.

Opcje etykietek ekranowych

Domyślnie etykietki ekranowe są stopniowo wyświetlane i ukrywane, a jQuery UI umieszcza je pod elementem, do którego zostały dodane. Jednak podobnie jak podczas tworzenia okien dialogowych (patrz strona 330), także i w wywołaniu funkcji `tooltip()` można przekazywać literał obiektowy (patrz strona 165) kontrolujący zachowanie etykietek. Poniżej przedstawiona została lista najbardziej przydatnych opcji.

- **`show`.** Właściwość `show` daje możliwość określenia animacji, która zostanie wykonana podczas wyświetlania etykietki ekranowej. Działa dokładnie tak samo jak właściwość `show` okien dialogowych, opisana na stronie 335. Innymi słowy, etykietka może się pojawiać stopniowo, zostać wsunięta na miejsce lub pojawić się w dowolny inny sposób określany przez dostępne efekty jQuery UI (patrz strona 461), poprawiając tym samym wizualną atrakcyjność strony.
- **`hide`.** Działa tak samo jak właściwość `show`, przy czym określa sposób ukrywania etykietki.
- **`track`.** Przypisanie tej właściwości wartości `true` sprawi, że etykietka ekranowa będzie przesuwała się wraz z ruchami wskaźnika myszy (o ile tylko będzie on umieszczony w obszarze elementu, do którego etykietka została dodana).

  ```
  track: true
  ```

  Takie przesuwające się etykietki bez wątpienia przyciągają uwagę użytkownika, jednak mogą także dekoncentrować i utrudniać czytanie umieszczonego w nich tekstu.
- **`tooltipClass`.** Właściwość pozwala podać nazwę klasy, do której będzie należeć etykietka.

  ```
  tooltipClass: 'tooltip'
  ```

  Przygotowanie własnej klasy oraz dodanie jej do etykietek umożliwia zmodyfikowanie lub rozszerzenie domyślnych stylów etykietek ekranowych określanych przez jQuery UI.
- **`position`.** Wartością tej właściwości ma być obiekt `position` jQuery UI (patrz ramka na stronie 343), określający położenie etykietki w odniesieniu do elementu docelowego (czyli elementu, którego wskazanie powoduje jej wyświetlenie).

Uwaga: Stosując właściwość `position`, należy zachować dużą ostrożność. Etykietki ekranowe znikają, kiedy wskaźnik myszy znajdzie się na obszarze strony, który same zajmują. Jeśli zatem umieścisz etykietkę na obszarze elementu, który je wyświetla, może się zdarzyć, że etykietka nigdy nie zostanie wyświetlona. Wynika to z faktu, że umieszczenie wskaźnika myszy w obszarze elementu powodującego wyświetlenie etykietki sprawi, że wskaźnik znajdzie się także w obszarze etykietki, co spowoduje jej natychmiastowe ukrycie.

Wszystkie te opcje podaje się w literale obiektowym, przekazywanym w wywołaniu funkcji `tooltip()`. Załóżmy na przykład, że chcesz, by etykietka stopniowo się pojawiała, znikała przy użyciu efektu `'explode'` i poruszała zgodnie z ruchami wskaźnika myszy. W takim przypadku powinieneś użyć wywołania funkcji `tooltip()` o następującej postaci:

```
$('[title]').tooltip({
  show: true,
  hide: 'explode',
  track: true
});
```

Umieszczanie w etykietkach treści HTML

Jak się mogłeś przekonać, dodawanie etykietek ekranowych przy użyciu jQuery UI jest bardzo proste. Co jednak można zrobić, gdy chcemy wyświetlić w etykietce więcej treści, na przykład zdjęcie lub kilka akapitów tekstu? Oczywiście takich treści nie można zapisać w atrybucie `title` — nie byłoby to prawidłowe i mogłoby doprowadzić do problemów na stronie. W takich sytuacjach widżet etykietek ekranowych zapewnia możliwość określenia alternatywnego źródła treści. Do tego celu służy właściwość `content` pozwalająca na wskazanie elementu zawierającego treść, która zostanie wyświetlona w etykietce.

Właściwości `content` można używać na kilka różnych sposobów. Najprostszym z nich jest zapisanie w niej łańcucha znaków zawierającego kod HTML, który stanie się treścią etykietki. Załóżmy, że umieściłeś na stronie odnośnik do strony z informacjami na swój temat.

```
<a href="coolest_person_on_earth.html" id="me">Strona o mnie</a>
```

Jeśli na witrynie jest dostępna Twoja fotografia, mógłbyś ją umieścić w etykietce, tak by była wyświetlana, gdy użytkownik wskaże odnośnik myszą.

```
$('#me').tooltip({
  content: '<img src="me.png" alt="Spójrz, to ja!">'
});
```

Wywołanie `$('#me')` powoduje wybranie odnośnika, a wywołanie `.tooltip()` dodaje do niego etykietkę ekranową. Właściwość `content` określona w literale obiektowym przekazywanym do funkcji `tooltip()` określa, że wewnątrz etykietki ma zostać umieszczony kod HTML — w tym przypadku jest to znacznik `<img>`.

Kolejnym sposobem wyświetlania kodu HTML w etykietkach ekranowych jest umieszczenie go w innym elemencie na stronie, a następnie użycie właściwości `content`, która sprawi, że zostanie on pobrany z elementu źródłowego i umieszczony w etykietce. Jak takie rozwiązanie działa? Skoro umieścimy kod HTML na stronie, zostanie on wyświetlony w oknie przeglądarki, a przecież nam zależy na tym, by był widoczny wyłącznie w etykietce. Jednym z możliwych rozwiązań tego problemu jest ukrycie takiego elementu przy użyciu metody `hide()` jQuery, a następnie wywołanie funkcji `tooltip()`, by pobrać niewidoczny kod HTML i wyświetlić go w etykietce.

Istnieje jednak inne, prostsze rozwiązanie, które ostatnio coraz częściej jest używane przez programistów JavaScript; polega ono na utworzeniu „szablonu" poprzez umieszczenie kodu HTML wewnątrz znaczników `<script>`. Ponieważ przeglądarki traktują zawartość tych znaczników jak kod JavaScript, zatem nie wyświetlą takiego kodu HTML. Jednak do tego kodu można się odwołać przy użyciu jQuery i w jakiś sposób wykorzystać — na przykład wyświetlając go w etykietce! Na przykład załóżmy, że chciałbyś wyświetlić w etykietce ekranowej nagłówek

<h2> oraz krótką listę wypunktowaną. W takim przypadku możesz zacząć od utworzenia szablonu umieszczonego w znaczniku <script>:

```
<script id="tooltipTemplate" type="text/template">
  <h2>O mnie</h2>
  <ul>
    <li>Jestem totalnie super.</li>
    <li>Nawet myję zęby trzy razy dziennie!</li>
    <li><img src="me.png" alt="Spójrz, to ja!"></li>
  </ul>
</script>
```

Ponieważ do tego szablonu musisz się jakoś odwołać, zatem bardzo ważne jest określenie wartości atrybutu id, choć oczywiście nie musi on mieć wartości tooltipTemplate . Wartość identyfikatora może być zupełnie dowolna. Opcjonalny jest także atrybut type= text/template , choć wielu programistów korzystających z tej techniki używa go, by jasno określić przeznaczenie znacznika <script>. Powszechnie stosowanym rozwiązaniem jest także umieszczanie tych znaczników na samym końcu strony, tuż przed zamykającym znacznikiem </body>.

Kiedy już przygotujesz szablon, możesz się do niego odwołać, pobrać umieszczony w nim kod HTML i użyć w etykietce. W poniższym kodzie pokazano, jak to zrobić:

```
$('#me').tooltip({
  content: $('#toolTipTemplate').html()
});
```

Zgodnie z informacjami podanymi na stronie 157, metoda html() jQuery pozwala na pobranie kodu HTML umieszczonego w wybranych elementach. W tym przypadku wybieramy znacznik <script> zawierający przygotowany szablon, a następnie pobieramy jego kod HTML. jQuery UI użyje potem tego kodu HTML do przygotowania zawartości etykietki ekranowej.

Miniprzykład — umieszczanie kodu HTML w etykietkach ekranowych

Tworzenie etykietek ekranowych zawierających kod HTML jest nieco bardziej złożone od opracowania zwyczajnych etykietek z krótkim fragmentem tekstu, jednak z pomocą sztuczki ze znacznikiem <script> i tak jest dosyć proste.

1. **W edytorze tekstów otwórz plik *advanced_tooltips.html*.**

   Do strony zostały już dołączone wszystkie niezbędne pliki arkuszy stylów jQuery UI oraz JavaScript. Kolejnym krokiem będzie dodanie kodu HTML, który w przyszłości stanie się treścią etykietki ekranowej.

2. **Tuż przed końcem strony, poniżej komentarza <!-- Tutaj umieść szablon. --> wpisz następujący kod HTML:**

   ```
   <script id="contactInfo" type="text/template">
     <p>Zadzwoń do nas na numer: 555-555-5555</p>
     <p><img src="images/map.png" title="Oto ja, mapa!"></p>
   </script>
   ```

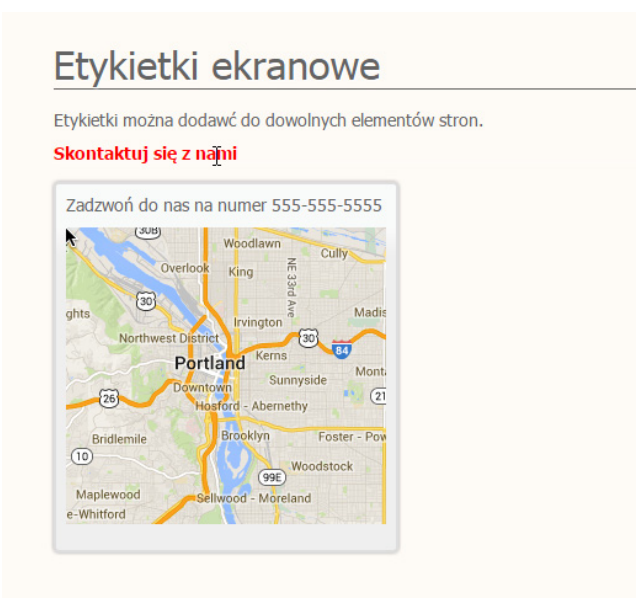
Zastosowaliśmy tu technikę opisaną nieco wcześniej na stronie 349, która pozwala ukrywać kod HTML wewnątrz znaczników `<script>`. Przeglądarka nie wyświetli takiego kodu, lecz wciąż będzie można odwołać się do niego przy użyciu jQuery.

3. **Wewnątrz wywołania funkcji `$(document).ready()` dodaj następujący fragment kodu:**

```
$('#contact').tooltip({
  content: $('#contactInfo').html()
}); // Koniec funkcji tooltip.
```

Takie wywołania funkcji wyglądają znajomo. W tym przypadku wywołanie wybiera element strony (a konkretnie znacznik `<p>` o identyfikatorze `contact`), a następnie wywołuje na jego rzecz funkcję `tooltip()`, określając przy tym zawartość etykietki. Wywołanie `$('#contactInfo')` wybiera znacznik `<script>`, który dodałeś do strony w poprzednim kroku, a wywołanie `.html()` pobiera jego zawartość.

Zapisz plik i wyświetl go w przeglądarce. Wskaż myszą tekst *Skontaktuj się z nami* — powinna pojawić się etykietka ekranowa pokazana na rysunku 9.7. Dokończona wersja przykładu jest dostępna w pliku *complete_advanced_tooltips.html*, umieszczonym w katalogu *R09*.

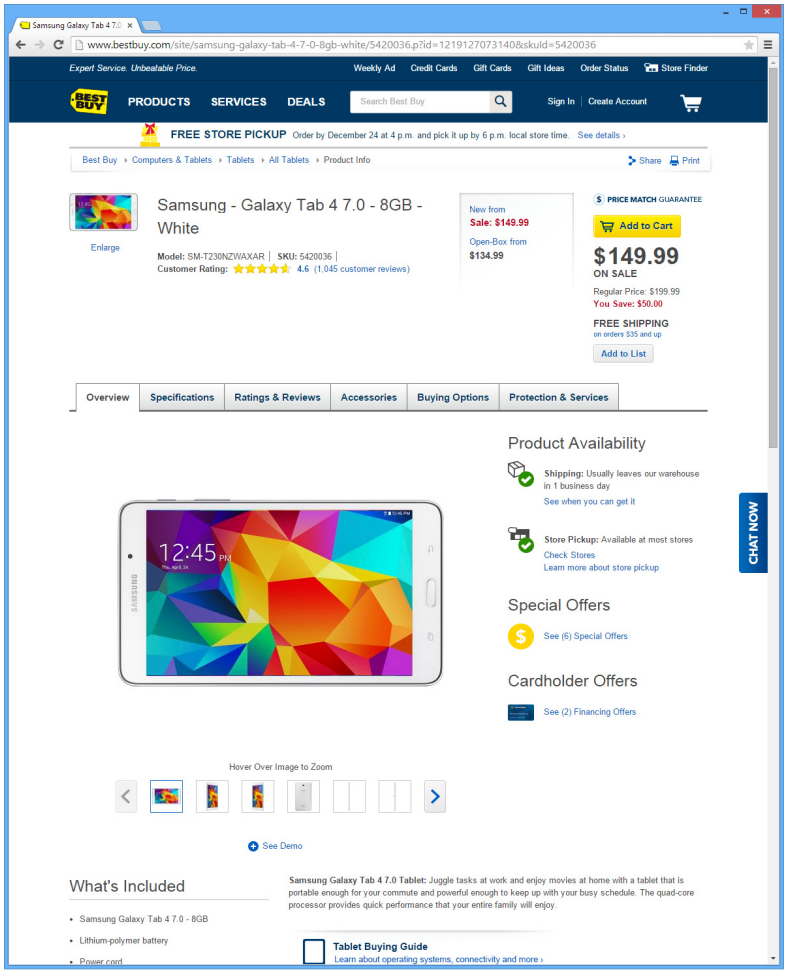


Rysunek 9.7. *Zawartość etykietek ekranowych nie musi ograniczać się do treści, które można zapisać w atrybucie title. Można w nich wyświetlać dowolny kod HTML, w tym także obrazki, akapity tekstu oraz klipy wideo. Pamiętaj jednak, że wskazanie etykietek myszą powoduje ich ukrycie, dlatego nie można w nich umieszczać elementów, z którymi użytkownik musi prowadzić interakcję, takich jak odnośniki lub pola formularzy. Takie elementy lepiej umieszczać w oknach dialogowych (patrz strona 330)*

Dodawanie zestawów kart

Czasami tworzenie stron WWW przypomina walkę o uwagę użytkownika. Może się zdarzyć, że na stronie trzeba będzie umieścić tak wiele informacji, że stanie się ona długa, przepełniona i nieczytelna. Jednym z rozwiązań problemów tego typu jest zastosowanie kart. Karty pozwalają podzielić stronę na fragmenty, z których

w danej chwili jest wyświetlany tylko jeden, wybierany przez użytkownika, który kliknął widoczny nagłówek. Komercyjne witryny, takie jak *Best Buy*, powszechnie korzystają z tego rozwiązania (patrz rysunek 9.8). Kiedy informacje zostaną rozdzielone na karty, użytkownicy będą w stanie znaleźć wszystko, czego potrzebują — na przykład specyfikację techniczną produktu, jego recenzje, opcje zapłaty i wysyłki — a jednocześnie nie będą przytłoczeni nadmiarem informacji prezentowanych na stronie.



Rysunek 9.8. *Karty są często stosowane na witrynach, które muszą prezentować bardzo dużo informacji. Przykładowo powszechnie są one stosowane na witrynach zajmujących się handlem elektronicznym, dzięki czemu potencjalni klienci nie są przytłaczani ilością dostępnych informacji, a mimo to mogą szybko dotrzeć do poszukiwanych informacji*

Zestawy kart jQuery UI (podobnie jak większość innych widżetów tej biblioteki) tworzy się w bardzo prosty sposób. Najważniejsza jest znajomość struktury kodu HTML, bo jQuery UI wymaga, by kod HTML zestawu kart miał ściśle określoną postać; konkretnie rzecz biorąc, musi się składać z trzech głównych komponentów. Oto one.

- **Element `<div>` pełniący rolę głównego kontenera.** Cały zestaw kart i paneli treści musi być umieszczony wewnątrz jednego elementu. Nie musi to być koniecznie element `div`, choć faktycznie jest najczęściej stosowany. To ten

element jest wybierany przy użyciu jQuery i on następnie informuje, gdzie należy szukać nagłówków kart i paneli treści. Warto więc dodać do tego elementu identyfikator, aby można go było łatwo wybrać.

Uwaga: Jeśli na jednej stronie ma być wyświetlonych kilka zestawów kart, warto do każdego z kontenerów dodać tę samą nazwę klasy, na przykład `<div class="tabbedPanels">`. Dzięki temu wszystkie elementy div pełniące rolę kontenerów dla zestawu kart będzie można wybrać za pomocą jednego wywołania `$('.tabbedPanels')` i utworzyć te zestawy, korzystając z tylko jednego wiersza kodu:

```
$('.tabbedPanels').tabs();
```

- **Nagłówki kart.** Nagłówki kart można utworzyć w formie listy wypunktowanej lub numerowanej. Każdy nagłówek karty jest reprezentowany przez jeden znacznik `<li>`. Wewnątrz każdego znacznika `<li>` należy umieścić znacznik `<a>`, którego atrybut `href` będzie zawierał identyfikator skojarzonego z nagłówkiem panelu treści. Załóżmy na przykład, że chcemy zbudować zestaw składający się z trzech kart. Ich nagłówki można by utworzyć przy użyciu następującej listy:

```
<ul>
  <li><a href="#details">Informacje szczegółowe</a></li>
  <li><a href="#reviews">Opinie</a></li>
  <li><a href="#order">Zamówienie</a></li>
</ul>
```

- **Panele treści.** Każdy panel treści jest jednym elementem blokowym HTML. Zazwyczaj są to znaczniki `<div>`, lecz stosuje się także znaczniki `<article>`, `<section>` bądź dowolne inne znaczniki blokowe. W tym znaczniku `<div>` należy podać identyfikator, którego wartość będzie odpowiadać adresowi odnośnika podanego w nagłówku karty. Oto przykład:

```
<div id="details">
  <!-- Tu należy umieścić treść panelu -->
</div>
```

  Ten identyfikator (atrybut `id`) jest bardzo ważny. jQuery UI używa go do skojarzenia nagłówka karty z odpowiednim panelem treści. Dodatkowo przydaje się w przeglądarkach, które nie obsługują języka JavaScript, gdyż zastosowanie etykiety (ang. *named anchor*) zapewnia możliwość przejścia z elementu listy do skojarzonego z nim elementu `div`. Wewnątrz takiego panelu treści można umieścić dowolną zawartość HTML: obrazki, teksty, listy, klipy wideo i tak dalej. Zawartość panelu zostanie wyświetlona wyłącznie po kliknięciu nagłówka karty.

  Poniżej przedstawiona została kompletna struktura kodu HTML zestawu kart:

```
<div id="tabbedPanel">
  <ul>
    <li><a href="#details">Informacje szczegółowe</a></li>
    <li><a href="#reviews">Opinie</a></li>
    <li><a href="#order">Zamówienie</a></li>
  </ul>
  <div id="details">
    <!-- Kod HTML panelu 1. -->
  </div>
  <div id="reviews">
    <!-- Kod HTML panelu 2. -->
  </div>
```

```
    <div id="order">
      <!-- Kod HTML panelu 3. -->
    </div>
  </div>
```

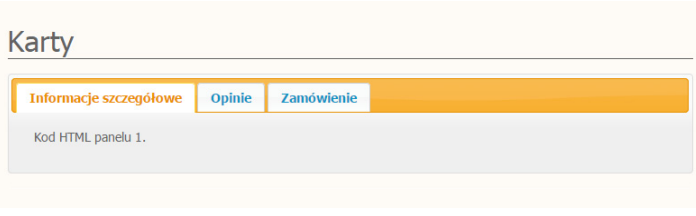
Nazwy używanych identyfikatorów są całkowicie dowolne. Dla głównego elementu kontenera nie musi to być `tabbedPanel`, podobnie dowolne wartości mogą mieć identyfikatory poszczególnych paneli treści — nie muszą to być `details`, `review` oraz `order`. Trzeba tylko pamiętać, by adresy podawane w znacznikach `<a>` na liście nagłówków kart odpowiadały identyfikatorom paneli treści.

Uwaga: Zestawy karty tworzone przez jQuery UI można także obsługiwać przy użyciu klawiatury. Użytkownik może nacisnąć strzałkę w prawo, aby wyświetlić następną kartę umieszczoną z prawej strony, oraz klawisz strzałki w lewo, by wyświetlić kartę umieszczoną po lewej stronie.

Aby przekształcić taki kod HTML na zestaw kart, wystarczy wybrać element pełniący rolę ich kontenera i wywołać funkcję `tabs()`:

```
$('#tabbedPanel').tabs();
```

Instrukcja spowoduje utworzenie kart przedstawionych na rysunku 9.9.



Rysunek 9.9. *Tworzenie karty przy użyciu jQuery UI jest naprawdę proste: wystarczy prosty kod HTML i odrobina JavaScriptu. Jeśli nie odpowiada Ci wygląd karty, możesz go zmodyfikować w internetowym narzędziu ThemeRoller (patrz strona 407)*

Opcje zestawów kart

Widżet `Tabs` tworzący zestaw karty, podobnie jak wszystkie inne widżety jQuery UI, zapewnia możliwość dostosowywania sposobu działania. Aby określić jego opcje, wystarczy przekazać w wywołaniu funkcji `tabs()` literał obiektowy, zawierający nazwy i wartości wybranych właściwości. Poniżej przedstawiono kilka najczęściej stosowanych.

- **`show`** oraz **`hide`.** Te dwie właściwości określają, w jaki sposób panele treści będą wyświetlane i ukrywane. Można w nich używać tych samych wartości, co w analogicznych właściwościach widżetu okna dialogowego. Aby na przykład panel treści był wsuwany podczas wyświetlania oraz wysuwany podczas ukrywania, w wywołaniu funkcji `tabs()` należy przekazać dwie poniższe właściwości:

  ```
  show: 'slideDown',
  hide: 'slideUp'
  ```

- **`active`.** Standardowo po wyświetleniu strony zawierającej zestaw kart prezentowana jest pierwsza z nich — nagłówek karty zostanie wyróżniony, a widoczna będzie zawartość pierwszego panelu treści. Jednak nic nie stoi na przeszkodzie,

by początkowo wyświetlić drugą, ostatnią bądź którąkolwiek inną kartę. Właśnie do tego celu służy właściwość `active`:

```
active: 1
```

Karty, podobnie jak elementy tablic JavaScript (patrz strona 77), są numerowane od 0. Oznacza to, że przypisanie właściwości `active` wartości `1` spowoduje wyświetlenie drugiej karty. Jeśli zamiast liczby przypiszemy tej właściwości wartość `false` — `active: false` — zostaną ukryte *wszystkie* panele treści. W takim przypadku któryś z paneli zostanie wyświetlony dopiero wtedy, gdy użytkownik kliknie jeden z nagłówków kart. Warto zwrócić uwagę, że możliwość ta jest dostępna wyłącznie w przypadku, gdy właściwości `collapsible` zostanie przypisana wartość `true`.

- **`collapsible`.** Tej wartości można przypisać wartość `true`, jeśli chcemy, by użytkownik mógł ukryć wszystkie panele treści. Zazwyczaj widoczny jest przynajmniej jeden z nich. Jeśli jednak właściwości `collapsible` przypiszemy wartość `true`, kliknięcie nagłówka widocznej karty spowoduje ukrycie powiązanego z nią panelu treści, a z całego zestawu kart będą widoczne wyłącznie nagłówki. Takie rozwiązanie można stosować, gdy na stronie jest naprawdę mało miejsca. Jednak w praktyce jest ono używane raczej sporadycznie, a większość użytkowników witryny nie będzie do niego przyzwyczajona. Trzeba pamiętać, że jeśli właściwości `active` przypiszemy wartość `false` — aby ukryć wszystkie panele treści — to właściwości `collapsible` koniecznie trzeba przypisać wartość `true`.
- **`event`.** Po kliknięciu nagłówka karty wyświetlany jest skojarzony z nim panel treści. A przynajmniej taki jest standardowy sposób działania tego widżetu. Jeśli jednak chcemy, by zmiana wyświetlonej karty następowała w wyniku innego zdarzenia, jego nazwę (patrz strona 179) należy podać we właściwości `event`. Poniżej pokazano, co należy zrobić, by karty były wyświetlane po wskazaniu ich nagłówka myszą:

  ```
  event: 'mouseover'
  ```

  Jednak do stosowania tej właściwości należy podchodzić z dużą ostrożność. Użytkownicy stron WWW są przyzwyczajeni do pewnych konwencji, a najpopularniejszą z nich jest ta, że aby coś się na stronie stało, należy kliknąć. Jeśli użytkownicy będą musieli dwukrotnie kliknąć nagłówek karty, aby ją wyświetlić (czyli użyć zdarzenia `dblclick`), to może się zdarzyć, że nigdy nie wpadną na to, jak dotrzeć do umieszczonych na niej treści.
- **`heightStyle`.** Właściwość kontroluje wysokość każdego z paneli treści i może przyjmować wartości `content`, `auto` oraz `fill`. Wartością domyślną jest `content`. Sprawia ona, że wysokość każdego panelu treści odpowiada wysokości jego zawartości. Jeśli jeden panel zawiera kilka akapitów tekstu, a drugi tylko jedno zdanie, podczas zmiany widocznej karty zmieni się także wysokość całego widżetu. Jeśli różnice w ilości treści na poszczególnych kartach są znaczące, zmiany wysokości związane z ich przełączeniem mogą rozpraszać użytkownika.

  Użycie opcji `auto` sprawi, że wszystkie panel treści będą miały taką samą wysokość, równą wysokości najwyższego z nich. Zatem w tym przypadku zmiana wybranej karty nie będzie miała wpływu na wysokość całego widżetu;

jednocześnie jednak na kartach, których zawartość jest krótka, będzie dużo wolnego miejsca. I w końcu ostatnia z opcji — `auto` — powoduje, że wysokość całego widżetu zostanie dostosowana do wysokości elementu strony, w którym został umieszczony. Użycie tej opcji sprawia, że zazwyczaj na każdej z kart jest dużo wolnego miejsca, co oznacza marnowanie cennego obszaru strony, dlatego tej opcji lepiej unikać.

Miniprzykład — dodawanie zestawu kart

W tym przykładzie przedstawiony zostanie proces dodawania do strony zestawu kart. Jego najtrudniejszym elementem jest przygotowanie odpowiedniego kodu HTML. Kod JavaScript jest bardzo prosty.

1. **W edytorze tekstów otwórz plik *tabs.html*.**

   Do tej przykładowej strony zostały już dołączone niezbędne pliki CSS i JavaScript. W ramach dodatkowego ułatwienia nie będziesz także musiał wpisywać całego kodu HTML niezbędnego do utworzenia kart. (Wymagane elementy tego kodu zostały opisane na stronie 352). Jednak przygotowany na stronie kod nie jest kompletny, więc będziesz musiał go uzupełnić. Konkretnie rzecz biorąc, brakuje identyfikatora kontenera zawierającego wszystkie karty oraz odnośników prowadzących z nagłówków karty do paneli treści.

2. **W kodzie HTML strony odszukaj znacznik `<h1>Zestaw kart</h1>`. Do znacznika `<div>` umieszczonego poniżej tego nagłówka dodaj atrybut `id="tabContainer"`:**

   ```
   <h1>Zestaw kart</h1>
   <div id="tabContainer">
   ```

   Widżet zestawu karty jQuery UI jest tworzony w elemencie strony zawierającym zestaw nagłówków kart oraz paneli treści — zazwyczaj jest to znacznik `<div>`. Dodając do niego identyfikator, zyskujesz możliwość wybrania tego elementu przy użyciu jQuery i wywołania na jego rzecz funkcji `tabs()`. A teraz zajmiesz się dodaniem odnośników.

3. **Odszukaj wypunktowaną listę (znacznik `<ul>`) umieszczoną poniżej otwierającego znacznika `<div>`, który zmodyfikowałeś w poprzednim kroku. Do każdego ze znaczników `<a>` umieszczonych w elementach tej listy dodaj odwołanie do etykiety:**

   ```
   <li><a href="#panel1">Karta 1.</a></li>
   <li><a href="#panel2">Karta 2.</a></li>
   <li><a href="#panel3">Karta 3.</a></li>
   ```

   To są odnośniki do etykiet — prowadzą one do konkretnych miejsc strony, określonych na podstawie podanych identyfikatorów. Innymi słowy, odnośnik umieszczony w pierwszym znaczniku `<li>` prowadzi do znacznika `<div>` pierwszego panelu treści, odnośnik umieszczony w drugim elemencie listy — do znacznika `<div>` drugiego panelu treści i tak dalej. Aby te odnośniki zaczęły działać prawidłowo, musisz dodać odpowiednie identyfikatory do znaczników `<div>` tworzących panele treści.

PORADNIA DLA ZAAWANSOWANYCH

Niestandardowe zdarzenia jQuery UI

Na stronach od 178 do 182 poznałeś standardowe zdarzenia przeglądarki, takie jak `click`, `mouseover`, `focus` czy też `resize`. Zdarzenia te są wbudowane w przeglądarkę i generowane, kiedy użytkownik wykona na stronie odpowiednie czynności, takie jak kliknięcie odnośnika lub przesłanie formularza. Zdarzenia są bardzo użyteczne, gdyż pozwalają na pisanie programów, które odpowiadają na czynności wykonywane przez użytkownika na stronie.

Widżety jQuery UI mają swoje własne zdarzenia — nie są one jednak dokładnie takie same jak zdarzenia generowane przez przeglądarkę, do których jesteśmy przyzwyczajeni. Odpowiadają one po prostu określonym etapom tworzenia, wykonywania oraz kończenia działania widżetu.

Przykładowo widżet zestawu kart udostępnia zdarzenie `beforeActivate`, pozwalające określić kod, który zostanie wykonany bezpośrednio przed wyświetleniem niewidocznej wcześniej karty. Może się ono przydać, jeśli na przykład będziemy chcieli wykonać jakąś operację za każdym razem, gdy użytkownik kliknie nagłówek karty. Zdarzenia `beforeActivate` możemy użyć do modyfikowania adresu wyświetlonego w pasku adresu przeglądarki za każdym razem, gdy zostanie kliknięty nagłówek karty, tak by był do niego dodawany znak # oraz identyfikator panelu; na przykład *`tabs.html#panel3`*. Stosując to rozwiązanie wraz z instrukcją warunkową podaną w kroku 9. na stronie 359, mógłbyś zapewnić użytkownikom możliwość zapisania adresu strony powodującego wyświetlenie odpowiedniej karty.

W tym celu musiałbyś przekazać w wywołaniu funkcji `tabs()` właściwość `beforeActivate` zawierającą odpowiednią funkcję, taką jak przedstawiona na przykładzie:

```
$('#tabContainer').tabs({
  beforeActivate: function(evt) {
    location.hash =
$(evt.currentTarget).attr('href');
  }
});
```

W tym kodzie dzieje się ca kiem sporo, jednak ogólnie rzecz biorąc, odnajduje on atrybut `href` kliknięty nagłówka karty (`#panel1`, `#panel2` lub `#panel3`) i zapisuje jego wartość we właściwości `hash` obiektu `location` (więcej informacji na jej temat można znaleźć w kroku 8. na stronie 359). Działającą wersję takiej strony możesz znaleźć w pliku *complete_tabs_with_custom_events.html* w katalogu *R09*.

Nie tylko widżet kart, lecz także wszystkie inne widżety jQuery UI zapewniają wiele sposobów wykonywania różnych operacji podczas ich tworzenia, modyfikowania i usuwania. Programiści nazywają rozwiązania tego typu „punktami zaczepienia" (ang. *hook*), gdyż pozwalają dołączać własny kod do już istniejącego kodu (na przykład biblioteki jQuery UI). Stosowanie tych rozwiązań jest dosyć złożonym, lecz jednocześnie intersującym zagadnieniem.

Najlepszym sposobem poznania tych niestandardowych zdarzeń widżetów jQuery UI jest poszukanie informacji na ich temat w dokumentacji API (API to skrót oznaczający *interfejs programowania aplikacji*, ang. *application programming interface* i określa zbiór wszystkich właściwości i funkcji, które są dostępne dla programisty). Na początku każdej strony znajduje się ramka *QuickNav*, a w niej, w kolumnie *Events*, lista wszystkich zdarzeń obsługiwanych przez dany widżet. Przykładowo strona dokumentacji widżetu `Dialog` (*http://api.jqueryui.com/dialog/*) przedstawia listę 11 zdarzeń, z których możemy skorzystać!

4. **Odszukaj znacznik `<div>` umieszczony tuż poniżej znacznika `<ul>` modyfikowanego w poprzednim kroku (przed nim znajduje się komentarz `<!-- Panel 1 -->`, dzięki czemu łatwo go zauważysz). Dodaj do niego atrybut `id="panel1"`:**

```
<!-- Panel 1 -->
<div id="panel1">
```

W taki sam sposób będziesz musiał dodać identyfikatory do pozostałych dwóch paneli treści.

5. **Powtórz czynności opisane w kroku 4., aby dodać identyfikatory do pozostałych dwóch paneli treści (łatwo je znajdziesz dzięki umieszczonym przed nimi komentarzom).**

Upewnij się, że do znaczników `<div>` dodałeś odpowiednie identyfikatory, pasujące do odnośników dodanych w kroku 3. Przykładowo znacznik `<div>` drugiego panelu treści powinien mieć postać: `<div id= panel2 >`. Teraz przekształcisz cały kod HTML na zestaw kart.

6. **Wewnątrz wywołania funkcji `$(document).ready()` dodaj wywołanie funkcji `tabs()`:**

```
$(document).ready(function() {
  $('#tabContainer').tabs();
}); // Koniec funkcji ready.
```

Zapisz plik w przeglądarce, po czym wyświetl go w tejże przeglądarce. Strona powinna wyglądać tak, jak na rysunku 9.10. Jeśli wygląda inaczej, wyświetl okno konsoli JavaScript (patrz strona 51) i sprawdź, czy są w nim widoczne jakieś komunikaty o błędach. Jeśli nie ma żadnych informacji o błędach, sprawdź, czy w kodzie HTML, w znacznikach `<div>` paneli treści zostały podane dobre identyfikatory.

Rysunek 9.10. *Gotowa strona prezentująca wygląd zestawu kart*

W następnym kroku dodasz efekty, które będą odtwarzane podczas zmiany prezentowanej karty; konkretnie rzecz biorąc, panel treści wyświetlanej karty będzie się stopniowo pojawiał, a karty, która była widoczna poprzednio — stopniowo zanikał.

7. **W wywołaniu funkcji `tabs()` wpisz następujący literał obiektowy (wyróżniony pogrubioną czcionką):**

```
$(document).ready(function() {
  $('#tabContainer').tabs({
    show: 'fadeIn',
    hide: 'fadeOut'
  });
}); // Koniec funkcji ready.
```

W ten sposób dodasz efekty przejść, które będą odtwarzane przy przełączaniu kart. Możesz wypróbować różne efekty, na przykład `'slideDown'` oraz `'slideUp'`. Zapisz stronę i wyświetl ją w przeglądarce.

Zestaw kart działa bardzo dobrze, choć jest z nim jeden drobny problem: bezpośrednio po wyświetleniu strony jest w nim zawsze widoczna pierwsza karta bądź karta określona przy użyciu właściwości `active` (patrz strona 354). Co mógłbyś zrobić, gdybyś chciał wysłać w e-mailu adres strony i to taki adres, którego kliknięcie spowoduje wyświetlenie konkretnej karty? Przykładowo wyobraź sobie, że pracujesz w dziale obsługi klienta i ktoś poprosił o przesłanie adresu strony ze specyfikacją techniczną jednego z oferowanych produktów. Na firmowej witrynie istnieje strona z taką specyfikacją umieszczoną na jednej z kart; problem polega jednak na tym, że domyślnie wyświetlana jest zawsze karta „Informacje o produkcie".

Gdybyś tak mógł przesłać adres w postaci *http://stronafirmowa.com.pl/produktA.html#specs*, a po przejściu na tę stronę zostałaby wyświetlona karta ze specyfikacją techniczną produktu? Cóż, można tak zrobić, choć wymaga to zastosowania prostej magii — fragmentu kodu JavaScript. Cała tajemnica polega na pobraniu z adresu URL fragmentu `#specs` i zastosowaniu go do wyświetlenia odpowiedniej karty.

8. **Poniżej wywołania funkcji `tabs()` dodaj nowy wiersz i wpisz w nim `var hash = location.hash;`.**

   Obiekt `window` udostępnia tak zwany obiekt `location`, zawierający wiele informacji dotyczących adresu URL aktualnie prezentowanej strony, w tym nazwę komputera (`location.hostname`), cały adres URL (`location.href`) oraz wiele innych (pełną listę dostępnych właściwości tego obiektu można znaleźć na stronie: *https://developer.mozilla.org/en-US/docs/Web/API/Location*). Właściwość `location.hash` zwraca jedynie tę część adresu, która zawiera fragment rozpoczynający się od znaku #.

   Przykładowo załóżmy, że podałeś adres *http://stronafirmowa.com.pl/produktA.html#specs*. W takim przypadku właściwość `location.hash` będzie zawierać fragment `#specs`. W następnym kroku skorzystasz z tej właściwości, by wyświetlić kartę odpowiadającą wartości podanej w adresie URL.

9. **Poniżej kodu wpisanego w poprzednim kroku dodaj instrukcję warunkową, tak by ostateczna postać skryptu miała następującą postać:**

```
$(document).ready(function() {
  $('#tabContainer').tabs({
    show: 'fadeIn',
    hide: 'fadeOut'
  });
  var hash = location.hash;
  if (hash) {
    $('#tabContainer').tabs('load', hash)
  }
}); // Koniec funkcji ready.
```

   Dodana instrukcja najpierw sprawdza, czy w adresie URL bieżącej strony znajduje się fragment zapisany po znaku #. Jeśli na przykład użytkownik wpisał jedynie adres strony w postaci *tabs.html*, takiego fragmentu w nim nie będzie. W takim przypadku reszta kodu jest pomijana, a po wyświetleniu strony zostanie standardowo wyświetlona pierwsza karta. Jeśli jednak użytkownik dopisał do adresu na przykład: `#panel1`, zostanie wykonany kod umieszczony wewnątrz instrukcji warunkowej. Całe jego działanie sprowadza się do wybrania elementu kontenera zestawu kart (`$('#tabContainer')`), wywołaniu funkcji

`tabs()` i przekazaniu do niej dwóch argumentów. Pierwszym z nich jest łańcuch znaków `'load'` — wbudowane polecenie jQuery UI, nakazujące funkcji `tabs()` wyświetlenie konkretnego panelu treści. Drugi argument, `hash`, zawiera identyfikator tego panelu, na przykład: `#panel`, `#panel2` lub `#panel3`.

10. **Zapisz stronę i wyświetl ją w przeglądarce.**

    Powinna zostać wyświetlona pierwsza karta; następnie w pasku adresu przeglądarki dopisz na końcu adresu strony `#panel3` (za *tabs.html*). Naciśnij klawisz *Enter*, aby wyświetlić stronę.

    Powinno to spowodować wyświetlenie trzeciej karty. (Jeśli tak się nie stało, spróbuj skopiować adres wyświetlony w pasku adresu, otworzyć nowe okno przeglądarki i wkleić do niego adres). Kompletna wersja przykładu znajduje się w pliku *complete_tabs.html* umieszczonym w katalogu *R09*.

Karty prezentujące zawartość

Biblioteka jQuery UI pozwala nawet pobierać zawartość wyświetlaną na poszczególnych kartach z innych stron WWW. Innymi słowy, zamiast tworzenia wypunktowanej listy odnośników prowadzących do elementów `div` na *tej samej stronie*, można utworzyć listę odnośników wskazujących na *inne* strony (bądź też na treści generowane przez serwer WWW). Z tego rozwiązania można skorzystać, gdy zawartość każdej z kart nieustannie się zmienia (są to notowania giełdowe, recenzje, wpisy na forum i tak dalej). Wyświetlając na karcie treści generowane dynamicznie — na przykład informacje pobierane z często aktualizowanej bazy danych przy użyciu takich technologii jak PHP, .NET czy też Ruby on Rails — można mieć pewność, że będą one cały czas aktualne.

Aby pobrać i wyświetlić w panelu treści zawartość uzyskaną z innej strony WWW lub z serwera, wystarczy utworzyć główny znacznik `<div>` zawierający zestaw kart, listę wypunktowaną z odnośnikami do odpowiednich stron i wywołać funkcję `tabs()`. Załóżmy, że chcemy, by każdy panel prezentował zawartość odrębnej strony WWW, a konkretnie rzecz biorąc — stron: *panel1.html*, *panel2.html* oraz *panel3.html*. W takim przypadku wypunktowana lista z odnośnikami powinna mieć następującą postać:

```
<div id="tabContainer">
  <ul>
    <li><a href="panel1.html">Karta 1</a></li>
    <li><a href="panel2.html">Karta 2</a></li>
    <li><a href="panle3.html">Karta 3</a></li>
  </ul>
</div>
```

Może się zdarzyć, że wyświetlając na kartach treści generowane dynamicznie, nie będziemy odwoływali się do stron, lecz do skryptów wykonywanych na serwerze i napisanych na przykład w języku PHP:

```
<div id="tabContainer">
  <ul>
    <li><a href="reviews.php?id=1298">Aktualne recenzje</a></li>
    <li><a href="forum.php?id=1298">Dyskusja na forum</a></li>
  </ul>
</div>
```

W przypadku budowania takich zestawów kart nie trzeba tworzyć znaczników `<div>` opisanych w kroku 3. na stronie 356. Zostaną one utworzone automatycznie, podczas generowania paneli treści. Aby opracować zestaw kart, wystarczy, tak samo jak wcześniej, wybrać element `<div>` i wywołać funkcję `tabs()`:

```
$('#tabContainer').tabs();
```

W ramach wykonywania funkcji `tabs()` jQuery UI pobierze kod HTML strony, która ma być prezentowana na panelu treści pierwszej widocznej karty. Bazując na powyższym przykładzie, podczas prezentowania strony jQuery UI pobierze zawartość pliku *panel1.html* i wyświetli ją w panelu treści, poniżej nagłówków kart. Kiedy użytkownik kliknie drugą kartę, jQuery UI pobierze zawartość strony wskazanej przez drugi odnośnik na liście i wygeneruje nowy panel treści oraz wyświetli w nim pobrany kod HTML.

W większości przypadków można nawet odwoływać się do zewnętrznych witryn, takich jak Google, Wikipedia, czy też do serwerów innych niż ten, z którego pochodzi strona prezentująca zestaw kart. Jednak te zewnętrzne witryny mogą blokować takie odwołania i w takim przypadku treści, takie jak czcionki, obrazy lub klipy wideo, nie będą wczytywane.

Takie rozwiązanie niesie ze sobą jeszcze jeden problem — w panelu treści zostanie wyświetlona *cała* zawartość pobranej strony. A zatem, jeśli odwołamy się do *kompletnej* strony WWW, zawierającej znaczniki `<header>`, `<title>`, odwołania do plików CSS i JavaScript, to wszystkie te treści i pliki zewnętrzne zostaną wczytane. W efekcie powstanie strona, wyświetlona wewnątrz strony. Jeśli jednak chcemy jedynie pobrać i wyświetlić w panelu treści danej karty fragment zawartości zewnętrznej strony, możemy to zrobić na dwa sposoby.

Łatwiejszym jest utworzenie *fragmentów* strony — plików HTML zawierających *wyłącznie* te fragmenty treści, które chcemy wyświetlać na kartach. Utworzenie takich fragmentów nie przysporzy żadnych problemów w przypadku pobierania stron dynamicznie z serwera WWW — w takim przypadku skrypt działający na serwerze powinien wygenerować wyłącznie ten fragment treści, który chcemy wyświetlić (natomiast wszystkie pozostałe fragmenty strony, takie jak sekcja `<head>` wymagana przez kompletną stronę WWW, powinny zostać pominięte).

Ewentualnie można także wczytać pełną stronę i ograniczyć treści, które pojawią się na panelu treści przy użyciu *zdarzeń niestandardowych* (patrz ramka na stronie 357). A tak działa takie rozwiązanie: widżet zestawu kart jQuery UI udostępnia zdarzenie o nazwie `load`. Pozwala ono na wykonanie funkcji bezpośrednio po wczytaniu treści z zewnętrznego źródła. Wewnątrz tej funkcji można użyć jQuery do odszukania tych fragmentów, które chcemy wyświetlić, następnie je pobrać i umieścić w panelu. Rozwiązanie to jest nieco złożone, lecz nie wymaga tworzenia bardzo rozbudowanego kodu.

Najpierw należy upewnić się, że istnieje możliwość pobrania konkretnych treści z zewnętrznej strony WWW. Prostym sposobem, by zapewnić tę możliwość, jest umieszczenie wybranego fragmentu treści wewnątrz znacznika `<div>` o jakimś identyfikatorze, na przykład `<div id= panelContent >`. Dzięki temu będzie można odwołać się do wybranego fragmentu zawartości strony bez jej niepotrzebnych fragmentów, takich jak znaczniki `<head>` i `<title>`.

Kolejną czynnością, jaką trzeba wykonać, jest przekazanie opcji `load` w wywołaniu funkcji `tabs()`. Opcja `load` określa zdarzenie niestandardowe, a jej wartością musi być funkcja anonimowa (patrz strona 168), która poinformuje jQuery, co należy zrobić po pobraniu zawartości z zewnętrznego źródła i wyświetleniu jej w panelu treści. Gdy generowane jest zdarzenie `load`, biblioteka jQuery UI utworzyła już panel treści *i* wstawiła do niego zawartość zewnętrznej strony. W tym momencie w panelu znajduje się cały kod HTML zewnętrznej strony, włącznie z jego niepożądanymi fragmentami. Jednak możemy szybko usunąć całą treść i zastąpić ją tylko tym fragmentem, który nas interesuje. Cały proces jest wykonywany tak szybko, że początkowo pobrany, pełny kod HTML zewnętrznej strony nawet nie jest wyświetlany w przeglądarce.

Oto przykładowy kod, który realizuje taką podmianę:

```
1.  $('#tabContainer').tabs({
2.    load: function(evt,ui) {
3.      var newHTML = ui.panel.find('#panelContent').html();
4.      ui.panel.html(newHTML);
5.    }
6.  })
```

Opcja `load` została zapisana w wierszach od 2. do 5. Biblioteka jQuery UI przekazuje do swoich niestandardowych zdarzeń dwie informacje (są to parametry `evt` oraz `ui` zapisane w wierszu 2.). Parametr `evt` to zwyczajny obiekt zdarzenia jQuery (patrz strona 194). Pozwala on na stosowanie dowolnych właściwości i metod zdarzenia, opisanych na stronach od 195 do 197. W tym przypadku interesuje nas drugi argument przekazany do funkcji anonimowej obsługującej zdarzenie, czyli `ui`. Obiekt `ui` reprezentuje aktualizowany element interfejsu użytkownika. W przypadku zestawów kart obiekt `ui` ma dwie właściwości: `ui.panel` oraz `ui.tab`. Pierwsza, `ui.panel`, reprezentuje nowo utworzony panel treści (czyli znacznik `<div>` utworzony przez jQuery UI podczas wczytywania treści ze źródła zewnętrznego).

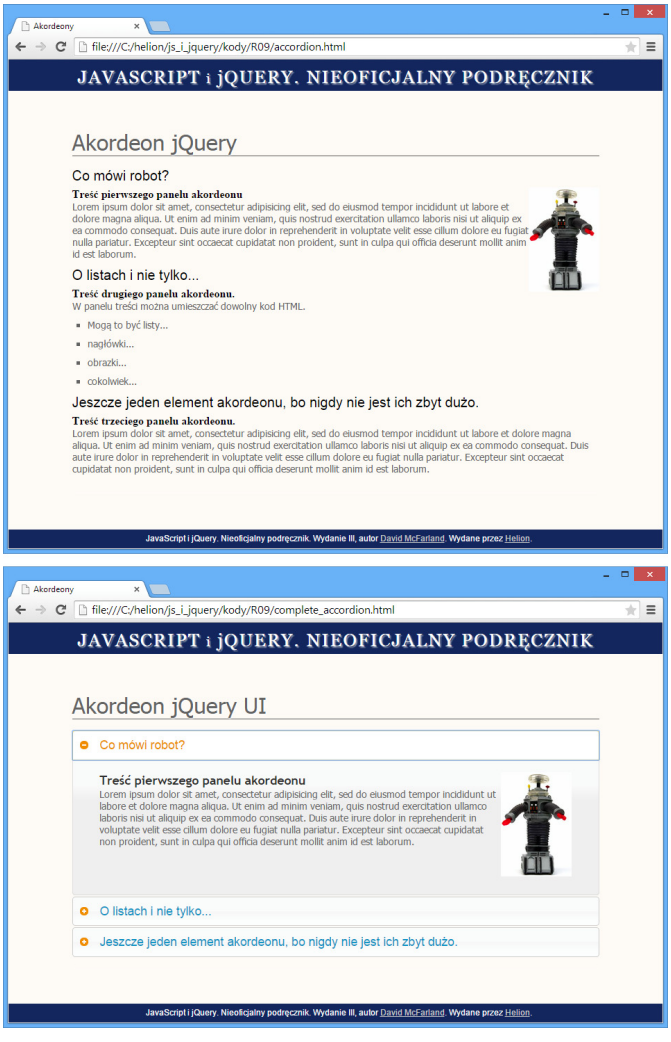
W wierszu 3. tworzymy nową zmienną, `newHTML`, w której zostanie zapisana ostateczna zawartość panelu (bez niepotrzebnego kodu HTML, takiego jak znacznik `<head>`). Kiedy jQuery UI tworzy nowy panel treści, wczytuje całą zawartość zewnętrznej strony WWW, dzięki czemu możemy ją odczytać i pobrać tylko ten fragment, który nas interesuje. Wywołanie `ui.panel.find('#panelContent')` pobiera zawartość nowego panelu treści i odnajduje wewnątrz niej element o identyfikatorze `panelContent` (metoda `find()` biblioteki jQuery została opisana na stronie 555). Następnie wywoływana jest metoda `html()` (opisana na stronie 148), która pobiera zawartość odnalezionego wcześniej elementu. Innymi słowy, powyższe wywołanie zapisuje w zmiennej `newHTML` interesujący nas fragment strony.

Uwaga: Przykład wykorzystania wyświetlania zawartości zewnętrznych stron na kartach można znaleźć w pliku *remote_tabs.html* umieszczonym w katalogu *R09*.

W wierszu 4. zawartość panelu treści (czyli cała zawartość zewnętrznej strony, włącznie z `<head>`, `<title>` oraz wszystkimi innymi niepotrzebnymi znacznikami) zostaje zastąpiona nowym kodem HTML (zawierającym tylko wybrany fragment strony). Powyższy kod jest wykonywany tak szybko, że przeglądarka nigdy nie zdąży wyświetlić na karcie pełnego kodu zewnętrznej strony — użytkownik zobaczy wyłącznie odpowiednio wybrany fragment treści.

Oszczędzanie miejsca z wykorzystaniem akordeonów

Akordeony (ang. *accordion*) jQuery UI, podobnie jak karty, są kolejnymi widżetami, których przeznaczeniem jest oszczędzanie miejsca na stronach. Przykład takiego widżetu można zobaczyć na rysunku 9.11. Jednak w odróżnieniu od kart, które są przełączane przy użyciu wiersza nagłówków, każdy z paneli akordeonu posiada własny nagłówek, umieszczony bezpośrednio nad nim, którego kliknięcie pozwala dany panel ukryć lub wyświetlić. Kliknięcie nagłówka akordeonu powoduje wyświetlanie panelu bezpośrednio poniżej i jednoczesne ukrycie panelu, który był widoczny wcześniej. Innymi słowy, także w widżecie akordeonu w danej chwili widoczny jest tylko jeden panel.



Rysunek 9.11. *Raz to widać, a raz nie. Biblioteka jQuery UI potrafi przekształcić zwyczajną kolekcję nagłówków i znaczników (widoczną u góry) na interaktywny widżet prezentujący kolekcję zwijanych i wyświetlanych paneli (u dołu)*

Ogólnie rzecz biorąc, akordeony działają bardzo podobnie do kart. Większość opcji stosowanych w obu widżetach jest taka sama i działa identycznie. Różnią się one jednak pod względem struktury kodu HTML, który jest całkowicie odmienny. Kod używany do tworzenia widżetu akordeonu składa się z trzech podstawowych fragmentów. Oto one.

- **Zewnętrzny znacznik zawierający cały widżet.** Do tego znacznika `<div>` trzeba będzie odwołać się przy użyciu jQuery, zatem warto określić w nim identyfikator lub nazwę klasy.
- **Nagłówek zawierający tekst.** Ten nagłówek jest widocznym elementem strony, który można klikać, by otwierać lub zamykać panel treści (przykładem może być pomarańczowy nagłówek „Co mówi robot?" widoczny na rysunku 9.11). Poziom użytych nagłówków nie odgrywa tu znaczenia — mogą to być znaczniki `<h2>`, `<h3>` lub inne. Jednak w całym widżecie muszą to być te same znaczniki. (Z technicznego punktu widzenia, nie muszą to nawet być znaczniki nagłówków — wystarczy dowolny element blokowy; jednak najczęściej są stosowane właśnie nagłówki).
- **Element blokowy umieszczony bezpośrednio pod nagłówkiem.** Zazwyczaj jest to znacznik `<div>` zawierający treści, które chcemy wyświetlić. Znacznik ten *musi* zostać umieszczony *bezpośrednio* za nagłówkiem.

Nagłówek oraz umieszczony bezpośrednio za nim element `div` stanowią jedną część akordeonu. Aby utworzyć więcej takich części, wystarczy dodać kolejne pary nagłówek – element blokowy. Przykładowo poniżej została przedstawiona podstawowa struktura kodu HTML akordeonu składającego się z trzech elementów:

```
<div id="accordion">
  <h3>Nagłówek wyświetla lub ukrywa pierwszy panel akordeonu</h3>
  <div>
    <!-- Zawartość pierwszego panelu akordeonu. -->
  </div>
  <h3>Nagłówek wyświetla lub ukrywa drugi panel akordeonu</h3>
  <div>
    <!-- Zawartość drugiego panelu akordeonu. -->
  </div>
  <h3>Nagłówek wyświetla lub ukrywa trzeci panel akordeonu</h3>
  <div>
    <!-- Zawartość trzeciego panelu akordeonu. -->
  </div>
</div>
```

Po przygotowaniu kodu HTML o wymaganej strukturze, dołączeniu niezbędnych plików CSS jQuery UI oraz plików JavaScript bibliotek jQuery i jQuery UI (patrz strona 133) utworzenie widżetu akordeonu jest bardzo proste — sprowadza się do wybrania elementu zawierającego opcje i wywołania funkcji `accordion()`:

```
$('#accordion').accordion();
```

Jak zwykle, jQuery UI zajmuje się wszystkimi najtrudniejszymi zadaniami i przekształca naszą prostą strukturę kodu HTML na widżet akordeonu przedstawiony w dolnej części rysunku 9.11. Podobnie jak w innych widżetach jQuery UI, także podczas tworzenia akordeonów można przekazywać obiekt zawierający opcje (wiele z nich działa dokładnie tak samo jak w przypadku zestawów kart).

- **active.** Opcja `active` działa dokładnie tak samo jak analogiczna opcja stosowana w widżecie zestawu kart (patrz strona 354). Jej wartością jest liczba określająca, który z paneli akordeonu ma być widoczny bezpośrednio po wyświetleniu strony. Aby na przykład wyświetlić drugi panel, należy użyć opcji:

  ```
  active: 1
  ```

  Podobnie jak w tablicach JavaScript, także numeracja paneli rozpoczyna się od 0. Jeśli tej właściwości zostanie przypisana wartość `false`, a wartości `collapsible` wartość `true`, zostanie utworzony akordeon pozwalający na ukrycie wszystkich paneli.

- **collapsible.** Jeśli tej opcji zostanie przypisana wartość `true`, a opcji `active` wartość `false`, bezpośrednio po wyświetleniu strony wszystkie panele akordeonu będą ukryte. Oprócz tego, jeśli tej opcji zostanie przypisana wartość `true`, nagłówki paneli będą działać jak przełączniki: jeśli panel poniżej nagłówka będzie widoczny, kliknięcie tego nagłówka spowoduje jego ukrycie; jeśli natomiast panel będzie ukryty, zostanie wyświetlony.

- **animate.** Standardowo panele akordeonu są wyświetlane i ukrywane przy użyciu efektów wsuwania i wysuwania. Ten domyślny sposób działania można wyłączyć, dzięki czemu panele treści będą się natychmiast pojawiały i znikały. W tym celu wystarczy przypisać opcji `animate` wartość `false`:

  ```
  animate: false
  ```

  Zastosowanie innych wartości pozwala na wprowadzenie kilku dalszych zmian działania widżetu. Podana liczba określi czas odtwarzania animacji (wyrażony w milisekundach). Aby naprawdę poważnie zdenerwować użytkowników, można zażądać, by panele akordeonu wyświetlały się i chowały przez boleśnie długi czas wynoszący 5 sekund:

  ```
  animate: 5000
  ```

  W tej opcji można także podać łańcuch znaków, określający nazwę jednej z kilkunastu dostępnych funkcji wyznaczających szybkość odtwarzania animacji. Zgodnie z informacjami podanymi na stronie 465, funkcje określają szybkość odtwarzania animacji w całym okresie trwania. Na przykład początkowo animacja może być odtwarzana wolno, a potem przyspieszać. By skorzystać z funkcji `easeInElastic`, należałoby użyć opcji `animate` o następującej postaci:

  ```
  animate: 'easeInElastic'
  ```

  Jednak niemal wszystkie efekty, oprócz domyślnego, dają raczej paskudne efekty.

- **event.** Opcja określa zdarzenie, które będzie powodowało wyświetlenie panelu akordeonu. Działa ona dokładnie tak samo jak w widżecie zestawu kart (patrz strona 355).

- **heightStyle.** Opcja działa tak samo jak analogiczna opcja stosowana w widżecie zestawu kart (patrz strona 355).

- **icons.** Jak widać na rysunku 9.11, jQuery UI dodaje z lewej strony nagłówków akordeonu niewielkie ikony. W nagłówku aktualnie widocznego panelu wyświetlana jest niewielka ikona strzałki w dół, natomiast w nagłówkach wszystkich pozostałych — niewidocznych — paneli wyświetlana jest ikona strzałki w prawo. Wszystkie tematy graficzne jQuery UI udostępniają obszerny

zestaw ikon (ich pełną listę można znaleźć na stronie *http://api.jqueryui.com/theming/icons/*). Zastosowane ikony można określić przy użyciu opcji `icons`, których wartością musi być literał obiektowy, podobny do przedstawionego poniżej:

```
icons : {
  header: "ui-icon-plus",
  activeHeader: "ui-icon-minus"
}
```

Literał obiektowy definiuje dwie ikony, które powinny być używane w widżecie. Właściwość `header` określa ikonę, która ma być wyświetlana w nagłówkach niewidocznych paneli akordeonu, natomiast właściwość `activeHeader` — ikonę wyświetlaną w nagłówku aktualnie wyświetlonego panelu. W powyższym przykładzie w nagłówkach paneli byłyby wyświetlanie niewielkie ikony „+" i „-".

Jak we wszystkich innych widżetach jQuery UI, także w widżecie akordeonu można połączyć dowolnie wiele opcji, określając w ten sposób, jak ma wyglądać i działać. Załóżmy, że chcielibyśmy zmienić zdarzenie powodujące wyświetlanie i ukrywanie paneli oraz ikony pokazywane w ich nagłówkach. Moglibyśmy to zrobić, używając wywołania funkcji `accordion()` o następującej postaci:

```
$('#accordion').accordion({
  event: 'mouseover',
  icons : {
    header: 'ui-icon-circle-plus',
    activeHeader: 'ui-icon-circle-minus'
  }
});
```

Miniprzykład — tworzenie akordeonu jQuery UI

Akordeony są bardzo podobne do zestawów kart. Większość pracy, jaką należy wykonać podczas ich stosowania, sprowadza się do sformatowania kodu HTML. Jednak struktura kodu używanego do tworzenia akordeonów jest nawet prostsza niż w zestawach kart. W tym przykładzie punktem wyjścia będzie już przygotowany plik zawierający kod HTML, który następnie przekształcisz na widżet akordeonu.

1. **Przejrzyj kod HTML, aby zobaczyć, co już zawiera. Otwórz w przeglądarce plik *accordion.html*.**

   Strona zawiera nagłówek — Akordeon jQuery — oraz prostą kolekcję nagłówków, tekstów oraz obrazków (patrz rysunek 9.11, u góry).

2. **W edytorze tekstów otwórz plik accordion.html i przeanalizuj kod HTML umieszczony poniżej znacznika `<h1>`.**

   Zwróć uwagę, że znajduje się tam znacznik `<div>` — to kontener, wewnątrz którego znajdują się wszystkie elementy przyszłego akordeonu. Wewnątrz niego zobaczysz znacznik `<h3>`, a za nim znacznik `<div>`. Pierwszy z nich, `<h3>`, reprezentuje nagłówek akordeonu, natomiast drugi, `<div>`, to panel treści. W kodzie strony znajdują się trzy takie pary znaczników `<h3>` – `<div>`, a zatem, kiedy już dodasz niezbędny kod JavaScript, powstanie widżet akordeonu zawierający trzy panele. Najpierw musisz zadbać o możliwość wygodnego wybrania znacznika `<div>` zawierającego cały kod akordeonu.

3. **Odszukaj znacznik `<div>` umieszczony poniżej znacznika `<h1>` i dodaj do niego atrybut `id`:**

```
<div id="accordion">
```

To cały kod HTML, który trzeba dodać do strony, by utworzyć widżet akordeonu. Teraz musisz zająć się kodem JavaScript.

4. **W wywołaniu `$(document).ready()` wybierz znacznik `<div>` zawierający kod HTML akordeonu i wywołaj funkcję `accordion()`:**

```
$(document).ready(function() {
  $('#accordion').accordion();
}); // Koniec funkcji ready.
```

Zapisz plik i wyświetl go w przeglądarce. Strona powinna wyglądać niemal tak samo jak przedstawiona u dołu rysunku 9.11. Dlaczego zatem ten przykład jest określony jako „mini"? To jQuery UI sprawia, że tworzenie akordeonów jest tak proste. Jednak Twoja praca jeszcze się nie skończyła. Chcesz, by bezpośrednio po wyświetleniu strony wszystkie panele akordeonu były niewidoczne.

5. **W wywołaniu funkcji `accordion()` dodaj poniższy literał obiektowy (wyróżniony pogrubioną czcionką):**

```
$(document).ready(function() {
  $('#accordion').accordion({
    active: false,
    collapsible: true
  });
}); // Koniec funkcji ready.
```

Kiedy wyświetlisz stronę w przeglądarce, przekonasz się, że wszystkie panele akordeonu początkowo będą niewidoczne. Kliknij jeden z nagłówków, aby wyświetlić panel treści; kiedy ponownie klikniesz ten sam nagłówek, panel zostanie ukryty. Panele akordeonu są ukrywane po kliknięciu ich nagłówka wyłącznie w przypadku, gdy opcji `collapsible` została przypisana wartość `true`. Spróbuj teraz zastosować w akordeonie jakieś inne ikony, wybrane spośród obszernego zbioru ikon udostępnianych przez jQuery UI (ich pełną listę możesz znaleźć na stronie *http://api.jqueryui.com/theming/icons/*).

6. **Do opcji przekazywanych w wywołaniu funkcji `accordion()` dodaj kolejną, czyli `icons`. Jej wartością powinien być kolejny literał obiektowy, zawierający dwie właściwości:**

```
$(document).ready(function() {
  $('#accordion').accordion({
    active: false,
    collapsible: true,
    icons: {
      header: 'ui-icon-circle-plus',
      activeHeader: 'ui-icon-circle-minus'
    }
  });
}); // Koniec funkcji ready.
```

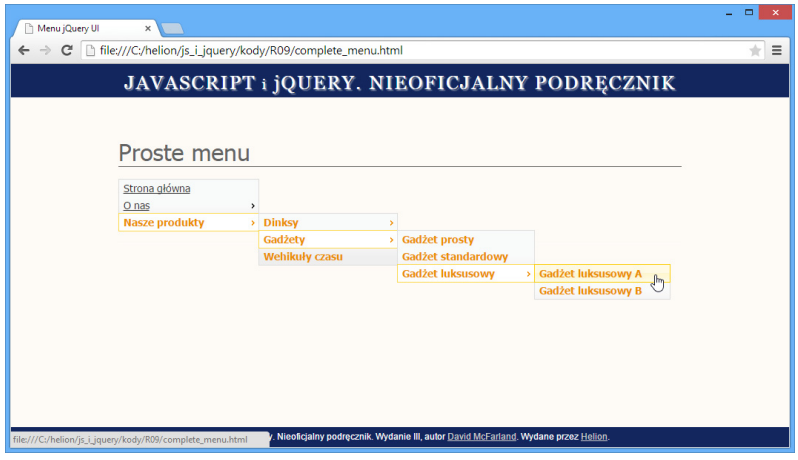
Nie zapomnij dodać przecinka za wartością `true` w trzecim wierszu kodu. Jak widać, JavaScript pozwala umieszczać jeden literał obiektowy wewnątrz innych. Czasami analizowanie takiego kodu może być dosyć trudne, trzeba jednak pamiętać, że literały obiektowe można traktować jak wszystkie inne obiekty — zmienne, liczby lub łańcuchy znaków — i używać jak wartości przypisywanych zmiennym lub właściwościom obiektów.

7. **Zapisz stronę i wyświetl ją w przeglądarce.**

   Strona powinna wyglądać tak, jak ta w dolnej części rysunku 9.11. Jeśli wygląda inaczej, dokładnie przejrzyj kod i sprawdź, jak wyglądała strona po wykonaniu czynności opisanych w kroku 4. Końcową wersję tego przykładu można znaleźć w pliku *complete_accordion.html* umieszczonym w katalogu *R09*.

Dodawanie menu

W skład jQuery UI wchodzi także widżet `Selectmenu` pozwalający na przekształcanie wypunktowanych, zagnieżdżonych list w rozwijalne menu. Z założenia, menu prezentowane jest w formie pionowej kolumny, w której kolejne opcje znajdują się jedna nad drugą, a podmenu są wyświetlane z prawej strony (patrz rysunek 9.12). Jeśli jednak ograniczymy się do jednego zestawu podmenu, można skłonić widżet do prezentowania zawartości głównego poziomu menu w układzie poziomym.



Rysunek 9.12. *Widżet menu jQuery UI pozwala na bardzo proste tworzenie wielopoziomowych menu, które swoim wyglądem odpowiadają innym widżetom jQuery UI. Doskonale nadaje się on do tworzenia menu aplikacji internetowych, które upodobnią je do tradycyjnych aplikacji komputerowych*

Podobnie jak wszystkie pozostałe widżety jQuery UI, także menu jest bardzo proste w użyciu. Najtrudniejszym zadaniem związanym z budową menu jest utworzenie odpowiedniej struktury kodu HTML. Z kodem o takiej samej strukturze spotkałeś się już przy okazji prezentowania wtyczki jQuery SmartMenus, na stronie 270. Tworzenie tego kodu zaczynamy od zapisania prostej listy wypunktowanej zawierającej odnośniki — staną się one przyciskami głównego poziomu menu, prezentowanymi bezpośrednio po wyświetleniu strony. Jeśli zechcemy dodać do jednego z tych przycisków rozwijane podmenu, wystarczy umieścić w wybranym elemencie `<li>` kolejną listę wypunktowaną. Poniższy przykład przedstawia menu zawierające trzy opcje poziomu głównego oraz podmenu, wyświetlane po wskazaniu myszą ostatniej opcji:

```
<ul id="mainMenu">
  <li><a href="about.html">O nas</a></li>
  <li><a href="contact.html">Kontakt</a></li>
  <li><a href="products.html">Produkty</a>
    <ul>
      <li><a href="a.html">Produkt A</a></li>
```

```
        <li><a href="b.html">Produkt B</a></li>
        <li><a href="c.html">Produkt C</a></li>
      </ul>
    </li>
  </ul>
```

Kiedy powyższy kod HTML zostanie skonwertowany na menu, użytkownik będzie mógł wskazać myszą opcję *Produkty*, a w efekcie zostanie wyświetlone podmenu zawierające trzy kolejne opcje. Jak we wszystkich widżetach jQuery UI, także podczas stosowania menu w elemencie zawierającym cały kod HTML widżetu warto podać identyfikator. Tu elementem tym będzie pierwszy znacznik `<ul>`, gdyż właśnie w nim znajdują się wszystkie opcje poziomu głównego oraz podmenu.

Uwaga: Aby wizualnie zorganizować opcje w menu, można dodać separator, który rozdzieli opcje w podmenu. W tym celu należy dodać znacznik <li>, w którym nie będzie odnośnika, a jedynie znak „-" (minus):

```
<li>-</li>
```

W takim przypadku, zamiast dodawać przycisk, jQuery UI wyświetli poziomą linię.

Stosowanie tego widżetu jest całkiem proste.

1. **Do tworzonej strony WWW dołącz plik CSS jQuery UI oraz pliki JavaScript jQuery i jQuery UI.**

   To te same podstawowe czynności związane ze stosowaniem wszystkich widżetów jQuery UI, które zostały opisane na stronie 133.

2. **Dodaj do strony wypunktowaną listę odnośników wraz z ewentualnymi kolejnymi, zagnieżdżonymi listami wypunktowanymi dla podmenu.**

   Warto upewnić się, że odnośniki umieszczone w opcjach głównego poziomu menu prowadzą do głównych stron witryny.

3. **Dodaj kod CSS ograniczający wielkość opcji głównego poziomu menu oraz podmenu.**

   Domyślny kod CSS stosowany przez widżet menu jQuery UI nie ogranicza szerokości opcji menu, dlatego też mogą one być wyjątkowo szerokie. Aby ograniczyć szerokość opcji głównego poziomu menu, należy utworzyć regułę stylu dla selektora `.ui-menu`, a w niej określić wartość właściwości `width`:

   ```
   .ui-menu {
     width: 10em;
   }
   ```

   Klasa `ui-menu` jest automatycznie stosowana przez jQuery UI podczas tworzenia widżetu menu. Jest dodawana do każdego znacznika `<ul>` wchodzącego w skład menu — zarówno menu głównego, jak i wszystkich podmenu.

   Określając szerokość menu, można stosować dowolne jednostki miary — `em`, `px` oraz `%` — choć w przypadku wartości procentowych należy zachować dużą ostrożność. Ponieważ szerokość każdego podmenu jest określana względem elementu rodzica, zatem każdy kolejny poziom podmenu będzie odpowiednio węższy od poprzedniego. Aby rozwiązać ten problem, wszystkie podmenu (czyli znaczniki `<ul>` umieszczone wewnątrz innego znacznika `<ul>`) powinny mieć szerokość `100%` (czyli taką samą jak ich rodzic). Oto przykład:

```
.ui-menu {
  width: 25%;
}
.ui-menu .ui-menu {
  width: 100%;
}
```

Ten kod CSS powinieneś dodać do arkusza stylów witryny, a nie jQuery UI. Jeśli w przyszłości zdecydujesz się zmienić używany temat graficzny lub skorzystać z nowszej wersji jQuery UI, wszystkie zmiany wprowadzone w pliku CSS jQuery UI zostaną stracone. Więcej informacji o stosowaniu stylów do określania wyglądu widżetów jQuery UI znajdziesz w rozdziale 11.

4. **Wybierz zewnętrzny znacznik `<ul>` menu i wywołaj metodę `menu()`:**

```
$('#mainMenu').menu();
```

Uwaga: Kompletny przykład widżetu menu można znaleźć w pliku *complete_menu.html* umieszczonym w katalogu *R09*.

Widżet menu, podobnie jak inne widżety jQuery UI, udostępnia kilka opcji pozwalających na dostosowywanie jego działania i wyglądu. Opcje te, opisane na poniższej liście, należy zapisywać w literale obiektowym, przekazywanym w wywołaniu funkcji `menu()`.

- **`icons`.** Jak widać na rysunku 9.12, jQuery UI dodaje niewielkie ikony z prawej strony wszystkich opcji, które zawierają podmenu. Ikona ta informuje użytkowników, że dana opcja ukrywa kolejny poziom menu. W każdym temacie graficznym jQuery UI dostępny jest obszerny zestaw ikon (ich pełną listę znajdziesz na stronie *http://api.jqueryui.com/theming/icons/*). Ikonę stosowaną w widżecie można zmienić, podając jej nazwę w opcji `icons`, jak pokazano na poniższym przykładzie:

```
icons : {
  submenu: "ui-icon-circle-triangle-e"
}
```

Niestety można podać tylko jedną nazwę ikony, co oznacza, że ikony wyświetlane w opcjach menu głównego będą takie same jak prezentowane w opcjach *niższych* poziomów menu.

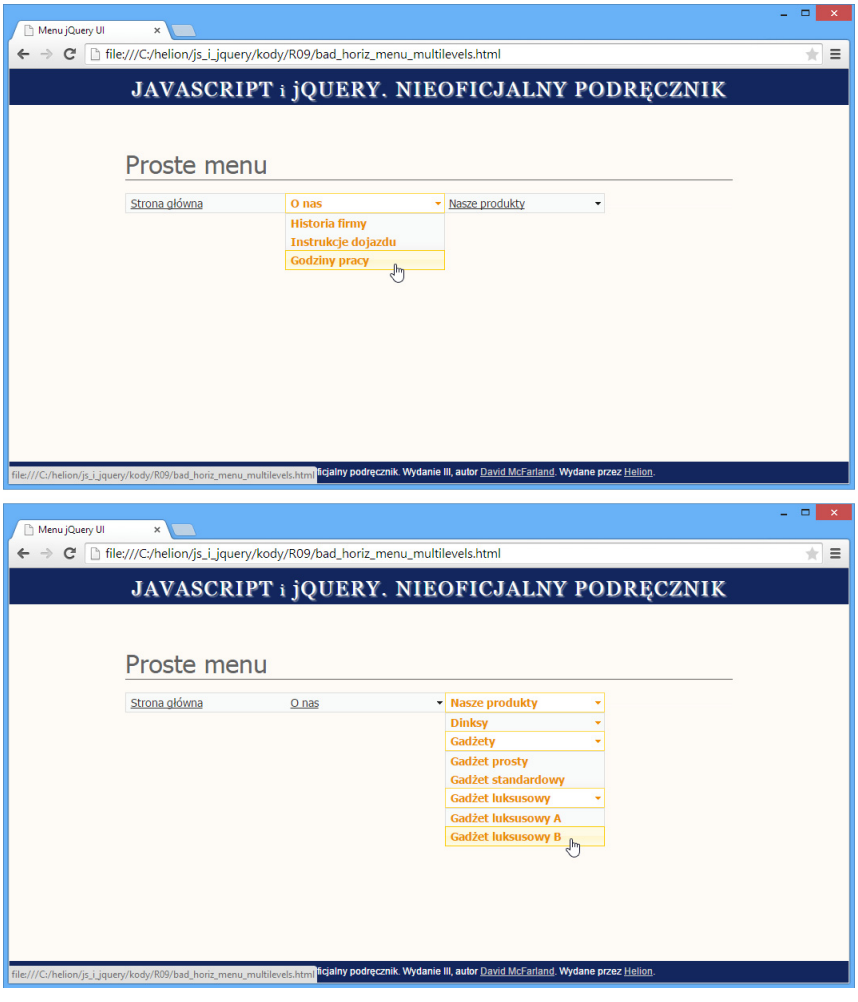
- **`position`.** Opcja `position` określa położenie podmenu względem opcji menu nadrzędnego, wewnątrz której zostały umieszczone. Zazwyczaj podmenu umieszczane są bezpośrednio z prawej strony tej opcji, jednak to położenie można zmieniać poprzez przekazanie obiektu `position`, opisanego na stronie 343. Aby na przykład pokazać podmenu bezpośrednio poniżej opcji, która je wyświetla, należałoby użyć opcji w następującej postaci:

```
position : {
  my: "center top",
  at: "center bottom"
}
```

Należy to rozumieć w następujący sposób: mój (ang. *my*, czyli podmenu) środek górnej krawędzi ma być wyświetlony przy (ang. *at*) środku dolnej krawędzi opcji menu nadrzędnego. Technika ta jest przydatna w przypadku tworzenia menu o układzie poziomego paska nawigacyjnego opisanego na stronie 373.

Tworzenie poziomego paska nawigacyjnego

Menu jQuery UI nie zostały napisane z myślą o budowaniu klasycznych pasków nawigacyjnych, takich jak te, które można zobaczyć na samej górze wszystkich przeglądarek. Jeśli właśnie o coś takiego chodzi, zapewne lepszym rozwiązaniem będzie zastosowanie wtyczki SmartMenus opisanej na stronie 270. Jednak można skłonić widżet do wyświetlenia menu poziomego, zawierającego jeden poziom podmenu; przykład takiego menu został przedstawiony w górnej części rysunku 9.13.



Rysunek 9.13. *Jeśli chcemy zachować spójność interfejsu użytkownika poprzez wykorzystanie arkusza stylów biblioteki jQuery UI oraz jej widżetów, możemy utworzyć w pełni funkcjonalne menu poziome , używając widżetu menu, o le tylko zadowoli nas jeden poziom podmenu (u góry). Widżet Selectmenu jQuery UI nie nadaje się najlepiej do tworzenia poziomych menu o wielu poziomach podmenu (u dołu). Ponieważ wszystkie podmenu są umieszczane w tym samym miejscu względem opcji menu nadrzędnego, zatem kolejne poziomy podmenu będą się wzajemnie przesłaniać. Oba rodzaje menu poziomych można znaleźć w plikach complete_horiz_menu.html oraz bad_horiz_menu_mult level.html umieszczonych w katalogu R09*

Wielopoziomowe menu w układzie poziomym nie działają najlepiej, gdyż widżet menu jQuery UI rozmieszcza wszystkie podmenu w taki sam sposób, określając ich położenie względem opcji menu nadrzędnego. W przypadku menu o układzie pionowym (patrz rysunek 9.12) takie rozwiązanie bardzo dobrze zdaje egzamin: podmenu są wyświetlane z prawej strony opcji menu nadrzędnego. Jednak w menu poziomym pierwszy poziom podmenu jest zazwyczaj wyświetlany bezpośrednio *poniżej* opcji menu nadrzędnego (patrz rysunek 9.13, u góry), natomiast kolejne poziomy podmenu — z prawej strony odpowiedniej opcji. Niestety, jeśli korzystamy z jQuery UI i jeden poziom umieścimy poniżej opcji menu nadrzędnego, *wszystkie* podmenu będą wyświetlane w taki sam sposób; w efekcie uzyskamy menu, w którym opcje kolejnych poziomów menu będą się przesłaniać, co pokazano u dołu rysunku 9.13.

1. **Aby utworzyć menu poziome, zacznij od umieszczenia na stronie wypunktowanej listy odnośników, a później dodaj do niej co najwyżej jeden poziom list zagnieżdżonych (jako przykład takiego menu może posłużyć kod HTML przedstawiony na stronie 368).**

   Teraz będziesz musiał dodać kod CSS, który sprawi, że opcje głównego poziomu menu będą wyświetlane w poziomie. Ten kod CSS powinieneś umieścić w arkuszu stylów witryny, a nie w pliku CSS jQuery UI.

2. **Dodaj kod CSS określający postać opcji głównego poziomu menu, dzięki któremu będą one wyświetlane w układzie poziomym:**

   ```
   #mainMenu > li {
     width: 10em;
     float: left;
   }
   ```

   Zastosowany selektor CSS — `#mainMenu > li` — wybiera wszystkie znaczniki `<li>`, które są bezpośrednimi potomkami elementu o identyfikatorze `mainMenu`. (Przykład zakłada, że zgodnie z kodem zamieszczonym na stronie 368, do głównego znacznik `<ul>` zawierającego całe menu dodałeś identyfikator). Znak większości (>) jest selektorem elementów dzieci i wybiera tylko te znaczniki `<li>`, które są *bezpośrednimi* dziećmi głównego znacznika `<li>`; oznacza to, że ta reguła nie będzie się odnosiła do znaczników `<li>` umieszczonych w podmenu.

   Reguła ta nadaje wszystkim opcjom menu określoną szerokość, a następnie sprawia, że będą umieszczane jedna obok drugiej. Kolejnym krokiem będzie określenie szerokości podmenu.

3. **Dodaj kolejną regułę CSS:**

   ```
   .ui-menu .ui-menu {
     width: 10em;
   }
   ```

   Ta reguła określa szerokość podmenu. jQuery UI dodaje do każdego menu — znacznika `<ul>` — klasę `.ui-menu`. Oznacza to, że powyższy selektor — `.ui-menu .ui-menu` — wybierze tylko te znaczniki `<ul>`, które są umieszczone wewnątrz innego znacznika `<ul>`. Innymi słowy, reguła ta odnosi się tylko do podmenu i określa ich szerokość.

4. **W końcu dodaj jeszcze jedną regułę, by rozwiązać problem z menu głównym:**

```
#mainMenu {
  float: left;
}
```

Ta reguła rozwiązuje problem określany jako „uciekający element pływający" (ang. *escaping float*) — chodzi o sytuację, w której wysokość elementu zawierającego wyłącznie elementy pływające jest redukowana do zera. A mówiąc po ludzku, problem ten oznacza, że obramowania i tło elementu nie wyglądają prawidłowo, gdy wszystkie elementy umieszczone wewnątrz są elementami pływającymi. Przedstawiona tu technika pozwala rozwiązać ten problem, a sprowadza się do zastosowania właściwości `float` także w elemencie nadrzędnym.

W ten sposób uporałeś się z kodem CSS, teraz dodasz odpowiedni kod JavaScript.

5. **Wywołaj metodę `menu()` w sposób opisany na stronie 370, zastosuj jednak kilka dodatkowych opcji, które określą położenie podmenu oraz zmienią używane ikony:**

```
$('#menu').menu({
  position: {
    my: 'center top',
    at: 'center bottom'
  },
  icons: {
    submenu: 'ui-icon-triangle-1-s'
  }
});
```

Opcja `position` (opisana na stronie 370) kontroluje rozmieszczenie podmenu. W tym przypadku podmenu jest wyświetlane bezpośrednio poniżej opcji menu nadrzędnego. Oprócz tego, w opcjach zawierających podmenu widżet `Selectmenu` wyświetla zazwyczaj strzałkę w prawo. Ponieważ teraz pasek menu jest wyświetlony w układzie poziomym, a podmenu są wyświetlane poniżej opcji menu głównego, dlatego zmieniłeś używaną ikonę na strzałkę w dół (`'ui-icon-triangle-1-s'`).

10

ROZDZIAŁ

Formularze raz jeszcze

Formularze to pierwsze interaktywne elementy stron WWW i cały czas, w takiej bądź innej formie, są stosowane w większości aplikacji internetowych. Formularze pozwalają na pobieranie informacji od użytkowników, umożliwiają im kupowanie towarów w sklepach internetowych, za ich pomocą członkowie internetowych społeczności mogą redagować i publikować swoje wpisy i tak dalej. Informacje zamieszczone w rozdziale 8. pokazały, jak możemy sprawić, by formularze były bardziej inteligentne i łatwiejsze w użyciu. Biblioteka jQuery UI udostępnia dodatkowe możliwości rozbudowy formularzy i zapewnia spójny projekt, dzięki któremu elementy formularzy będą wyglądały i funkcjonowały podobnie.

W tym rozdziale dowiesz się, jak używać czterech widżetów jQuery UI — kalendarza (Datepicker), automatycznie uzupełnianej listy (Autocomplete), listy wyboru (Selectmenu) oraz przycisków (Button) — które naprawdę mogą sprawić, by formularze doskonale wyglądały i działały.

Wybieranie dat ze stylem

Wiele formularzy wymaga wybierania dat. Formularze służące do dodawania zdarzeń w kalendarzu, rezerwowania lotów bądź stolika w restauracji wymagają wskazania konkretnej daty. Wiele zawiera instrukcje typu: „Wpisz datę w formacie 12.11.2014", jednak zwyczajne poproszenie użytkownika o wpisanie daty w polu tekstowym może dać bardzo różne efekty. Przede wszystkim w takim przypadku zdajemy się na to, że użytkownik nie popełni błędu. Poza tym, ponieważ osoby mieszkające w różnych krajach wpisują daty w odmienny sposób (na przykład w USA daty są zapisywane w kolejności: miesiąc, dzień, rok, natomiast w wielu innych krajach stosuje się zapis: dzień, miesiąc, rok), takie ręcznie wpisywane daty mogą być mylące i niedokładne.

Na szczęście widżet Datepicker jQuery UI sprawia, że wybieranie dat staje się banalnie proste. Zamiast wpisywać je ręcznie, użytkownik klika pole formularza, co powoduje wyświetlenie wizualnego kalendarza umożliwiającego wybór daty (patrz rysunek 10.1). Widżet kalendarza jest łatwy w użyciu i zapewnia możliwości dostosowywania jego wyglądu i działania do własnych potrzeb.

Rysunek 10.1. *Widżet kalendarza jQuery UI zapewnia możliwość prostego i wygodnego wybierania dat w formularzach*

Podobnie jak wszystkie inne widżety jQuery UI, także widżet kalendarza jest bardzo łatwy w użyciu.

1. **Wykonaj czynności opisane na stronie 329, aby dołączyć do strony plik CSS jQuery UI oraz niezbędne pliki JavaScript.**

   Do strony musisz także dołączyć plik JavaScript biblioteki jQuery. Zawsze wtedy, gdy używasz widżetów jQuery UI, do strony muszą być dołączone następujące pliki: arkusz CSS jQuery UI, plik JavaScript biblioteki jQuery oraz plik JavaScript jQuery UI (w podanej kolejności).

Uwaga: Wszystkie teksty wyświetlane w widżecie kalendarza — nazwy miesięcy oraz dni tygodnia — są domyślnie zapisane w języku angielskim. Na szczęście polonizacja kalendarza jest, jak większość czynności związanych ze stosowaniem widżetów jQuery UI, bardzo prosta. Sprowadza się do dołączenia do strony jednego pliku JavaScript, *datepicker-pl.js*, który można pobrać ze strony *https://github.com/jquery/jquery-ui/tree/master/ui/i18n* (jest on także umieszczony w przykładach dołączonych do książki, w katalogu *R10*). Plik ten musi zostać dołączony do strony za plikiem JavaScript jQuery UI.

2. **Dodaj do strony formularz, a w nim pole tekstowe `<input>`, do którego będzie wpisywana data.**

   Powinieneś także zapewnić prosty sposób odwołania się do tego pola przy użyciu jQuery. Możesz na przykład podać jego identyfikator:

   ```
   <input type="text" name="birthdate" id="birthday">
   ```

   Ewentualnie, jeśli formularz zawiera kilka pól, w jakich trzeba będzie wpisać daty (na przykład datę przyjazdu i wyjazdu), możesz użyć nazwy klasy, by określić wszystkie pola, do których należy dodać widżet kalendarza:

   ```
   <input type="text" name="arrival" class="date">
   <input type="text" name="departure" class="date">
   ```

3. **Dodaj do strony funkcję `$(document).ready()`:**

   ```
   $(document).ready(function() {

   }); // Koniec funkcji ready.
   ```

4. **Zastosuj jQuery, by wybrać element (lub elementy), do którego chcesz dodać widżet kalendarza, a następnie wywołaj funkcję** `datepicker()`**:**

```
$(document).ready(function() {
  $('#birthdate').datepicker();
}); // Koniec funkcji ready.
```

   Ewentualnie, jeśli zastosowałeś nazwę klasy, aby wyróżnić kilka pól, do których mają być dodane kalendarze (jak w kroku 2. powyżej), użyj poniższego wywołania:

```
$(document).ready(function() {
  $('.date').datepicker();
}); // Koniec funkcji ready.
```

To jedyne czynności niezbędne do utworzenia kalendarza do wyboru daty, takiego jak przedstawiony na rysunku 10.1. I jeśli to już wszystko, co widżet ma robić, to doskonale zda egzamin. Jednak widżet kalendarza jQuery UI udostępnia bardzo wiele opcji służących do modyfikowania jego wyglądu i sposobu działania.

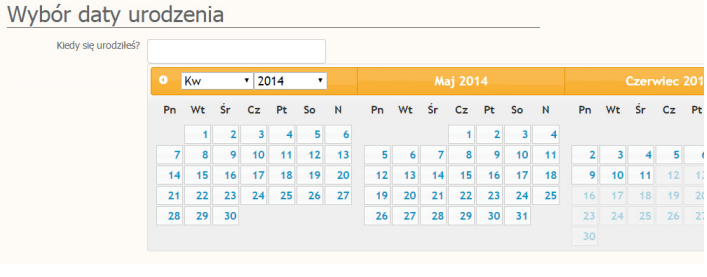
Uwaga: Język HTML5 oferuje specjalny typ pól formularzy, `date`, którego zadaniem jest udostępnianie niektórych możliwości widżetu kalendarza jQuery UI, lecz bez stosowania kodu JavaScript. Niestety, nie jest on obsługiwany przez wszystkie przeglądarki, ani nie zapewnia możliwości określania wyglądu wyświetlanego kalendarza. Co więcej, widżet jQuery UI daje wiele dodatkowych możliwości, których nie uzyskamy podczas korzystania ze standardowego pola `date` HTML5.

Określanie właściwości kalendarzy

Właściwości widżetu kalendarza — takie jak używany format zapisu dat, które będą wstawiane do pola tekstowego — można określać za pomocą przekazania literału obiektowego w wywołaniu funkcji `datepicker()`. Literał ten zawiera opcje widżetu oraz ich wartości.

I tak opcja `numberOfMonths` pozwala określić, ile miesięcy ma być widocznych w kalendarzu po wyświetleniu. Normalnie prezentowany jest tylko jeden, co pokazano na rysunku 10.1, jednak istnieje możliwość wyświetlenia do trzech miesięcy (patrz rysunek 10.2). W tym celu należy opcji `numerOfMonth` przypisać wartość `3`:

```
$('.date').datepicker({
  numberOfMonths : 3
});
```



Rysunek 10.2. *Widżet kalendarza jQuery UI zapewnia bardzo duże możliwości dostosowywania. Można w nim zmienić liczbę prezentowanych miesięcy, umieścić numer roku przed nazwą miesiąca oraz podać inne nazwy miesięcy i dni tygodnia, dostosowując go do użycia na stronach napisanych w innym języku*

Poniżej przedstawionych zostało kilka najczęściej używanych opcji konfiguracyjnych widżetu kalendarza.

- **changeMonth.** Normalnie użytkownik może zmieniać miesiąc wyświetlony w widżecie kalendarza, klikając strzałki umieszczone w jego górnej części, przy lewej i prawej krawędzi (patrz rysunek 10.1). Przyciski te powodują odpowiednio przejście do poprzedniego (lewy przycisk) oraz następnego (prawy przycisk) miesiąca. Jeśli jednak trzeba się cofnąć o dziewięć miesięcy, takie rozwiązanie może być uciążliwe i niewygodne. Przypisanie opcji `changeMonth` wartości `true` pozwala wyświetlić w widżecie rozwijaną listę, z której użytkownik może wygodnie wybrać miesiąc (patrz rysunek 10.2).

  ```
  changeMonth : true
  ```

- **changeYear.** Ta opcja przypomina opcję `changeMonth`. Jeśli przypiszemy jej wartość `true`, jQuery UI wyświetli w widżecie rozwijaną listę umożliwiającą wybranie roku. Opcja ta jest zazwyczaj używana wraz z opcją `yearRange` (patrz strona 370), pozwalającą na określenie zakresu lat, które zostaną wyświetlone na liście.

  ```
  changeYear : true
  ```

- **dateFormat.** Opcja pozwala na podanie łańcucha znaków, definiującego format, w jakim jQuery UI będzie zapisywać daty wybierane w widżecie w polu tekstowym formularza. Format ten określa się przy użyciu predefiniowanych kodów. I tak `dd` oznacza dzień miesiąca, `mm` miesiąc, a `yy` rok. W łańcuchu formatującym można także umieszczać znaki, takie jak znaki odstępu, ukośniki (/),minusy (-) bądź kropki (.). Na przykład załóżmy, że ktoś wybrał w widżecie kalendarza datę 27 stycznia 2015 roku i chcemy ją wyświetlić w formacie 27.01.2015. W takim przypadku należałoby użyć następującej opcji `dateFormat`:

  ```
  dateFormat : 'dd.mm.yy'
  ```

  Biblioteka jQuery UI udostępnia wiele kodów służących do formatowania dat. Kilka najczęściej stosowanych zostało przedstawionych w tabeli 10.1. Pełna lista wszystkich kodów formatujących stosowanych w opcji `dateFormat` widżetu kalendarza jQuery UI jest dostępna na stronie *http://api.jqueryui.com/datepicker/#utility-formatDate*.

Uwaga: Widżet kalendarza jQuery UI udostępnia także wiele innych opcji, zdarzeń i metod. Informacje o nich można znaleźć na stronie *http://api.jqueryui.com/datepicker/*.

- **monthNames.** Opcja zawiera tablicę składającą się z 12 łańcuchów znaków określających nazwy miesięcy zapisane w innym języku niż angielski. Aby na przykład widżet kalendarza wyświetlał polskie nazwy miesięcy, w literale obiektowym przekazywanym w wywołaniu funkcji `datepicker()` należy podać opcję `monthNames` o następującej postaci:

  ```
  monthNames : ['Styczeń','Luty','Marzec','Kwiecień','Maj','Czerwiec',
  ➥'Lipiec','Sierpień','Wrzesień','Październik','Listopad','Grudzień']
  ```

- **numberOfMonths.** W tej opcji można podać liczbę miesięcy, które mają być wyświetlone w widżecie kalendarza. Zazwyczaj prezentowany jest tylko jeden miesiąc, jednak można zażądać, by było ich więcej. Jeśli zażądamy wyświetlenia więcej niż trzech miesięcy (patrz rysunek 10.2), prezentowany

Tabela 10.1. *Użyteczne łańcuchy formatujące, stosowane w opcji dateFormat*

Przykład	Znaczenie	Przykładowa postać daty
`'yy-mm-dd'`	Pełny numer roku, dwucyfrowa liczba określająca numer miesiąca i dwucyfrowa liczba określająca numer dnia. Ten format zapisu dat jest używany na przykład w bazie danych MySQL.	`2015-02-05`
`'m/d/y'`	Jedno- lub dwucyfrowa liczba określająca numer dnia, jedno- lub dwucyfrowa liczba określająca numer miesiąca i dwucyfrowy numer roku.	`2/5/15`
`'D, d M, yy'`	Skrócona nazwa dnia tygodnia, przecinek, odstęp, jedno- lub dwucyfrowy numer dnia, odstęp, skrócona nazwa miesiąca, przecinek, odstęp i pełny numer roku.	`Wt, 5 Lu, 2015`
`'DD, dd MM, yy'`	Pełna nazwa dnia tygodnia, przecinek, odstęp, jedno- lub dwucyfrowy numer dnia, odstęp, pełna nazwa miesiąca, przecinek, odstęp i pełny numer roku.	`Wtorek, 5 Luty, 2015`
`'@'`	Znacznik czasu systemu Unix. Określa liczbę milisekund, które upłynęły do północy 1 stycznia 1970 roku (patrz strona 595).	`1423123200000`

widżet kalendarza stanie się nieco nieporęczny. Kolejne miesiące zawsze są prezentowane jeden obok drugiego, a zatem, jeśli zażądamy pokazania więcej niż 3, zapewne wyświetlenie któregoś ze skrajnych miesięcy będzie wymagało przewinięcia zawartości widżetu w prawo lub w lewo. Dlatego też najlepiej przypisywać tej opcji jedynie wartości z zakresu od 1 do 3.

- **`maxDate`.** Określa najpóźniejszą datę, jaką użytkownik może wybrać w widżecie. Opcji tej można używać na przykład w hotelowym systemie rezerwacji miejsc. Wiele hoteli nie pozwala na rezerwowanie pokoi z wyprzedzeniem większym niż jeden rok; dlatego widżet kalendarza może ograniczyć zakres wybieranych dat, tak by nie wyprzedzały daty bieżącej o więcej niż jeden rok. Aby na przykład data wybierana w przyszłości nie była odległa o więcej niż 30 dni od daty bieżącej, należałoby użyć następującej opcji:

  ```
  maxDate : 30
  ```

  Innym rozwiązaniem jest podanie w tej opcji łańcucha znaków określającego odległość tej maksymalnej daty do dnia dzisiejszego, wyrażoną jako ilość lat (litera `y`), miesięcy (litera `m`), tygodni (litera `w`) oraz dni (litera `d`). By przykładowo ograniczyć zakres dat do roku w przód, należałoby użyć następującej opcji:

  ```
  maxDate : '+1y'
  ```

  Poszczególne znaki należy od siebie oddzielać odstępami. Aby maksymalna data nie była odległa od dnia dzisiejszego o więcej niż trzy miesiące, dwa tygodnie i pięć dni, należałoby użyć właściwości `maxDate` o postaci:

  ```
  maxDate : '+3m +2w +5d'
  ```

  Przykład zastosowania tej opcji można znaleźć w dalszej części rozdziału, na stronie 382.

- **minDate.** Opcja jest odwrotnością opcji `maxDate`. Określa *najwcześniejszą* datę, którą będzie można wybrać z kalendarza. Jest szczególnie użyteczna we wszelkiego typu formularzach rezerwacyjnych — w końcu nie ma większego sensu rezerwowanie czegoś od dnia wczorajszego (chyba że od momentu opublikowania tej książki udało się opanować podróże w czasie). Tę najwcześniejszą datę określa się dokładnie tak samo jak wartości opisywanej wcześniej opcji `maxDate`. Aby na przykład uniemożliwić użytkownikom wskazanie jakiejkolwiek daty z przeszłości, należy przypisać opcji `minDate` wartość `0`:

  ```
  minDate : 0
  ```

  W przypadku tej opcji zastosowanie wartości dodatnich oznacza, że użytkownik będzie musiał wybierać daty z przyszłości. Jeśli przykładowo wszystkie pokoje w hotelu zostały już zarezerwowane na najbliższe trzy tygodnie, można uniemożliwić użytkownikom wybieranie dat z tego zakresu, używając poniższej właściwości:

  ```
  minDate : '+3w'
  ```

  Zastosowanie wartości ujemnych pozwoli na wybieranie dat z przeszłości. Załóżmy, że utworzyłeś formularz do przeszukiwania firmowej bazy archiwalnych wiadomości poczty elektronicznej. Aby firmowa baza danych nie rozrosła się do zbyt dużych rozmiarów, przechowywane są w niej wiadomości wyłącznie z dwóch ostatnich lat, dlatego też nie ma sensu pozwalać użytkownikom na wybieranie wcześniejszych dat — sprzed trzech, czterech lub nawet pięciu lat — gdyż te e-maile i tak zostały już dawno usunięte. Poniżej pokazano, jak można ograniczyć zakres wybieranych dat do dwóch lat wstecz:

  ```
  minDate : '-2y'
  ```

  Także w tym przypadku można łączyć liczby dni, miesięcy i lat. Oto sposób, w jaki można ograniczyć zakres dat do jednego roku, dwóch miesięcy i trzech dni wstecz:

  ```
  minDate : '-1y -2m -3d'
  ```

Uwaga: W opcjach `minDate` oraz `maxDate` można także zapisywać obiekty dat JavaScript. Na przykład załóżmy, że firma zaczęła działalność od 13 marca 2010 roku. W takim przypadku najwcześniejszą datę, jaką można wybrać w widżecie kalendarza, można by określić w następujący sposób:

```
minDate : new Date(2010, 2, 13)
```

Informacje dotyczące tworzenia obiektów dat w języku JavaScript zostały podane na stronie 592.

- **yearRange.** Opcja jest stosowana wraz z opcją `changeYear` (patrz strona 378) i określa liczbę lat wyświetlanych na rozwijanej liście. Załóżmy, że chcesz pobrać datę urodzenia użytkownika. Możesz zatem przypisać opcji `changeYear` wartość `true`, tak by użytkownicy mogli wyświetlić w kalendarzu daty sprzed 20, 30 lub 50 lat. Standardowo przypisanie opcji `changeYear` wartości `true` sprawia, że w rozwijanej liście wyświetlanych jest dziesięć poprzednich i dziesięć przyszłych lat. Jednak w naszym przypadku lepiej byłoby wyświetlić kilkadziesiąt lat wstecz i żadnego roku z przyszłości (chyba że wynaleziono sposób podróży w czasie). W tym celu możesz przypisać opcji `yearRange` łańcuch znaków zawierający liczbę ujemną, dwukropek i kolejną liczbę — dodatnią lub ujemną. Pierwsza z liczb określa pierwszy rok prezentowany w rozwijanej liście, a druga — ostatni.

Wracając do przykładu z datami urodzenia, chciałbyś zapewne wyświetlić na liście ostatnich 120 lat i żadnego roku z przyszłości. Oto jak to zrobić:

```
yearRange : '-120:+0'
```

Ta opcja nakazuje jQuery UI wyświetlić w widżecie rozwijaną listę do wyboru roku, przy czym pierwszy z nich będzie sprzed 120 lat, a ostatni będzie bieżącym rokiem. Jeśli wciąż nie do końca rozumiesz, o co w tym chodzi, zajrzyj do przykładu zamieszczonego na stronie 382.

Przykład — pole do wyboru daty urodzenia

Nadszedł czas, żebyś spróbował zastosować widżet kalendarza. W tym przykładzie przekształcisz zwyczajne pole tekstowe w inteligentny komponent ułatwiający użytkownikowi określanie daty urodzenia.

Uwaga: Informacje o tym, jak pobrać przykłady prezentowane w książce, zostały zamieszczone na stronie 46.

1. **W edytorze tekstów otwórz plik *birthdate.html* umieszczony w katalogu *R10*.**

   Plik zawiera już odwołania do wszystkich plików jQuery i jQuery UI (w tym do pliku *datepicker-pl.js*, w którym zamieszczone zostały ustawienia pozwalające na polonizację kalendarza) oraz wywołanie funkcji `$(document).ready()` (patrz strona 190). Kolejnym krokiem będzie wybranie pola tekstowego.

2. **Wewnątrz wywołania funkcji `$(document).ready()` wpisz:**

   ```
   $('#dob')
   ```

   Jeśli przyjrzysz się kodowi HTML edytowanej strony, zauważysz w nim pole tekstowe, służące do podawania daty: `<input type= text id= dob name= birthdate >`. Pole to ma identyfikator `dob`, a zatem podany wyżej selektor pozwoli je pobrać. Kolejnym krokiem będzie utworzenie widżetu kalendarza.

3. **Wpisz kropkę oraz `datepicker();`, tak by kod wyglądał dokładnie tak, jak na poniższym przykładzie:**

   ```
   $('#dob').datepicker();
   ```

   I to już wszystko!

4. **Zapisz plik i wyświetl go w przeglądarce. Kliknij widoczne na stronie pole tekstowe.**

   W magiczny sposób poniżej niego zostanie wyświetlony kalendarz. W tym przypadku korzysta on z tematu graficznego jQuery UI o nazwie Lightness, lecz już w następnym rozdziale dowiesz się, jak można zmieniać wygląd widżetów.

   Jeśli teraz spróbujesz wybrać datę, przekonasz się, że jest to dosyć kłopotliwe. Musisz aż 12 razy kliknąć strzałkę w lewo umieszczoną w górnej części kalendarza, aby cofnąć się tylko o jeden rok! Dlatego ułatwisz użytkownikom wybór dat, dodając do kalendarza rozwijane listy do wyboru miesięcy i lat.

5. **Wróć do edytora tekstów, wewnątrz wywołania funkcji `datepicker()` dodaj literał obiektowy:**

   ```
   $('#dob').datepicker({
   });
   ```

Aby zmienić opcje kalendarza, musisz przekazać w wywołaniu funkcji `datepicker()` literał obiektowy — `{ }` — zawierający odpowiednie opcje — pary nazwa – wartość. Najpierw zajmiesz się rozwijaną listą do wyboru miesiąca.

6. **Do literału obiektowego dodaj właściwość `changeMonth : true`:**

```
$('#dob').datepicker({
  changeMonth : true
});
```

Jeśli teraz zapiszesz stronę, wyświetlisz ją w przeglądarce, a następnie klikniesz widoczne na niej pole tekstowe, zostanie wyświetlony kalendarz, a w jego górnej części będzie umieszczona rozwijana lista zawierająca nazwy 12 miesięcy. Dzięki niej znacznie łatwiej będzie można wybrać datę sprzed 9 miesięcy.

W kolejnym kroku zajmiesz się podobną listą do wyboru lat.

7. **Na końcu wpisanego wcześniej wiersza kodu dodaj przecinek, naciśnij klawisz *Enter* i wpisz `changeYear : true`:**

```
$('#dob').datepicker({
  changeMonth : true,
  changeYear : true
});
```

Zastosowanie tej opcji spowoduje dodanie do kalendarza kolejnej rozwijanej listy. Niestety, domyślna zawartość listy lat sięga jedynie 10 lat wstecz, czyli nie jest dokładnie tym, o co chodziło. Jeśli docelową grupą użytkowników witrynie nie są dzieci w wieku poniżej 10 lat bądź nie są to osoby posiadające wehikuł czasu, lata prezentowane na tej liście nie na wiele się im przydadzą. Na szczęście całkiem łatwo można zmienić zakres lat wyświetlanych na tej liście.

8. **Na końcu wpisanego wcześniej wiersza kodu dopisz przecinek, naciśnij klawisz *Enter* i wpisz: `yearRange : '-120:+0'`:**

```
$('#dob').datepicker({
  changeMonth : true,
  changeYear : true,
  yearRange : '-120:+0'
});
```

Ta opcja zmienia zakres lat wyświetlanych na liście. Teraz zaczynają się one od roku przypadającego 120 lat wstecz i kończą na bieżącym roku. I to jest to! Gdybyś jednak przetestował teraz stronę, przekonałbyś się, że w kalendarzu można wybrać datę z następnego tygodnia lub miesiąca. Powinieneś ograniczyć użytkowników przynajmniej do noworodków urodzonych w bieżącym dniu.

9. **Ponownie na końcu wiersza wpisz przecinek, naciśnij *Enter* i wpisz: `maxDate : 0`:**

```
$('#dob').datepicker({
  changeMonth : true,
  changeYear : true,
  yearRange : '-120:+0',
  maxDate : 0
});
```

Teraz nie będzie już można wybierać dat późniejszych od bieżącej. W końcu zmienisz format zapisu, by daty umieszczane w polu tekstowym były zapisywane tak, jak w przykładzie, czyli 27.1.2015.

10. **Jeszcze raz wpisz przecinek, naciśnij *Enter* i wpisz: `formatDate : 'dd-m-yy':`**

```
$('#dob').datepicker({
  changeMonth : true,
  changeYear : true,
  yearRange : '-120:+0',
  maxDate : 0,
  dateFormat : 'dd-m-yy'
});
```

Teraz dysponujesz już kalendarzem dostosowanym do własnych potrzeb, który świetnie nadaje się do wybierania dat urodzin.

11. **Zapisz plik i wyświetl go w przeglądarce.**

Kiedy klikniesz pole tekstowe widoczne na stronie, zostanie wyświetlony widżet kalendarza w swojej ostatecznej, dostosowanej postaci (patrz rysunek 10.3). Kompletną wersję tego przykładu możesz znaleźć w pliku *complete-birthdate.html* umieszczonym w katalogu *R10*.

Rysunek 10.3. *Widżet kalendarza jQuery UI jest koniecznym dodatkiem do wszystkich stron wymagających wybierania dat. Można go dostosowywać na niezliczoną ilość sposobów, na przykład pozwolić wyłącznie na wybieranie przyszłych dat lub dat z przeszłości. W tym przypadku daty późniejsze od 11 czerwca 2014 roku są szare i niedostępne, jednak dwie rozwijane listy u góry widżetu ułatwiają wybór roku z odległej przeszłości i dowolnego miesiąca*

Stylowe rozwijane listy

Tematy graficzne jQuery UI umożliwiają zapewnienie jednolitego wyglądu różnych elementów interfejsu użytkownika. Przykładowo widżet kalendarza bardzo przypomina widżet zestawu kart. Rozwijane listy — te elementy formularzy pozwalające na wybranie jednej z opcji z listy wyświetlanej po kliknięciu pola — nie dają dobrych możliwości określania wyglądu przy użyciu arkuszy stylów. Każda przeglądarka wyświetla je w nieco inny sposób, który niejednokrotnie jest zależny do używanego systemu operacyjnego (Windows, Mac, Linux), a co więcej, przeglądarki nie pozwalają na stosowanie w nich wszystkich dostępnych właściwości CSS.

Na szczęście jQuery UI udostępnia wygodny widżet Selectmenu, który przekształca standardową listę rozwijaną HTML, nadając jej znacznie bardziej atrakcyjną postać, dostosowaną do wyglądu pozostałych widżetów jQuery UI (patrz rysunek 10.4). Widżet ten dosłownie odtwarza listę rozwijaną, zapisując ją w postaci listy wypunktowanej i grupy znaczników `<span>`, których postać można znacznie łatwiej określać przy użyciu arkuszy stylów. Korzystając ze sprytnego kodu JavaScript,

Rysunek 10.4. *Standardowy wygląd rozwijanej listy zależy od przeglądarki i systemu operacyjnego (z lewej). Jednak dzięki zastosowaniu jQuery UI można przekształcić bezbarwne listy na stylowe elementy interfejsu użytkownika, dopasowane wyglądem do pozostałych widżetów jQuery UI*

widżet ukrywa początkową listę rozwijaną i pozwala użytkownikom na wybieranie opcji z listy obsługiwanej przez kod JavaScript. Opcje wybierane przez użytkownika są zaznaczane także w początkowym elemencie formularza, dzięki czemu w momencie przesyłania formularza informacje o wybranej opcji zostaną prawidłowo przesłane na serwer.

Biblioteka jQuery UI, w całkowicie niezauważalny sposób, wykonuje tu całkiem skomplikowaną programistyczną magię, a dla nas, użytkowników, tworzenie stylowej listy wyboru nie może być prostsze.

1. **Wykonaj czynności opisane na stronie 329, aby dodać do strony plik CSS jQuery UI oraz pliki JavaScript jQuery i jQuery UI.**

   Jak wcześniej, będziesz musiał dołączyć także plik JavaScript biblioteki jQuery, więc w przypadku korzystania z widżetów jQuery UI do strony dołączone będą następujące pliki: arkusz CSS jQuery UI, plik JavaScript biblioteki jQuery oraz plik JavaScript jQuery UI (dokładnie w takiej kolejności).

2. **Do strony dodaj formularz, a wewnątrz niego rozwijaną listę — znacznik `<select>` zawierający grupę znaczników `<option>`:**

   ```
   <select name="meal" id="meal">
     <option>brak</option>
     <option>wegański</option>
     <option>bezglutenowy</option>
     <option>wegetariański</option>
     <option>mięsny</option>
   </select>
   ```

   Przy okazji powinieneś zadbać o możliwość wygodnego pobrania znacznika `<select>` przy użyciu jQuery. W tym celu bądź to określ jego identyfikator, bądź też, jeśli na stronie chcesz umieścić kilka list rozwijanych, dodaj do każdej z nich taką samą nazwę klasy, na przykład `class= select` .

3. **Umieść na stronie wywołanie funkcji `$(document).ready()`:**

   ```
   $(document).ready(function() {

   }); // Koniec funkcji ready.
   ```

   Zgodnie z informacjami podanymi na stronie 191, stosowanie tego rozwiązania jest konieczne wyłącznie w przypadku, gdy chcesz umieścić kod JavaScript w sekcji `<head>` strony.

4. **Skorzystaj z możliwości jQuery, by wybrać rozwijaną listę, a następnie wywołaj funkcję `selectmenu()`:**

```
$(document).ready(function() {
  $('#meal').selectmenu();
}); // Koniec funkcji ready.
```

   Jeśli zastosowałeś nazwę klasy, by za jej pomocą wyróżnić więcej niż jedną rozwijaną listę na stronie, użyj wywołania o następującej postaci:

```
$(document).ready(function() {
  $('.select').selectmenu();
}); // Koniec funkcji ready.
```

   Jeżeli opcje umieszczone na liście nie są wyjątkowo krótkie, widżet Selectmenu jQuery nie wyświetli pierwszej z nich w całości. Innymi słowy, może się okazać, że widżet nie będzie dostatecznie szeroki, by w całości wyświetlić pierwszą opcję. Będzie to wyglądać dosyć dziwnie, więc zawsze powinieneś określić szerokość listy.

5. **W wywołaniu funkcji `selectmenu()` przekaż literał obiektowy określający wartość właściwości `width`:**

```
$(document).ready(function() {
  $('#meal').selectmenu({
    width: 200
  });
}); // Koniec funkcji ready.
```

   Podobnie jak w przypadku pozostałych widżetów jQuery UI (takich jak kalendarz przedstawiony na stronie 377), istnieje możliwość określania różnych opcji listy rozwijanej za pomocą przekazywania literału obiektowego zawierającego pary opcja – wartość. Opcja `width` jest wymagana niemal zawsze i powinna zawierać wartość liczbową określającą szerokość listy wyrażoną w pikselach (więcej informacji na temat tej opcji można znaleźć w następnym punkcie rozdziału).

   I to są wszystkie czynności konieczne do utworzenia listy rozwijanej, takiej jak przedstawiona na rysunku 10.4.

Określanie właściwości list rozwijanych

W widżecie Selectmenu nie znajdziesz zbyt wielu opcji. Przede wszystkim jest on narzędziem, które ma sprawić, by rozwijane listy wyglądały podobnie do pozostałych widżetów jQuery UI. Udostępnia on jednak kilka opcji, które mogą się przydać.

- **`width`.** Opcja niemal zawsze jest wymagana. Zazwyczaj jQuery UI tworzy rozwijaną listę, która nie jest na tyle szeroka, by były w niej w całości widoczne nazwy opcji. Dlatego praktycznie zawsze trzeba poszerzyć listę, aby wyświetlić całe nazwy. Co gorsza, jeśli lista nie jest na tyle szeroka, by można było na niej w całości wyświetlić nazwę opcji składającą się z dwóch lub więcej wyrazów, taka nazwa zostanie pokazana w dwóch wierszach. Jeśli szerokość listy ma być określona w pikselach, opcji `width` możemy użyć w następujący sposób:

```
width : 300
```

Szerokość można także podać w formie wartości procentowej lub przy użyciu jednostek `em`. W takich przypadkach wartość należy podać jako łańcuch znaków, za liczbą trzeba dodać odpowiednio: znak procenta (`%`) lub litery `em`. Aby na przykład dostosować menu do szerokości elementu rodzica (przykładowo znacznika `<div>`, w którym zostało umieszczone), moglibyśmy zażądać, by jego szerokość wynosiła 100%:

```
width : '100%'
```

Ewentualnie, jeśli ktoś woli jednostki `em`, poniższy przykład pokazuje, w jaki sposób nadać rozwijanej liście szerokość 6 em:

```
width : '6em'
```

- **`icons`**. Istnieje także możliwość wyświetlenia z prawej strony menu jednej z wielu ikon jQuery UI, takiej jak trójkąt widoczny na rysunku 10.4. Każdy temat graficzny jQuery UI udostępnia obszerny zestaw ikon (ich pełna lista jest podana na stronie *http://api.jqueryui.com/theming/icons/*). Ikonę, którą chcemy wyświetlić w widżecie listy, można określić, używając opcji `icons` i przypisując jej literał obiektowy o następującej postaci:

```
icons : {
  button: "ui-icon-circle-triangle-s"
}
```

Uwaga: Można się zastanawiać, dlaczego opcja `icons` wymaga podania kolejnego literału obiektowego zawierającego parę nazwa – wartość. W końcu byłoby znacznie łatwiej przypisać nazwę ikony bezpośrednio właściwości `icons`. Byłoby to całkiem sensowne rozwiązanie, gdyby Selectmenu był jedynym widżetem jQuery UI, w którym można określać używaną ikonę. Jednak ikony są wyświetlane także w innych widżetach, a niektóre z nich, takie jak widżet akordeonu (patrz strona 363), pozwalają na określanie *więcej* niż jednej ikony. W tym przypadku do widżetu należy przekazać literał obiektowy zawierający więcej par nazwa – wartość określających każdą z używanych ikon. Właśnie w celu zachowania spójności z innymi widżetami jQuery UI także w widżecie rozwijanej listy wartością opcji `icons` musi być literał obiektowy.

- **`position`**. Opcja pozwala na określanie położenia listy opcji. Domyślnie jest ona wyświetlana bezpośrednio poniżej widocznego pola z wybraną opcją — tak standardowo działają rozwijane listy. Jednak można zażądać, by lista opcji była wyświetlana z prawej lub z lewej strony pola. W tym celu należy określić położenie tej listy przy użyciu obiektu `position` jQuery UI. Ponieważ umieszczanie listy opcji gdziekolwiek indziej niż poniżej pola powodującego jej wyświetlenie jest rozwiązaniem niestandardowym, zatem stosując je, należy zachować ostrożność, gdyż może być mylące dla użytkowników strony. (Obiekt `position` oraz sposób jego stosowania został opisany w poprzednim rozdziale, w ramce na stronie 343).

Wykonywanie operacji po wybraniu opcji z listy

Zazwyczaj, kiedy użytkownik wybierze jedną z opcji dostępnych na liście, coś powinno się zdarzyć. Załóżmy na przykład, że tworzymy formularz do zamawiania ubrań, zawierający rozwijaną listę dostępnych kolorów. Kiedy użytkownik wybierze kolor z tej listy, chcemy zmienić wyświetlany na stronie obrazek, tak by prezentował dany artykuł w wybranym kolorze. Innymi słowy chcemy, by wybranie

opcji z listy spowodowało zmianę obrazka prezentowanego na stronie. Biblioteka jQuery UI zapewnia możliwość wywoływania określonej funkcji po każdej zmianie opcji wybranej na liście.

W tym celu należy skorzystać z opcji `change`. Jest ona wybierana dokładnie tak samo jak wszystkie inne opcje widżetów jQuery UI opisane w tym rozdziale, czyli należy ją podać w literale obiektowym przekazywanym w wywołaniu funkcji `selectmenu()`. Wartością tej opcji powinna być funkcja. Załóżmy, że na stronie znajduje się rozwijana lista o identyfikatorze `colors`. Chcemy przekształcić ją w widżet jQuery UI, określić szerokość i opcję `change`; a tak możemy to zrobić:

```
$('#colors').selectmenu({
  width : 300,
  change : function(event, ui) {
    // Tutaj umieść kod obsługujący zmianę wybranej opcji.
  }
});
```

Za każdym razem, gdy użytkownik wybierze z listy nową opcję (czyli *zmieni* ją), zostanie wywołana funkcja określona w opcji `change`. Funkcja ta ma dwa parametry — `event` oraz `ui`. Pierwszy z nich, `event`, zawiera obiekt `event` jQuery UI (opisany na stronie 194). Zazwyczaj nie trzeba stosować tego parametru w kodzie funkcji — zawiera ona jedynie informacje o zdarzeniu, takie jak współrzędne wskaźnika myszy w momencie kliknięcia oraz inne, które raczej nie są przydatne podczas obsługi wyboru opcji z listy.

Jednak drugi parametr, `ui`, zawiera kilka użytecznych informacji na temat listy. Można z niego odczytać indeks wybranej opcji — czyli jej położenie na liście, liczone od zera. Z parametru tego można także odczytać etykietę oraz wartość wybranej opcji menu. Parametr `ui` jest obiektem zawierającym kilka różnych właściwości, do których można się odwoływać przy użyciu zapisu z kropką (patrz strona 86).

- **`ui.item.index`:** Właściwość zawiera wartość indeksu wybranej opcji listy. Opcje list są numerowane tak samo jak elementy tablic, czyli pierwsza opcja ma indeks `0`, druga — indeks `1` i tak dalej.
- **`ui.item.label`:** Właściwość zawiera etykietę wybranej opcji listy. Jest to słowo lub kilka słów, które użytkownik widzi na liście. W kodzie HTML strony stanowią one treść znacznika `<option>`. Gdyby na stronie była umieszczona lista o następującej postaci:

  ```
  <select id="colors">
    <option>Czerwony</option>
    <option>Zielony</option>
    <option>Niebieski</option>
  </select>
  ```

  to etykietami poszczególnych opcji listy byłyby odpowiednio słowa: `Czerwony`, `Zielony` i `Niebieski`.
- **`ui.item.value`:** Właściwość zawiera wartość wybranej opcji listy. Wartości te są określane przy użyciu atrybutu `value` znacznika `<option>`. Bardzo często etykieta opcji oraz jej wartość są takie same. W takim przypadku określanie wartości w kodzie HTML strony nie jest konieczne. Jednak czasami zdarza się także, że do skryptu przetwarzającego dane na serwerze będziemy chcieli przesłać inną wartość. Przykładowo firma może używać specjalnych

kodów, które precyzyjnie identyfikują używane kolory. Użytkownik może wybrać „czerwoną" (ang. *red*) koszulę, lecz dla firmy odzieżowej, która korzysta z różnych odcieni tego koloru zależnie od typu produktu, kolor czerwony na koszulkach z krótkim rękawkiem może mieć kod „R785".

W takim przypadku poprzedni, przykładowy formularz musiałby mieć następującą postać:

```
<select id="colors">
  <option value="R785">Czerwony</option>
  <option value="G101">Zielony</option>
  <option value="B498">Niebieski</option>
</select>
```

Dla takiej listy wartościami kolejnych opcji będą odpowiednio łańcuchy znaków `R785` , `G101` oraz `B498` , a jej etykietami: `Czerwony` , `Zielony` i `Niebieski` .

A teraz zastanówmy się, w jaki sposób można by zmienić obrazek wyświetlony na stronie po zmianie opcji wybranej na liście. Załóżmy, że na stronie znajduje się przedstawiona powyżej lista z trzema opcjami: `Czerwony`, `Zielony` oraz `Niebieski`. Kiedy użytkownik wybierze jedną z nich, przeglądarka powinna wczytać obrazek koszulki w wybranym kolorze. Załóżmy, że w momencie wczytywania strony znajduje się w niej następujący znacznik:

```
<img src="red_shirt.jpg" id="shirt" alt="Kupuj nasze koszulki!">
```

Zgodnie z informacjami podanymi na stronie 239, zmieniając wartość atrybutu `img` obrazka, można kazać przeglądarce, by wczytała i wyświetliła na jego miejscu nowy obrazek. A zatem, aby wyświetlić na przykład obrazek niebieskiej koszulki, wystarczyłoby wybrać obrazek i zmienić jego atrybut `src`, używając następującego wywołania:

```
$('#shirt').attr('src', 'blue_shirt.jpg');
```

Jeśli złożymy wszystkie te fragmenty kodu w całość, wywołanie pozwalające utworzyć widżet rozwijanej listy i zmieniać wyświetlany obrazek po wybraniu jednej z opcji będzie mieć następującą postać:

```
$('#colors').selectmenu({
  width : 300,
  change : function (event, ui) {
    var newImage;
    if (ui.item.label === 'Czerwony') {
      newImage = 'red_shirt.jpg';
    } else if (ui.item.label === 'Zielony') {
      newImage = 'green_shirt.jpg';
    } else {
      newImage = 'blue_shirt.jpg';
    }
    $('#shirt').attr('src', newImage);
  }
});
```

Opcja `change` pozwala na wykonywanie przeróżnych operacji, takich jak aktualizowanie kodu HTML, dodawanie drugiego widżetu listy zawierającego zestaw opcji powiązanych z aktualnie wybraną opcją i tak dalej.

Uwaga: Widżet Selectmenu udostępnia także inne opcje, zdarzenia oraz metody. Ich kompletna lista jest dostępna na stronie *http://api.jqueryui.com/selectmenu/*.

Stylowe przyciski

Biblioteka jQuery UI udostępnia także widżet, który pozwala nadać spójny wygląd i sposób działania różnym rodzajom przycisków dostępnych w języku HTML. Widżet przycisku służy do określania wyglądu przycisków przesyłających formularz (`submit`), przycisków przywracających początkową wartość pól formularza (`reset`) oraz elementów `<input>` typu `button`: `<input type= button >`. Oprócz tego, można go także używać do określania postaci elementu `<button>`, tak by był dopasowany do wyglądu pozostałych widżetów jQuery UI (patrz rysunek 10.5).

1. **Dołącz do strony plik CSS jQuery UI oraz pliki JavaScript jQuery i jQuery UI, zgodnie z informacjami podanymi na stronie 329.**

   Jak wcześniej, będziesz musiał dołączyć także plik JavaScript biblioteki jQuery, więc w przypadku korzystania z widżetów jQuery UI do strony dołączone będą następujące pliki: arkusz CSS jQuery UI, plik JavaScript biblioteki jQuery oraz plik JavaScript jQuery UI (dokładnie w takiej kolejności).

2. **Dodaj do strony jakiś przycisk. Może to być przycisk typu `reset`, `submit`, `button` lub element `<button>`. Oto przykład:**

   ```
   <input type="submit" id="submit" value="Wyślij formularz!">
   <input type="reset" id="reset" value="Wyczyść formularz.">
   <input type="button" id="inputButton" value="Zwyczajny przycisk.">
   <button id="button">Element Button</button>
   ```

   Nie zapomnij o zapewnieniu sobie możliwości łatwego wybrania tego przycisku przy użyciu jQuery. Możesz określić jego identyfikator bądź też, jeśli formularz ma zawierać więcej takich przycisków, możesz użyć w nich wszystkich tej samej klasy, na przykład `class= button` .

3. **Dodaj do strony wywołanie funkcji `$(document).ready()`:**

   ```
   $(document).ready(function() {

   }); // Koniec funkcji ready.
   ```

   Użycie tego wywołania jest konieczne wyłącznie w przypadku, gdy kod JavaScript jest umieszczany w sekcji `<head>` strony, zgodnie z informacjami podanymi na stronie 190.

Rysunek 10.5. *Widżet przycisku jQuery UI zapewnia prosty i szybki sposób przekształcania bezbarwnych elementów formularzy, takich jak przyciski opcji, pola wyboru i różne przyciski HTML (pokazane u góry), w stylowo wyglądające elementy interfejsu użytkownika cechujące się spójnym sposobem prezentacji (u dołu)*

4. **Skorzystaj z możliwości jQuery, by wybrać element przycisku, a następnie wywołaj funkcję `button()`:**

```
$(document).ready(function() {
  $('#submit').button();
}); // Koniec funkcji ready.
```

Bądź też, jeśli przyciski są identyfikowane przy użyciu nazwy klasy, użyj wywołania o następującej postaci:

```
$(document).ready(function() {
  $('.button').button();
}); // Koniec funkcji ready.
```

Po wywołaniu tej funkcji widżet nada wybranym elementom strony wygląd pasujący do używanego tematu graficznego jQuery UI (patrz rysunek 10.5).

Dostosowywanie przycisków

Widżet przycisku jQuery UI nie zapewnia zbyt szerokich możliwości dostosowywania. W elementach `<input>` tekst widoczny na przycisku jest określany przez atrybut `value`. Natomiast w elementach `<button>` tekstem prezentowanym na przycisku jest zawartość umieszczona pomiędzy znacznikami otwierającym i zamykającym, na przykład `<button>Jestem przyciskiem</button>`. Jednak i tak można dostosowywać postać tych widżetów na kilka różnych sposobów, wystarczy przekazać opcje w wywołaniu funkcji `button()`. Oto te opcje.

- **`icons`:** Przyciski jQuery UI pozwalają na wyświetlanie jednej ikony z lewej strony przycisku, a drugiej — z jego prawej strony. (Przykładem może być element `<button>` przedstawiony na rysunku 10.5). Każdy temat graficzny jQuery UI udostępnia obszerny zestaw ikon (ich pełna lista jest podana na stronie *http://api.jqueryui.com/theming/icons/*). Ikony można dodawać do przycisków przy użyciu opcji `icons`, której wartością powinien być literał obiektowy, taki jak przedstawiony poniżej:

```
icons : {
  primary : "ui-icon-gear",
  secondary : "ui-icon-trinagle-1-s"
}
```

Ikona określana jako *główna* (ang. *primary*) jest wyświetlana z lewej strony przycisku, natomiast *pomocnicza* (ang. *secondary*) z prawej strony. Nie trzeba opisywać obu ikon jednocześnie — zazwyczaj wygląda to dosyć dziwnie. Przeważnie określana jest tylko jedna z dwóch możliwych ikon. Aby na przykład wyświetlić strzałkę w prawo we wszystkich elementach `<button>` umieszczonych na stronie, należałoby użyć poniższego fragmentu kodu:

```
$('button').button({
  icons : { secondary : 'ui-icon-circle-arrow-e' }
});
```

Uwaga: Tych ikon nie można wyświetlać w przyciskach jQuery UI tworzonych przy użyciu elementów `<input>` — czyli w przyciskach przesyłających formularz i czyszczących jego pola. Jest to możliwe wyłącznie w przyciskach budowanych za pomocą elementów `<button>`.

- **text:** Podczas stosowania elementów `<button>` i wyświetlania na nich ikon można całkowicie ukryć tekst umieszczony na przycisku i wyświetlić wyłącznie ikonę. Aby to zrobić, należy użyć opcji `text` i przypisać jej wartość `false`. Przykładowo załóżmy, że na stronie znajduje się przycisk z napisem `Dalej` :

  ```
  <button id="next">Dalej</button>
  ```

  Aby przekształcić ten element w widżet jQuery UI i zamiast tekstu wyświetlić na nim ikonę strzałki w prawo, należałoby użyć następującego fragmentu kodu:

  ```
  $('#next').button({
    icons : { secondary : 'ui-icon-arrowthick-1-e' },
    text : false
  });
  ```

Uwaga: Widżet Button udostępnia także inne opcje, zdarzenia oraz metody. Ich kompletna lista jest dostępna na stronie *http://api.jqueryui.com/button/*.

Poprawianie wyglądu przycisków opcji i pól wyboru

Przyciski opcji oraz pola wyboru są kolejnymi elementami formularzy HTML, które w swojej standardowej postaci nie wyglądają najlepiej (patrz rysunek 10.5 u góry). Przeglądarki wyświetlają przyciski i pola wyboru w sposób zależny od używanego systemu operacyjnego i nie pozwalają na dostosowywanie ich wyglądu przy użyciu arkuszy stylów w równie szerokim zakresie, jak innych elementów HTML.

Na szczęście biblioteka jQuery UI udostępnia widżet, który pozwala upodabniać przyciski opcji i pola wyboru do pozostałych widżetów (patrz rysunek 10.5, u dołu). Bardzo dobrą wiadomością jest to, że skorzystanie z tego widżetu w celu upiększenia elementów formularzy wymaga jedynie minimalnego nakładu pracy — wszystko załatwia wywołanie funkcji `buttonset()`. Co ciekawe, działanie tej funkcji sprowadza się do wywołania funkcji `.button()` na rzecz każdego przycisku opcji lub pola wyboru we wskazanej grupie.

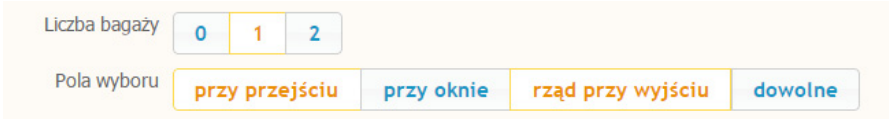
Nasze zadanie polega jedynie na prostym przygotowaniu odpowiedniego kodu HTML: wystarczy umieścić wszystkie przyciski w jakimś kontenerze, takim jak znacznik `<div>`, i zapewnić możliwość jego łatwego wybrania przy użyciu jQuery. Załóżmy na przykład, że tworzymy formularz do rezerwowania przelotów, na którym klienci mogą zaznaczyć, ile będą mieli toreb lub walizek — 0, 1 lub 2. Załóżmy, że kod HTML formularza ma następującą postać:

```
<div id="bags">
  <p class="label">Liczba bagaży</p>

  <input type="radio" id="none" name="bags" checked="checked">
  <label for="none">0</label>

  <input type="radio" id="one" name="bags">
  <label for="one">1</label>

  <input type="radio" id="two" name="bags">
  <label for="two">2</label>
</div>
```

W powyższym kodzie znajdują się trzy przyciski opcji — `<input type= radio >` — umieszczone wewnątrz elementu `div` o identyfikatorze `radio`. Aby przekształcić je w grupy przycisków jQuery UI, wystarczy wykonać kroki od 1. do 3. ze strony 329, by dołączyć do strony niezbędne pliki CSS i JavaScript bibliotek jQuery i jQuery UI, a następnie wywołać funkcję `$(document).ready()`. Wewnątrz tego wywołania należy umieścić poniższy wiersz kodu:

```
$('#radio').buttonset();
```

I to już wszystko. Wywołanie `$('#radio')` wybiera element `div` zawierający przyciski opcji, a wywołanie `.buttonset()` odnajduje każdy przycisk opcji lub pole wyboru i przekształca je w widżet przycisku jQuery UI. Unikalną cechą funkcji `buttonset()` jest to, że umieszcza wszystkie elementy w taki sposób, jakby stanowiły jedną całość (patrz rysunek 10.6).



Rysunek 10.6. *Z grupy przycisków opcji można wybrać tylko jeden przycisk (u góry). W tym przykładzie wybrany (i wyróżniony kolorem) jest środkowy przycisk, ten z cyfrą „1". Natomiast w grupie pól wyboru można zaznaczyć dowolnie wiele pól (u dołu). W tym przykładzie zaznaczono pola pierwsze („przy przejściu") oraz trzecie („rząd przy wyjściu")*

Uwaga: Biblioteka jQuery UI domyślnie formatuje grupę przycisków opcji oraz pól wyboru, nadając im postać jednego elementu (patrz rysunek 10.6). Oznacza to, poszczególne pola lub przyciski są wyświetlane w poziomie, jedne za drugimi. Gdybyśmy jednak chcieli, by przyciski nie stykały się ze sobą i były umieszczone obok siebie, wystarczyłoby wywołać funkcję `button()` na rzecz każdego z nich osobno. Jeśli przykładowo wrócimy do poprzedniego przykładu i grupy przycisków opcji, poniższe wywołanie JavaScript pozwoli przekształcić je w grupę niezależnych przycisków:

```
$('#radio input').button();
```

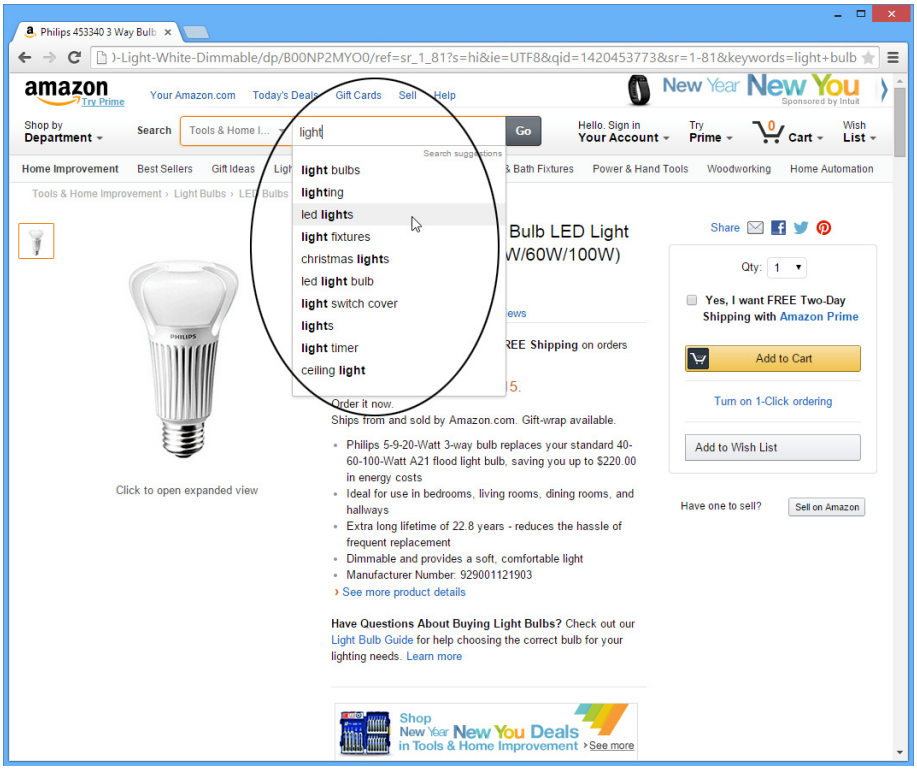
Powyższa instrukcja sprawia, że funkcja `button()` zostanie wywołana niezależnie dla każdego przycisku opcji umieszczonego w elemencie `div`.

Dokładnie to samo dotyczy pól wyboru. Wystarczy umieścić je w jakimś kontenerze, na przykład `<div id= check >`, wybrać go przy użyciu jQuery, a następnie wywołać funkcję `buttonset()`. Niezależnie od tego, czy operujemy na grupie pól wyboru, czy też przycisków opcji, jQuery UI sformatuje je w taki sam sposób — przedstawiony na rysunku 10.6. Jednak te dwa rodzaje elementów sterujących działają w inny sposób. W przypadku przycisków opcji w danej chwili może być wybrany tylko jeden przycisk w grupie. W odróżnieniu od nich, w grupie pól wyboru zawsze można zaznaczyć dowolnie wiele z nich (może to także oznaczać, że żadne pole wyboru nie będzie zaznaczone). jQuery UI wyróżnia wybrane pola i przyciski innym kolorem.

Funkcja `buttonset()` nie pobiera żadnych argumentów. Jej działanie w niezauważalny dla nas sposób sprowadza się do wywołania funkcji `button()` na rzecz każdego z elementów — pól wyboru lub przycisków opcji — w grupie. A to oznacza, że postać przycisków można dodatkowo modyfikować, zgodnie z informacjami podanymi na stronie 389.

Dostarczanie podpowiedzi przy użyciu automatycznego uzupełniania

Wiele witryn zawierających pola do przeszukiwania zawartości udostępnia także bardzo przydatną możliwość, polegającą na wyświetlaniu sugestii pasujących do tekstu, który użytkownik zaczął wpisywać w polu. Wystarczy wejść na stronę *Amazon.com* i wpisać `light`, a poniżej zostanie wyświetlona lista sugerowanych kategorii i produktów zawierających wpisane słowo: light bulbs (żarówki), led lights (oświetlenie ledowe) i tak dalej (patrz rysunek 10.7). Zamiast ręcznie wpisywać resztę słowa, wystarczy kliknąć jedną z podpowiedzi albo nawet skorzystać z klawiatury i zaznaczyć ją przy użyciu strzałek w górę i w dół.



Rysunek 10.7. *Czy nie bolą Cię palce od tego ciągłego wpisywania? Wiele witryn stara się pomóc użytkownikom i oszczędzić wpisywania poprzez sugerowanie wyszukiwanych słów, pasujących do liter, które użytkownik zaczął wpisywać w polu*

Takie rozwiązanie jest nazywane *automatycznym uzupełnianiem*, a jQuery UI udostępnia wygodny widżet, który pozwoli zaimplementować je na własnej witrynie. Możliwość automatycznego uzupełniania można dodać do dowolnego pola tekstowego w formularzu. Załóżmy na przykład, że tworzymy stronę do rezerwacji biletów lotniczych, na której użytkownik musi określić początkowy port lotniczy. Kiedy zacznie wpisywać jego nazwę, poniżej pola zostanie wyświetlona

rozwijana lista, zawierająca sugerowane lotniska pasujące do pierwszych wpisanych liter. Dzięki temu użytkownik będzie mógł kliknąć jedną z opcji na liście, a nazwa portu lotniczego zostanie umieszczona w polu bez konieczności wpisywania pełnej nazwy.

Jeszcze ciekawsza jest możliwość zastosowania widżetu automatycznego uzupełniania do umieszczania w polu tekstowym innej zawartości. Kontynuujmy zatem poprzedni przykład formularza do rezerwacji biletów lotniczych: użytkownik mógłby wpisywać nazwę początkowego portu lotniczego, a po wybraniu lotniska w polu formularza byłby umieszczany jego *kod*. Przykładowo wpisanie `Portland` mogłoby powodować wyświetlenie listy portów lotniczych zawierającej między innymi „Portland International Airport". Gdyby użytkownik wybrał tę opcję, w polu zostałby wpisany odpowiedni kod portu lotniczego, w tym przypadku PDX. Dokładnie tego samego rozwiązania można by użyć do odszukania produktu z katalogu. Kiedy użytkownik wpisze nazwę produktu i wybierze go z listy podpowiedzi, w polu zostanie zapisany jego numer.

Aby korzystać z widżetu Autocomplete, należy dostarczyć danych, których jQuery UI będzie mogła użyć do odnalezienia podpowiedzi pasujących do tego, co użytkownik wpisał w polu, i wyświetlenia ich w liście. W naszym przykładzie powinna to być lista nazw portów lotniczych, z którą jQuery UI będzie mogła porównać nazwę wpisaną przez użytkownika w polu tekstowym. Te dane można określić na dwa sposoby. Pierwszym z nich jest przekazanie tablicy JavaScript zawierającej niezbędne informacje, a drugim — wykorzystanie technologii AJAX w celu przesłania na serwer poszukiwanej frazy, na podstawie której serwer odszuka pasujące podpowiedzi i prześle je z powrotem do przeglądarki. W tym rozdziale zostaną przedstawione oba rozwiązania, jednak na początku musisz poznać podstawy stosowania widżetu Autocomplete.

1. **Dołącz do strony plik CSS jQuery UI oraz pliki JavaScript jQuery i jQuery UI, zgodnie z informacjami podanymi na stronie 329.**

   Jak wcześniej, będziesz musiał dołączyć także plik JavaScript biblioteki jQuery, więc w przypadku korzystania z widżetów jQuery UI do strony dołączone będą następujące pliki: arkusz CSS jQuery UI, plik JavaScript biblioteki jQuery oraz plik JavaScript jQuery UI (dokładnie w takiej kolejności).

2. **Dodaj do strony formularz zawierający pole tekstowe:**

   ```
   <input type="text" id="airport" name="airport">
   ```

   Powinieneś także pamiętać o zapewnieniu wygodnego sposobu wybrania tego pola tekstowego przy użyciu jQuery. Możesz na przykład dodać do niego identyfikator.

3. **Dodaj do strony wywołanie funkcji `$(document).ready()`:**

   ```
   $(document).ready(function() {

   } // Koniec funkcji ready.
   ```

   Użycie tego wywołania jest konieczne wyłącznie w przypadku, gdy kod JavaScript zostanie umieszczony w sekcji `<head>` strony, zgodnie z informacjami podanymi na stronie 190.

4. **Skorzystaj z możliwości jQuery, by pobrać pole tekstowe, a następnie wywołaj funkcję `autocomplete()`:**

```
$(document).ready(function() {
  $('#airport').autocomplete();
} // Koniec funkcji ready.
```

W najprostszym przypadku w wywołaniu funkcji `autocomplete()` musisz przekazać obiekt zawierający właściwość `source`, której wartością będzie bądź to tablica danych, bądź też adres URL programu działającego na serwerze, który będzie zwracał listę podpowiedzi pasujących do liter wpisanych przez użytkownika w polu tekstowym. W dwóch kolejnych punktach zostały opisane oba rozwiązania — zastosowanie tablicy z danymi bądź skryptu działającego na serwerze.

Generowanie podpowiedzi przy użyciu tablicy danych

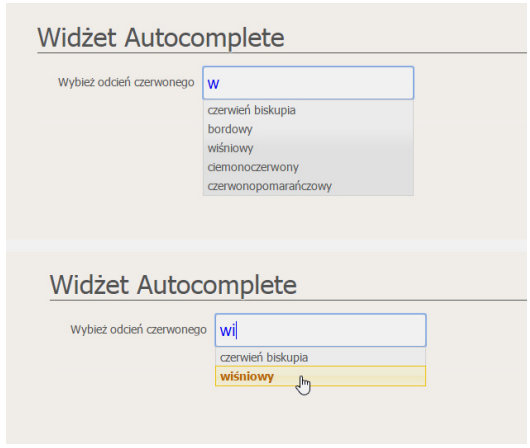
Widżet Autocomplete wymaga listy danych, których jQuery UI będzie używać w celu odszukania podpowiedzi pasujących do znaków wpisanych przez użytkownika. Najprostszym sposobem dostarczenia tych danych jest przekazanie tablicy, której jQuery UI będzie używać do odnajdywania pasujących podpowiedzi. Przykładowo załóżmy, że na stronie zostało umieszczone pytanie: „Jaki jest Twój ulubiony odcień czerwonego?". W takim przypadku moglibyśmy podać nazwy kolorów w tablicy i przekazać ją w wywołaniu funkcji `autocomplete()`:

```
var colors = ['szkarłat', 'czerwień biskupia', 'bordowy', 'wiśniowy',
              'rodopsyna', 'cynober', 'ciemnoczerwony',
              'czerwonopomarańczowy'];
$('#redInput').autocomplete( { source : colors } );
```

Innymi słowy, zaczęliśmy od utworzenia tablicy — `colors`. Następnie w wywołaniu funkcji `autocomplete()` przekazaliśmy literał obiektowy zawierający jedną właściwość, `source`, której wartością była tablica `colors`: `{ source : colors }`.

Po wykonaniu takiego kodu, kiedy użytkownik wpisze w polu tekstowym literę `w`, poniżej pola zostanie wyświetlona lista zawierająca podpowiedzi: `czerwień biskupia`, `bordowy`, `wiśniowy`, `ciemnoczerwony` oraz `czerwonopomarańczowy` (patrz rysunek 10.8, u góry). Jednak kiedy użytkownik dopisze literkę `i`, tak że w polu tekstowym znajdzie się łańcuch `wi`, lista dostępnych podpowiedzi zostanie ograniczona do dwóch: `czerwień biskupia` i `wiśniowy` (patrz rysunek 10.8, u dołu).

Jednak tablica zawierająca kilka elementów nie będzie bardzo pomocna. W końcu, gdyby dostępnych opcji było naprawdę tylko tyle, można by je umieścić na normalnej liście rozwijanej. Mechanizm automatycznego uzupełniania sprawdza się najlepiej w tych przypadkach, gdy możemy wyświetlać użytkownikowi *wiele* podpowiedzi — znacznie więcej niż można by, w wygodny sposób, umieścić na liście rozwijanej. W takim przypadku dobrym rozwiązaniem jest utworzenie odrębnego pliku JavaScript zawierającego wyłącznie dane, które mają być przekazane jako wartość właściwości `source` widżetu. Przykładowo załóżmy, że chcemy wyświetlać podpowiedzi zawierające nazwy portów lotniczych. Moglibyśmy utworzyć odrębny plik JavaScript, *airports.js*, zawierający odpowiednią tablicę z danymi, taką jak przedstawiona poniżej:



Rysunek 10.8. *Widżet Autocomplete jQuery UI ułatwia wpisywanie często używanych haseł poprzez wyświetlanie podpowiedzi, pasujących do słów wpisywanych przez użytkownika w polu tekstowym*

```
var airports = [
  'Aberdeen Regional Airport, Aberdeen, Dakota Południowa',
  'Abilene Regional Airport, Abilene, Teksas',
  'Abraham Lincoln Capital Airport, Springfield, Illinois',
  'Adak Airport, Adak Island, Alaska',
  'Adirondack Regional Airport, Saranca Lake, Nowy York'
]; // Wiele, wiele innych portów lotniczych.
```

Uwaga: To tylko początek listy wszystkich portów lotniczych na terenie USA. W rzeczywistości ta lista byłaby bardzo długa — a zatem musielibyśmy używać bardzo długiej tablicy z nazwami lotnisk.

Ten plik byłby całkiem duży, lecz nie trzeba by go używać na wszystkich stronach witryny; dlatego też takie dane najlepiej umieszczać w odrębnych plikach i dołączać jedynie do tych stron, na których jest wykorzystywany widżet Autocomplete. Taki plik należałoby dołączyć do strony dokładnie w taki sam sposób, w jaki są dołączane wszystkie inne pliki JavaScript (patrz strona 49):

```
<script src="ariports.js"></script>
```

Trzeba się jednak upewnić, że ten plik zostanie dołączony do strony *przed* utworzeniem widżetu, gdyż tablica z danymi musi być dostępna przed wywołaniem funkcji `autocomplete()`. Przykładowo kod dołączający pliki JavaScript, umieszczony w sekcji `<head>` strony, mógłby wyglądać tak:

```
<link href="css/jquery-ui.min.css" rel="stylesheet">
<link href="css/site.css" rel="stylesheet">
<script src="js/jquery.min.js"></script>
<script src="js/jquery-ui.min.js"></script>
<script src="js/airports.js"></script>
<script>
$(document).ready(function() {
  $('#airport').autocomplete( { source : airports} );
}); // Koniec funkcji ready.
</script>
```

Warto zwrócić uwagę, że plik *airports.js* będzie wczytany przed wywołaniem funkcji `autocomplete()`, dzięki czemu mamy pewność, że tablica `airports` zostanie utworzona i będzie można ją przekazać do widżetu.

Stosowanie osobnych etykiet i wartości

Wybór jednej opcji z listy podpowiedzi nakazuje jQuery UI zapisanie tego elementu w polu tekstowym. Jeśli w formularzu pokazanym na rysunku 10.8 po wpisaniu litery `w` użytkownik kliknie opcję *wiśniowy*, jQuery UI wpisze to słowo w polu tekstowym.

Jednak można poinstruować jQuery UI, by zapisywała w polu *coś innego*, a nie tekst widoczny na liście podpowiedzi. W ramach przykładu wróćmy do formularza z listą portów lotniczych i załóżmy, że użytkownik wpisał w polu tekstowym litery `Ab`. W takim przypadku widżet Autocomplete wyświetli listę z trzema podpowiedziami: `Aberdeen Regional Airport`, `Abilene Regional Airport` oraz `Abraham Lincoln Capital Airport`. Można nakazać, by zamiast nazwy portu lotniczego jQuery UI umieściła w polu tekstowym jego kod, a zatem, jeśli użytkownik wybierze opcję `Abilane Regional Airport`, w polu tekstowym zostanie zapisany kod `ABI`.

W tym celu należy przekazać tablicę *obiektów*. Każdy z tych obiektów musi zawierać dwie właściwości: `label` oraz `value`. Pierwsza z nich, `label`, określa treść podpowiedzi wyświetlanej na rozwijanej liście, natomiast druga, `value`, wartość, jaka zostanie wpisana do pola formularza po dokonaniu wyboru. W naszym przykładzie z portami lotniczymi wartością właściwości `label` będzie nazwa lotniska, natomiast wartością właściwości `value` — jego kod. Zatem obiekt reprezentujący lotnisko w Abilane będzie mieć następującą postać:

```
{
  label : 'Abilene Regional Airport, Abilene, Teksas',
  value : 'ABI'
}
```

W takim przypadku tablica z danymi o portach lotniczych zaczynałaby się następująco:

```
var airports = [
  {
    label : 'Aberdeen Regional Airport, Aberdeen, Dakota Południowa',
    value : 'ABR'
  },
  {
    label : 'Abilene Regional Airport, Abilene, Teksas',
    value : 'ABI'
  },
  {
    label : 'Abraham Lincoln Capital Airport, Springfield, Illinois',
    value : 'SPI'
  },
  {
    label : 'Adak Airport, Adak Island, Alaska',
    value : 'ADK'
  },
  {
    label : 'Adirondack Regional Airport, Saranca Lake, Nowy York'
    value : 'SLK'
  }
]; // Wiele, wiele innych portów lotniczych.
```

Uwaga: Teraz też lista portów lotniczych na terenie USA nie jest kompletna. Rzeczywisty plik *airports.js* byłby ogromny.

Każdy z obiektów zapisanych w tej tablicy ma dwie właściwości. Taką tablicę należy przekazać w wywołaniu funkcji `autocomplete()` dokładnie w taki sam sposób jak wcześniej — biblioteka jQuery UI została napisana tak, by wiedzieć, co zrobić, gdy zostanie przekazana tablica takich obiektów.

```
$('#airport').autocomplete( { source : airports } );
```

W przypadku przekazania tablicy obiektów widżet Autocomplete stara się dopasować tekst wpisany przez użytkownika do wartości właściwości `label` tych obiektów. Kiedy użytkownik wybierze jedną z sugerowanych podpowiedzi wyświetlonych na liście, w polu formularza zostanie zapisana wartość właściwości `value` wybranego obiektu. Działanie takiego rozwiązania znajdziesz w przykładzie prezentowanym w dalszej części rozdziału, a konkretnie w kroku 12., na stronie 405.

Pobieranie danych automatycznego uzupełniania z serwera

Widżet Autocomplete jest najbardziej użyteczny w tych sytuacjach, gdy danych, które mogą być przeszukiwane i wyświetlane jako podpowiedzi, jest bardzo dużo. Czy pamiętasz przykład zamieszczony na stronie 395, w którym podano jedynie osiem odcieni czerwieni? Niewielka liczba dostępnych opcji nie będzie stanowiła dla użytkownika dużego ułatwienia. Jednak tworzenie pliku zawierającego bardzo dużą tablicę nie jest łatwe, a w wielu przypadkach nawet niewykonalne. Przykładowo mechanizm automatycznego sugerowania używany na stronach sklepu Amazon.com na pewno nie korzysta z jednego pliku JavaScript zawierającego listę wszystkich kategorii produktów wraz z ich opisami: taki plik byłyby ogromny, a jego wczytywanie trwałoby zbyt długo.

Wiele witryny korzystających z tego mechanizmu (takich jak Google lub Amazon) stosuje skrypty wykonywane na serwerze, by przesyłać do przeglądarki znacznie mniejszą listę podpowiedzi. Oto sposób działania takiego rozwiązania.

1. **Użytkownik zaczyna wpisywać coś w polu tekstowym.**
2. **Znaki wpisane przez użytkownika są przesyłane na serwer.**

   Używając technologii AJAX, przeglądarka przesyła dane na serwer i oczekuje na odpowiedź. W tym przypadku przesyłane są znaki wpisane przez użytkownika.
3. **Serwer przesyła w odpowiedzi tablicę haseł pasujących do znaków wpisanych do tej pory przez użytkownika.**

   Serwer przygotowuje i przesyła listę pasujących haseł. Zazwyczaj do tego celu używany jest jakiś program, który przeszukuje bazę danych, pobiera wyniki i zapisuje je w formie tablicy, która następnie jest przesyłana z powrotem do przeglądarki.
4. **Pasujące hasła są wyświetlane na liście podpowiedzi.**

Jak widać, znaczna część tej magii dzieje się przy użyciu technologii AJAX, która umożliwia skorzystanie z kodu JavaScript do przesyłania i odbierania informacji między przeglądarką i serwerem, bez konieczności wyświetlania zupełnie nowej strony (technologię AJAX poznasz w rozdziale 13.). Prezentacja kodu działającego po stronie serwera i obsługującego takie rozwiązania wykracza poza ramy tematyczne

tej książki — aby dowiedzieć się, jak napisać taki program w językach PHP, Ruby, Python lub nawet JavaScript (używając Node.js), będziesz musiał skorzystać z innego źródła informacji. Jednak poniżej opisany został sposób przygotowania widżetu Autocomplete jQuery UI tak, by pobierał dane z serwera: zamiast przekazywać we właściwości `source` tablice potencjalnych podpowiedzi, należy w niej podać adres URL skryptu na serwerze.

Załóżmy, że na stronie jest umieszczone pole tekstowe używane do przeszukiwania firmowej bazy danych zawierającej informacje o 250 tysiącach produktów. Aby ułatwić użytkownikom poszukiwania, chcemy wzbogacić to pole i dodać mechanizm automatycznego uzupełniania. Niestety katalog produktów jest zbyt duży, by można go było zapisać w jednym pliku JavaScript. Zamiast tego możemy przygotować program działający na serwerze; jest to plik *products.php* umieszczony w głównym katalogu serwera. Możemy zapisać ścieżkę dostępu do tego pliku we właściwości `source`. Załóżmy, że pole tekstowe ma identyfikator `productSearch`:

```
<input type="text" id="productSearch" name="productSearch">
```

Następnie w kodzie JavaScript strony moglibyśmy użyć poniższego wywołania, które utworzy widżet Autocomplete i doda go do pola tekstowego, nakazując jednocześnie, by pobierał dane z serwera:

```
$('#productSearch').autocomplete( { source : '/products.php'} );
```

Można także podać pełny adres URL, zawierający określenie protokołu oraz nazwę domeny:

```
$('#productSearch').autocomplete( { source :
'http://myCompany.com/products.php'} );
```

Po przekazaniu we właściwości `source` tablicy elementów (w sposób opisany na stronie 395) jQuery UI filtruje tę tablicę, odnajdując w niej elementy pasujące do znaków, które użytkownik wpisał w polu tekstowym. Następnie jQuery UI wyświetla tylko pasujące elementy. Kiedy jednak we właściwości `source` zostanie podany adres URL do skryptu na serwerze, widżet Autocomplete zachowuje się w nieco inny sposób. Otóż w tym przypadku wyświetlane są *wszystkie* przesłane z serwera podpowiedzi, a widżet w żaden sposób ich nie filtruje. Gdyby na przykład użytkownik wpisał słowo `lampa`, a serwer zwrócił tablicę zawierającą łańcuchy `ciemny`, `karczoch` i `struś`, jQuery UI wyświetliłaby na liście podpowiedzi wszystkie trzy łańcuchy.

Innymi słowy, zwrócenie prawidłowych danych leży wyłącznie w gestii serwera. Oznacza to, że my musimy napisać całą logikę generującą prawidłową listę podpowiedzi i wysłać tę listę do przeglądarki. Aby ułatwić to zadanie, jQuery UI dodaje do adresu URL parametr, który informuje serwer, czego właściwie ma szukać. Parametr ten nosi nazwę `term`, a jego wartością jest łańcuch znaków wpisany przez użytkownika w polu tekstowym. Gdyby w naszym wcześniejszym przykładzie przeszukiwania produktów użytkownik wpisał słowo `lampa`, jQuery UI przesłałaby żądanie pod następujący adres:

```
http://myCompany.com/products.php?term=lampa
```

Sposób wykorzystania tych danych zależy wyłącznie od strony *products.php*; na przykład może ich użyć do przeszukania bazy danych i znalezienia w niej produktów związanych z oświetleniem. Mogłoby to być poszukiwanie wystąpień

konkretnego łańcucha znaków, w takim przypadku zwróconymi wynikami mogłyby być: „lampa jarzeniowa", „lampa stojąca" i tak dalej. Jednak równie dobrze program mógłby wykonywać poszukiwanie kontekstowe i zwrócić takie hasła jak „oświetlenie studyjne" czy „żarówki dekoracyjne". Innymi słowy, wyłącznie od programisty zależy, w jaki sposób zostanie przetworzony poszukiwany termin i jakie dane zostaną przesłane z powrotem do przeglądarki.

Uwaga: Widżet Autocomplete umożliwia także przesyłanie danych do innej domeny i na inny serwer. Jednak ze względu na ograniczenia technologii AJAX związane z zapewnieniem bezpieczeństwa serwer docelowy musi być skonfigurowany w taki sposób, by generowane odpowiedzi były zapisane w specjalnym formacie JSONP. Więcej informacji na temat technologii AJAX i formatu JSONP można znaleźć na stronie 506.

Program działający na serwerze musi zwrócić listę podpowiedzi w formie tablicy łańcuchów znaków:

```
[
  "lampa jarzeniowa",
  "lampa błyskowa",
  "lampa stojąca"
]
```

Ewentualnie program mógłby też zwrócić dane w formie tablicy obiektów zawierających etykiety i wartości, opisanej dokładniej na stronie 397. Jednak takie dane musiałyby być zapisane zgodnie z regułami formatu JSON, opisanymi na stronie 500; oznacza to, że zarówno nazwy właściwości, jak i wartości musiałyby być zapisane w cudzysłowach, co pokazano na poniższym przykładzie:

```
var airports = [
  {
    "label" : "Aberdeen Regional Airport, Aberdeen, Dakota Południowa",
    "value" : "ABR"
  },
  {
    "label" : "Abilene Regional Airport, Abilene, Teksas",
    "value" : "ABI"
  },
  {
    "label" : "Abraham Lincoln Capital Airport, Springfield, Illinois",
    "value" : "SPI"
  },
  {
    "label" : "Adak Airport, Adak Island, Alaska",
    "value" : "ADK"
  },
  {
    "label" : "Adirondack Regional Airport, Saranca Lake, Nowy York"
    "value" : "SLK"
  }
];
```

Opcje widżetu Autocomplete

Nie ma zbyt wielu opcji kontrolujących sposób działania widżetu Autocomplete. Najważniejszą z nich (i jednocześnie wymaganą) jest opcja `source` opisana na stronach od 395 do 400. Jednak istnieje także kilka innych opcji, które mogą się

przydać. Podobnie jak w przypadku wszystkich innych widżetów jQuery UI, opcje należy zapisywać w obiekcie przekazywanym w wywołaniu funkcji `autocomplete()`. Oto przykład takiego wywołania:

```
$('#airport').autocomplete({
  source : '/airportSearch.php',
  delay : 500,
  minLength : 2
});
```

Poniżej przedstawione zostały najbardziej użyteczne opcje widżetu Autocomplete.

- **`source`.** To najważniejsza i jedyna wymagana opcja widżetu. Służy do przekazywania bądź to tablicy danych, bądź adresu URL programu działającego na serwerze. W pierwszym przypadku może to być tablica wartości lub tablica obiektów zawierających właściwości `label` oraz `value` (patrz strona 397). Kiedy przekazywany jest adres URL, musi wskazywać program, który zwraca dane zapisane w formie tablicy (patrz strona 395).
- **`delay`.** Podczas pobierania danych z programu działającego na serwerze jQuery UI przesyła żądanie za każdym razem, gdy użytkownik wpisze coś w polu tekstowym. Jednak częste przesyłanie wielu żądań może spowolnić działanie serwera, a tym samym także widżetu Autocomplete. Z tego względu można zastosować niewielkie opóźnienie, dzięki któremu na serwer nie będzie trafiać tak wiele żądań. Wielkość tego opóźnienia podaje się przy użyciu właściwości `delay`, której wartością jest liczba, określająca długość opóźnienia w milisekundach. Aby na przykład przed wysłaniem żądania odczekać pół sekundy, należy użyć poniższej właściwości:

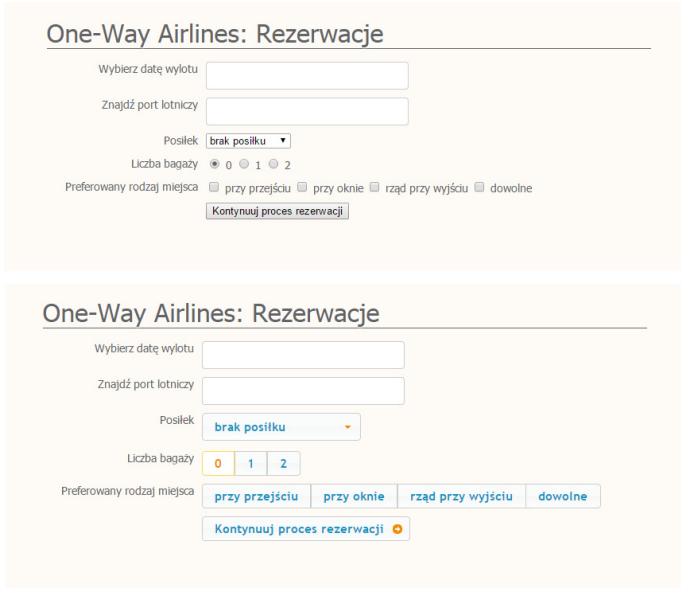
  ```
  delay : 500
  ```

- **`minLength`.** Opcja określa minimalną liczbę znaków, które użytkownik musi wpisać w polu, zanim jQuery UI zacznie wyświetlać listę podpowiedzi. Jeśli źródłem danych dla podpowiedzi są setki lub tysiące rekordów, być może warto będzie zażądać, by ta minimalna liczba znaków wynosiła 3. W końcu, gdyby użytkownik wpisał literę *a*, to w przypadku korzystania z bardzo dużego źródła danych mogłyby zostać wygenerowane tysiące podpowiedzi.

Uwaga: Widżet Autocomplete udostępnia także inne opcje, zdarzenia oraz metody. Ich kompletna lista jest dostępna na stronie *http://api.jqueryui.com/autocomplete/*.

Przykład — widżety UI usprawniające formularze

W tym przykładzie punktem wyjścia będzie formularz do rezerwacji biletów lotniczych; znajdują się na nim pola tekstowe, przyciski opcji oraz element `<button>` (patrz rysunek 10.9, u góry). Kiedy wykorzystasz jQuery oraz widżety jQuery UI, nie tylko poprawisz wizualną atrakcyjność tego formularza, lecz także wzbogacisz możliwości interakcji użytkownika ze stroną oraz jej funkcjonalność (patrz rysunek 10.9, u dołu).



Rysunek 10.9. *Biblioteka jQuery UI ułatwia uatrakcyjnienie wyglądu bezbarwnego formularza i dostosowanie go do reszty aplikacji. Pozwala przekształcać takie elementy jak listy rozwijane, przyciski opcji oraz pola wyboru (których postać zazwyczaj jest zależna od używanej przeglądarki) w interaktywne i przyciągające uwagę elementy interfejsu użytkownika*

Uwaga: Informacje o tym, jak pobrać przykłady prezentowane w książce, zostały zamieszczone na stronie 46.

1. **W edytorze tekstów otwórz plik *form.html* umieszczony w katalogu *R10*.**

   Do pliku zostały już dołączone wszystkie niezbędne pliki bibliotek jQuery i jQuery UI, jak również znacznik `<script>` zawierający wywołanie funkcji `$(document).ready()` (patrz strona 190). Poniżej przedstawiono kod HTML formularza umieszczonego na stronie:

```
<form>
  <div>
    <label for="departure" class="label">Wybierz datę wylotu</label>
    <input type="text" id="departure" name="departure">
  </div>
  <div>
    <label for="airport" class="label">Znajdź port lotniczy</label>
    <input type="text" id="airport" name="airport">
  </div>
  <div>
    <label for="meal" class="label">Posiłek</label>
    <select name="meal" id="meal">
      <option>brak posiłku</option>
      <option>wegański</option>
      <option>bezglutenowy</option>
      <option>wegetariański</option>
      <option>mięsny</option>
    </select>
  </div>
  <div id="bags">
    <p class="label">Liczba bagaży</p>
    <input type="radio" id="none" name="bags" checked="checked">
    <label for="none">0</label>

    <input type="radio" id="one" name="bags">
    <label for="one">1</label>
```

```
      <input type="radio" id="two" name="bags">
      <label for="two">2</label>
    </div>
    <div id="seatTypes">
      <p class="label">Preferowany rodzaj miejsca</p>
      <input type="checkbox" id="aisle" name="aisle">
      <label for="aisle">przy przejściu</label>

      <input type="checkbox" id="window" name="window">
      <label for="window">przy oknie</label>

      <input type="checkbox" id="exit" name="exit">
      <label for="exit">rząd przy wyjściu</label>

      <input type="checkbox" id="any" name="any">
      <label for="any">dowolne</label>
    </div>
    <div>
      <button id="next">Kontynuuj proces rezerwacji</button>
    </div>
  </form>
```

Ważne elementy formularza, w których zastosujesz widżety jQuery UI, zostały wyróżnione pogrubioną czcionką. Zaczniesz od dodania do pierwszego z nich widżetu kalendarza. Zwróć uwagę, że to pole tekstowe ma identyfikator `departure`.

2. **W wywołaniu funkcji `$(document).ready()` wpisz:**

```
$('#departure')
```

Teraz musisz utworzyć widżet kalendarza.

3. **Wpisz kropkę, a za nią `datepicker();`, tak by kod wyglądał w następujący sposób:**

```
$('#departure').datepicker();
```

To już wszystko, co musisz zrobić, by dodać do strony wyskakujący kalendarz do wybierania daty. Jeśli jednak zapiszesz stronę i wyświetlisz ją w przeglądarce, przekonasz się, że kalendarz pozwala na wybieranie dat, które już minęły. Oczywiście trochę trudno byłoby dostać się na lot, który miał miejsce w zeszłym tygodniu, dlatego będziesz musiał zadbać, by nie można było wybrać daty wcześniejszej niż dzisiejsza.

4. **Kliknij wewnątrz wywołania funkcji `datepicker()` i wpisz `{`. Dwukrotnie naciśnij klawisz *Enter* i wpisz `}`, tak by kod wyglądał w następujący sposób:**

```
$('#departure').datepicker({

});
```

Para nawiasów klamrowych — `{ }` — to pusty literał obiektowy. W następnym kroku przekażesz do widżetu opcję, dodając do tego literału właściwości.

5. **W pustym wierszu wewnątrz literału obiektowego wpisz `minDate : 0`.**

Wartość opcji `minDate` określa liczbę dni, licząc od dziś: 0 *oznacza* dzień dzisiejszy, –7 — dzień wypadający tydzień temu, a 7 — dzień dokładnie za tydzień.

Możesz także określić *maksymalną* dopuszczalną datę, którą użytkownik będzie mógł wybrać w kalendarzu. W naszym przykładzie linie lotnicze nie gromadzą informacji o lotach, które odbędą się za więcej niż jeden rok; dlatego zadbasz o to, by użytkownik nie mógł wybrać daty z tak dużym wyprzedzeniem.

6. **Na końcu wiersza wpisanego w poprzednim punkcie dodaj przecinek, naciśnij klawisz *Enter*, po czm wpisz: `maxDate : '+1y'`, tak by kod wyglądał w poniższy sposób:**

```
$('#departure').datepicker({
  minDate : 0,
  maxDate : '+1y'
});
```

W obu właściwościach, `minDate` i `maxDate`, można podać wartość liczbową określającą liczbę dni. Jednak oprócz tego można w nich także zapisać łańcuch znaków zawierający liczby i litery ustalające długość okresu czasu; na przykład: `'+1y'` oznacza „za jeden rok", `'-2w'` oznacza „dwa tygodnie temu", a `'+1m +10d'` oznacza „za miesiąc i dziesięć dni".

7. **Zapisz stronę i wyświetl ją w przeglądarce. Kliknij pole „Wybierz datę wylotu".**

Na stronie zostanie wyświetlony kalendarz. Zauważ, że z kalendarza nie można wybrać żadnej daty z przeszłości ani daty z wyprzedzeniem większym niż jeden rok od daty dzisiejszej (jeśli chcesz się o tym przekonać, użyj przycisku strzałki w prawo, aby przejść o 12 miesięcy do przodu).

Teraz użyjesz widżetu Autocomplete, by ułatwić użytkownikowi wybór lotniska. Jednak zanim to zrobisz, przyjrzyj się danym, których będziesz używał. Zgodnie z informacjami podanymi na stronie 395, funkcja `autocomplete()` wymaga przekazania bądź to tablicy danych, bądź też adresu URL skryptu działającego na serwerze. Aby uprościć ten przykład, dane zapiszemy w odrębnym pliku JavaScript.

8. **W edytorze tekstów otwórz plik *airports.js*.**

To prosty plik JavaScript zawierający instrukcję przypisania — tworzymy w nim tablicę, którą zapisujemy w zmiennej `airports`. Tablica ta zawiera obiekty składające się z dwóch właściwości: `label` oraz `value`. Etykieta (wartość właściwości `label`) będzie wyświetlana na rozwijanej liście podpowiedzi, natomiast wartość (wartość właściwości `value`) zostanie zapisana w polu tekstowym po wybraniu jednej z opcji wyświetlonych na liście.

Oczywiście ten plik nie zawiera kompletnej listy portów lotniczych, a jedynie niewielką liczbę danych pozwalającą na wypróbowanie działania widżetu. Aby skorzystać z danych zapisanych w tym pliku, będziesz musiał dołączyć go do strony.

9. **W edytorze tekstów otwórz plik *form.html*. Poniżej ostatniego wiersza ze znacznikiem `<script>` — `<script src="datepicker-pl.js"></script>` — dodaj jeszcze jeden znacznik `<script>`, który dołączy do strony plik z danymi:**

```
<script src="airports.js"></script>
```

Ten wiersz dołącza do strony zewnętrzny plik JavaScript, a kiedy przeglądarka już go pobierze, wykona umieszczony w nim kod. W tym przypadku działanie kodu sprowadza się do utworzenia tablicy z danymi, których użyjesz w widżecie Autocomplete.

10. **Poniżej kodu widżetu kalendarza dodaj pusty wiersz i wpisz w nim:**

```
$('#airport').autocomplete({ source : airports });
```

Pole tekstowe, w którym użytkownik ma wybrać port lotniczy, ma identyfikator `airport`. A zatem wywołanie `$('#airport')` wybiera to pole, a wywołanie `.autocomplete()` dodaje do niego widżet Autocomplete.

11. **Zapisz stronę i wyświetl ją w przeglądarce. Kliknij pole „Znajdź port lotniczy" i wpisz w nim `Port`.**

    Poniżej pola pojawi się lista podpowiedzi, która być może będzie zawierać ten port lotniczy, o który Ci chodzi. Zwróć uwagę, że widżet odnajduje porty lotnicze, w których nazwie słowo „port" występuje w dowolnym miejscu, na przykład „La Guardia Air*port*".

12. **Kliknij opcję „Portland International Airport, Portland, OR".**

    Zwróć uwagę, że w polu pojawiła się wartość „PDX". Wynika to z faktu, że źródłem danych dla widżetu jest tablica obiektów zawierających etykiety i wartości. Etykietą wyświetloną na liście była nazwa „Portland International Airport, Portland, OR", jednak wartością, którą jQuery UI zapisze w polu, będzie kod lotniska — „PDX".

    Teraz zajmiesz się przekształceniem listy rozwijanej w piękny, interaktywny widżet.

13. **Wróć do edytora tekstów i kodu strony *form.html*. Poniżej kodu dodanego w kroku 10. wpisz następny wiersz:**

    ```
    $('#meal').selectmenu();
    ```

    Być może chciałbyś wpisać więcej kodu, ale to już wszystko, czego potrzebujesz, by przekształcić listę rozwijaną. Jeśli jednak zapiszesz stronę i wyświetlisz ją w przeglądarce, zobaczysz, że lista wygląda dosyć dziwnie. Jej pierwszy element nie będzie widoczny w całości! Jeśli opcje prezentowane na liście nie są naprawdę krótkie, będziesz musiał określić jej szerokość.

14. **Wewnątrz nawiasów funkcji `selectmenu()` wpisz: `{ width : 200 }`, tak by kod wyglądał w następujący sposób:**

    ```
    $('#meal').selectmenu( { width : 200 } );
    ```

    Opcja `width` widżetu Selectmenu (patrz strona 385) pozwala określić szerokość listy widocznej na stronie. W tym przypadku liczba `200` oznacza, że lista będzie mieć 200 pikseli szerokości. Zgodnie z informacjami podanymi na stronie 386, możesz także używać innych sposobów określania długości, takich jak wartości procentowe lub jednostki `em`. Zapisz stronę i wyświetl ją w przeglądarce. Kto powiedział, że programowanie jest trudne?

    Teraz przekształcisz grupę przycisków opcji i pól wyboru w coś, co jest znacznie lepiej dopasowane wyglądem do powstającego formularza.

15. **Poniżej kodu wpisanego w poprzednim kroku dodaj dwa nowe wiersze:**

    ```
    $('#bags').buttonset();
    $('#seatTypes').buttonset();
    ```

    Funkcja `buttonset()` operuje zarówno na przyciskach opcji, jak i polach wyboru. Ostatnim krokiem jest przekształcenie znacznika `<button>` na przycisk jQuery UI.

16. **Do kodu programu dodaj jeszcze jeden wiersz:**

    ```
    $('#next').button();
    ```

To wywołanie przekształca bezbarwny przycisk HTML w coś, co znacznie lepiej pasuje wyglądem do formularza i umieszczonych na nim pól wyboru i przycisków opcji. Do elementów `<button>` można także dodawać ikony; informacje na ten temat znajdziesz na stronie 390. Właśnie tym zajmiesz się w kolejnym kroku.

17. **Wewnątrz nawiasów funkcji `button()` wpisz poniższy fragment kodu:**

```
{
  icons : {
    secondary : 'ui-icon-circle-arrow-e'
  }
}
```

A tak powinna wyglądać ostateczna wersja kodu:

```
$(document).ready(function() {
  $('#departure').datepicker({
    minDate : 0,
    maxDate : '+1y'
  });
  $('#airport').autocomplete({ source : airports});
  $('#meal').selectmenu({width : 200});
  $('#bags').buttonset();
  $('#seatTypes').buttonset();
  $('#next').button({
    icons : {
      secondary : 'ui-icon-circle-arrow-e'
    }
  });
}); // Koniec funkcji ready.
```

Tych kilka wierszy kodu pozwoliło całkowicie zmienić wygląd i funkcjonalność formularza.

18. **Zapisz stronę i wyświetl ją w przeglądarce.**

Obecnie strona powinna wyglądać tak, jak ta z rysunku 10.9. Jeśli jest inaczej, upewnij się, że Twój kod JavaScript wygląda dokładnie tak samo jak przedstawiony powyżej, w kroku 17. Możesz także wyświetlić konsolę JavaScript (patrz strona 51) i sprawdzić, czy są w niej pokazane jakieś błędy. Musisz jednak pamiętać, że jQuery niejednokrotnie ukrywa błędy i nie wyświetla ich w oknie konsoli, co nieco utrudnia diagnozowanie problemów w kodzie korzystającym z tej biblioteki.

Pełną wersję tego przykładu możesz znaleźć w pliku *complete_form.html*, umieszczonym w katalogu *R10*.

11

ROZDZIAŁ

Dostosowywanie wyglądu jQuery UI

Widżety jQuery UI mają ujednolicony wygląd — kalendarz do wyboru dat wygląda podobnie jak karty, które z kolei wyglądają podobnie jak okno dialogowe oraz etykietki ekranowe. Jeśli zgromadzisz kolekcję odrębnych wtyczek jQuery implementujących te same widżety, lecz napisanych przez różnych autorów, stracisz bardzo dużo czasu na modyfikowanie arkuszy stylów po to, by zapewnić ich spójny wygląd. Jednolity sposób prezentacji, jaki zapewniają wszystkie widżety jQuery UI, oznacza, że można tworzyć aplikacje o spójnym wyglądzie bez konieczności poświęcania niezliczonych godzin na samodzielne modyfikowanie kodu CSS.

Co jednak zrobić, gdy już dysponujemy witryną o własnym, charakterystycznym wyglądzie i chcemy dostosować wygląd jQuery UI do jej projektu? Z myślą o takich sytuacjach zespół twórców jQuery UI przygotował wiele praktycznych porad, a nawet opracował bardzo pomocne narzędzie. W tym rozdziale znajdziesz informacje, w jaki sposób można nadpisywać lub modyfikować istniejące style jQuery UI oraz jak tworzyć nowe.

Prezentacja narzędzia ThemeRoller

Biblioteka jQuery UI zawiera wiele elementów, zatem utworzenie arkuszy CSS zapewniających wspaniały (i jednocześnie spójny) wygląd wszystkich jej widżetów jest naprawdę dużym zadaniem. Na szczęście zespół twórców biblioteki udostępnił w internecie narzędzie o nazwie *ThemeRoller*. Pozwala ono na wybór jednego z 24 gotowych tematów graficznych, określających wygląd biblioteki jQuery UI. Dodatkowo ThemeRoller oferuje także specjalne narzędzie pozwalające na modyfikowanie istniejących tematów — wybierane czcionek, zmienianie kolorów tak, by pasowały do projektu witryny.

Aby skorzystać z narzędzia ThemeRoller, należy wejść na stronę *http://jqueryui.com/themeroller/* (patrz rysunek 11.1). Kolekcję gotowych tematów graficznych można wyświetlić, klikając kartę *Gallery* (zaznaczoną kółkiem). Główna część strony prezentuje wygląd różnych widżetów jQuery UI: wybór jednego z tematów graficznych, których miniaturki są przedstawione w lewej kolumnie, powoduje jego natychmiastowe zastosowanie i możliwość oglądnięcia sposobu prezentacji widżetów w wybranym temacie.

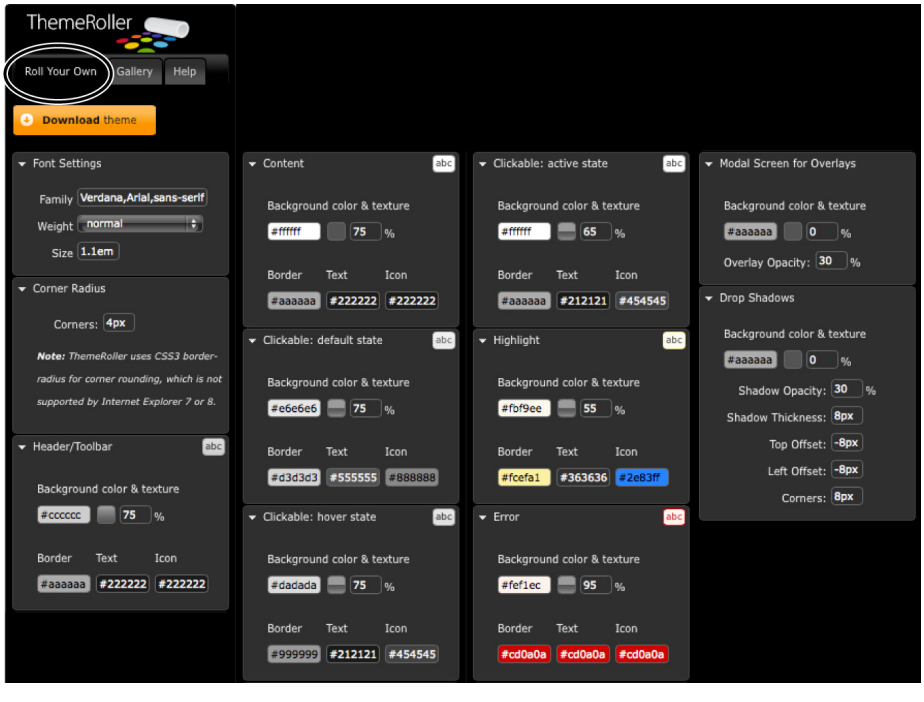
Rysunek 11.1. *Ze strony ThemeRoller biblioteki jQuery UI pobierzesz przygotowane tematy graficzne. Można wybierać spośród 24 gotowych projektów. Aby pobrać wybrany temat, wystarczy kliknąć przycisk Download. Te gotowe tematy mogą także stanowić punkt wyjścia podczas prób tworzenia własnych projektów. W tym celu należy kliknąć przycisk Edit umieszczony poniżej miniaturki tematu. W efekcie ThemeRoller wczyta wybrany temat; teraz można modyfikować używane w nim czcionki i kolory przy użyciu opcji dostępnych na karcie Roll Your Own*

Jeśli podoba Ci się wygląd jakiegoś tematu widocznego na karcie *Gallery*, wystarczy kliknąć przycisk *Download* umieszczony poniżej miniaturki tematu; spowoduje to wyświetlenie strony *jQuery Download Builder*, opisanej na stronie 327. Na niej można wybrać widżety, metody interakcji oraz efekty, których planujesz używać, a następnie kliknąć przycisk *Download* umieszczony u dołu, aby pobrać pliki.

Uwaga: Zajrzyj na stronę 327, aby dowiedzieć się, w jaki sposób można pobrać oraz zorganizować pliki jQuery, byś mógł z nich korzystać na swojej witrynie.

Jeśli chcesz dodać jQuery UI do już istniejącej witryny, skorzystaj z narzędzia ThemeRoller w celu opracowania własnego tematu graficznego, dostosowanego do jej projektu. Kliknij kartę *Roll Your Own* umieszczoną w kolumnie z lewej strony (zakreśloną na rysunku 11.2). Narzędzie to zapewnia dostęp do ustawień związanych między innymi z czcionkami oraz kolorami. Ustawienia są podzielone na kategorie, które można wyświetlać i ukrywać, klikając strzałkę widoczną z lewej strony nazwy kategorii. Poniżej przedstawiona została lista dostępnych kategorii ustawień.

- ***Font settings*:** Ta kategoria pozwala na określanie używanych czcionek, ich wielkości oraz wagi. Nazwy czcionek, których chcemy używać, należy wpisać w polu *Family*. Zazwyczaj podawane są trzy wersje używanych czcionek: właściwa czcionka, z której chcielibyśmy skorzystać, czcionka awaryjna oraz ogólny typ czcionki awaryjnej. Jeśli właściwa czcionka nie będzie dostępna na komputerze użytkownika, zostanie zastosowana czcionka awaryjna; jeśli także i ona nie będzie osiągalna, przeglądarka wybierze jedną z dostępnych czcionek należących do tego samego typu ogólnego (`serif`, `sans-serif`, `monospace` lub `fantasy`).



Rysunek 11.2. *Aby wyświetlić kategorię ustawień, kliknij strzałkę umieszczoną z lewej strony nazwy wybranej kategorii. Domyślnie wszystkie kategorie są ukryte, jednak tutaj pokazano, jak wyglądają po wyświetleniu (aby wszystkie zmieść ły się na obrazku, rozmieszczono je jedna obok drugiej). Na stronie ThemeRoller wszystkie te kategorie są wyświetlone jedna nad drugą, w kolumnie po lewej stronie*

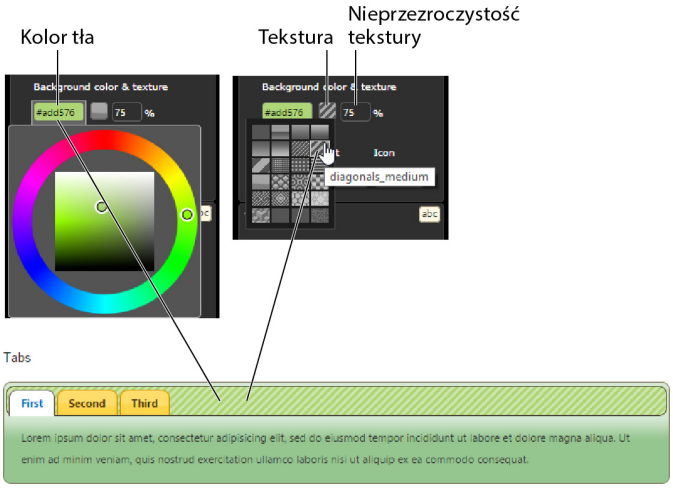
Należy tu podać tę samą listę czcionek, które są używane na witrynie. Jeśli na przykład na Twojej witrynie główną używaną czcionką jest Helvetica Neue, powinieneś tu wpisać: `Helvetica Neue , Arial, sans-serif`. (Kiedy nazwa używanej czcionki składa się z więcej niż jednego wyrazu, na przykład Helvetica Neue, powinieneś ją zapisywać w cudzysłowach). jQuery UI zastosuje tę samą czcionkę we wszystkich widżetach oraz ich elementach, dotyczy to także kart oraz paneli. Można także określać odrębne czcionki, które będą stosowane w różnych komponentach; informacje na ten temat zostały podane na stronie 415.

Uwaga: Jeśli używasz czcionki, która jest dostępna na Twoim serwerze lub jest dostarczana przez serwisy, takie jak TypeKit lub Google Fonts, bądź też dowolnej innej czcionki niezainstalowanej na Twoim komputerze, nie będzie ona widoczna w podglądzie prezentowanym na stronie *ThemeRoller*.

Wagę czcionki można określić jako `normal` lub `bold` . Zastosowanie wartości `bold` nie sprawi, że wszystkie teksty prezentowane w widżetach jQuery UI będą pogrubione — ustawienie to odnosi się wyłącznie do niektórych elementów, takich jak teksty prezentowane na kartach lub przyciskach, wybrana opcja menu czy też nagłówki w widżecie accordion. Wszystkie pozostałe teksty prezentowane są zwyczajną czcionką.

Wielkość czcionki określa bazową wielkość czcionki dla wszystkich widżetów jQuery UI. Jednak teksty prezentowane w niektórych widżetach mogą być większe. Przykładami takich elementów, w których tekst prezentowany jest większą czcionką, są tytuły okien dialogowych oraz teksty prezentowane w etykietkach ekranowych.

- ***Corner Radius.*** Większość widgetów jQuery UI ma wierzchołki, a promień tych wierzchołków określany w tej kategorii, sprawia, że będą one bardziej lub mniej okrągłe. Zastosowanie wartości 0 sprawi, że wierzchołki będą prostokątne, a im wyższa wartość zostanie podana, tym będą bardziej okrągłe. Warto wpisywać różne wartości i sprawdzić, które ustawienie najbardziej Ci odpowiada.
- ***Header/Toolbar.*** Ta kategoria ustawień pozwala na określanie kolorów oraz tła nagłówka w kalendarzach lub oknach dialogowych, a także koloru pasków postępu i suwaków. Udostępnia ona sześć ustawień.
- **Kolor tła.** Kliknięcie tego pola spowoduje wyświetlenie okna do wyboru koloru. Można w nim wybrać odcień (w tym celu należy kliknąć zewnętrzny okrąg), a następnie konkretny kolor (klikając obszar pośrodku okna). Ta grupa opcji została przedstawiona na rysunku 11.3.
- **Tekstura tła.** Biblioteka jQuery UI daje możliwość wyboru tekstur (takich jak poziome lub skośne paski), które będą wyświetlane w tle elementów. Wystarczy kliknąć to pole, aby wyświetlić paletę dostępnych tekstur. Następnie można kliknąć jedną z tekstur lub jednolity kolor (w lewym górnym rogu palety), aby zrezygnować ze stosowania tekstury tła. Tekstura ta jest tworzona poprzez wyświetlanie w tle elementu określonego, małego obrazka (patrz rysunek 11.3).



Rysunek 11.3. *Wiele kategorii narzędzia ThemeRoller pozwala na określanie tych samych trzech ustawień: koloru tła, tekstury oraz nieprzezroczystości tekstury. Kiedy przyzwyczaisz się do tych ustawień, będziesz w stanie określać postać tła wszystkich widżetów jQuery UI*

- **Nieprzezroczystość tekstury tła.** Opcja określa, w jakim stopniu będzie widoczna tekstura tła. Jeśli przypiszesz jej wartość 0, tekstura w ogóle nie będzie widoczna. Wartość 10% sprawi, że wzorek będzie bardzo delikatny, a wartość 75%, że będzie wyraźnie widoczny (patrz rysunek 11.3).
- **Kolor obramowania.** Opcja pozwala określić kolor obramowania wyświetlanego wokół nagłówka lub paska narzędzi. Kliknięcie pola spowoduje wyświetlenie okna do wyboru koloru. Jeśli nie chcesz, by widżet miał obramowanie, nadaj mu taki sam kolor, jaki ma tło — dzięki temu tło i obramowanie nie będą się odróżniać. (Na rysunku 11.3 przedstawiono to samo okno do wyboru koloru, które jest używane do określania wszystkich ustawień związanych z kolorami).
- **Kolor tekstu.** Kliknij to pole, by wyświetlić okno wyboru koluru, w jakim będą prezentowane teksty w nagłówkach i paskach narzędzi.
- **Kolor ikon.** W niektórych widżetach są prezentowane ikony. Przykładem może być kalendarz, w którym są wyświetlane ikony poprzedniego i następnego miesiąca. Aby określić ich kolor, wystarczy kliknąć to pole i w wyświetlonym oknie dialogowym wybrać kolor pasujący do projektu witryny.

Uwaga: Określając wartości wszystkich opcji związanych z kolorami, można także kliknąć pole tekstowe i wpisać szesnastkową wartość wybranego koloru, na przykład `#e53c4c`.

- ***Content.*** Ta kategoria pozwala na określanie tła, obramowania oraz koloru tekstu obszaru zawartości widżetów. Dotyczy to paneli widżetu accordion, paneli treści na kartach, elementów menu w widżecie Selectmenu oraz cyfr widocznych w widżecie kalendarza. Dostępne tu opcje są takie same jak w kategorii *Header/Toolbar* — pozwalają na określanie koloru tła, jego tekstury i nieprzezroczystości oraz koloru obramowania, tekstu i ikon.
- ***Clickable items.*** Narzędzie *ThemeRoller* udostępnia trzy kategorie elementów, które można klikać, czyli tych fragmentów widżetów, które klikasz, by zapewnić ich działanie; mogą to być nagłówki w widżecie accordion, karty, przyciski

numerów w kalendarzu, ikony, przyciski w oknach dialogowych i opcje menu. Trzy dostępne kategorie odpowiadają trzem stanom, w których mogą się znajdować takie elementy; są to *stan domyślny* (ang. *default state*, który określa wygląd elementu bezpośrednio po wyświetleniu strony), *stan po wskazaniu myszą* (ang. *hover state*, który występuje, gdy użytkownik umieści wskaźnik myszy w obszarze elementu) oraz *stan aktywny* (ang. *active state*). Ten ostatni stan występuje, gdy użytkownik wybrał jakąś kartę, zaznaczył datę w kalendarzu bądź przycisk (na przykład przycisk opcji). Każda z tych kategorii udostępnia grupę tych samych sześciu opcji, które zostały opisane w kategorii *Header/Toolbar*, na stronie 410, czyli kolor tła, teksturę tła, nieprzezroczystość tła oraz kolory obramowania, tekstu i ikon.

- Kategorie ***Highlight*** oraz ***Error***. Ustawienia dostępne w tych kategoriach nie określają wyglądu żadnych gotowych komponentów. Dotyczą one dwóch klas CSS — `.ui-state-highlight` oraz `.ui-state-error` — których można używać w widżetach bądź dowolnych innych elementach strony. Przykładowo widżet okna dialogowego (przedstawiony na stronie 330) pozwala na zastosowanie w oknie dialogowym klasy o podanej nazwie. A zatem, jeśli zależy Ci na wyświetleniu wyróżnionego okna dialogowego, wystarczy przekazać odpowiednią nazwę klasy (bez kropki z przodu) podczas tworzenia okna, w sposób przedstawiony na poniższym przykładzie:

  ```
  $('#dialogDiv').dialog({ dialogClass : 'ui-state-highlight' });
  ```

  W podobny sposób możesz zastosować klasę `.ui-state-error` w dowolnym elemencie strony, który chcesz wyróżnić. Jeśli na przykład w jakimś polu formularza użytkownik wpisał nieprawidłową wartość, możesz pobrać etykietę tego pola i dynamicznie dodać do niej klasę reprezentującą błąd:

  ```
  $('#userNameLabel').addClass('ui-state-error');
  ```

  W tym przypadku nie używasz żadnego widżetu jQuery UI, a jedynie klasy CSS utworzonej przez narzędzie *ThemeRoller*. Okazuje się, że w taki sam sposób, w dowolnych elementach strony możesz stosować wszystkie klasy dostępne w arkuszu stylów jQuery UI (więcej informacji na ten temat można znaleźć na stronie 415).

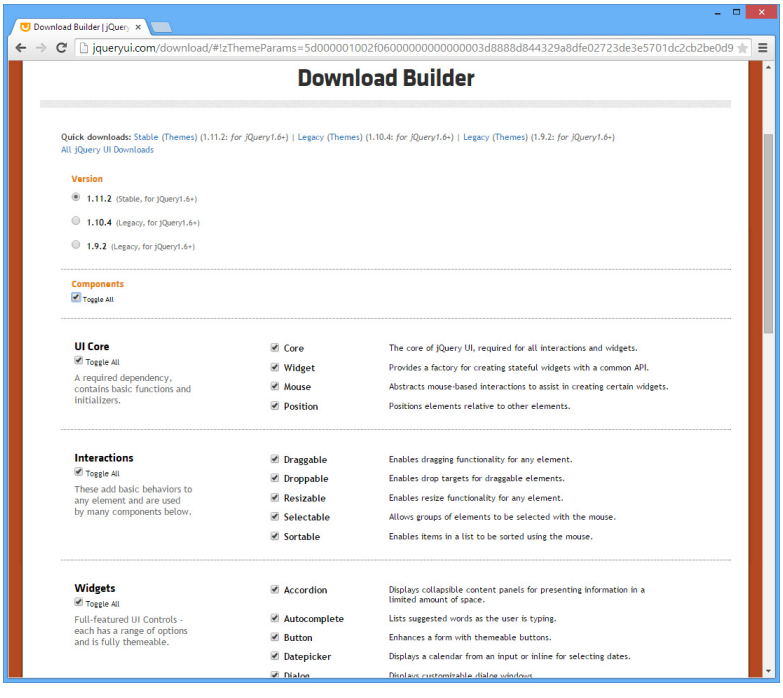
- ***Modal Screen for overlays.*** Widżet `Dialog` (patrz strona 330) można stosować do tworzenia modalnych okien dialogowych — czyli okien, które aż do momentu zamknięcia uniemożliwiają użytkownikom interakcję z wszelkimi innymi elementami strony. Ten typ okien dialogowych jest stosowany do prezentowania ważnych komunikatów oraz wykonywania istotnych czynności, takich jak „Czy jesteś pewny, że chcesz całkowicie usunąć swój profil na Facebooku?". Wyświetlając takie okno dialogowe, biblioteka jQuery UI zaciemnia całą pozostałą zawartość strony, wyświetlając nad nią (a jednocześnie pod oknem dialogowym) specjalną „nakładkę". Wygląd tej nakładki można określać, dobierając jej kolor tła, teksturę oraz nieprzezroczystość tekstury (czyli korzystając z tych samych ustawień, które zostały opisane w kategorii *Header/Toolbar* na stronie 410).

- ***Drop shadows.*** Tę kategorię można zignorować, gdyż dostępne w niej ustawienia nie są używane w żadnym z widżetów jQuery UI — nawet w widżecie `Tooltip`, który *ma* cień. To dziwne, ale prawdziwe.

Wskazówka: Kiedy korzystasz z narzędzia ThemeRoller w celu przygotowania własnego tematu graficznego dla widżetów jQuery UI, zmienia się także adres URL strony: każda zmiana ustawień powoduje modyfikację tego adresu. Dlaczego tak się dzieje? Wynika to z faktu, że *ThemeRoller* w bardzo inteligentny sposób tworzy adres URL, który dokładnie odwzorowuje ustawienia budowanego tematu. A to oznacza, że można ten adres URL zapisać i w bardzo prosty sposób wrócić na stronę *ThemeRoller*, na której zostaną odtworzone wybrane wcześniej ustawienia. Można także skopiować ten adres i przesłać go znajomemu lub koledze, żeby mogli błyskawicznie skopiować przygotowany temat.

Pobieranie i stosowanie nowego tematu

Po przygotowaniu nowego tematu jQuery UI wystarczy kliknąć przycisk *Download Theme* umieszczony z lewej strony, u góry karty *Roll Your Own*. Spowoduje to wyświetlenie strony *Download Builder* (patrz rysunek 11.4). Aby pobrać wszystkie widżety oraz wszystkie możliwości funkcjonalne biblioteki, należy po prostu kliknąć przycisk *Download* umieszczony u dołu strony. Jeśli jednak chcesz pobrać tylko część możliwości — na przykład tylko widżet okna dialogowego (*Dialog*), kalendarz (*Datepicker*) oraz karty (*Tabs*) — usuń zaznaczenie z pól wyboru przy wszystkich pozostałych widżetach, metodach interakcji (*Interactions*) i efektach animacji (*Effects*) i dopiero potem kliknij przycisk *Download*.



Rysunek 11.4. *Strona Download Builder pozwalająca na pobranie wersji biblioteki jQuery dostosowanej do naszych potrzeb. Pozwala ona na wybranie numeru wersji biblioteki, widżetów, sposobów interakcji oraz efektów animacji, których będziemy używać i które chcemy pobrać. Dzięki temu możemy wyeliminować te możliwości biblioteki, które nie są potrzebne, zmniejszając tym samym jej wielkość i czas konieczny do pobrania*

Dodawanie własnego tematu do istniejących stron WWW

Jeśli już używałeś jQuery UI na swojej witrynie i jedynie chcesz zaktualizować wygląd widżetów, korzystając z nowego tematu graficznego, będziesz potrzebował jedynie nowych stylów. Możesz wykonać ten sam proces, który został opisany w poprzednim punkcie rozdziału, aby przygotować i pobrać nowy schemat graficzny. Pamiętaj jednak, aby na stronie *Download Builder* (przedstawionej na rysunku 11.4) zaznaczyć te same opcje — widżety, sposoby interakcji oraz efekty — których aktualnie używasz. W przeciwnym razie pobierzesz arkuszy stylów, który jest zbyt mały lub niepotrzebnie duży.

Jeśli na przykład na swojej witrynie używasz wszystkich widżetów jQuery UI, lecz podczas pobierania biblioteki nie zaznaczysz widżetu *Dialog*, nie pobierzesz arkuszy stylów niezbędnych do określania wyglądu okien dialogowych. I podobnie, jeśli używasz wyłącznie widżetu okna dialogowego, ale pobierzesz cały zestaw możliwości biblioteki jQuery UI, to pobrany arkusz stylów będzie *większy*, niż to konieczne, a to z kolei oznacza, że użytkownicy będą marnowali czas na pobieranie niepotrzebnych stylów CSS.

Kiedy już pobierzesz nową kopię jQuery UI, pojawią się także pliki, których już nie będziesz potrzebował (jeśli na przykład już używasz jQuery UI, nie będziesz musiał zastępować istniejących plików JavaScript). Jedynymi elementami biblioteki, które będziesz musiał zastąpić, są katalog *images* oraz pliki CSS.

1. **Zastąp wcześniejszy plik lub pliki CSS biblioteki jQuery UI plikami nowego tematu graficznego.**

   Zgodnie z informacjami podanymi na stronie 329, w skład biblioteki jQuery UI wchodzi sześć plików CSS. Większość z nich stanowią powtarzające się style, zapisane w różnych formatach. Mówiąc krótko, jedynym plikiem, którego *naprawdę* potrzebujesz, jest arkusz *jquery-ui.min.css*. Zawiera on wszystkie style CSS wymagane przez bibliotekę i zajmuje mało miejsca, gdyż jest zminimalizowany. Ogólnie rzecz biorąc, powinieneś używać tych plików, w których nazwie znajduje się człon *.min*, gdyż są zazwyczaj najmniejsze. Ten nowy plik CSS powinieneś umieścić dokładnie w tym samym miejscu, w którym znajdował się poprzedni — na przykład w katalogu *css* umieszczonym w głównym katalogu witryny.

2. **Zastąp dotychczasowy katalog *images* nowym katalogiem o tej samej nazwie.**

   W różnych tematach używane są ikony o odmiennej kolorystyce oraz różnych teksturach tła. Ikony i grafika nowego tematu zapewne nie będą odpowiadały tym, które były stosowane w dotychczas używanym temacie graficznym, dlatego też powinieneś je zastąpić. Katalog *images* powinien znaleźć się w tym samym katalogu, w którym przechowywane są arkusze stylów jQuery UI; na przykład może to być katalog *css*, umieszczony w głównym katalogu witryny, bądź też katalog *jquery-ui*. Innymi słowy, niezależnie od tego, gdzie konkretnie zostanie umieszczony plik CSS biblioteki, w tym samym katalogu musisz umieścić katalog *images*.

Uwaga: Podczas pobierania nowego tematu graficznego upewnij się, że nie uległa zmianie wersja biblioteki jQuery UI. Biblioteka ta zmienia się dosyć często, dlatego, jeśli zdecydujesz się na aktualizację tematu graficznego, może się okazać, że w międzyczasie zmienił się także sam kod biblioteki. W takim przypadku *powinieneś* zastąpić style CSS, katalog *images* oraz plik JavaScript biblioteki jQuery UI (*jquery-ui.min.js*).

Więcej informacji o arkuszach stylów jQuery UI

W skład jQuery UI wchodzi sześć plików CSS. Konkretnie rzecz biorąc, są to dwie grupy po trzy pliki: pierwszą grupę tworzą pliki zminimalizowane lub spakowane (na przykład *jquery-ui.min.css*), a drugą pliki CSS, których zawartość można bez trudu przejrzeć i przeanalizować (na przykład *jquery-ui.css*).

Zgodnie z informacjami podanymi wcześniej, do używania na produkcyjnych witrynach najlepiej nadają się zminimalizowane wersje kodów JavaScript, gdyż ich pobieranie zajmuje najmniej czasu. Pozostałe pliki doskonale nadają się do wyświetlania w edytorze tekstów i umożliwiają sprawdzenie, jak są utworzone style określające wygląd biblioteki.

Największy z tych plików — *jquery-ui.css* — jest połączeniem dwóch pozostałych (*jquery-ui.structure.css* oraz *jquery-ui.theme.css*). Zespół pracujący nad jQuery rozdzielił ten główny plik na dwie części, gdyż istnieją pewne style, które zawsze są stosowane we *wszystkich* elementach, niezależnie od używanych w nich czcionek i kolorów. Style te stanowią „strukturę" widżetów oraz interakcji jQuery UI i zawierają takie właściwości jak wartość z-index okien dialogowych oraz położenie etykietek ekranowych. Wszystkie te style znajdują się w pliku o nazwie *jquery-ui.structure.css*.

Jednak pozostałe style zmieniają się w zależności od wybranego schematu graficznego. Przykładowo rodziny czcionek używanych do wyświetlenia tekstu w widżetach czy też koloru tła nagłówków w widżetach accordion są ustawieniami charakterystycznymi dla konkretnego tematu graficznego. Wszystkie takie style zostały umieszczone w pliku *jquery-ui-theme.css*. Możesz go otworzyć, by przeanalizować style mające wpływ na teksty i tła elementów.

Dlatego też do swoich stron powinieneś dołączyć arkusz *jquery-ui.min.css*, natomiast pliku *jquery-ui-theme.css* używać jako źródła informacji o nazwach stylów i właściwościach różnych widżetów jQuery UI.

Przesłanianie stylów jQuery UI

Tematy graficzne jQuery UI wyglądają całkiem dobrze, jednak w naszym przypadku wcale nie muszą nadawać się do użycia. Przykładowo korzystając z ustawień określonych na stronie *ThemeRoller* (patrz strona 407), mamy do dyspozycji tylko jedno ustawienie opisujące czcionki używane w oknach dialogowych, widżecie do wyboru daty, kartach i tak dalej. A co zrobić, gdybyśmy chcieli skorzystać z innych czcionek w kartach, a innych w panelach zawierających treści stron?

Taki problem można rozwiązać na kilka różnych sposobów, choć jest jedna rzecz, której definitywnie *nie należy robić* — nie należy samodzielnie modyfikować arkuszy stylów jQuery UI. Arkusze te zmieniają się wraz z kolejnymi wersjami biblioteki; jeśli zatem wprowadzisz jakieś zmiany w pliku *jquery-ui.css*, a następnie zaktualizujesz jQuery UI, narazisz się na trudne zadanie odtworzenia wszystkich zmian, które kiedyś do tego pliku wprowadziłeś.

A jeśli nawet nie aktualizujesz biblioteki jQuery UI, nie będziesz mógł używać strony *ThemeRoller* do tworzenia nowych projektów, bez narażania się na konieczność uciążliwego powielania wprowadzonych wcześniej zmian.

Zamiast tego nowe style powinieneś tworzyć w odrębnym pliku CSS. Może to być osobny plik przeznaczony specjalnie na nowe style związane z jQuery UI bądź też może to być główny arkusz stylów witryny. Jak się niebawem przekonasz, istnieje kilka technik przesłaniania stylów jQuery UI, jednak wszystkie bazują na stosowanym w kaskadowych arkuszach stylów pojęciu *szczegółowości*. Już niedługo dowiesz się znacznie więcej na ten temat, jednak na razie wystarczy, że zapamiętasz jedno: arkuszy stylów przesłaniający domyślne style jQuery UI należy dołączyć *po* arkuszu stylów biblioteki:

```
<link href="css/jquery-ui.min.css" rel="stylesheet">
<link href="css/site.css" rel="stylesheet">
```

Dołączając arkusz stylów witryny za arkuszem jQuery UI, możesz tworzyć style zawierające *dokładnie te same* nazwy, co domyślne style biblioteki. W takim przypadku style opracowane przez Ciebie będą miały wyższy priorytet, gdyż zostaną dołączone później niż style jQuery UI.

Zasada szczegółowości

Style CSS często współdziałają ze sobą lub wywołują konflikty. Może istnieć wiele stylów określających postać tego samego elementu strony, a przeglądarka określa, która właściwość z którego stylu zostanie ostatecznie użyta. To właśnie stąd wzięło się słowo „kaskadowe" w nazwie „kaskadowe arkusze stylów". Ta cecha stylów wiąże się z bardzo ważnym pojęciem — ze *szczegółowością* (ang. *specificity*). W CSS bardziej szczegółowe reguły zawsze są ważniejsze od mniej szczegółowych. Oto kilka najważniejszych zasad związanych ze szczegółowością.

- **Selektory identyfikatorów są bardziej szczegółowe od selektorów klas, a te z kolei od selektorów elementów.** Oto przykład:

  ```
  <p id="susan" class="person">Zuzanna Mazur: bohaterka CSS</p>
  ```

  W tym przykładzie, w znaczniku `<p>` został umieszczony zarówno atrybut `id` określający identyfikator, jak i atrybut `class` określający klasę. Teraz załóż, że dysponujesz następującym arkuszem stylów:

  ```
  #susan {
    color: green;
    font-size: 24px;
  }
  .person {
    color: blue;
    text-align: center;
    font-weight: bold;
  ```

```
}
p {
  color: orange;
  font-weight: normal;
  font-family: Arial, sans-serif;
}
```

Wszystkie trzy powyższe style opisują postać akapitu i w każdym z nich została określona wartość właściwości `color`. Ponieważ selektory identyfikatorów są najbardziej szczegółowe, zatem tekst w akapicie zostanie wyświetlony na zielono, gdyż selektor identyfikatora przesłania dwa pozostałe. Jeśli jednak pomiędzy stylami nie występują żadne konflikty — na przykład właściwość `text-align` została określona wyłącznie w regule z selektorem `.person` — to taka właściwość zostanie zastosowana niezależnie od tego, z której reguły pochodzi.

W tym przykładzie tekst w akapicie będzie mieć wielkość 24 pikseli i zielony kolor (pochodzący z reguły z selektorem `#susan`), oprócz tego będzie pogrubiony i wyśrodkowany (z reguły z selektorem `.person`), a dodatkowo zostanie wyświetlony czcionką Arial (z reguły z selektorem `p`).

Ponieważ style biblioteki jQuery UI są określane w oparciu o klasy, znajomość zasady szczegółowości i jej działania w przypadku selektorów innych typów ma duże znaczenie.

- **Wygrywają reguły zdefiniowane później.** Jeśli ta sama reguła pojawi się w dwóch różnych arkuszach stylów (bądź nawet w dwóch miejscach tego samego arkusza), wygra ostatnia z nich. To właśnie dlatego arkusz stylów biblioteki jQuery UI należy dołączać jako pierwszy. Jeśli w takim przypadku do drugiego arkusza dodasz nowy styl używający takiej samej nazwy, co klasa jQuery UI, ta domyślna klasa zostanie nadpisana. Przykładowo w arkuszu stylów jQuery UI można znaleźć następujący styl:

```
/* in jquery-ui stylesheet */
 .ui-widget-content a {
   color: #222222;
}
```

Jeśli dołączysz arkusz stylów swojej witryny *za* arkuszem stylów jQuery UI, będziesz mógł do niego dodać style charakterystyczne dla swojej witryny i całkowicie nadpisać style jQuery UI dla wybranego elementu:

```
/* in jquery-ui stylesheet */
 .ui-widget-content a {
   color: #FFO;
}
```

- **Szczegółowość się sumuje.** Stosowanie selektorów elementów potomnych jest doskonałym sposobem tworzenia bardzo szczegółowych stylów. Przeglądarka dodaje wszystkie fragmenty takiego selektora, wyznaczając w ten sposób sumaryczną wartość jego szczegółowości. Prosty sposób określenia tej wartości polega na przyjęciu, że selektor identyfikatora ma wartość 100 punktów, selektor klasy — 10 punktów, a selektor elementu — 1 punkt. Przykładowo styl z selektorem `#main a` przesłoni styl z selektorem `.ui-state-default a`, gdyż ma wartość `101` punktów (jeden selektor identyfikatora i jeden elementu), natomiast ten drugi tylko `11` (jeden selektor klasy i jeden elementu). Innymi słowy, możesz także przesłaniać style jQuery UI, podając selektory elementów potomnych, których szczegółowość ma wyższą wartość punktową.

- **W trudnych okolicznościach można zastosować deklarację `!important`.** Kaskadowe arkusze stylów udostępniają rozwiązanie, które można by określić jako „opcję nuklearną". Jest nim deklaracja `!important`. Jeśli dodasz `!important` za deklaracją stylu, jego wartość przesłoni wszystkie inne style, niezależnie od tego, jak bardzo będą szczegółowe. Załóżmy na przykład, że arkusz podany w pierwszym punkcie tej listy zawiera styl określający wygląd znaczników `<p>` o następującej postaci:

  ```
  p {
    color: orange !important;
    font-weight: normal !important;
    font-family: Arial, sans-serif !important;
  }
  ```

  Ponieważ każda z właściwości podanych w tym stylu zawiera deklarację `!important`, zatem przesłonią one wartości w bardziej szczegółowych stylach `#susan` oraz `.person`. Jednak deklarację `!important` naprawdę należy stosować jako ostatnią deskę ratunku. Gdybyś zaczął korzystać z niej naprawdę często, szybko doprowadziłoby to do powstania wielu stylów z tymi deklaracjami, a to z kolei znacząco utrudniłoby określanie szczegółowości konkretnych reguł stylów. (Gdyby każda właściwość w każdym stylu zawierała deklarację `!important`, byłyby stosowane zwyczajne reguły szczegółowości, czyli selektor identyfikatora miałby wyższy priorytet od selektora klasy, a ten miałby wyższy priorytet od selektora elementu).

Jak są określane style widżetów jQuery UI?

Biblioteka jQuery UI wykorzystuje modularne podejście do określania postaci swoich widżetów: zamiast definiować jeden styl `ui-dialog`, określający wszystkie aspekty wyglądu okna dialogowego, jego ostateczny wygląd jest ustalany przy użyciu wielu różnych stylów. Przykładowo poniżej przedstawiony został styl `.ui-widget`, stosowany we wszystkich widżetach:

```
.ui-widget {
  font-family: Verdana,Arial,sans-serif;
  font-size: 1.1em;
}
```

Ponieważ ta klasa jest wykorzystywana we wszystkich widżetach, zatem wszystkie używają tych samych czcionek, tej samej wielkości. Jednak postać okien dialogowych określa także wiele innych stylów. W samych tylko oknach dialogowych, w ich zewnętrznym kontenerze stosowane są następujące style:

```
class="ui-dialog ui-widget ui-widget-content
ui-corner-all ui-front ui-draggable ui-resizable"
```

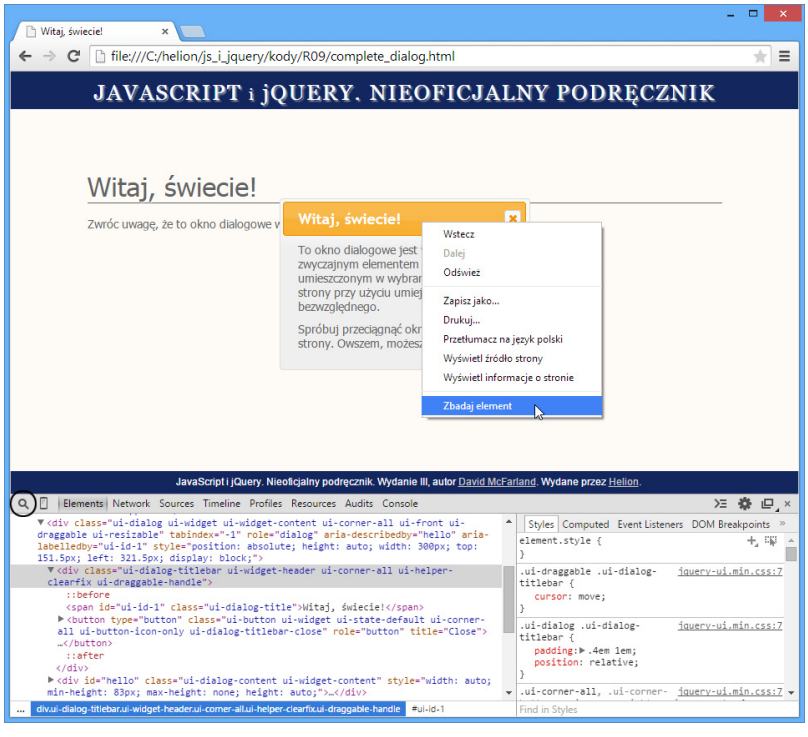
Każda z tych klas określa jakieś drobne aspekty ostatecznego wyglądu okna dialogowego. Poprzez takie połączenie wielu klas jQuery UI może ich wielokrotnie używać do określania postaci różnych widżetów. Przykładowo poniżej przedstawione zostały klasy używane w zewnętrznym kontenerze menu jQuery UI (patrz strona 368):

```
class="ui-menu ui-widget ui-widget-content"
```

Zauważ, że dwie spośród tych klas — `ui-widget` oraz `ui-widget-content` — były także zastosowane w widżecie okna dialogowego. Klasy te zawierają aspekty wyglądu używane w obu tych widżetach.

Mogłeś już się przekonać, że zwyczajny element `<div>` całkiem łatwo można zamienić w złożone okno dialogowe (patrz strona 330), a listę zagnieżdżoną — w rozwijane menu (patrz strona 368). Biblioteka jQuery UI za kulisami modyfikuje nasz prosty kod HTML, dodając do niego nowe warstwy elementów `div`, `span` oraz różnych innych obiektów HTML i uzyskując w ten sposób ostateczny wygląd widżetu. W efekcie, patrząc na tworzony przez nas kod i na style jQuery UI, nie jesteśmy w stanie określić, w jaki sposób uzyskiwany jest ten ostateczny efekt. Aby dowiedzieć się, jak są formatowane widżety biblioteki jQuery UI, trzeba zajrzeć do *wygenerowanego* kodu HTML (czyli faktycznego kodu HTML utworzonego z pomocą magicznych sztuczek biblioteki).

Najlepszym sposobem przeanalizowania sposobu, w jaki style CSS określają wygląd widżetów jQuery UI, jest skorzystanie z narzędzi inspekcji kodu, wbudowanych w przeglądarkę. Większość przeglądarek udostępnia narzędzia pozwalające na sprawdzanie wygenerowanego kodu HTML strony. W przeglądarce Chrome wystarczy kliknąć widoczny na stronie widżet i z menu podręcznego wybrać opcję *Inspect Element*. U dołu okna przeglądarki zostanie wyświetlone dodatkowe okno, a w nim na karcie *Elements* kod HTML (patrz rysunek 11.5).

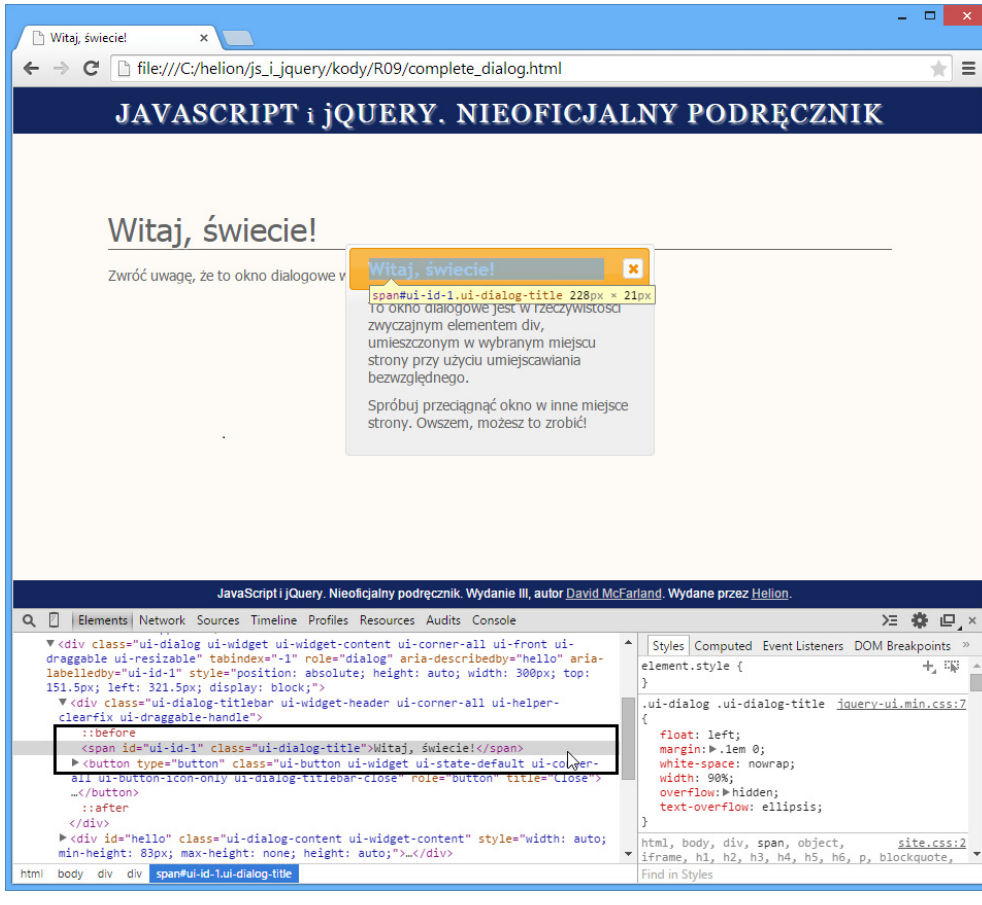


Rysunek 11.5. *Kiedy w oknie konsoli zostanie już wyświetlona karta Elements, będzie można wskazać myszą interesujący nas fragment kodu HTML. Elementy widoczne w oknie przeglądarki i odpowiadające temu fragmentowi zostaną wyróżnione, tak jakby zostały wskazane myszą. Bez najmniejszego problemu można dopasować kod HTML do jego wizualnej reprezentacji w przeglądarce*

Kliknięcie ikony szkła powiększającego (zakreślonej na rysunku 11.5) aktywuje narzędzie inspekcji kodu. Po wybraniu tej ikony można kliknąć dowolny element na stronie, a na karcie *Elements* zostanie wyświetlony odpowiadający mu kod HTML. To doskonały sposób, by odszukać na przykład nagłówek okna dialogowego i przekonać się, jakie style zostały w nim użyte.

Kiedy już uda Ci się zidentyfikować style, będziesz mógł we własnym arkuszu utworzyć style o takich samych nazwach. Przykładowo używając narzędzi przeglądarki Chrome, możesz się przekonać, że w kodzie HTML tworzącym tytuł okna dialogowego zastosowano klasę o nazwie `.ui-dialog-title` (patrz rysunek 11.6). Aby zmienić kolor nagłówków okien dialogowych na czerwony, możesz dodać do swojego arkusza stylów następującą regułę:

```
.ui-dialog-title {
  color: red;
}
```



Rysunek 11.6. *Korzystając z narzędzia inspektora (szkła powiększającego), można błyskawicznie zlokalizować wygenerowany kod HTML odpowiadający elementom widocznym w oknie przeglądarki. Na tym rysunku pokazany został kod HTML wygenerowany przez jQuery UI w celu utworzenia paska tytułu okna dialogowego*

12

ROZDZIAŁ

Interakcje i efekty jQuery UI

Biblioteka jQuery UI ma do zaoferowania znacznie więcej, nie tylko widżety interfejsu użytkownika. Udostępnia także kilka użytecznych możliwości służących do wzbogacania interaktywności tworzonych stron WWW oraz aplikacji internetowych. Te *interakcje* (ang. *interactions*) jQuery UI pozwalają na przesuwanie elementów stron. Niektóre widżety można przeciągać i upuszczać, co oznacza, że użytkownik może przeciągnąć jeden element i umieścić go w obszarze innego elementu — w ramach przykładu można sobie wyobrazić przeciągnięcie pliku do kosza lub produktu do koszyka z zakupami. jQuery UI pozwala także na tworzenie list, w których kolejność elementów można zmieniać przy użyciu ich przeciągania — z tego rozwiązania można skorzystać przykładowo podczas tworzenia listy zadań do zrobienia lub listy odtwarzanych utworów muzycznych.

Dodatkowo jQuery UI udostępnia zestaw efektów, które można zastosować, by dodać do witryny atrakcyjne wizualnie animacje. Można ich użyć i przyciągnąć uwagę użytkownika poprzez wyróżnienie koloru tła jakiegoś elementu. Można zastosować animację odbicia podczas przeciągania elementu, czy też sprawić, by wybrany element zatrząsł się lub zmienił wielkość. Więcej informacji o tych efektach można znaleźć na stronie 461.

Widżet Draggable

Dla osób korzystających z komputerów przeciąganie jest czymś naturalnym — wszyscy przeciągają pliki z jednego katalogu do drugiego, usuwają je, przeciągając ich ikony do kosza, przeciągają okna, by zrobić więcej miejsca na pulpicie i tak dalej. Te interakcje są czymś całkowicie naturalnym dla generacji osób, która wychowała się na komputerach osobistych i myszach. Biblioteka jQuery UI za pomocą swoich dwóch widżetów — Draggable oraz Droppable (patrz strona 434) — przenosi te same interakcje do świata stron WWW.

Widżet Draggable zapewnia możliwość przesuwania wybranego elementu strony. Działanie tego widżetu poznałeś już na przykładzie okna dialogowego jQuery UI (patrz strona 330): okna dialogowe są wyświetlane dokładnie pośrodku ekranu, lecz można je przeciągnąć i umieścić w dowolnym innym miejscu. Oba widżety — Draggable oraz Droppable — można zastosować jednocześnie, by utworzyć interaktywne koszyki z zakupami. Kupujący mogą przeciągać obrazki produktów do koszyka umieszczonego na stronie, produkty te zostaną do niego dodane.

Dodawanie widżetu Draggable do strony

Każdy element strony można zmodyfikować w taki sposób, by można go było przeciągać. Oczywiście trzeba pamiętać o tym, że możliwość przeciągania wybranego elementu powinna mieć sens. Przykładowo zapewnienie możliwości przeciągania akapitów tekstu nie poprawi czytelności strony i nie będzie żadną zaletą dla czytelników. Z drugiej strony, zapewnienie możliwości przeciągania wyskakującego okna dialogowego jest bardzo sensowne, a w przypadku tworzenia internetowej wersji gry w warcaby jest wręcz niezbędne.

Podstawowy sposób wykorzystania widżetu Draggable jest bardzo prosty.

1. **Wykonaj czynności opisane na stronie 329, by dołączyć do strony niezbędne pliki CSS i JavaScript.**

   Pamiętaj, że biblioteka jQuery UI ma własne pliki CSS oraz JavaScript; należy je dołączyć do strony *za* plikiem JavaScript biblioteki jQuery.

2. **Umieść na stronie bądź w kolejnym zewnętrznym pliku JavaScript wywołanie funkcji `$(document).ready()`:**

   ```
   $(document).ready(function() {

   }); // Koniec funkcji ready.
   ```

   Zgodnie z informacjami podanymi na stronie 190, krok ten jest niezbędny wyłącznie wtedy, gdy kod JavaScript jest umieszczany w sekcji `<head>` strony, przed właściwym kodem HTML stanowiącym jej treść. Niektórzy programiści umieszczają swój kod JavaScript na końcu strony, bezpośrednio przed zamykającym znacznikiem `</body>`; w takim przypadku umieszczanie kodu w wywołaniu funkcji `$(document).ready()` nie jest konieczne.

3. **Użyj jQuery, żeby wybrać element strony i zastosować w nim widżet Draggable:**

   ```
   $(document).ready(function() {
     $('#dialog').draggable();
   }); // Koniec funkcji ready.
   ```

   W tym przykładzie wybraliśmy jeden element, używając selektora identyfikatora. Gdybyśmy jednak chcieli zapewnić możliwość przeciągania większej liczby elementów strony, moglibyśmy dodać do ich znaczników określoną klasę, na przykład `<div class= draggable >`, a następnie podać ją w selektorze, by za jednym zamachem użyć widżetu Draggable we wszystkich tych znacznikach:

   ```
   $('.draggable').draggable();
   ```

4. **Zapisz plik i wyświetl go w przeglądarce.**

   I to już wszystko. Jednak można sądzić, że będziemy chcieli dodatkowo skonfigurować możliwości przeciągania. Na szczęście jQuery UI udostępnia wiele opcji pozwalających na dostosowanie działania tego widżetu. Informacje na ich temat zostały podane na stronie 424.

Miniprzykład — zastosowanie widżetu Draggable

Nadszedł czas, by przetestować widżet Draggable i przekonać się, jak łatwo można zapewnić możliwość przeciągania elementu strony.

Uwaga: Więcej informacji o tym, jak pobrać przykłady prezentowane w książce, można znaleźć na stronie 46.

1. **W edytorze tekstów otwórz plik *draggable.html* umieszczony w katalogu *R12*.**

   Strona ta zawiera już odwołania do wymaganych plików bibliotek jQuery i jQuery UI, jak również wywołanie funkcji `$(document).ready()` (patrz strona 190).

   Przejrzyj kod strony i odszukaj w nim znacznik `<div>` o identyfikatorze `note`, zawierający nagłówek i krótki akapit tekstu. Zaraz przekształcisz ją w element, który można przeciągać.

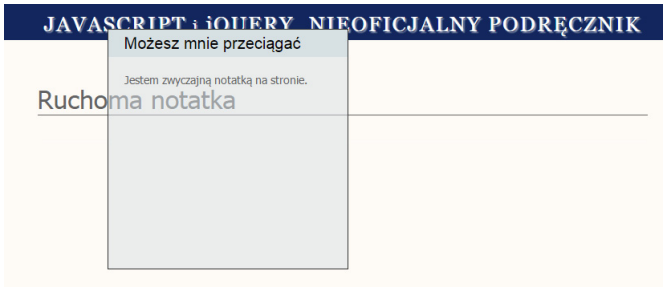
2. **W pustym wierszu umieszczonym wewnątrz wywołania funkcji `$(document).ready()` wpisz wiersz kodu wyróżniony poniżej pogrubioną czcionką:**

```
$(document).ready(function() {
  $('#note').draggable();
}); // Koniec funkcji ready.
```

   To wywołanie wybiera znacznik `<div>` i zapewnia możliwość jego przeciągania na stronie.

3. **Zapisz stronę i wyświetl ją w przeglądarce (patrz rysunek 12.1).**

   Kliknij gdziekolwiek w obszarze nagłówka wewnątrz elementu `div` „Możesz mnie przeciągać" i przeciągnij go w inne miejsce strony. Działanie widżetu Draggable można konfigurować na wiele różnych sposobów, o czym przekonasz się w następnym punkcie rozdziału.



Rysunek 12.1. *Przy użyciu tylko jednego wiersza kodu można zmienić statyczny znacznik w interaktywny komponent, który można przeciągać. Widżet Draggable jest także używany przez okna dialogowe (widżet Dialog) jQuery UI (patrz strona 330)*

Uwaga: Kompletna wersja tego przykładu jest dostępna w pliku *complete_draggable.html*.

Opcje widżetu Draggable

Widżet jQuery UI Draggable ma bardzo duże możliwości i udostępnia wiele opcji. Można określać dozwolony kierunek przeciągania elementu oraz odległość, sposób zachowania elementu podczas przeciągania, a nawet fragment elementu służący do przeciągania. Podobnie jak wszystkie inne widżety jQuery UI, także i ten jest konfigurowany za pomocą przekazania odpowiedniego obiektu w wywołaniu funkcji, w tym przypadku jest to funkcja `draggable()`. Aby na przykład zmienić kształt wskaźnika myszy na dłoń z wyciągniętym palcem wskazującym i wskazać, że „uchwytem" (ang. *handle*) służącym do przeciągania elementu będzie umieszczony w nim znacznik `<h2>`, należy przekazać w wywołaniu funkcji `draggable()` przedstawiony poniżej literał obiektowy z dwoma właściwościami:

```
$('#controls').draggable({
  cursor : 'pointer',
  handle : 'h2'
});
```

Poniżej przedstawione zostały najczęściej używane opcje.

- **`axis`**. Właściwość pozwala ograniczyć możliwości ruchu elementu do przeciągania w prawo i w lewo bądź w górę i w dół. Przykładowo załóżmy, że chcemy przeciągać jakiś element na osi czasu. Element ten musi być cały czas wyświetlony przy linii, zatem nie będziemy chcieli, by ktoś mógł przeciągnąć go na dół strony. Właściwości tej można przypisać wartość `'x'` (by ograniczyć możliwości przeciągania elementu do ruchu w poziomie) lub wartość `'y'` (by można go było przeciągać tylko w pionie).

  ```
  axis : 'x'
  ```

- **`cancel`**. Za pomocą tej opcji można uniemożliwić przeciąganie, kiedy zostanie kliknięty podany element. Załóżmy, że utworzyliśmy ramkę na komunikaty, którą użytkownik może przeciągać w oknie przeglądarki. Komunikat ma nagłówek, lecz także kilka akapitów tekstu zawierających ważne informacje, takie jak adres lub numer telefonu. Domyślnie, gdyby użytkownik chciał zaznaczyć taki adres (na przykład po to, by skopiować go i wkleić w Google Maps), okazałoby się, że przeciąga całą ramkę z komunikatami. Aby wyłączyć możliwość przeciągania po kliknięciu akapitu, należy użyć następującej opcji:

  ```
  cancel : 'p'
  ```

  Jeśli teraz użytkownik kliknie akapit w ogłoszeniu, nie spowoduje to rozpoczęcia przeciągania ramki i będzie można zaznaczyć interesujący fragment tekstu. Podobne działanie widżetu można zapewnić, używając opcji `handle` opisanej na stronie 427.

- **`connectToSortable`**. Opcja pozwala wskazać kolekcję „sortowalnych" elementów, do której będzie można dodać element. Biblioteka jQuery UI udostępnia widżet Sortable, który zapewnia możliwość określania kolejności grupy elementów; z powodzeniem można go zastosować do utworzenia listy zadań do zrobienia (więcej informacji na temat widżetów Sortable można znaleźć na stronie 449). Aby skorzystać z tej opcji, należy podać identyfikator elementu stanowiącego kolekcję „sortowalnych" elementów (patrz strona 449.):

  ```
  connectToSortable : '#toDoList'
  ```

- **`containment`**. Korzystając z tej opcji, można uniemożliwić przeciągnięcie elementu poza obszar kontenera, w którym jest umieszczony. Na przykład załóżmy, że tworzymy stronę, której fragment będzie działał jak lodówka, na której można przyczepiać magnesiki ze słowami, aby utworzyć z nich jakieś zdanie. W tym celu możemy opracować grupę elementów `div` ze słowami i umieścić je w wybranym obszarze strony. Nie chcemy jednak, by użytkownik mógł przeciągać magnesiki po całej stronie, na przykład po części nawigacyjnej, nagłówku czy też stopce. Opcja `containment` pozwala ograniczyć możliwości przeciągania elementu do mniejszego obszaru strony. Może ona przyjmować kilka różnych wartości. Oto one.
  - **Selektor.** Jeśli zostanie podana nazwa selektora, jQuery UI ograniczy możliwości przeciągania wyłącznie do wnętrza elementu wskazanego przez ten selektor. Gdyby na stronie znajdował się znacznik `<div>` o identyfikatorze `refrigerator`, to aby ograniczyć możliwość przeciągania wyłącznie do jego obszaru, należy użyć opcji `containment` w następujący sposób:

    ```
    containment : '#refrigerator'
    ```

  - **`parent`, `document` lub `window`.** Aby ograniczyć możliwości przeciągania wyłącznie do obszaru elementu rodzica, należy użyć wartości `parent`. Jeśli chcemy przeciągać znacznik `<div>` umieszczony wewnątrz innego znacznika `<div>` i jednocześnie chcemy ograniczyć obszar, w którym może znaleźć się element przeciągany do zewnętrznego znacznika `<div>`, powinniśmy przypisać opcji `containment` wartość `parent` w następujący sposób:

    ```
    containment : 'parent'
    ```

    Wartości `document` oraz `window` działają podobnie, przy czym pierwsza z nich sprawia, że przeciągany element będzie mógł znaleźć się gdziekolwiek w obszarze dokumentu, a druga, `window`, że będzie on mógł częściowo wychodzić poza obszar okna przeglądarki.

  - **Tablica współrzędnych.** Można także podać tablicę współrzędnych, reprezentujących odpowiednio górny lewy oraz prawy dolny wierzchołek obszaru, w którym będzie można przeciągać element. Współrzędne te są podawane w pikselach, względem lewego, górnego wierzchołka okna przeglądarki. Załóżmy, że chcemy ograniczyć obszar, w którym będzie można przeciągać element, do prostokąta rozpoczynającego się 50 pikseli na prawo od lewej krawędzi okna i 100 pikseli poniżej górnej krawędzi okna, a kończącego 500 pikseli od lewej krawędzi i 600 pikseli poniżej górnej krawędzi. Można to zrobić, używając następującej opcji `containment`:

    ```
    containment : [50, 100, 500, 600]
    ```

    Ponieważ ten sposób określania obszaru, w którym będzie można przeciągać element, wymaga podania precyzyjnych współrzędnych, zatem nie nadaje się on do użycia w przypadku stosowania elastycznych projektów stron. W takich sytuacjach lepiej skorzystać z elementu zawierającego i podać jego selektor.
- **`cursor`**. Istnieje możliwość zmiany postaci wskaźnika myszy podczas przeciągania. Normalnie wskaźnik myszy ma postać strzałki (jeśli klikniemy w pustym miejscu elementu) bądź też charakterystycznej kreski do zaznaczania

tekstu (jeśli klikniemy na przykład tekst nagłówka). Jednak można za pomocą jQuery UI zmienić postać wskaźnika myszy podczas przeciągania — wystarczy podać odpowiednią wartość w opcji `cursor`. Listę wszystkich dostępnych wartości można znaleźć na stronie *https://developer.mozilla.org/en-US/docs/Web/CSS/cursor*, jednak najczęściej używane są wartości `pointer`, `crosshair` oraz `default` (czyli strzałka):

```
cursor : 'pointer'
```

- **`cursorAt`**. Można także określać miejsce, w którym, będzie umieszczony wskaźnik myszy podczas przeciągania. Zazwyczaj jest on umieszczony w miejscu, w którym znajdował się, gdy użytkownik wcisnął przycisk myszy. Jeśli na przykład wskaźnik myszy znajdował się u dołu elementu `div`, kiedy zaczęliśmy go przeciągać, pozostanie u dołu tego elementu, aż do momentu zwolnienia przycisku myszy. Może się jednak zdarzyć, że będziemy chcieli, by wskaźnik myszy był widoczny w konkretnym miejscu — na przykład w obszarze wizualnego uchwytu używanego do przeciągania elementu. Aby to zrobić, wystarczy określić ten obszar przy użyciu opcji `cursorAt`. Jej właściwością powinien być obiekt zawierający wybrane właściwości `left`, `right`, `top` oraz `bottom`. Aby na przykład umieścić wskaźnik myszy w lewym, górnym wierzchołku przeciąganego elementu, należy użyć właściwości `left` oraz `top`:

  ```
  cursorAt : {
    left : 5,
    top : 5
  }
  ```

  Nie można wykorzystać jednocześnie właściwości `left` i `right`, gdyż wyznaczają one dwa miejsca na tej samej osi; z tego samego powodu nie można jednocześnie użyć właściwości `top` i `bottom`. Jednak można skorzystać z dowolnej właściwości określającej położenie na danej osi (`left` lub `right` oraz `top` lub `bottom`). Można także użyć tylko jednej właściwości. Jeśli będzie nas interesować tylko to, by wskaźnik znajdował się wewnątrz paska u góry przeciąganego elementu, wystarczy określić wartość właściwości `top`. W takim przypadku położenie wskaźnika w poziomie będzie takie samo, jakie było w momencie rozpoczynania przeciągania:

  ```
  cursorAt : { top : 5 }
  ```

- **`disabled`**. Właściwość określa, czy będzie można przeciągać element. Jeśli tej właściwości zostanie przypisana wartość `true`, przeciąganie nie będzie możliwe. Można uznać, że gdyby ktoś *nie chciał* przeciągać elementu, najprościej byłoby w ogóle nie wywoływać funkcji `draggable()`. To prawda, zapewne mało kto używa tej opcji bezpośrednio podczas tworzenia widżetu Draggable.

  Jednak może się ona przydać *potem*, kiedy element został już przeciągnięty. Przykładowo możemy chcieć, by użytkownik mógł przeciągnąć produkt do koszyka zakupów, lecz nie mógł go z niego wyciągnąć. W takim przypadku wystarczyłoby zastosować tę opcję po zakończeniu operacji przeciągania (patrz strona 440).

- **`grid`**. Normalnie element podczas przeciągania może się dowolnie przesuwać w dopuszczalnym obszarze. Jednak może się zdarzyć, że będziemy chcieli wymusić, by był on przesuwany o określoną liczbę pikseli w poziomie i pionie.

Wyobraźmy sobie, że tworzymy internetową wersję gry w warcaby. Siatka pól ma wymiary 8 na 8 pikseli. Pionek może być przeciągany przez krawędzie pól, lecz plansza byłaby niechlujna, gdyby można go było umieszczać zupełnie dowolnie. Dlatego też gra będzie wyglądała lepiej, jeśli pionki będzie można przesuwać o odcinki odpowiadające wymiarom pól planszy.

jQuery UI zapewnia możliwość określenia przyrostów, poziomego i pionowego, o jakie będzie przesuwany element. W opcji `grid` należy podać tablicę zawierającą dwie liczby. Pierwszą z nich jest przyrost w poziomie, a drugą — w pionie. Załóżmy, że chcemy, by przeciągany element był przesuwany w poziomie na odległość 50 pikseli. Oprócz tego chcemy, by podczas przeciągania w pionie był przesuwamy o 100 pikseli. Moglibyśmy to zrobić, używając opcji `grid`:

```
grid : [ 50, 100 ]
```

- **`handle`**. Opcja pozwala ograniczyć obszar, który użytkownik musi kliknąć, by rozpocząć przeciąganie elementu. Normalnie element można przeciągać, klikając go w dowolnym miejscu. Jednak może się zdarzyć, że będziemy chcieli ograniczyć ten obszar wyłącznie do nagłówka lub jakiegoś wizualnego fragmentu elementu. W takim przypadku w opcji `handle` można podać selektor określający element wewnątrz przeciąganego elementu, który posłuży jako „uchwyt" do przeciągania. Przykładowo poniższa opcja określa, że element można przeciągać wyłącznie za pomocą umieszczonego w nim nagłówka `<h2>`:

  ```
  handle : 'h2'
  ```

- **`helper`**. Może się zdarzyć, że nie będziemy chcieli, by podczas przeciągania przesuwany był sam wybrany element. Załóżmy, że tworzymy stronę koszyka z zakupami. Na stronie są wyświetlane produkty, a użytkownik może je dodawać do koszyka, przeciągając zdjęcie produktu i upuszczając je na ikonie koszyka. Jednak nie chcemy przy tym, by użytkownik przeciągał faktyczne zdjęcie produktu, gdyż spowodowałoby to jego zniknięcie ze strony. Lepszym rozwiązaniem byłoby przeciąganie kopii tego zdjęcia i pozostawienie jego oryginału w początkowym miejscu. Taki efekt można łatwo uzyskać, przypisując opcji `helper` wartość `'clone'`:

  ```
  helper : 'colone'
  ```

  Nieco bardziej złożonym rozwiązaniem jest przekazanie w tej opcji funkcji generującej kod HTML, który następnie jQuery UI wyświetli podczas przeciągania elementu. Aby na przykład wyświetlić pomocniczy znacznik `<div>` o ogólnym charakterze (a nie przeciągać elementu umieszczonego na stronie), można użyć opcji `helper` o następującej postaci:

  ```
  helper : function( event ) {
    return $("<div class='ui-widget-header'>Jestem elementem
  pomocniczym</div>" );
  }
  ```

  Cokolwiek zwróci ta funkcja, przekazana jako wartość opcji `helper`, zostanie wyświetlone przez jQuery UI i użyte podczas przeciągania wskaźnika myszy. Trzeba jednak pamiętać, że musi to być obiekt jQuery — czyli konieczne jest wywołanie funkcji `$()`. Zwrócenie zwykłego łańcucha znaków z kodem HTML nie da zamierzonego efektu.

- **opacity**. Podczas przeciągania elementu można zmienić jego nieprzezroczystość. Można ją zmniejszyć do 50%, sprawiając wrażenie, jakby po stronie był przesuwany jego półprzezroczysty duch. Technika jest często stosowana w celu zasygnalizowania, że element zostanie przeniesiony z jednego miejsca w drugie. Tej opcji należy przypisywać wartości z zakresu od `0` (przeciągany element będzie całkowicie niewidoczny) do `1` (element będzie całkowicie nieprzezroczysty). Opcja `opacity` działa dokładnie tak samo jak właściwość CSS o tej samej nazwie. Aby na przykład przeciągany element był w 50% przezroczysty, należy przypisać jej wartość `0.5`:

  ```
  opacity : 0.5
  ```

- **revert**. Opcja określa, czy po zakończeniu operacji przeciągany element ma wrócić w miejsce, w którym się początkowo znajdował. Domyślną wartością tej opcji jest `false`, co oznacza, że element pozostanie w miejscu, w którym został upuszczony. W przypadku przeciągania okien i okienek dialogowych jest to właściwe rozwiązanie — przeciągamy je zazwyczaj po to, by odsłonić jakieś inne elementy strony.

  Jednak zdarzają się także przypadki, kiedy przeciągany element powinien wrócić do swojej początkowej lokalizacji. Jeśli na przykład używamy widżetu Droppable (opisanego na stronie 434) i przeciągany element nie zostanie upuszczony w odpowiednim miejscu, będziemy chcieli, by wrócił w miejsce, w którym się początkowo znajdował. Załóżmy, że tworzymy aplikację do organizowania zdjęć. Jej użytkownik może przeciągać miniaturki zdjęć do różnych katalogów widocznych na stronie bądź też upuszczać je na ikonie kosza na śmieci. Może się jednak zdarzyć, że użytkownik przeciągnie miniaturkę na nagłówek strony, na jej stopkę lub w inne nieodpowiednie miejsce. W takim przypadku będziemy chcieli, by miniaturka wróciła w miejsce, w którym się wcześniej znajdowała, i nie przesłaniała nagłówka bądź stopki strony.

  W opcji `revert` można podawać kilka różnych wartości. Wartość `true` powoduje, że przeciągany element zawsze wróci w początkowe położenie (witamy kolejną stronę, która żartuje sobie z użytkowników); wartość `'invalid'` sprawia, że element wraca w początkowe położenie wyłącznie w przypadku, gdy nie zostanie upuszczony w obszarze prawidłowego widżetu Droppable (patrz strona 434). W hipotetycznej aplikacji do organizowania zdjęć można by użyć wartości `invalid`:

  ```
  revert : 'invalid'
  ```

  I w końcu, wartością opcji `revert` może być także funkcja. Jeśli ta funkcja zwróci wartość `true`, element wróci na swoje początkowe położenie. Tego rozwiązania można użyć podczas tworzenia internetowej wersji gry w warcaby: kiedy użytkownik zrobi ruch i upuści pionek na innym polu planszy, funkcja mogłaby sprawdzić, czy ruch jest prawidłowy (czy pionek nie został upuszczony na zajęte pole lub został przesunięty zbyt daleko). Gdyby się okazało, że ruch nie jest prawidłowy, funkcja zwróciłaby wartość `true`, a pionek powróciłby w swoje początkowe położenie.

  ```
  revert : function() {
   // Ta funkcja powinna wykonywać jakiś test i na jego podstawie zwracać
   // wartość true, aby element wrócił na swoje początkowe położenie, bądź
   // wartość false, by pozostał w miejscu, w którym został upuszczony.
  }
  ```

- **revertDuration**. Kiedy jQuery UI przenosi przeciągany element w miejsce, w którym się początkowo znajdował, jego ruch jest animowany. Domyślny czas trwania tej animacji wynosi 500 milisekund, czyli pół sekundy. Wygląda to świetnie, lecz nic nie stoi na przeszkodzie, by animacja trwała krócej (liczba milisekund była mniejsza od 500) lub dłużej (liczba milisekund była większa od 500). Efekt ten można uzyskać, korzystając z opcji `revertDuration`, której wartością jest liczba określająca czas trwania animacji wyrażony w milisekundach. Aby animacja powrotu trwała ćwierć sekundy (czyli 250 milisekund), należy użyć następującej opcji:

  ```
  revertDuration : 250
  ```

- **scope**. Opcja `scope` pozwala grupować elementy, w których zastosowano widżety Draggable i Droppable. Przykładowo załóżmy, że nasza strona prezentuje kalendarz. Użytkownik może przeciągać zdarzenia z obszaru reprezentującego jeden dzień i upuszczać na innym dniu. Wszystkie takie zdarzenia należą do kalendarza, zatem podczas ich tworzenia w wywołaniu funkcji `draggable()` można przypisać opcji `scope` wartość `calendar`:

  ```
  scope : 'calendar'
  ```

  W tym przypadku łańcuch znaków `'calendar'` nie jest ani selektorem, ani żadnym elementem strony. Jest jedynie nazwą używaną do grupowania elementów, które mogą być przeciągane (widżetów Draggable) oraz elementów wyznaczających miejsca, gdzie można je upuszczać (widżetów Droppable). Ta nazwa może być całkowicie dowolna; trzeba tylko pamiętać, by tej samej nazwy użyć w tworzonych widżetach Droppable (patrz strona 434).

  Opcja `scope` jest potrzebna wyłącznie wtedy, gdy na stronie mogą być używane różne typy elementów przeciąganych i elementów, w których można je upuszczać. Gdyby na stronie z kalendarzem znajdowała się także internetowa układanka, zapewne nie chcielibyśmy, by użytkownik mógł przeciągać zdarzenia z kalendarza na układankę ani elementy układanki na kalendarz. Dzięki zastosowaniu innej wartości opcji `scope` w każdej z tych grup elementów można kontrolować, gdzie będzie można upuszczać konkretne przeciągane elementy (więcej informacji na temat widżetu Droppable podano na stronie 434).

Uwaga: Opcje te możesz wypróbować na stronie testowej — *draggable.html* — którą utworzyłeś na stronie 423. Jeśli nie wykonałeś tego przykładu, otwórz teraz plik *complete_draggable.html* i dodaj kilka z tych opcji do literału obiektowego przekazywanego w wywołaniu funkcji `draggable()`.

- **snap**. Można nakazać, by przeciągany element był przyciągany do innego elementu strony lub innego przeciąganego elementu. Przykładowo załóżmy, że tworzymy internetową układankę: kilkanaście prostokątnych fragmentów należy poukładać tak, by odtworzyć początkowe zdjęcie. W takim przypadku zastosowanie opcji `snap` pozwoliłoby uzyskać efekt, w którym jeden element układanki byłby przyciągany do innego. Opcja ta może przyjmować dwie wartości.

  - **true**. Wartość sprawia, że dany element będzie przyciągany do każdego innego elementu strony, w którym zastosowano widżet Draggable. Zastosowanie tej opcji będzie preferowane w przypadku internetowej układanki, gdyż na takiej stronie znajduje się wiele elementów, które można przeciągać.

- **Nazwa selektora**. Może to być nazwa dowolnego selektora. Aby przeciągany element był przyciągany do elementu `div` o identyfikatorze `photoholder`, należałoby użyć poniższej opcji `snap`:

```
snap : '#photoholder'
```

Sposób działania tego mechanizmu przyciągania jest określany przez dwie inne opcje, `snapMode` oraz `snapTolerance`, które zostały opisane poniżej.

- **`snapMode`**. Opcja działa wyłącznie wtedy, gdy została zastosowana także opcja `snap`. Można jej przypisać jedną z trzech dopuszczalnych wartości: `inner`, `outer` oraz `both`. Jeśli chcemy, by przeciągany element był przyciągany do innego elementu tylko wtedy, gdy będzie się znajdował wewnątrz niego, należy użyć wartości `inner`. W tym przypadku przeciągany element będzie przyciągany do dowolnej krawędzi elementu określonego przez opcję `snap`.

  Jeśli chcemy, by przeciągany element był przyciągany do zewnętrznych krawędzi innego elementu strony, należy zastosować wartość `outer`. Ta wartość doskonale nadaje się do zastosowania w internetowej układance, gdyż będziemy chcieli, by jeden jej kawałek był przyciągany do innych.

  I w końcu, użycie ostatniej z dostępnych wartości, czyli `both`, sprawia, że przeciągany element będzie przyciągany zarówno do wewnętrznych, *jak i* do zewnętrznych krawędzi elementu wybranego w opcji `snap`:

```
snapMode : 'both'
```

- **`snapTolerance`**. Opcja określa, jak blisko musi się znaleźć przeciągany element, by został przyciągnięty do krawędzi. Im większa wartość, tym większa ta odległość. Wartość tej właściwości określa odległość wyrażoną w pikselach:

```
snapTolerance : 30
```

- **`zIndex`**. Opcja pozwala określić wartość `z-index` przeciąganego elementu. W kaskadowych arkuszach stylów właściwość `z-index` określa kolejność, w jakiej będą wyświetlane elementy zajmujące ten sam obszar strony (nachodzące na siebie). Elementy posiadające wyższą wartość właściwości `z-index` będą wyświetlane ponad elementami, w których wartość tej właściwości jest mniejsza. Opcja ta jest przydatna, kiedy chcemy mieć pewność, że przeciągany element będzie wyświetlany ponad pozostałą treścią strony. (Została ona zastosowana w przykładzie przedstawionym na stronie 446).

```
zIndex : 100
```

Uwaga: Jeszcze więcej opcji służących do konfigurowania działania tego widżetu można znaleźć na stronie *http://api.jqueryui.com/draggable/* (patrz rysunek 12 2).

Zdarzenia widżetu Draggable

Widżet Draggable obsługuje kilka różnych zdarzeń, z których każde jest zgłaszane w innym momencie realizacji procesu przeciągania. Aby program zrobił coś w odpowiedzi na zdarzenie, należy dodać do niego funkcję.

Rysunek 12.2. *Dokumentacja jQuery UI jest bardzo obszerna. Na stronie poświęconej widżetowi Draggable można znaleźć informacje o bardzo wielu opcjach, metodach i zdarzeniach, z których można korzystać w celu dostosowywania działania tego widżetu do własnych potrzeb*

Załóżmy, że napisaliśmy grę, która wymaga, by użytkownik szybko przeciągnął element do celu. Im szybciej to zrobi, tym wyższy będzie jego wynik. W takiej aplikacji możemy sprawdzać czas najpierw w momencie rozpoczynania przeciągania, a następnie robić to ponownie po zakończeniu przeciągania.

Dostępne są trzy zdarzenia odpowiadające trzem różnym etapom procesu przeciągania.

Zdarzenie create

Zdarzenie jest zgłaszane za każdym razem, gdy wywołana zostanie funkcja `draggable()` tworząca nowy, przeciągany element. Można je wykorzystać na przykład do wyświetlenia okna dialogowego z instrukcjami, takimi jak „Przeciągnij ten produkt do koszyka”. Jest ono zgłaszane tylko jeden raz — podczas tworzenia widżetu Draggable. Aby obsłużyć to zdarzenie, w obiekcie przekazywanym w wywołaniu funkcji `draggble()` należy dodać właściwość `create`, której wartością będzie funkcja:

```
$('.product').draggable({
  create : function (event){
    // To znajdzie się kod funkcji.
  }
});
```

Zdarzenie start

To zdarzenie jest zgłaszane natychmiast po rozpoczęciu przeciągania elementu. Aby je obsługiwać, należy podać odpowiednią właściwość w obiekcie opcji, przekazywanym w wywołaniu funkcji `draggable()`. Należy w tym celu użyć właściwości `start`, której wartością będzie funkcja — jQuery UI wywoła tę funkcję, gdy użytkownik zacznie przeciągać element.

Załóżmy, że na stronie znajduje się znacznik `<div>` o identyfikatorze `raceCar`. Aby przekształcić go w przeciągany element i jednocześnie określić funkcję, która będzie obsługiwać zdarzenie `start`, należy użyć następującego fragmentu kodu:

```
$('#raceCar').draggable({
  start : function (event, ui) {
    // Tu znajdzie się kod funkcji.
  }
});
```

Funkcja przypisywana właściwości `start` ma dwa parametry, `event` oraz `ui`. Pierwszy z nich jest obiektem `event` jQuery i udostępnia wiele informacji na temat elementu, do którego zostało skierowane zdarzenie, współrzędnych położenia wskaźnika myszy i tak dalej. (Więcej informacji na temat obiektu `event` można znaleźć na stronie 194). Z kolei parametr `ui` jest obiektem zawierającym cztery właściwości.

- Właściwość **`ui.helper`** to obiekt jQuery zawierający odwołanie do elementu, który jQuery UI przeciąga na stronie. Zazwyczaj jest to ten sam element, na rzecz którego została wywołana funkcja `draggable()`, jednak nie zawsze musi tak być. Jeśli podczas tworzenia widżetu Draggable została użyta właściwość `helper` o wartości `'clone'` lub przekazana funkcja generująca pomocniczy element HTML (patrz strona 427), właściwość `ui.helper` będzie zawierać odwołanie do elementu innego niż utworzony wcześniej element przeciągany, a konkretnie — do jego kopii bądź do pomocniczego obiektu jQuery (patrz strona 427). To rozwiązanie może się przydać na przykład na stronie sklepu internetowego, gdy przeciąganie widocznego na stronie produktu do koszyka mogłoby zaburzyć początkowy układ strony. W takich sytuacjach użytkownik nie powinien przeciągać samego elementu, lecz jego *kopię*.

  Właściwości `ui.helper` należy używać zawsze, gdy chcemy wykonać jakąś operację na tym elemencie strony, który jest wizualnie przeciągany. Przykładowo załóżmy, że chcemy, by po rozpoczęciu przeciągania wielkość elementu została podwojona. Można to zrobić, odpowiednio modyfikując właściwość CSS w obiekcie `ui.helper`:

  ```
  $('#photo').draggable({
    start : function (event, ui) {
      ui.helper.css('transform', 'scale(2)');
    }
  });
  ```

  W tym przykładzie użyta została metoda `css()` jQuery (patrz strona 163) i za pomocą właściwości CSS o nazwie `transform` zażądano przeskalowania elementu o współczynnik `2` (co spowoduje dwukrotne powiększenie elementu). Można by także użyć metody `addClass()` jQuery, by dodać do elementu jakąś klasę w momencie, gdy użytkownik zacznie go przeciągać. Taka klasa mogłaby na przykład wyróżnić element, zmieniając jego kolor. Później, kiedy użyt-

kownik skończy przeciągać element (czyli w ramach obsługi zdarzenia stop opisanego na stronie 434), dodaną do niego wcześniej nazwę klasy można usunąć, wywołując w tym celu metodę jQuery `removeClass()`.

Uwaga: Więcej informacji na temat przekształceń CSS można znaleźć na stronie *https://developer.mozilla.org/en-US/docs/Web/Guide/CSS/Using_CSS_transforms*.

- Właściwość **`ui.position`** zawiera współrzędne określające położenie lewego górnego wierzchołka *elementu pomocniczego* (czyli elementu, który jest wizualnie przeciągany na stronie). Informacje o położeniu określone są na podstawie arkusza stylów i mogą być zależne od elementu rodzica (bądź też innego umiejscowionego przodka). Jeśli na przykład przeciągany jest element umieszczony wewnątrz znacznika `<div>`, a znacznik ten został umiejscowiony na stronie w sposób bezwzględny, właściwości `left` i `top` obiektu `position` będą zawierały współrzędne określone względem lewego, górnego wierzchołka umiejscowionego znacznika `<div>`.

  Wartością właściwości `ui.position` jest obiekt JavaScript składający się z dwóch właściwości, `top` oraz `left`. Właściwość `top` określa współrzędną pionową, czyli wyrażoną w pikselach odległość od górnej krawędzi najbliższego umiejscowionego przodka (znacznika umiejscowionego w sposób względny bądź bezwzględny, wewnątrz którego znajduje się przeciągany element). Właściwość `left` określa współrzędną poziomą, czyli odległość od lewej krawędzi najbliższego umiejscowionego przodka. Właściwości tej można używać do określania położenia elementu pomocniczego w momencie, gdy użytkownik zakończy operację przeciągania bądź też w jej trakcie (patrz opisane poniżej zdarzenia `drag` i `stop`).

  Do wartości tych dwóch właściwości można się odwoływać w następujący sposób: `ui.position.left` oraz `ui.position.top`.

- Właściwość **`ui.offset`** także udostępnia obiekt zawierający dwie właściwości, `top` i `left`. Jednak to położenie jest wyznaczane względem lewego górnego wierzchołka *okna przeglądarki*. Jeśli przeciągany element nie będzie znajdował się wewnątrz żadnego innego elementu strony, w którym użyto właściwości `position: absolute` lub `position: relative`, wartości `ui.offset` oraz `ui.position` będą takie same. Właściwość `ui.offset.top` określa odległość elementu od górnej krawędzi okna przeglądarki, a właściwość `ui.offset.left` — odległość od lewej krawędzi okna. Obie te odległości są wyrażone w pikselach.

- Właściwość **`ui.originalPosition`** zawiera te same dwie właściwości — `top` oraz `left` — co właściwości `ui.position` oraz `ui.offset`. Jednak właściwość `ui.originalPosition` określa współrzędne początkowego położenia przeciąganego elementu, czyli miejsce, w którym się znajdował w momencie rozpoczynania przeciągania. Jej wartość, podobnie jak wartość właściwości `ui.position`, jest wyznaczana względem umiejscowionego elementu przodka.

Zdarzenie drag

Kolejnym zdarzeniem udostępnianym przez elementy przeciągane jest `drag` — zdarzenie generowane podczas przeciągania elementu na stronie. Można zatem utworzyć funkcję, która będzie wielokrotnie wywoływana podczas przeciągania

elementu. Można by jej użyć, by wyświetlać iskierki wyznaczające ślad przeciąganego elementu. Ponieważ zdarzenie to jest generowane wielokrotnie, należy ograniczać ilość operacji wykonywanych w ramach jego obsługi. Jeśli podczas obsługi zdarzenia będziemy wykonywać wiele skomplikowanych operacji, może to doprowadzić do spadku wydajności działania przeglądarki.

Funkcja obsługująca zdarzenia `drag` akceptuje dwa parametry — `event` oraz `ui` — te same, które zostały opisane na stronie 432 w opisie zdarzenia `start`.

Załóżmy, że chcemy wyświetlać bieżące położenie przeciąganego elementu, a na stronie znajdują się dwa znaczniki `<span>` o identyfikatorach `left` i `top`, służące do prezentowania tych współrzędnych. Poniższy fragment kodu pokazuje, w jaki sposób można utworzyć widżet Draggable, którego położenie będzie na bieżąco aktualizowane podczas przeciągania:

```
$('#raceCar').draggable({
  drag : function (event, ui) {
    $('#left').text(ui.position.left);
    $('#top').text(ui.position.top);
  }
});
```

Zdarzenie stop

Zdarzenie `stop` widżetu Draggable działa tak samo jak zdarzenie `start`, przy czym jest zgłaszane w momencie, gdy użytkownik zwolni przycisk myszy. Nie oznacza to wcale, że przeciągany element został upuszczony w obszarze któregoś z istniejących na stronie widżetów Droppable (opisanych w następnym podrozdziale). Nie oznacza to nawet, że użytkownik zakończył operację przeciągania. Przykładowo użytkownik mógł kliknąć element i zaczął go przeciągać (co spowodowało zgłoszenie zdarzenia `start`), przeciągał przez chwilę (generując sekwencję zdarzeń `drag`), następnie skończył przeciąganie (generując zdarzenie `stop`), po czym ponownie kliknął element i rozpoczął cały ten proces od początku.

Zdarzenie `stop` obsługuje się dokładnie w tak samo jak zdarzenie `start`. Załóżmy, że użyliśmy przedstawionego wcześniej fragmentu kodu do utworzenia elementu, który jest dwukrotnie powiększany po rozpoczęciu przeciągania. W takim przypadku poniższa funkcja pozwoli przywrócić jego początkowe wymiary po zakończeniu przeciągania:

```
$('#photo').draggable({
  stop : function (event, ui) {
    ui.helper.css('transform', 'scale(1)');
  }
});
```

Widżet Droppable

Sam widżet Draggable może być przydatny do tworzenia okien dialogowych oraz innych elementów stron, których położenie można zmieniać (takich jak na przykład przybornik z narzędziami). Kiedy jednak uzupełnimy go o kolejny widżet jQuery UI — Droppable — uzyskamy możliwość tworzenia wysoce interaktywnych aplikacji, w których przeciągnięcie elementu i upuszczenie go na innym może spowodować wykonanie jakiejś akcji.

Przykładowo aplikacja do udostępniania zdjęć może pozwalać użytkownikom na usuwanie zdjęcia po przeciągnięciu jego miniaturki i upuszczeniu jej na ikonie kosza na śmieci. Witryna do nauki języka mogłaby sprawdzać poziom znajomości słownictwa użytkowników, prosząc o przeciągnięcie słowa na odpowiedni obrazek.

Stosowanie widżetu Droppable

Widżety Droppable nie są przydatne bez widżetów Draggable. Widżet Droppable pełni rolę „obszaru, w którym można upuścić" przeciągany widżet Draggable. W momencie upuszczenia elementu widżet Droppable może dodatkowo wykonać określoną funkcję. Wszystkie elementy strony można przekształcić na widżety Droppable. Oczywiście możliwość przeciągnięcia jednego elementu i upuszczenia go na innym powinna mieć sens; na przykład ikona koszyka z zakupami oraz kosza na śmieci są jedynie wizualnymi metaforami, które w zrozumiały sposób informują o możliwości upuszczenia elementu. Jednak widżetami Draggable mogą także być zwyczajne znaczniki `<div>`, o ile tylko jesteśmy w stanie w zrozumiały sposób pokazać użytkownikom, że coś można przeciągnąć i upuścić w ich obszarze.

Stosowanie widżetu Droppable jest bardzo proste.

1. **Dołącz do strony niezbędne pliki CSS i JavaScript, zgodnie z informacjami podanymi na stronie 329.**

   Pamiętaj, że biblioteka jQuery UI ma własne pliki CSS oraz JavaScript i należy je dołączyć do strony *za* plikiem JavaScript biblioteki jQuery.

2. **Umieść na stronie bądź w kolejnym zewnętrznym pliku JavaScript wywołanie funkcji `$(document).ready()`:**

   ```
   $(document).ready(function() {

   }); // Koniec funkcji ready.
   ```

   Zgodnie z informacjami podanymi na stronie 190, krok ten jest niezbędny wyłącznie w przypadku, gdy kod JavaScript umieszczamy w sekcji `<head>` strony, przed właściwym kodem HTML stanowiącym jej treść. Niektórzy programiści umieszczają swój kod JavaScript na końcu strony, bezpośrednio przed zamykającym znacznikiem `</body>`; w takim przypadku umieszczanie kodu w wywołaniu funkcji `$(document).ready()` nie jest konieczne.

   Stosowanie widżetów Droppable nie ma większego sensu, jeśli na stronie nie zostaną umieszczone elementy, które można przeciągać. Dlatego nie zapomnij dodać przynajmniej jednego widżetu Draggable.

3. **Użyj jQuery żeby wybrać element strony i zastosować w nim widżet Draggable:**

   ```
   $(document).ready(function() {
     $('.product').draggable();
   }); // Koniec funkcji ready.
   ```

   W tym przykładzie dysponujemy stroną prezentującą zawartość katalogu, na której umieszczono kilka produktów. Zdjęcie każdego produktu należy do klasy `product`, zatem powyższy fragment kodu przekształci wszystkie zdjęcia na elementy, które można przeciągać.

4. **Użyj jQuery, by wybrać element, w którym chcesz upuszczać zdjęcia produktów, i przekształć go w widżet Droppable:**

```
$(document).ready(function() {
  $('.product').draggable();
  $('#cart').droppable();
}); // Koniec funkcji ready.
```

W ten sposób element o identyfikatorze `cart` został przekształcony w obszar, w którym można upuszczać przeciągane elementy. Aby po upuszczeniu elementu coś się stało, by na przykład została przeliczona całkowita wartość koszyka, za każdym razem gdy użytkownik coś w nim umieści, trzeba określić różne opcje widżetu Droppable. W tym celu musisz przekazać w wywołaniu funkcji `droppable()` obiekt, zawierający odpowiednie opcje i funkcje.

5. **Dodaj opcje do wywołania funkcji `droppable()`:**

```
$(document).ready(function() {
  $('.product').draggable();
  $('#cart').droppable({
    activeClass : 'highlight',
    drop : function (event,ui) {
      alert('Produkt został dodany');
    }
  });
}); // Koniec funkcji ready.
```

Najważniejsze opcje tego widżetu poznasz już w następnym punkcie rozdziału, jednak teraz w ramach drobnej prezentacji jego możliwości przyjrzymy się dwóm opcjom podanym w powyższym kodzie. Pierwsza z nich, `activeClass`, informuje jQuery UI, że należy dodać do elementu klasę `highlight`. Innymi słowy, pozwala na dodawanie do elementu określonej klasy CSS. Taka klasa może w widoczny sposób wyróżnić widżet Droppable, dodając do niego jaskrawe tło bądź czerwone obramowanie.

Z kolei druga opcja, `drop`, określa procedurę obsługi zdarzeń. Pozwala na wykonywanie określonej funkcji, kiedy jakiś element zostanie upuszczony w obszarze danego widżetu Droppable.

Opcje widżetu Droppable

Widżety Droppable nie udostępniają aż tak wielu opcji jak widżety Draggable (ich podstawowym przeznaczeniem jest przyjmowanie upuszczanych elementów, a w zasadzie wykonywanie określonych funkcji, kiedy to się zdarzy).

- **`accept`**. Opcja określa, które widżety Draggable będzie można dodawać do danego widżetu Droppable. Jej wartością może być bądź to selektor, bądź funkcja. W pierwszym przypadku będzie to selektor przeciąganych elementów. Jeśli na przykład na stronie znajduje się grupa obrazków, z których każdy należy do klasy `photo`, i chcemy, by można je było upuszczać w danym widżecie Droppable, możemy użyć następującej opcji `accept`:

```
accept : '.photo'
```

- **`activeClass`**. Opcja pozwala wyróżnić widżet Droppable, w momencie gdy użytkownik zacznie przeciągać dowolny widżet Draggable, który można w nim upuścić. Przykładowo załóżmy, że napisaliśmy aplikację do obsługi systemu

plików, pozwalającą na przeglądanie plików umieszczonych na serwerze, przenoszenie ich do innych katalogów, zmienianie nazw i tak dalej. Można także przeciągać pliki do kosza, aby je usuwać. Ikona kosza na śmieci byłaby widżetem Droppable, natomiast wszystkie pliki — widżetami Draggable. Kiedy użytkownik zacznie przeciągać plik, nasza aplikacja może dodawać do ikony kosza na śmieci wybraną klasę, aby go w jakiś sposób wyróżnić (na przykład zmienić obraz tła, by wyświetlić kosz z otwartą klapą, gotowy na wyrzucenie pliki). W tym celu wystarczy przypisać tej opcji nazwę klasy (bez kropki), a jQuery UI doda ją do widżetu Droppable, gdy użytkownik zacznie przeciągać odpowiedni widżet Draggable:

```
activeClass : 'hightlight'
```

- **`disabled`**. Określa, czy dany element jest aktywny i można w nim upuszczać przeciągane elementy. Jeśli wartością tej opcji będzie `true`, upuszczanie elementów nie będzie możliwe. Wartość tej właściwości można określać *po* utworzeniu widżetu jako sposób jego włączania i wyłączania. Przykładowo załóżmy, że na stronie utworzyliśmy widżet Droppable, lecz chcemy zezwolić na upuszczenie na nim jedynie pięciu elementów. Kiedy już te pięć elementów zostanie upuszczonych, możemy wyłączyć widżet, dzięki czemu użytkownik nie będzie już mógł na nim nic więcej upuścić. (Taką funkcję wyłączającą działanie widżetu można by wykonywać w ramach obsługi zdarzeń `drop`, opisanych na stronie 439).

  ```
  $('#dropZone').droppable({
    disabled : true
  });
  ```

  Można by pomyśleć, że gdybyśmy nie chcieli zapewniać możliwości upuszczania elementów na danym elemencie, moglibyśmy w ogóle nie wywoływać funkcji `droppable()`. To prawda, dlatego zazwyczaj opcja `disabled` nie jest stosowana już w momencie tworzenia widżetu.

  Jednak może się ona przydać później, kiedy użytkownik *już upuścił* jakiś element na widżecie Droppable. Może się zdarzyć, że będziemy chcieli pozwolić użytkownikowi na dodanie do koszyka jedynie pięciu produktów. W takim przypadku po dodaniu piątego elementu wyłączymy widżet, dzięki czemu użytkownik nie będzie już mógł dodać nic więcej do koszyka.

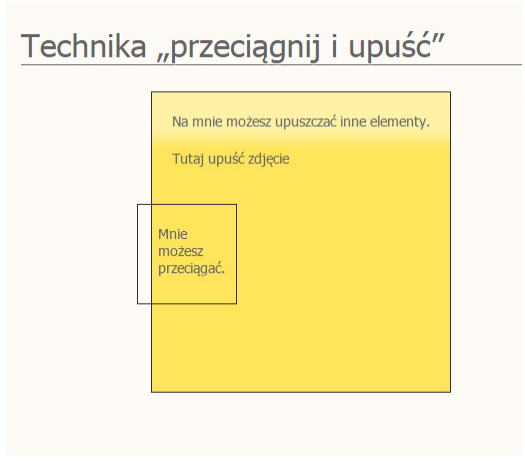
- **`hoverClass`**. Można także podać klasę, która będzie dodawana do widżetu Droppable, kiedy w jego obszarze będzie przeciągany jakiś element, który można w nim upuścić (patrz rysunek 12.3). Przykładowo do elementu z ikoną kosza na śmieci można by dodawać jakąś klasę *tylko* wtedy, gdy w jego obszarze zostanie umieszczony przeciągany plik:

  ```
  hoverClass : 'openTrashcan'
  ```

- **`scope`**. Opcja działa tak samo jak analogiczna opcja widżetu Draggable (patrz strona 429). Pozwala na grupowanie powiązanych ze sobą widżetów Draggable i Droppable.

  ```
  scope : 'calendar'
  ```

  Zastosowana nazwa — w tym przykładzie jest nią `'calendar'` — nie ma znaczenia. Można użyć dowolnej, sensownej nazwy; trzeba tylko upewnić się, że nazwa użyta we wszystkich widżetach będzie taka sama.



Rysunek 12.3. *Przy użyciu opcji hoverClass widżetu Droppable można dodać do elementu nazwę klasy, kiedy inny element będzie przeciągany w jego obszarze. W tym przykładzie kiedy mały kwadrat będzie przeciągany w obszarze dużego, duży kwadrat zostanie wyróżniony żółtym tłem (zmiana koloru jest określana przez style CSS). Zastosowany styl dodaje do elementu zarówno obraz tła, jak i kolor tła, jednak można by opracować styl, który dodawałby grube obramowanie obszaru lub nawet używał animacji CSS, by element pulsował, zmieniając kolor tła między białym a innym, wybranym kolorem*

- **`tolerance`**. Opcja określa, kiedy widżet uzna, że przeciągany element znajduje się w jego obszarze. Można jej przypisać jedną z czterech wartości.
  - `'fit'` — przeciągany element musi się całkowicie zawierać w obszarze widżetu Droppable.
  - `'intersect'` — przeciągany element musi się zawierać w obszarze widżetu Droppable przynajmniej w 50% na wysokość i szerokość. Innymi słowy, większa część przeciąganego elementu musi się znajdować w obszarze widżetu Droppable. Stanowi ona domyślne ustawienie widżetu.
  - `'pointer'` — jedynie wskaźnik myszy musi się znajdować w obszarze widżetu Droppable.
  - `'touch'` — przeciągany element musi dotykać widżetu Droppable tylko z jednej strony.

  Wartość `'fit'` jest przydatna w sytuacjach, gdy chcemy potwierdzić, że użytkownik naprawdę chce umieścić przeciągany element w wyznaczonym obszarze, lecz jednocześnie wymaga, by przeciągany element był mniejszy od widżetu Droppable, w którym ma zostać umieszczony. Najczęściej stosowaną wartością jest `'intersect'`, gdyż nie wymaga od użytkownika zachowania dużej precyzji podczas przeciągania:

```
tolerance : 'intersect'
```

Uwaga: Pełną listę opcji, metod i zdarzeń widżetu Droppable można znaleźć na stronie *http://api.jqueryui.com/droppable/*.

Zdarzenia widżetu Droppable

Prawdziwa zabawa z widżetem Droppable zaczyna się dopiero wtedy, gdy zaczniemy wykonywać jakieś operacje w odpowiedzi na upuszczenie elementu w jego obszarze bądź gdy element jest w nim przeciągany… lub przeciągany poza jego obszar. Przykładowo możemy wyliczać całkowitą wartość zakupów, za każdym razem

gdy użytkownik upuści produkt na koszyku, a następnie aktualizować ją, kiedy użytkownik usunie jakiś produkt z koszyka.

Widżet Droppable udostępnia kilka różnych zdarzeń, z których każde jest zgłaszane w wyniku innej interakcji. Aby nasz program w jakiś sposób odpowiedział na przeciąganie elementu, usunięcie go z obszaru widżetu bądź też upuszczenie w obszarze widżetu, należy dodać do odpowiedniego zdarzenia funkcję. Załóżmy, że napisaliśmy aplikację obsługującą listę zadań: za każdym razem, gdy użytkownik przeciągnie zadanie z listy i upuści je na ikonie kosza na śmieci, element tego zadania jest usuwany z listy oraz w ogóle ze strony. W takim przypadku zdarzeniem, które należy wykorzystać, jest zdarzenie `drop`.

W kolejnych podpunktach rozdziału przedstawione zostały zdarzenia widżetu Droppable, począwszy od tych, które są najczęściej stosowane.

Zdarzenie drop

Zdarzenie wykonuje skojarzoną z nim funkcję, kiedy przeciągany element zostanie upuszczony w obszarze widżetu Droppable. Upuszczany element musi być elementem *akceptowanym* — czyli określonym przez opcję `accept` (patrz strona 437) — bądź mieć taką samą wartość opcji `scope` (patrz strona 429), co widżet Droppable.

Zdarzenie wraz z obsługującą je funkcją jest określane jako właściwość obiektu przekazywanego podczas tworzenia widżetu Droppable. Przykładowo załóżmy, że na stronie znajduje się znacznik `<div>` o identyfikatorze `trashcan`. Poniższy fragment kodu przekształca go w widżet Droppable i jednocześnie określa funkcję obsługującą zdarzenia `drop`:

```
$('#trashcan').droppable({
  drop : function (event, ui) {
    // Kod obsługujący zdarzenie.
  }
});
```

Funkcja obsługująca to zdarzenie ma dwa parametry, `event` oraz `ui`. Pierwszy z nich, `event`, jest obiektem `event` jQuery (patrz strona 194) zawierającym takie informacje jak element, do którego zostało skierowane zdarzenie, współrzędne wskaźnika myszy i tak dalej. Drugi parametr, `ui`, przypomina analogiczny parametr używany w funkcjach obsługujących zdarzenia widżetu Droppable (patrz strona 430).

- Właściwość **`ui.helper`** jest obiektem jQuery zawierającym odwołanie do elementu, który jQuery UI przeciąga na stronie, i ma tę samą wartość, co właściwość `ui.helper` widżetów Draggable, opisana na stronie 432.

  Właściwości `ui.helper` należy używać zawsze, gdy chcemy wykonać jakąś operację na elemencie strony, który jest wizualnie przeciągany. Załóżmy, że chcemy, by po upuszczeniu przeciągany element został ukryty przy użyciu jednego z dostępnych efektów. W poniższym kodzie pokazano, jak to zrobić:

  ```
  $('#trashcan').droppable({
    drop : function (event, ui) {
      ui.helper.hide('explode');
    }
  });
  ```

Użycie wartości `'explode'` spowoduje zastosowanie jednego z naprawdę atrakcyjnych efektów wizualnych jQuery UI, który został dokładniej przedstawiony na stronie 463.

- Właściwość **`ui.draggable`** jest obiektem jQuery zawierającym odwołanie do elementu, na rzecz którego została wywołana funkcja `draggable()`. W większości przypadków to ten sam obiekt, który jest zapisany we właściwości `ui.helper`. Jednak w przypadku, gdy właściwości `helper` (patrz strona 427) przeciąganego elementu została przypisana wartość `'clone'`, na stronie będzie przeciągana kopia, a sam element pozostanie w początkowym położeniu.

  Właściwość może się przydać podczas tworzenia strony sklepu internetowego z koszykiem zakupów. Załóżmy, że napisaliśmy system dla sklepu, który obsługuje zarówno magazyn, jak i sprzedaż produktów. Strona katalogu może przedstawiać zdjęcie produktu oraz informacje o tym, ile jego egzemplarzy jest dostępnych w magazynie, na przykład 10. Kiedy użytkownik kliknie produkt, może przeciągnąć kopię zdjęcia na ikonę koszyka. Po upuszczeniu produktu funkcja obsługująca zdarzenie `drop` mogłaby zaktualizować element `ui.draggable`, zmniejszając o jeden liczbę dostępnych egzemplarzy danego produktu (a jednocześnie mogłaby, korzystając z technologii AJAX — opisanej w rozdziale 13. — zapisać odpowiednie informacje na serwerze bazy danych).

- Właściwość **`ui.position`** zawiera współrzędne określające położenie lewego górnego wierzchołka elementu pomocniczego (czyli elementu, który jest wizualnie przeciągany na stronie). Odpowiada ona właściwości `ui.position` opisanej na stronie 433 przy okazji prezentowania widżetu Draggable.

- Właściwość **`ui.offset`** określa położenie górnego lewego rogu przeciąganego elementu wyrażone względem okna przeglądarki. Odpowiada ona właściwości `ui.offset` opisanej na stronie 433 przy okazji prezentowania widżetu Draggable.

- Właściwość **`ui.originalPosition`** zawiera współrzędne określające położenie górnego lewego rogu widżetu Draggable, zanim użytkownik zaczął go przeciągać. Odpowiada ona właściwości `ui.originalPosition` opisanej na stronie 433 przy okazji prezentowania widżetu Draggable.

Podczas stosowania widżetu Droppable bardzo często będziemy korzystać ze zdarzenia `drop`. Jedną z operacji, które być może będziemy musieli wykonać, może być uniemożliwienie przeciągnięcia upuszczonego elementu poza obszar widżetu Droppable. Wróćmy do przykładu z internetową wersją gry w warcaby: ograniczenie to może się przydać, by nie pozwolić graczowi na cofnięcie wykonanego ruchu. Innymi słowy, możemy „zablokować" pionek na polu, w którym gracz go umieścił. Można to zrobić, wyłączając przeciągany element, w momencie gdy zostanie upuszczony:

```
$('.square').drop({
  drop : function (event, ui) {
    ui.helper.draggable({
      disabled : true
    });
  }
});
```

Wykorzystaliśmy obiekt `helper` parametru `ui` (czyli element upuszczony w widżecie Droppable) i wywołaliśmy jego metodę `draggable()`, by przypisać właściwości `disabled` wartość `true` i wyłączyć w ten sposób możliwość jego przeciągania. Zwróć uwagę, że wykorzystaliśmy właściwość `disabled` przeciąganego elementu (patrz strona 426), a nie obszaru, w którym został upuszczony. Aby pozwolić na dalsze przesuwanie elementu, będziemy musieli ponownie zmienić wartość jego właściwości `disabled`, przypisując jej tym razem wartość `false` (patrz strona 426).

Zdarzenie activate

Kiedy użytkownik zacznie przeciągać element, który może zostać upuszczony w danym widżecie Droppable (patrz opisane wcześniej opcje `accept` oraz `scope` widżetu Droppable), zostaje zgłoszone zdarzenie `activate`. Można je wykorzystać i dodać do widżetu funkcję, która je obsłuży. Załóżmy, że tworzymy aplikację, która pozwala użytkownikowi przeciągnąć kilka miniaturek zdjęć, upuścić je w obszarze elementu `div`, a następnie kliknąć przycisk *Wyślij*, aby wysłać wszystkie do znajomego. W takim przypadku moglibyśmy wyświetlać w elemencie `div` komunikat „Tutaj upuść zdjęcie", lecz wyłącznie wtedy, gdy użytkownik zacznie przeciągać zdjęcie.

Aby to zrobić, możemy dodać do zdarzenia `activate` funkcję, która wyświetli w obszarze elementu `div` odpowiedni komunikat. Przy założeniu, że element `div` stanowiący obszar, w którym użytkownik może upuszczać zdjęcia, ma identyfikator `photoZone`, poniższy fragment kodu pozwala przekształcić go w widżet Droppable i wyświetlać w nim komunikat za każdym razem, gdy użytkownik zacznie przeciągać zdjęcie:

```
$('#photoZone').droppable({
  activate : function (event, ui) {
    $(this).append('<p id="dropMessage">Tutaj upuść zdjęcie</p>');
  }
});
```

Wyrażenie `$(this)` odwołuje się do bieżącego elementu (jeśli musisz sobie przypomnieć, co oznacza to wyrażenie i jak działa, zajrzyj na stronę 169), czyli widżetu Droppable. Metoda `append()` dodaje podany kod HTML na samym końcu wybranego elementu (patrz strona 157).

Funkcja `activate` pobiera dwa parametry — `event` oraz `ui` — te same, które są używane w funkcji obsługującej zdarzenia `drop` (patrz strona 439).

Zdarzenie deactivate

Zdarzenie `deactivate` to przeciwieństwo zdarzenia `activate`. Jest zgłaszane w momencie, gdy użytkownik *skończy* przeciągać element, który można upuścić w obszarze danego widżetu Droppable, czyli zwolni przycisk myszy. Można je wykorzystać, by cofnąć efekty czynności wykonanych w ramach obsługi zdarzenia `activate`. W poniższym fragmencie kodu pokazano, w jaki sposób można wyświetlać komunikat w widżecie Droppable, gdy użytkownik zacznie przeciągać element, a następnie usunąć go po zakończeniu przeciągania.

```
$('#photoZone').droppable({
  activate : function (event, ui) {
    $(this).append('<p id="dropMessage">Tutaj upuść zdjęcie</p>');
  },
```

```
  deactivate : function (event, ui) {
    $('#dropMessage').remove();
  }
});
```

Zdarzenie over

Można nawet wykonywać funkcję podczas przeciągania elementu w obszarze widżetu Droppable. Zdarzenie `over` zaczyna być zgłaszane od razu, gdy przeciągany element znajdzie się w obszarze widżetu Droppable, a przestaje być zgłaszane, gdy element zostanie upuszczony. Tego zdarzenia można używać, by na przykład wyświetlać potencjalnemu klientowi komunikat zachęcający do upuszczenia produktu na ikonie koszyka, taki jak: „Wiesz, że w tych butach będziesz wyglądał świetnie. Nie wahaj się!".

Załóżmy, że utworzyliśmy na stronie obszar reprezentowany przez ikonę kosza na śmieci, na którym użytkownik może upuścić zdjęcie, aby je usunąć. Normalnie kosz na śmieci jest zamknięty, kiedy jednak użytkownik umieści na nim zdjęcie, możemy użyć zdarzenia `over`, by wyświetlić ikonę kosza z otworzoną klapą. Podobnie, kiedy użytkownik już upuści zdjęcie, możemy użyć zdarzenia `drop`, by zamiast pustego kosza wyświetlić ikonę wypełnionego.

Zdarzenie `over` obsługuje się dokładnie tak samo jak wszystkie inne zdarzenia widżetów — wystarczy przypisać funkcję właściwości `over`.

```
$('#trashcan').droppable({
  over : function (event, ui) {
    $('#trashCanImage').attr('src','images/open-lid-can.png');
  }
});
```

W tym przypadku zmieniamy jedynie wyświetlony obrazek, jednak funkcja obsługi tego zdarzenia (jak również wszystkich innych zdarzeń) może być znacznie bardziej złożona.

Uwaga: Czynnikiem określającym, kiedy jQuery uzna, że przeciągany element znajduje się w obszarze widżetu Droppable, kiedy został upuszczony oraz kiedy został usunięty z obszaru widżetu, jest właściwość `tolerance` (opisana na stronie 438).

Zdarzenie out

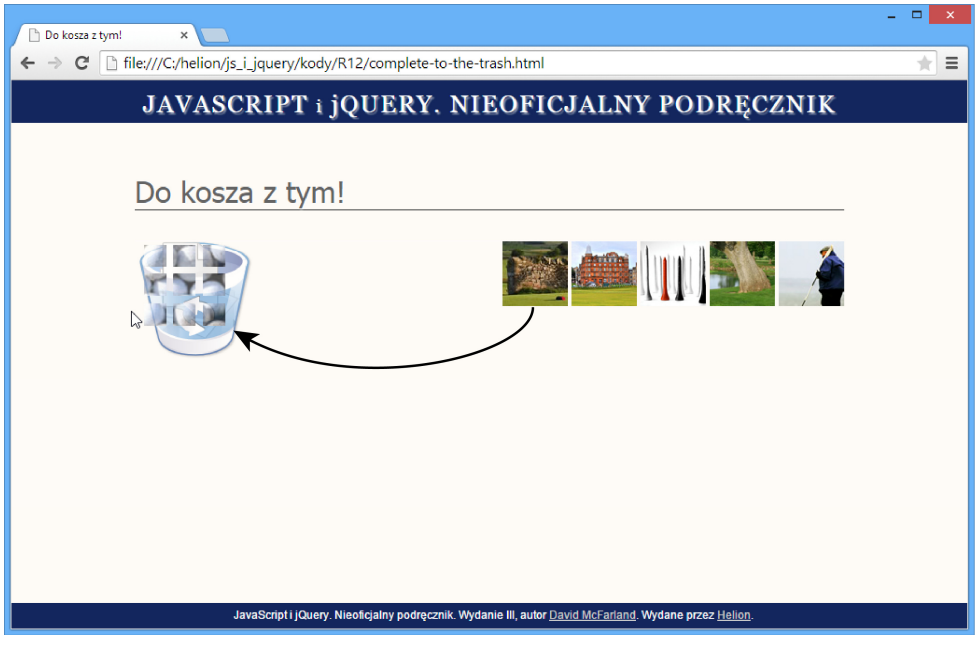
I w końcu można także zażądać wykonania określonej funkcji, w momencie gdy użytkownik przeciągnął element poza obszar widżetu Droppable. Załóżmy, że użytkownik przeciągnął produkt do obszaru koszyka z zakupami, co spowodowało, że prezentowana gdzieś na stronie całkowita wartość zakupów została zaktualizowana. Później jednak użytkownik uznał, że nie chce kupować danego produktu, i przeciągnął go poza obszar koszyka. W takim przypadku zdarzenie `out` daje możliwość odjęcia ceny produktu od całkowitej wartości zakupów:

```
$('#shoppingCart').droppable({
  drop : function (event, ui) {
    // Wyliczenia wartości zakupów po dodaniu produktu do koszyka.
  },
  out : function (event, ui) {
    // Wyliczenia wartości zakupów po usunięciu produktu z koszyka.
  }
});
```

Tych kilka przykładów wyraźnie pokazuje, że zdarzenia zazwyczaj będą używane parami, pozwalającymi na wykonywanie odwrotnych czynności. Takimi przykładowymi parami mogą być zdarzenia `out` oraz `drop` bądź `deactivate` oraz `activate`.

Przykład — technika „przeciągnij i upuść”

W tym przykładzie wykorzystasz oba widżety, Draggable i Droppable, w jednym programie. Napiszesz w nim prostą aplikację korzystającą z techniki „przeciągnij i upuść”, prezentującą podstawowe możliwości i sposoby użycia obu tych widżetów. Dokończona wersja strony będzie pozwalała na przeciąganie zdjęcia do kosza na śmieci i usunie je ze strony w bardzo widowiskowy sposób (przedstawiony na rysunku 12.4).



Rysunek 12.4. *Umieszczenie na stronie elementów, które można przeciągać i upuszczać, może dawać zabawne, wciągające i interesujące wrażenia. W tym przykładzie nauczysz się, jak tworzyć elementy, które można przeciągać, i jak sprawić, by znikły po upuszczeniu w wybranym miejscu strony*

Uwaga: Więcej informacji o tym, jak pobrać przykłady prezentowane w książce, można znaleźć na stronie 46.

1. **W edytorze tekstów otwórz stronę *to-the-trash.html*, umieszczoną w katalogu *R12*.**

   Do pliku zostały już dołączone niezbędne pliki CSS i JavaScript, jak również znacznik `<script>` zawierający wywołanie funkcji `$(document).ready()` (patrz strona 190).

Jeśli wyświetlisz stronę w przeglądarce, zobaczysz, że z jej lewej strony jest umieszczona ikona kosza na śmieci, a z prawej strony — kilka miniaturek zdjęć (patrz rysunek 12.4). Poniżej przedstawiony został kod HTML ikony kosza oraz zdjęć:

```
<img src="../_images/trashcan-empty-icon.png" id="trashcan">
<div id="photos">
  <img src="../_images/small/slide1_h.jpg">
  <img src="../_images/small/slide2_h.jpg">
  <img src="../_images/small/slide3_h.jpg">
  <img src="../_images/small/slide4_h.jpg">
  <img src="../_images/small/slide5_h.jpg">
  <img src="../_images/small/slide6_h.jpg">
</div>
```

Obrazek kosza na śmieci ma identyfikator `trashcan`, a wszystkie miniaturki zdjęć zostały umieszczone wewnątrz elementu `div` o identyfikatorze `photos`. Zaczniesz od zapewniania możliwości przeciągania miniaturek zdjęć.

2. **W pustym wierszu wewnątrz wywołania funkcji `$(document).ready()` dodaj wiersz kodu wyróżniony poniżej pogrubioną czcionką:**

```
$(document).ready(function() {
  $('#photos img').draggable();
}); // Koniec funkcji ready.
```

To wywołanie powoduje wybranie elementów `img` umieszczonych wewnątrz elementu `div` i przekształcenie ich w widżety Draggable. Ponieważ celem tego przykładu jest zapewnienie możliwości dodawania miniaturek zdjęć do kosza na śmieci, zatem chcesz mieć pewność, że w przypadku, gdy użytkownik upuści miniaturkę w innym miejscu strony niż na ikonie kosza, wróci ona w swoje początkowe położenie.

3. **W wywołaniu funkcji `draggable()` dodaj literał obiektowy, zawierający właściwość `revert` o wartości `'invalid'`:**

```
$('#photos img').draggable({
  revert : 'invalid'
});
```

Opcja `revert` opisana na stronie 428 spowoduje, że po upuszczeniu przeciągany element wróci do swojego początkowego położenia. Przypisując tej opcji wartość `'invalid'`[1], stwierdzasz: „Jeśli ten element zostanie upuszczony w niewłaściwym miejscu, należy go wrócić w początkowe położenie".

4. **Zapisz stronę i wyświetl ją w przeglądarce. Spróbuj przeciągnąć zdjęcie i upuścić je w innym miejscu strony.**

Zdjęcie powinno powrócić do początkowego położenia. Jeśli tak się nie stało, sprawdź kod i zajrzyj także do konsoli JavaScript przeglądarki (patrz strona 51).

Następnie użyjesz zdarzeń widżetu Draggable, aby wykonać jakieś czynności w momencie rozpoczynania przeciągania. Konkretnie rzecz biorąc, zmodyfikujesz nieco wygląd miniaturki zdjęcia z wykorzystaniem przekształcenia CSS.

5. **Za łańcuchem `'invalid'` wpisz przecinek, naciśnij klawisz *Enter*, a następnie dodaj funkcję obsługującą zdarzenia `start` (jej kod został wyróżniony pogrubioną czcionką):**

[1] Słowo „invalid" w języku angielskim oznacza: nieważny, błędny — *przyp. tłum.*

```
$('#photos img').draggable({
  revert : 'invalid',
  start : function (event, ui) {

  }
});
```

Wartością opcji `start` musi być funkcja, która pełni rolę procedury obsługi zdarzeń, podobnie jak te, których używałeś wcześniej w funkcjach jQuery `click()` lub `mouseover()` (patrz strona 182). „Start” nie jest jednak prawdziwym zdarzeniem generowanym przez przeglądarkę. Jest to zdarzenie niestandardowe, utworzone przez autorów biblioteki jQuery UI. Jest ono zgłaszane w momencie, gdy użytkownik zacznie przeciągać element. Funkcja, którą dodałeś do kodu w tym kroku, jest pusta, zatem będziesz musiał dopisać kod, który coś zrobi. W tym przypadku zmodyfikujesz nieco wygląd miniaturki zdjęcia przy użyciu funkcji `css()` jQuery (patrz strona 163) i prostych przekształceń CSS.

6. **Wewnątrz funkcji dodanej w poprzednim kroku wpisz poniższy wiersz kodu (wyróżniony pogrubioną czcionką):**

```
$('#photos img').draggable({
  revert : 'invalid',
  start : function (event, ui) {
    ui.helper.css('transform', 'rotate(5deg) scale(1.5)');
  }
});
```

Zgodnie z informacjami podanymi na stronie 432, funkcje obsługujące zdarzenia widżetów Draggable mają dwa parametry: obiekt `event` oraz obiekt `ui`, reprezentujący element przeciągany w oknie przeglądarki. Parametru `ui` można użyć, by odwołać się do przeciąganego elementu i coś z nim zrobić. Właściwość `ui.helper` (patrz strona 432) zawiera faktyczny element, który jest przeciągany w oknie przeglądarki, a ponieważ jest to obiekt jQuery, można go użyć do wywoływania wszelkich funkcji jQuery. Tutaj funkcja `css()` określa wartość właściwości `transform` CSS, dzięki czemu element zostanie nieco obrócony i powiększony 1,5 raza. Innymi słowy, przeciągana miniaturka zdjęcia zostanie obrócona i powiększona.

W następnym kroku zapewnisz, że upuszczona miniaturka wróci na swoje początkowe położenie.

Uwaga: Więcej informacji na temat właściwości CSS transform można znaleźć na stronie *http://www.sitepoint.com/css3-transformations-2d-functions/*.

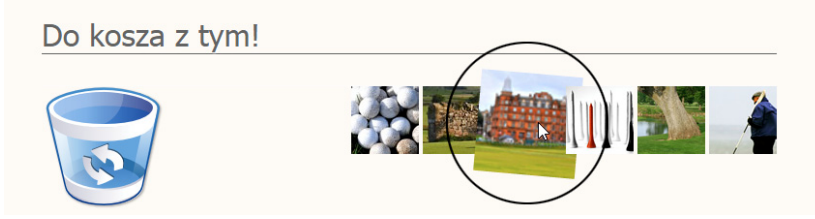
---

7. **Do obiektu opcji dodaj zdarzenie `stop`:**

```
$('#photos img').draggable({
  revert : 'invalid',
  start : function (event, ui) {
    ui.helper.css('transform', 'rotate(5deg) scale(1.5)');
  },
  stop : function (event, ui) {
    ui.helper.css('transform', 'rotate(0deg) scale(1)');
  }
});
```

Nie zapomnij o przecinku za zamykającym nawiasem klamrowym funkcji `start`.

Funkcja ta jedynie odtwarza stan elementu sprzed modyfikacji wprowadzonych w funkcji `start` — czyli usuwa obrót i przywraca początkową wielkość miniaturki zdjęcia. Jeśli teraz zapiszesz plik i wyświetlić go w przeglądarce, zauważysz coś dziwnego: gdy klikniesz miniaturkę i zaczniesz ją przeciągać, zobaczysz, że jest wyświetlana *poniżej* miniaturki z jej prawej strony (patrz rysunek 12.5). Przyczyna takiego zachowania została wyjaśniona na rysunku 12.5. Rozwiązanie tego problemu jest całkiem proste — wystarczy zmienić wartość z-index przeciąganego elementu.



Rysunek 12.5. *Na wartość z-index elementu ma wpływ k lka czynników. Przykładowo kolejność rozmieszczenia elementów w kodzie HTML wpływa na sposób, w jaki będą się one wzajemnie przesłaniać. W tym przypadku znacznik <img> wybranego zdjęcia hotelu (zakreślonego) jest umieszczony w kodzie strony przed znacznikiem <img> obrazka przedstawiającego podstawki pod piłeczki golfowe (patrz krok 1. na stronie 443). Z tego powodu zdjęcie hotelu jest przesłonięte przez zdjęcie podstawek*

8. **Do literału obiektowego dodaj ostatnią już opcję — `zIndex`.**

```
$('#photos img').draggable({
  revert : 'invalid',
  start : function (event, ui) {
    ui.helper.css('transform', 'rotate(5deg) scale(1.5)');
  },
  stop : function (event, ui) {
    ui.helper.css('transform', 'rotate(0deg) scale(1)');
  },
  zIndex : 100
});
```

Nie zapomnij o przecinku za zamykającym nawiasem klamrowym funkcji `stop`. Opcja `zIndex` (opisana na stronie 430) zmienia jedynie wartość właściwości CSS `z-index` elementu. Im wyższa jest wartość tej właściwości, tym „wyżej” będzie położony dany element na stosie elementów zajmujących ten sam obszar i wzajemnie się przesłaniających. Wartość `100` jest na tyle duża, by zapewnić, że przeciągana miniaturka zdjęcia nie zostanie przesłonięta przez żaden inny element strony.

9. **Zapisz plik i wyświetl go w przeglądarce. Spróbuj przeciągnąć zdjęcie na stronie.**

Kiedy zaczniesz przeciągać miniaturkę zdjęcia, zostanie ona nieco obrócona i powiększona, a później, kiedy zwolnisz przycisk myszy, wróci do początkowych wymiarów i orientacji. Co więcej, kiedy zaczniesz przeciągać miniaturkę, będzie ona wyświetlana powyżej wszelkich innych miniaturek umieszczonych na stronie.

W ten sposób zakończyłeś pracę nad zdjęciami. Teraz zajmiesz się przekształceniem ikony kosza na śmieci w obszar, w którym będzie można coś upuścić.

10. **Poniżej wywołania funkcji `draggable()` utwórz nowy wiersz i dodaj do niego poniższy kod:**

```
$('#trashcan').droppable();
```

To wywołanie przekształca ikonę kosza na śmieci w widżet Droppable. Jeśli zapiszesz stronę i wyświetlisz ją w przeglądarce, zauważysz, że można już upuszczać miniaturki na ikonie kosza. Jednak upuszczone na nim miniaturki nie wracają na swoje początkowe położenie, gdyż kosz na śmieci jest obecnie prawidłowym widżetem Droppable.

Aby strona wyglądała jeszcze atrakcyjniej, zadbasz o wyróżnienie ikony kosza, gdy użytkownik zacznie przeciągać jakąś miniaturkę. Konkretnie rzecz biorąc, sprawisz, że kosz stanie się nieco jaśniejszy, co będzie stanowiło wizualny sygnał informujący, że można na nim upuścić miniaturkę.

11. **W wywołaniu funkcji `droppable()` dodaj literał obiektowy z opcjami:**

```
$('#trashcan').droppable({
  activeClass : 'highlight'
});
```

Opcja `activeClass` (patrz strona 436) sprawia, że kiedy tylko użytkownik zacznie przeciągać jakiś element, który można w danym widżecie upuścić, do widżetu Droppable zostanie dodana wskazana nazwa klasy. W tym przykładzie dodawana jest klasa `highlight`, zdefiniowana w pliku *interactions.css*. Zmienia ona wartość właściwości `opacity` kosza na śmieci na `1` (czyli element ten będzie całkowicie nieprzezroczysty). Inna reguła umieszczona w tym samym pliku sprawia, że początkowa wartość właściwości `opacity` ikony kosza na śmieci będzie wynosić `.6` (czyli będzie nieprzezroczysta w 60%). Dzięki zmianie nieprzezroczystości z `.6` na `1` po rozpoczęciu przeciągania ikona kosza na śmieci będzie się wydawać jaśniejsza.

Choć ta niewielka zmiana wygląda bardzo atrakcyjnie i stanowi wizualną podpowiedź dla użytkowników, to jednak nie zrobiłeś jeszcze nic by zapewnić, że po upuszczeniu miniaturki na ikonie kosza coś się stanie. Aby to zrobić, musisz skorzystać ze zdarzenia `drop` widżetu Droppable.

12. **Za łańcuchem `'highlight'` wpisz przecinek, naciśnij klawisz *Enter* i dodaj funkcję obsługującą zdarzenia `drop`:**

```
$('#trashcan').droppable({
  activeClass : 'highlight',
  drop : function (event, ui) {

  }
});
```

Wartością opcji `drop` musi być funkcja, tak samo jak w przypadku zdarzeń `start` i `stop` widżetów Draggable. Funkcja widoczna powyżej jest pusta, jednak możesz sprawić, by coś robiła, na przykład odtworzyła jakiś atrakcyjny efekt wizualny jQuery UI!

13. **Wewnątrz funkcji `drop` dodaj przedstawiony poniżej wiersz kodu (wyróżniony pogrubioną czcionką):**

```
$('#trashcan').droppable({
  activeClass : 'highlight',
  drop : function (event, ui) {
    ui.helper.hide('explode');
  }
});
```

Właściwość `ui.helper` odwołuje się do przeciąganego elementu (patrz strona 432). Metody `hide()` używałeś już wcześniej w tej książce (na stronie 212), jednak jQuery UI udostępnia dodatkowe efekty wizualne pozwalające na ukrywanie i wyświetlanie elementów. Opcja `explode` tworzy zabawny, animowany efekt — to wesoły sposób pozwalający na ukrycie elementu strony. (Więcej informacji o tych efektach można znaleźć na stronie 461).

Została Ci do zrobienia jeszcze jedna rzecz: po upuszczeniu miniaturki zdjęcia na koszu musisz zmienić jego ikonę i wyświetlić kosz, który jest pełny.

14. **Wewnątrz funkcji `drop` dodaj jeszcze jeden wiersz kodu:**

```
$('#trashcan').droppable({
  activeClass : 'highlight',
  drop : function (event, ui) {
    ui.helper.hide('explode');
    $(this).attr('src','../_images/trashcan-full-icon.png');
  }
});
```

Wyrażenie `$(this)` odwołuje się do kosza na śmieci — znacznika `<img>` o identyfikatorze `trashcan`. Całe to wywołanie zmienia wartość właściwości `src` tego znacznika i zapisuje w nim adres nowego obrazka. (W rozdziale 7. dowiedziałeś się, jak to robić).

Dokończony kod JavaScript używany w tym przykładzie powinien mieć następującą postać:

```
$(document).ready(function() {
  $('#photos img').draggable({
    revert : 'invalid',
    start : function (event, ui) {
      ui.helper.css('transform', 'rotate(5deg) scale(1.5)');
    },
    stop : function (event, ui) {
      ui.helper.css('transform', 'rotate(0deg) scale(1)');
    },
    zIndex : 100
  });
  $('#trashcan').droppable({
    activeClass : 'highlight',
    drop : function (event, ui) {
      ui.helper.hide('explode');
      $(this).attr('src','../_images/trashcan-full-icon.png');
    }
  });
}); // Koniec funkcji ready.
```

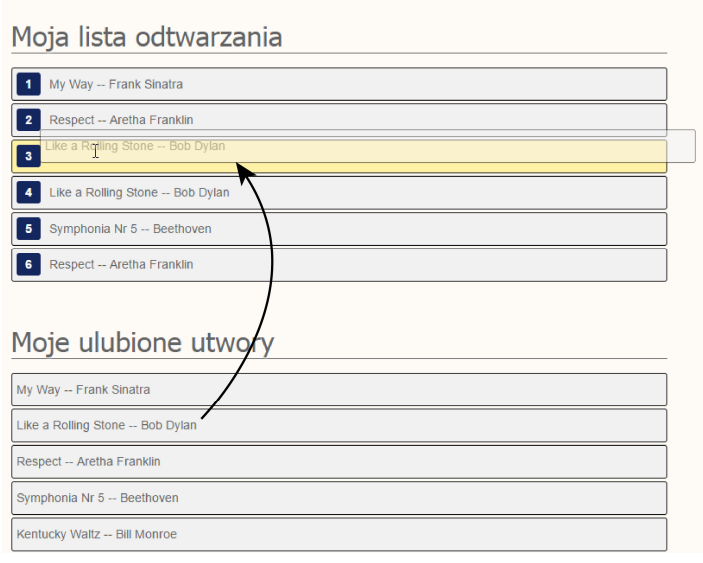
15. **Zapisz plik i wyświetl go w przeglądarce.**

Teraz możesz już przeciągać miniaturki zdjęć i upuszczać na ikonie kosza, a one będą wybuchać i znikać ze strony (patrz rysunek 12.4). Do funkcji obsługującej zdarzenia `drop` w tym przykładzie mógłbyś dodać znacznie więcej kodu. Mógłbyś skorzystać z technologii AJAX, by przesłać na serwer żądanie usunięcia wybranego zdjęcia z konta użytkownika (technologia AJAX została opisana w rozdziale 13.). Ten krótki przykład może być dla Ciebie inspiracją do wymyślenia wielu potencjalnych zastosowań widżetów Draggable i Droppable.

Uwaga: Końcowa, gotowa wersja tego przykładu jest dostępna w pliku *complete-to-the-trash.html*.

Sortowanie elementów strony

Biblioteka jQuery UI udostępnia także widżet pozwalający na tworzenie list, na przykład list zadań do zrobienia, list odtwarzania, a nawet list umieszczonych wewnątrz innych, takich jak listy plików w katalogach. Widżet Sortable zapewnia możliwość zmiany kolejności elementów na liście poprzez ich przeciągnięcie i upuszczenie w wybranym miejscu. Jest on bardzo przydatny do zarządzania, na przykład, listami piosenek — dzięki niemu użytkownik może tworzyć swoje własne listy odtwarzania, przeciągając na nie nowe piosenki, a nawet zmieniać ich kolejność po przeciągnięciu ich w inne miejsca (patrz rysunek 12.6).



Rysunek 12.6. *Widżet Sortable biblioteki jQuery UI sprawia, że zmiana kolejności elementów na liście jest banalnie prosta i szybka. Bez trudu można przeciągnąć element z jednej listy, na przykład z listy „Moje ulubione utwory" na inną, taką jak „Moja lista odtwarzania". Taki sposób interakcji jest bardzo popularny w wielu klasycznych programach komputerowych, takich jak iTunes, a nawet w programach prezentujących listy katalogów i plików*

Widżet Sortable może operować na dowolnej zgrupowanej kolekcji elementów. Choć naturalnym rozwiązaniem jest używanie go w wypunktowanych listach, jednak można go także stosować w grupach elementów `div`, akapitów i obrazów, by przekształcać je w sortowalne listy.

Stosowanie widżetu Sortable

Widżety Sortable są kolekcjami elementów, które można przeciągać w celu zmiany ich kolejności: na przykład w liście zadań do zrobienia każde z zadań można przeciągnąć i wstawić w dowolnym miejscu listy. A zatem taka lista jest widżetem Sortable, czyli elementem sortowalnym, a jej zawartość to elementy, które można sortować. Innymi słowy, widżet Sortable musi być kontenerem, takim jak lista wypunktowana (`<ul>`) lub numerowana (`<ol>`), bądź też elementem `<div>` zawierającym inne elementy, na przykład akapity, obrazy czy też inne elementy `<div>`.

Stosowanie widżetu Sortable jest proste.

1. **Wykonaj czynności opisane na stronie 329, aby dodać do strony niezbędne pliki CSS i JavaScript.**

   Pamiętaj, że biblioteka jQuery UI ma własne pliki CSS oraz JavaScript i należy je dołączyć do strony *za* plikiem JavaScript biblioteki jQuery.

2. **Umieść na stronie element kontenera.**

   Może to być lista wypunktowana lub znacznik `<div>`:

   ```
   <ul id="playlist">
   </ul>
   ```

   Dobrym pomysłem będzie dodanie do tego znacznika identyfikatora lub nazwy klasy, aby można było łatwo odwołać się do niego przy użyciu jQuery.

3. **Wewnątrz elementu kontenera dodaj elementy listy.**

   Dla list wypunktowanych i numerowanych byłaby to grupa znaczników `<li>`. Elementy tego nadrzędnego kontenera reprezentują obiekty, które będzie można przeciągać na stronie i upuszczać w innych miejscach listy. Jeśli elementem kontenera będzie znacznik `<div>`, wewnątrz niego można będzie umieszczać akapity, obrazy lub nawet kolejne znaczniki `<div>` (będą one tworzyć zawartość widżetu, którą będzie można sortować).

   ```
   <ul id="playlist">
     <li>My Way -- Frank Sinatra</li>
     <li>Like a Rolling Stone -- Bob Dylan</li>
     <li>Respect -- Aretha Franklin</li>
   </ul>
   ```

4. **Na stronie lub w zewnętrznym pliku JavaScript umieść wywołanie funkcji `$(document).ready()`:**

   ```
   $(document).ready(function() {

   }); // Koniec funkcji ready.
   ```

   Zgodnie z informacjami podanymi na stronie 190, krok ten jest niezbędny wyłącznie w przypadku, gdy kod JavaScript umieszczamy w sekcji `<head>` strony, przed właściwym kodem HTML stanowiącym jej treść. Niektórzy programiści umieszczają swój kod JavaScript na końcu strony, bezpośrednio przed zamykającym znacznikiem `</body>`; w takim przypadku umieszczanie kodu w wywołaniu funkcji `$(document).ready()` nie jest konieczne.

5. **Skorzystaj z możliwości jQuery, by wybrać element dodany do strony w kroku 2.:**

   ```
   $(document).ready(function() {
     $('#playlist').sortable();
   }); // Koniec funkcji ready.
   ```

   Powyższy kod wybiera element kontenera (znacznik `<ul>` reprezentowany przez kod dodany w kroku 3.) i przekształca w widżet Sortable, w którym możemy zmieniać kolejność elementów. Postać i sposób działania widżetu można modyfikować na wiele różnych sposobów, opisanych dalej w tym podrozdziale.

6. **W wywołaniu funkcji `sortable()` dodaj obiekt z opcjami:**

   ```
   $(document).ready(function() {
     $('#playlist').sortable({
       opacity : 0.5,
   ```

```
      placeholder : 'ui-state-hightlight'
    });
});  // Koniec funkcji ready.
```

Opcje widżetu Sortable poznasz już niebawem, jednak ich wybrane możliwości możesz ustalić, analizując powyższy fragment kodu. Pierwsza z zastosowanych opcji, `opacity`, nakazuje, by jQuery UI zmieniła nieprzezroczystość elementu podczas jego przeciągania. Z kolei druga opcja, `placeholder`, nakazuje zastosowanie podanego stylu do elementu reprezentującego puste miejsce, w którym można upuścić przeciągany element.

Uwaga: Dokładniej wypróbujesz działanie widżetu Sortable w przykładowej aplikacji, którą napiszesz w rozdziale 14.

Opcje widżetu Sortable

Widżet Sortable został wyposażony w wiele opcji. Można określać kierunek i odległość, na jaką mogą być przesuwane elementy, sposób, w jaki się mają zachowywać podczas przeciągania, a nawet to, za który fragment trzeba je przeciągać. Podobnie jak w innych widżetach jQuery UI, także i w tym opcje należy podawać w obiekcie przekazywanym w wywołaniu funkcji `sortable()`. Aby na przykład zmienić wygląd wskaźnika myszy na dłoń z wyciągniętym palcem wskazującym i określić, że elementy mogą być przeciągane wyłącznie za umieszczony wewnątrz nich nagłówek `<h2>`, należałoby użyć następującego wywołania:

```
$('#playlist).sortable({
  cursor : 'pointer',
  handle : 'h2'
});
```

Poniżej przedstawione zostały najczęściej używane opcje widżetu Sortable.

- **`axis`**. Przy użyciu tej właściwości można ograniczyć możliwości przesuwania elementów umieszczonych w widżecie wyłącznie do ruchu w pionie lub poziomie. Załóżmy, że utworzyliśmy poziomą grupę elementów `div`, których kolejność użytkownik może zmieniać w ramach gry (patrz rysunek 12.8). Możemy ograniczyć ruch tych elementów wyłącznie do przesuwania w prawo i lewo. Właściwość ta może przyjmować dwie wartości: `'x'` (element będzie można przesuwać wyłącznie w poziomie) lub `'y'` (element będzie można przesuwać wyłącznie w pionie):

  ```
  axis : 'x'
  ```

- **`cancel`**. Opcja umożliwia zabronienie przesuwania, jeśli kliknięty zostanie określony element. Załóżmy, że na stronie jest umieszczona (wypunktowana) lista utworów muzycznych. Obok nazwy każdego z utworów jest wyświetlona ikona kosza na śmieci; użytkownik może kliknąć tę ikonę, aby usunąć utwór z listy. Jeśli jednak użytkownik umieści na tej ikonie wskaźnik myszy i wciśnie jej przycisk, będzie mógł przeciągnąć utwór w inne miejsce listy. Aby wskazać konkretny element umieszczony wewnątrz elementów widżetu Sortable, którego nie będzie można użyć jako uchwytu do przeciągania, należy podać jego selektor w opcji `cancel`:

  ```
  cancel : '.trashicon'
  ```

Jeśli teraz użytkownik kliknie ikonę kosza (przy założeniu, że została w niej użyta klasa `trashicon`), nie będzie mógł przeciągnąć elementu. W tej opcji można także podać więcej niż jeden element — wystarczy rozdzielić ich selektory przecinkami:

```
cancel : '.trashicon, .addToFavorites'
```

Opcja `handle` opisana na stronie 453 pozwala określić konkretny element, który ma pełnić rolę „uchwytu" do przeciągania.

- **`cancelWith`**. Umożliwia określenie selektorów *innych* widżetów Sortable, czyli innych list, w których będzie można umieszczać elementy z tego widżetu. Załóżmy, że na stronie znajdują się dwie listy: lista życzeń zawierająca produkty, które chcielibyśmy kupić, oraz lista koszyka z zakupami zawierająca wszystkie produkty, które mamy zamiar kupić. Użytkownik powinien mieć możliwość zmiany kolejności produktów na liście życzeń — aby na przykład na jej początku umieścić produkt, na którym najbardziej mu zależy — lecz także przeciągania elementów z listy życzeń do koszyka. W tym przypadku na stronie powinniśmy umieścić dwie listy i osobno wywołać funkcję `sortable()` dla każdej z nich. Jednak dzięki zastosowaniu opcji `connectWith` możemy zapewnić użytkownikom możliwość przeciągania elementów z jednej listy do drugiej.

  ```
  $('#wishList').sortable({
    connectWith : '#shoppingCart'
  });
  ```

  Opcja `connectWith` określa połączenie jednokierunkowe. Innymi słowy, powyższy kod pozwoli użytkownikowi przeciągać produkty z listy życzeń do koszyka, lecz nie z koszyka na listę życzeń. Aby zdefiniować dwukierunkowe połączenie między oboma listami, musielibyśmy dodać opcję `connectWith` także do drugiego widżetu Sortable:

  ```
  $('#shoppingCart').sortable({
  connectWith : '#wishList'
  });
  ```

  Wartością opcji `connectWith` powinien być selektor odpowiadający elementowi, na rzecz którego została wywołana funkcja `sortable()`.

- **`containment`**. Można także uniemożliwić użytkownikom przeciąganie elementów listy poza obszar kontenera, w którym się znajdują. Opcja ta działa dokładnie tak samo jak analogiczna opcja widżetu Draggable (patrz strona 425). W praktyce okazuje się, że elementy widżetu Sortable same są widżetami Draggable, które można umieszczać wewnątrz elementów listy. Opcja `containment` może mieć kilka wartości.

  - **Selektor**. Jeśli wartością opcji będzie selektor, jQuery UI zadba o to, by elementy listy mogły znajdować się wyłącznie w obszarze wskazanego elementu. Gdyby na przykład na stronie znajdował się znacznik `<div>` o identyfikatorze `mainContent`, moglibyśmy zażądać, by podczas przeciągania elementy listy mogły się znajdować wyłącznie w jego obszarze, przy użyciu następującej właściwości:

    ```
    containment : '#mainContent'
    ```

- **Wartości `parent`, `document` lub `window`**. Aby element listy musiał się znajdować wewnątrz elementu jego rodzica, należy przypisać właściwości `containment` wartość `parent`. Jeśli na przykład elementy widżetu są umieszczone wewnątrz wypunktowanej listy i chcemy, by podczas przeciągania mogły się znajdować wyłącznie w jej obszarze, powinniśmy użyć następującej właściwości:

  ```
  containment : 'parent'
  ```

  Wartości `document` oraz `window` działają niemal identycznie, przy czym pierwsza z nich sprawia, że obszar, w którym będzie można przeciągać elementy, będzie odpowiadał obszarowi dokumentu. Z kolei w przypadku zastosowania wartości `window` fragmenty przeciąganego elementu będą mogły wychodzić poza okno przeglądarki (co nie wygląda najlepiej, więc przed użyciem tej opcji lepiej się dwa razy zastanowić).

- **`cursor`**. Działa tak samo jak analogiczna opcja widżetu Draggable (patrz strona 425).
- **`cursorAt`**. Działa tak samo jak analogiczna opcja widżetu Draggable (patrz strona 426).
- **`delay`**. Liczba milisekund, o którą należy opóźnić rozpoczęcie przeciągania elementu. To opóźnienie może się przydać, gdy dojdziemy do wniosku, że bardzo łatwo można rozpocząć przeciąganie elementu przypadkowo, podczas przesuwania wskaźnika myszy po stronie. Opóźnienie to określa, jak długo użytkownik będzie musiał trzymać wciśnięty przycisk myszy, zanim będzie można rozpocząć przeciąganie elementu.

  ```
  delay : 100
  ```

- **`distance`**. Wyrażona w pikselach odległość określająca, jak daleko trzeba będzie przeciągnąć element listy, zanim zacznie być wstawiany w innych miejscach. Opcja ta jest przydatna, gdy użytkownik może klikać elementy widżetu w innych celach niż przeciągnięcie. Jeśli na przykład element listy zawiera przycisk służący do jego usunięcia, moglibyśmy użyć opcji `distance`, by uniemożliwić przypadkowe przeciągnięcie elementu, gdy użytkownik będzie chciał kliknąć przycisk. Wartości przypisywane tej opcji powinny być stosunkowo małe (inaczej użytkownik będzie musiał daleko przeciągnąć element, zanim zauważy, że można zmieniać jego położenie na liście):

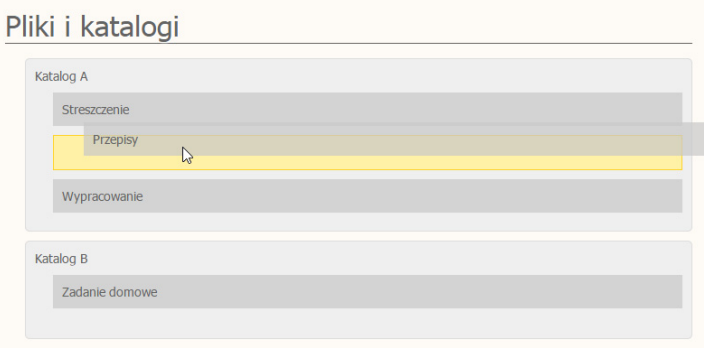
  ```
  distance : 10
  ```

- **`grid`**. Opcja sprawia, że elementy listy będą przyciągane do punktów siatki. Ustawienie to może działać bardzo dobrze, jeśli na stronie będą się znajdować obrazki lub elementy `div` o tej samej wielkości: w takim przypadku elementy te mogą być przyciągane do punktów oddalonych od siebie o szerokość elementu. Opcja ta działa tak samo jak analogiczna opcja widżetu Draggable (patrz strona 426).
- **`handle`**. Opcja określa fragment elementu, który użytkownik musi kliknąć, by przeciągnąć element. Normalnie użytkownik może przeciągać element listy, klikając go w dowolnym miejscu. Może się jednak zdarzyć, że będziemy chcieli zmusić użytkownika do kliknięcia w konkretnym miejscu, na przykład na nagłówku. Opcja ta działa tak samo jak analogiczna opcja widżetów Draggable (opisana na stronie 427).

- **`items`**. Opcja pozwala określić, które elementy listy będzie można przeciągać. Załóżmy, że przekształciliśmy w widżet Sortable listę hierarchiczną, czyli listę zawierającą inne listy (patrz rysunek 12.7). Nie chcemy jednak, by można było zmieniać kolejność listy najwyższego poziomu (w przypadku listy z rysunku 12.7 są to elementy *Katalog A* i *Katalog B*). Chcemy natomiast, by można było przeciągać elementy list zagnieżdżonych — przeciągać je do innych list lub zmieniać kolejność w ramach tej samej listy zagnieżdżonej.

  W takim przypadku moglibyśmy przypisać opcji `items` wartość `'li li'`, informując tym samym, że przeciągać będzie można wyłącznie elementy listy umieszczone wewnątrz elementów innej listy (czyli elementy list zagnieżdżonych). Elementy listy najwyższego poziomu nie są zagnieżdżone, więc nie będzie można ich przeciągać:

```
items : 'li li'
```



Rysunek 12.7. *Listy zagnieżdżone (czyli listy umieszczone wewnątrz innych list) można tworzyć, dodając znacznik listy wypunktowanej (<ul>) lub numerowanej (<ol>) wewnątrz elementu innej listy: Katalog A
Lista zagnieżdżona . Biblioteka jQuery UI umożliwia określenie, które elementy widżetu Sortable będzie można przeciągać i zmieniać ich kolejność; na przykład operacje te można ograniczyć wyłącznie do elementów list zagnieżdżonych*

- **`opacity`**. Podczas przeciągania elementu można zmienić jego nieprzezroczystość. Na przykład można sprawić, że element stanie się półprzezroczysty i będzie przypominał cień poruszający się po stronie. Taka zmiana jest często stosowanym wizualnym sposobem sygnalizowania użytkownikom, że element jest przeciągany z jednego miejsca w inne. Opcji tej można przypisywać wartości z zakresu od `0` (element jest zupełnie niewidoczny) do `1` (element jest całkowicie nieprzezroczysty). Działa ona tak samo jak właściwość `opacity` arkuszy stylów. Aby na przykład przeciągany element był półprzezroczysty, należy tej opcji przypisać wartość `0.5`:

```
opacity : 0.5
```

- **`placeholder`**. Opcja określa nazwę klasy, którą jQuery UI zastosuje w pustym elemencie listy, w którym zostanie umieszczony przeciągany element. Miejsce to można wyróżnić, stosując na przykład klasy zdefiniowane przez bibliotekę jQuery UI (przedstawione na stronie 418):

```
placeholder : 'ui-style-hightlight'
```

Uwaga: Informacje o wszystkich opcjach widżetu Sortable można znaleźć na stronie *http://api.jqueryui.com/sortable/*.

Zdarzenia widżetu Sortable

Kiedy użytkownik prowadzi interakcję z widżetem Sortable, jQuery UI generuje bardzo wiele różnego rodzaju zdarzeń; są one zgłaszane, kiedy na przykład użytkownik zacznie przeciągać jeden z elementów listy bądź go upuści. Biblioteka jQuery UI zgłasza ponad dziesięć różnych zdarzeń, a każde z nich można obsługiwać. Niektóre są zgłaszane w tak krótkich odstępach czasu, że można uznać, iż zachodzą jednocześnie.

Podobnie jak było w przypadku widżetów Draggable i Droppable, także i zdarzenia widżetu Sortable są obsługiwane za pomocą dodania funkcji do obiektu przekazywanego w wywołaniu funkcji `sortable()`. Przykładowo poniższy fragment kodu tworzy widżet Sortable, który po przeciągnięciu elementu będzie wyświetlać okno informacyjne z komunikatem:

```
$('#playList').sortable({
  stop : function (event, ui) {
    alert('Tak oto lista została uporządkowana!');
  }
});
```

Niektóre zdarzenia odnoszą się do wszystkich list, choć istnieją także dwa takie, które są zgłaszane wyłącznie wtedy, gdy na stronie są umieszczone dwa (lub więcej) widżety Sortable, a użytkownik przeciąga elementy między nimi. Przedstawione zostaną teraz zdarzenia dotyczące wszystkich widżetów Sortable. Są one zgłaszane w określonym porządku i w takiej samej kolejności zostały opisane.

- **`create`**. Zdarzenie jest zgłaszane za każdym razem, gdy użyjemy funkcji `sortable()` w celu utworzenia nowej sortowalnej listy. Można z niego skorzystać, aby na przykład wyświetlić okno dialogowe z instrukcjami typu: „Możesz przeciągać utwory na liście, by zmienić ich kolejność". Jest ono wywoływane tylko jeden raz — podczas tworzenia widżetu Sortable.

Uwaga: W odróżnieniu od wszystkich innych zdarzeń widżetu Sortable, do funkcji obsługującej zdarzenie `create` nie jest przekazywany parametr `ui` (opisany na stronie 432).

- **`start`**. Zdarzenie jest zgłaszane, jak tylko użytkownik zacznie przeciągać element listy. Do funkcji obsługującej te zdarzenia przekazywane są dwa argumenty; pierwszym z nich jest obiekt `event` (patrz strona 194), a drugim — obiekt `ui` zawierający informacje o widżecie. Obiekt `ui` zawiera siedem innych obiektów, z których każdy udostępnia ważne informacje na temat widżetu.
  - **`ui.helper`**. Właściwość `helper` jest obiektem jQuery reprezentującym element przeciągany w oknie przeglądarki. Biblioteka jQuery UI tworzy kopię właściwego elementu listy, zatem podczas przeciągania istnieją w kodzie HTML dwa różne elementy: pomocnicza kopia oraz element właściwy. Kiedy użytkownik zakończy przeciąganie, kopia elementu jest usuwana z dokumentu. Ponieważ jest to normalny obiekt jQuery, zatem można go używać do wywoływania wszystkich metod tej biblioteki, takich jak `css()`, `animate()` czy też `find()`.
  - **`ui.item`**. Właściwość zawiera obiekt reprezentujący element widżetu, który użytkownik klika, by rozpocząć przeciąganie, na przykład element `<li>`.

Jest to faktyczny element HTML, który później zostanie umieszczony w odpowiednim miejscu listy, kiedy użytkownik zakończy przeciąganie elementu pomocniczego. Także ta właściwość zawiera obiekt jQuery, którego można używać do wywoływania metod jQuery.

- **`ui.position`**. Właściwość zawiera współrzędne górnego lewego wierzchołka elementu pomocniczego (czyli tego, który jest wizualnie przeciągany w oknie przeglądarki), wyznaczone względem najbliższego, umiejscowionego elementu przodka. Jeśli widżet Sortable jest umieszczony wewnątrz jakiegoś elementu umiejscowionego (względnie lub bezwzględnie), właściwość `ui.position` będzie zawierać współrzędne górnego lewego wierzchołka elementu pomocniczego wyrażone względem górnego lewego wierzchołka elementu umiejscowionego.

  Do samych współrzędnych można się odwoływać, używając wyrażeń `ui.position.top` oraz `ui.position.left`.

- **`ui.originalPosition`**. Określa początkowe położenie elementu listy — czyli miejsce, w którym się znajdował, zanim użytkownik zaczął go przeciągać. Podobnie jak w przypadku właściwości `ui.position`, także ten obiekt zawiera dwie właściwości — `top` oraz `left`.

- **`ui.offset`**. Wartością tej właściwości jest obiekt zawierający dwie właściwości — `top` i `left`. Jednak w tym przypadku współrzędne są wyznaczane względem górnego lewego wierzchołka okna przeglądarki. Właściwość `ui.offset.top` określa, jak daleko poniżej górnej krawędzi okna przeglądarki znajduje się element pomocniczy.

  Z kolei właściwość `ui.offset.left` określa odległość, wyrażoną w pikselach, pomiędzy lewą krawędzią okna przeglądarki a przeciąganym elementem pomocniczym.

- **`ui.sender`**. Właściwość jest stosowana wyłącznie podczas przeciągania elementu z jednego widżetu Sortable do drugiego. Zawiera obiekt jQuery, reprezentujący widżet, w którym początkowo znajdował się przeciągany element.

- **`placeholder`**. Właściwość zawiera obiekt jQuery reprezentujący pusty obszar tworzony na liście podczas przeciągania elementu.

  Obiekt `ui` jest przekazywany także do wszystkich innych funkcji obsługi zdarzeń widżetów Sortable (takich jak `activate`, `over`, `sort` i tak dalej) z wyjątkiem zdarzeń `create`.

- **`activate`**. Zdarzenie jest zgłaszane bezpośrednio po zdarzeniu `start`, przy czym praktycznie następuje to w tym samym momencie. Można je wykorzystać, jeśli chcemy dodać drugą funkcję, która będzie wykonywana bezpośrednio po funkcji obsługującej zdarzenie `start`.

- **`sort`**. Zdarzenie `sort` jest zgłaszane za każdym razem, gdy podczas przeciągania elementu listy zostanie przesunięty wskaźnik myszy. Innymi słowy, jest ono zgłaszane nieustannie, dlatego w ramach jego obsługi nie należy wykonywać żadnych czasochłonnych operacji ani operacji, które mogą znacząco obciążać procesor. W przeciwnym razie ciągłe wywoływanie tej funkcji może pogorszyć szybkość działania strony, a także utrudnić przeciąganie elementów listy.

- **change**. Zdarzenie jest zgłaszane, jak tylko przeciągany element zmieni miejsce na liście. Kiedy na przykład przeciągamy element z samego początku listy w dół, element, który wcześniej był drugi, wskoczy na pierwsze miejsce i wygeneruje tym samym zdarzenie `change`. Można go użyć przykładowo do wyróżnienia dwóch elementów listy, których położenie zostało zmienione.
- **beforestop**. Zdarzenie jest zgłaszane bezpośrednio przed zakończeniem operacji przeciągania elementu listy. Jest ostatnim zdarzeniem, które ma dostęp do elementu pomocniczego — `ui.helper`. Po zakończeniu jego obsługi element pomocniczy (czyli kopia faktycznego elementu listy) jest usuwany.
- **update**. Zdarzenie jest zgłaszane, kiedy wszystkie elementy widżetu Sortable są już na miejscu i DOM strony został zaktualizowany.
- **deactivate**. Zdarzenie jest zgłaszane po zakończeniu przeciągania, bezpośrednio po zdarzeniu `update`.
- **stop**. Kiedy element listy zostanie upuszczony na miejsce, jest zgłaszane zdarzenie `stop`. Jest to ostatnie zdarzenie w sekwencji i zawsze następuje po zdarzeniach `beforestop` i `deactivate`. Funkcję obsługi tego zdarzenia można określać w przypadku, gdy chcemy mieć pewność, że będzie to ostatnia czynność wykonana po upuszczeniu elementu listy. Można go na przykład użyć, by sprawdzić status listy i upewnić się, że jej elementy znajdują się w pewnej, predefiniowanej kolejności (patrz rysunek 12.7).

Powyższe zdarzenia są zgłaszane w kolejności, w jakiej zostały opisane. Istnieją jednak inne zdarzenia, które są zgłaszane w innych momentach i dla innych typów widżetów Sortable.

- **out.** Zdarzenie `out` zostaje zgłoszone, kiedy element widżetu Sortable zostanie przeciągnięty poza jego obszar; kiedy na przykład użytkownik przeciągnie element listy poza nią, na pusty obszar strony. Jest ono zgłaszane także w przypadku, gdy element zostanie przeciągnięty poza obszar jednego widżetu Sortable i umieszczony w obszarze innego widżetu Sortable.
- **over.** Zdarzenie jest zgłaszane, kiedy element zostanie przesunięty i umieszczony w obszarze listy powiązanej z danym widżetem Sortable. Jeśli na przykład na stronie są umieszczone dwie listy — dwa widżety Sortable — i przeciągniemy element jednej z nich w obszar drugiej, zostanie zgłoszone zdarzenie `over`. Jest ono zgłaszane także w przypadku, gdy przesuniemy element zupełnie poza obszar widżetu, a następnie ponownie go w nim umieścimy.
- **receive.** Zdarzenia `receive` można używać, kiedy na stronie umieszczono kilka powiązanych ze sobą widżetów Sortable (patrz właściwość `connectWith`, opisana na stronie 451). Gdy do jednego z widżetów zostanie przeciągnięty element umieszczony wcześniej na innej liście, zostanie zgłoszone zdarzenie `receive`. Można go używać w połączeniu z obiektem `ui.sender` (patrz strona 456), by określić, skąd pochodzi nowy element listy.
- **remove.** Zdarzenie jest zgłaszane, kiedy element zostanie usunięty z widżetu Sortable. Załóżmy, że na stronie znajdują się dwa widżety Sortable — lista życzeń oraz koszyk z zakupami. Jeśli użytkownik przeciągnie produkt z listy życzeń do koszyka, w widżecie listy życzeń zostanie zgłoszone zdarzenie `remove` (jak również zdarzenie `out`). (W tym przypadku w widżecie koszyka z zakupami zostaną natomiast zgłoszone zdarzenia `over` i `receive`).

Metody widżetów Sortable

Widżet Sortable udostępnia kilka metod, czyli funkcji, które można na jego rzecz wywoływać. Nie są one jednak wywoływane w taki sam sposób, jak zwyczajne metody biblioteki jQuery, czyli poprzez wybranie odpowiedniego elementu strony, a następnie zastosowanie na nim jakiejś metody, na przykład `$('body').hide()`. Zamiast tego nazwę metody podaje się w formie łańcucha znaków i przekazuje w wywołaniu funkcji `sortable()`. Załóżmy, że chcemy wywołać metodę `destroy`, która usuwa widżet ze strony — metoda ta zwraca elementy stanowiące zawartość widżetu jako zwyczajne elementy listy, których nie można przeciągać. Poniżej przedstawiony został kod pozwalający na wywołanie tej metody:

```
$('#sortableItems').sortable('destroy');
```

Selektor zastosowany w tym przykładzie — `#sortableItems` — powinien odpowiadać selektorowi widżetu Sortable umieszczonego na stronie. Metody te są bardzo często używane w odpowiedzi na zdarzenia generowane przez widżet, opisane w poprzednim punkcie rozdziału. Jeśli na przykład utworzymy grę, która wymaga od użytkownika umieszczenia grupy elementów w określonej kolejności, moglibyśmy usuwać widżet Sortable, kiedy użytkownik skończy grę, i oszczędzić mu tym samym konieczności ponownego ustawiania elementów.

Pełną listę metod widżetu Sortable można znaleźć na stronie *http://api.jqueryui.com/sortable/*, jednak poniżej opisanych zostało kilka najbardziej przydatnych.

- **`cancel`.** Wywołanie tej metody anuluje jakiekolwiek zmiany wprowadzone w kolejności elementów listy. Innymi słowy, przerywa ona sortowanie listy. Można jej używać na przykład wraz ze zdarzeniem `receive` (opisanym na stronie 457), aby odrzucić element przeciągnięty z innego widżetu Sortable. Ewentualnie, w ramach obsługi zdarzenia `stop` (patrz strona 457) można zastosować funkcję, która będzie sprawdzać, w którym miejscu został umieszczony upuszczony element listy, i anulować całą operację, jeśli nie zostaną spełnione określone kryteria. Przykładowo w aplikacji do zarządzania listą zadań mogą się pojawić zadania zależne od innych zadań. Jeśli użytkownik spróbuje przeciągnąć jakieś zadanie na sam początek listy, lecz przed nim musi zostać wykonane inne, aplikacja może anulować operację i wyświetlić okno dialogowe z komunikatem tłumaczącym problem.
- **`destroy`.** Metoda pozwala całkowicie usunąć ze strony możliwości funkcjonalne widżetu Sortable.
- **`disable`.** Metoda pozwala tymczasowo zablokować możliwości funkcjonalne widżetu Sortable. Można jej używać, aby tymczasowo uniemożliwić użytkownikowi sortowanie elementów listy, aż do momentu, kiedy zostanie spełniony określony warunek. Pełne możliwości funkcjonalne widżetu można ponownie włączyć, wywołując metodę `enable` (opisaną poniżej).
- **`enable`.** Ta metoda ponownie włącza wyłączony wcześniej widżet Sortable — czyli widżet, w którym wywołano metodę `disable`.
- **`serialize`.** Metoda pozwala przesyłać uporządkowane elementy listy na serwer, przy wykorzystaniu technologii AJAX bądź zwykłego formularza HTML. Można z niej korzystać w przypadkach, gdy zapisanie kolejności elementów

listy na serwerze jest niezbędne w celu ich późniejszego odtworzenia. Gdyby na przykład gdyby menadżer porządkował listę zadań dla swoich pracowników, ich kolejność można by przesłać na serwer, zapisać w bazie danych, a następnie odtwarzać, gdy do aplikacji zalogują się pracownicy chcący sprawdzić, jakie prace należy wykonać (i w jakiej kolejności). Aby skorzystać z tej możliwości, elementy listy należy przygotować w odpowiedni sposób.

- Każdy element listy musi mieć identyfikator.
- Każdy identyfikator musi się zaczynać od tego samego słowa, które będzie pełnić rolę identyfikatora grupowego dla danej listy; za tym słowem należy umieścić znak podkreślenia (_).
- Za znakiem podkreślenia należy podać unikalny identyfikator danego elementu listy.

Przykładowo załóżmy, że na stronie jest umieszczona lista utworów muzycznych. Jej kod HTML mógłby wyglądać w następujący sposób:

```
<ul id="playlist">
  <li id="song_1">My Way -- Frank Sinatra</li>
  <li id="song_2">Like a Rolling Stone -- Bob Dylan</li>
  <li id="song_3">Respect -- Aretha Franklin</li>
</ul>
```

Zwróć uwagę, że każdy z elementów listy rozpoczyna się od łańcucha znaków `song_` , po którym został umieszczony unikalny identyfikator utworu z bazy danych. Chodzi o to, by przekazać na serwer informacje o nowej kolejności elementów na liście.

Poniższe wywołanie pokazuje, w jaki sposób można pobrać informacje o kolejności elementów na liście:

```
var listOrder = $('#playList').sortable('serialize');
```

Wywołanie metody `serialize` zwraca łańcuch znaków o następującej, przykładowej postaci:

```
song[]=2&song[]=3&song[]=1
```

Ten łańcuch określa kolejność elementów listy. W powyższym przykładzie liczby `2`, `3` i `1` oznaczają, że pierwszy element listy został przeciągnięty na jej koniec. Taki łańcuch znaków można dołączyć do adresu URL i przesłać na serwer bądź też przekazać go przy użyciu technologii AJAX (opisanej w rozdziale 13.).

Uwaga: Metoda `serialize` udostępnia opcje pozwalające określać format zwracanego łańcucha znaków. Więcej informacji na jej temat można znaleźć na stronie *http://api.jqueryui.com/sortable/#method-serialize*.

- **`toArray`.** Metoda `toArray`, podobnie jak `serialize`, jest sposobem na uzyskanie uporządkowanej listy elementów widżetu. Aby z niej skorzystać, wszystkie elementy widżetu muszą mieć identyfikatory. Jednak, w odróżnieniu od metody `serialize`, identyfikatory te mogą być zupełnie dowolne — nie trzeba ich zapisywać w żadnym określonym formacie. Metoda ta zwraca tablicę zawierającą identyfikatory wszystkich elementów widżetu, zapisane w takiej kolejności, w jakiej elementy są umieszczone na liście. Przykładowo

załóżmy, że na stronie znajduje się widżet Sortable o identyfikatorze `colorList`. Wewnątrz niego są umieszczone trzy elementy, z których każdy reprezentuje inny kolor.

```
<ul id="colorList">
  <li id="red">Czerwony</li>
  <li id="green">Zielony</li>
  <li id="blue">Niebieski</li>
</ul>
```

Załóżmy teraz, że użytkownik zmienił kolejność elementów w taki sposób, że `Niebieski` jest pierwszy, `Czerwony` jest drugi, a `Zielony` — trzeci. Jeśli teraz wywołamy metodę `toArrray`, aby pobrać informacje o kolejności elementów listy:

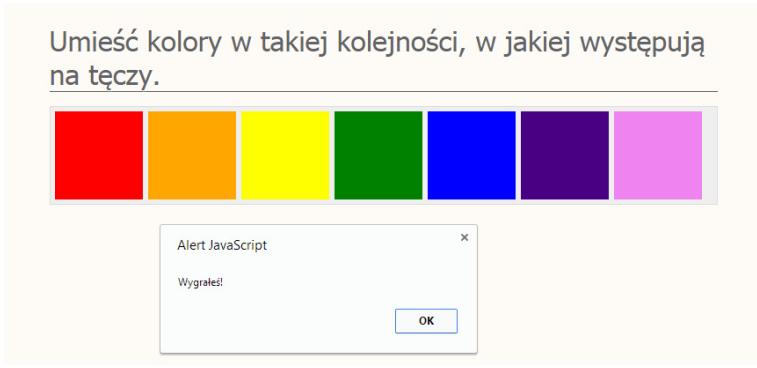
```
var colors = $('#colorList').sortable('toArray');
```

to w zmiennej `color` zostanie zapisana tablica o następującej zawartości:

```
['blue', 'red', 'green']
```

Jak widać, metoda ta zwraca zwyczajną tablicę JavaScript (patrz strona 77), na której można operować przy użyciu dowolnych metod opisanych na stronach od 78 do 83.

Jednym z potencjalnych zastosowań metody `toArray` jest sprawdzanie, czy kolejność elementów listy odpowiada jakiemuś predefiniowanemu stanowi. Załóżmy, że napisaliśmy grę, która wyświetla grupę kolorowych kwadratów w losowej kolejności. Zadaniem użytkownika jest rozmieszczenie ich w takiej kolejności, w jakiej poszczególne kolory występują w tęczy (pamiętasz? Są to kolory: czerwony, pomarańczowy, żółty, zielony, niebieski, indygo i fioletowy). W takiej aplikacji, w funkcji obsługującej zdarzenie `stop` (patrz strona 457) moglibyśmy pobierać kolejność kolorowych elementów przy użyciu metody `toArray`, a następnie porównywać z tablicą zawierającą prawidłową odpowiedź (patrz rysunek 12.8).



Rysunek 12.8. *Metoda toArray widżetu Sortable pozwala pobrać tablicę identyfikatorów elementów, zapisanych w takiej kolejności, w jakiej aktualnie występują one na liście. Można jej użyć, by porównać bieżącą kolejność elementów na liście z jakimś predefiniowanym stanem. W tej grze każde przesunięcie kolorowego kwadratu powoduje porównanie kolejności z prawidłowym wynikiem. Jeśli bieżąca kolejność odpowiada prawidłowej odpowiedzi, gracz wygrywa. Przedstawioną tu stronę — pattern-game.html — można znaleźć w przykładach dołączonych do książki, dla rozdziału R12*

Uwaga: Biblioteka jQuery UI udostępnia także jeszcze dwa inne widżety rozszerzające możliwości interakcji użytkownika ze stroną. Pierwszy z nich, widżet Resizable, jest używany przez okna dialogowe jQuery (patrz strona 330) i pozwala użytkownikom zmieniać ich wielkość poprzez przeciąganie uchwytów umieszczonych w rogach. Można go stosować, by zapewnić możliwość zmiany wielkości pływających okien. Więcej informacji na temat tego widżetu można znaleźć na stronie *http://api.jqueryui.com/resizable/*.

Widżet Selectable pozwala użytkownikom wybierać elementy (wyróżniać je) poprzez ich kliknięcie. Z powodzeniem można by go użyć na stronie do przesyłania zdjęć na serwer: „Zaznacz zdjęcia, które chcesz przesłać". Więcej informacji o tym widżecie można znaleźć na stronie *http://api.jqueryui.com/selectable/*.

Efekty jQuery UI

W skład biblioteki jQuery UI wchodzi także zestaw wizualnych, animowanych efektów, które mogą ożywiać nasze aplikacje internetowe. Przykładowo efekt `explode`, który zastosowałeś w przykładzie przedstawionym na stronie 447, powoduje, że element rozpada się na części i stopniowo zanika. Już za chwilę dowiesz się więcej o dostępnych efektach, jednak najpierw musisz nauczyć się je stosować.

Efekty są przeznaczone do wyświetlania elementów na stronie, ukrywania elementów, które są widoczne, bądź też wizualnego wyróżniania elementów — na przykład poprzez zmianę koloru lub szybkie przesuwanie tam i z powrotem, co sprawia wrażenie, jakby element się trząsł. Efekty te można stosować albo przy użyciu niektórych funkcji jQuery (jak robiliśmy w przykładach przedstawionych wcześniej w książce), albo też za pomocą wywołania specjalnej funkcji jQuery UI, czyli `effect()`. Aby na przykład zastosować efekt `drop`, w którym element wydaje się spadać na stronę, można wywołać metodę `show()` i podać w niej nazwę efektu oraz czas trwania animacji:

```
$('#pageElement').show('drop', 1000);
```

To wywołanie spowoduje „upuszczenie" elementu na stronę, przy czym odtwarzanie efektu będzie trwało jedną sekundę (1000 milisekund). Metoda `show()` powinna już wyglądać znajomo (patrz strona 212). To funkcja jQuery, która wyświetla ukryty wcześniej element. Jednak biblioteka jQuery UI wzbogaca funkcję `show()` oraz dwie inne funkcje jQuery o kilka dodatkowych możliwości. jQuery UI udostępnia cztery różne funkcje pozwalające na dodawanie efektów do elementów. Oto one.

- **`show()`.** Biblioteka jQuery UI rozszerza funkcję `show()`; pozwala wyświetlać ukryte wcześniej elementy strony na 15 różnych sposobów. Przed wywołaniem funkcji `show()` należy się upewnić, że element jest niewidoczny (patrz funkcja `hide()` opisana na stronie 212), gdyż w przeciwnym razie element najpierw szybko zniknie, a potem się pojawi.
- **`hide()`.** Aby zastosować jeden z efektów jQuery UI podczas ukrywania elementu, należy wywołać funkcję `hide()`. Działa ona podobnie do funkcji `hide()` jQuery (patrz strona 212), czyli sprawia, że element znika ze strony; używa przy tym specjalnego efektu. Przed zastosowaniem tej funkcji należy się upewnić, że element jest widoczny.

- **`toggle()`.** Funkcja `toggle()` naprzemiennie wyświetla i ukrywa element. Jeśli element jest ukryty, wywołanie funkcji `toggle()`wyświetli go, używając podanego efektu. Jeśli natomiast element jest widoczny, wywołanie `toggle()` ukryje go.
- **`effect()`.** Większość efektów jQuery UI zostało zaprojektowanych po to, by w widowiskowy sposób wyświetlać i ukrywać elementy. Istnieje także kilka efektów — `bounce`, `hightlight`, `pulsate` oraz `shake` — które wyróżniają element widoczny na stronie bez jego ukrywania. Efekt `highlight` wyróżnia element, wyświetlając w nim szybko jasny kolor tła. Stanowi to doskonały sposób zwrócenia uwagi użytkownika na konkretne miejsce strony. Funkcja `effect()` jest udostępniana przez jQuery UI, a nie przez bibliotekę jQuery.

Uwaga: Funkcji `effect()` użyjesz w przykładowej aplikacji, którą napiszesz w rozdziale 14.

Efekty

Efekty jQuery UI są użytecznymi, animowanymi narzędziami dostępnymi w naszym przyborniku. Każdy z nich można przekazać, w formie łańcucha znaków, w wywołaniu jednej z funkcji opisanych w poprzednim punkcie rozdziału. Każdy generuje także inny efekt wizualny.

Podstawowy sposób stosowania efektów jQuery UI przy użyciu jednej z wcześniej opisanych funkcji wygląda następująco:

```
$('#element').hide('nazwaEfektu', { nazwaOpcji : wartoscOpcji },
➥czasTrwania, funkcjaZwrotna);
```

Gdybyśmy chcieli ukryć element o identyfikatorze `deleteThis`, rozrywając go na 16 części, które zanikną w ciągu pół sekundy, a następnie wyświetlić komunikat „Buuum!", należałoby użyć następującego wywołania:

```
$('#deleteThis').hide('explode', { pieces : 16 }, 500, function () {
  alert('Buuum!');
});
```

Większość tych efektów pozwala na przekazywanie jednej lub kilku opcji dodatkowo kontrolujących ich działanie. Opcje te są przekazywane w formie obiektu zawierającego pary nazwa – wartość. W powyższym przykładzie taką opcją jest liczba fragmentów, na które zostanie podzielony wybuchający element:

```
{ pieces : 16 }
```

Każdy efekt posiada inne opcje, a niektóre nie mają ich w ogóle. Poniżej przedstawiona została lista efektów jQuery UI.

- **`blind`**. Efekt `blind` wyświetla bądź ukrywa element; przykładem może być opuszczanie lub podnoszenie rolety. Efekt umożliwia przekazanie jednej opcji, `direction`, określającej kierunek ruchu rolety: `up` (w górę), `down` (w dół), `left` (w lewo) lub `right` (w prawo). Opcję tę należy przekazywać w postaci obiektu — pary nazwa – wartość — jako drugi argument wywołania funkcji. A tak można użyć tego efektu do ukrycia elementu strony:

```
$('#element').hide('blind', { direction : 'left'}, 1000);
```

- **bounce**. Efekt `bounce` można wykorzystać podczas wyświetlania i ukrywania elementów. Można go także stosować przy użyciu funkcji `effect()`, aby po prostu poruszać widocznym elementem w górę i w dół; stanowi to doskonały sposób zwrócenia uwagi na element — jakby element „wołał": „Hej, popatrz na mnie!!". Efekt ten ma dwie opcje.
- **distance**. Największa odległość (wyrażona w pikselach), na jaką element zostanie odsunięty podczas odbijania. Im większa ta liczba, tym dalej element będzie się odbijać i tym bardziej będzie przyciągać uwagę użytkownika.
- **times**. Opcja określa, ile razy element zostanie odbity.

  W poniższym kodzie pokazano, w jaki sposób można sprawić, by element odbił się 20 razy na odległość 100 pikseli:

```
$('#theElement').click(function () {
  $(this).effect('bounce', {
      distance : 100,
      times : 20
    },
    1000
  );
});
```

- **clip**. Efekt `clip` polega na zastosowaniu właściwości CSS `clip`, by element pojawiał się, rozwijając się do pełnych wymiarów w pionie lub w poziomie. Efekt ten obsługuje tylko jedną opcję, `direction`, która może przyjmować jedną z dwóch wartości: `vertical` (w pionie) lub `horizontal` (w poziomie).

```
{ direction : 'horizontal' }
```

- **drop**. Efekt umożliwia wyświetlanie lub ukrywanie elementu; stopniowo zmienia jego nieprzezroczystość i jednocześnie przesuwa go w górę, dół, lewo lub prawo. W efekcie można zastosować jedną opcję, `direction`, która może przyjmować jedną z czterech wartości: `up` (w górę), `down` (w dół), `left` (w lewo) i `right` (w prawo).
- **explode**. Efekt `explode` dzieli element na określoną liczbę fragmentów, następnie animuje ich ruch na zewnątrz względem środka elementu i jednocześnie sprawia, że stopniowo zanikają. Zastosowany w funkcji `show()` spowoduje efekt eksplodującego elementu oglądanego wstecz: na początku pojawiają się fragmenty, które następnie zbliżają się do siebie, tworząc w końcu jedną całość. Efekt ten udostępnia tylko jedną opcję, `pieces`, określającą liczbę fragmentów, na które zostanie podzielony element. Jej wartości muszą być kwadratami kolejnych liczb całkowitych, czyli mają to być liczby: 1, 4, 9, 16, 25 i tak dalej (nie używaj jednak wartości większej od 25, gdyż sprawi to, że efekt będzie odtwarzany bardzo wolno):

```
{ pieces: 16 }
```

- **fade**. Efekt działa jak funkcje jQuery `fadeIn()` oraz `fadeOut()`; innymi słowy, nie jest szczególne efektowny.
- **fold**. Efekt pozwala wyświetlać i chować element poprzez jego rozkładanie i składanie. Działanie tego ciekawego efektu można modyfikować przy użyciu dwóch opcji. Pierwsza z nich, `size`, określa wielkość elementu (wyrażoną w pikselach), do której powinien zostać zmniejszony, zanim zacznie być rozwijany lub składany wzdłuż drugiej osi. Druga opcja, `horizFirst`, przyjmuje

wartość logiczną i określa, czy element najpierw ma być rozkładany lub składany wzdłuż osi pionowej (to domyślny sposób działania efektu) czy też wzdłuż osi poziomej.

```
{ size : '50%', horizFirst : true }
```

- **hightlight**. Efekt `highlight` błyskawicznie zmienia kolor tła elementu, aby przyciągnąć uwagę użytkownika. To kolejny z grupy efektów, których można używać na widocznym elemencie strony bez jego ukrywania. Kolor tła, jaki zostanie zastosowany, określany jest przy użyciu opcji `color`. Aby na przykład wyróżnić element, zmieniając jego kolor tła na czerwony na okres 1 sekundy, należałoby użyć następującego wywołania funkcji `effect()`:

```
$('#element').effect('highlight', { color : '#ff0000' }, 1000);
```

- **puff**. Efekt zmienia wielkość elementu i sprawia, że jednocześnie stopniowo zanika lub się pojawia. Udostępnia on tylko jedną opcję, `percent`, określającą (w procentach) wielkość, do jakiej element zostanie powiększony (podczas ukrywania) lub od jakiej zacznie się zmniejszać do docelowych wymiarów (podczas wyświetlania).

```
{ percent : 200 }
```

- **pulsate**. To kolejny z grupy efektów, których można używać w celu przyciągnięcia uwagi użytkownika bez ukrywania elementu. Efekt naprzemiennie ukrywa i wyświetla element. Przy użyciu opcji `times` można określić, ile razy element ma zniknąć i ponownie się pojawić. Poniżej przedstawiony został przykład użycia tego efektu:

```
$('#element').effect('pulsate', { times : 20 }, 2000);
```

- **scale**. Efekt `scale` pozwala na zmianę wielkości elementu. Można go używać zarówno w metodzie `hide()`, jak i `show()`. Zazwyczaj, kiedy wywołamy metodę `show()`, by wyświetlić ukryty element, będzie on powiększany od maleńkiego punktu aż do swoich pełnych wymiarów. Jeśli natomiast element jest widoczny na stronie i wywołamy metodę `hide()`, element będzie się stopniowo zmniejszał, aż całkowicie zniknie.

- **shake**. Efekt można zastosować przy użyciu funkcji `effect()` (patrz strona 462) na elemencie, który jest widoczny. Powoduje on „potrząśnięcie" elementem w podanym kierunku, na podaną odległość i w określonym kierunku. To kolejny z grupy efektów typu „Proszę zwrócić na mnie uwagę!", choć można go także używać do ważniejszych celów, takich jak zasygnalizowanie wystąpienia błędu; na przykład można „potrząsnąć" oknem dialogowym, jeśli użytkownik nie zaznaczył pola wyboru „Akceptuję regulamin" w formularzu rejestracyjnym. Postać tego efektu można modyfikować przy użyciu trzech opcji. Pierwsza z nich, `direction`, określa kierunek, w którym element będzie potrząsany. Może ona przyjmować wartości: `up`, `down`, `left` lub `right`. Druga opcja, `distance`, określa odległość (wyrażoną w pikselach), na jaką element zostanie przesunięty przy każdym potrząśnięciu; im większa jej wartość, bym bardziej zauważalny będzie efekt. I w końcu ostatnia opcja, `times`, określa, ile razy element ma zostać potrząśnięty. Aby na przykład potrząsnąć elementem w lewo (i dalej przesuwać go w poziomie) 10 razy na odległość 50 pikseli, należałoby użyć następujących opcji:

```
{ direction : 'left', distance : 50, times : 10 }
```

- **`size`**. Efekt zmienia wielkość elementu do podanych wymiarów. Udostępnia trzy opcje. W pierwszej z nich, `to`, należy zapisać obiekt zawierający dwie właściwości, `width` oraz `height`. Określają one, jakie mają być docelowe wymiary elementu (podczas jego ukrywania) bądź też jego wymiary początkowe (podczas wyświetlania).

  Opcja `origin` określa położenie punktu, do którego będzie przesuwany zmniejszający się element podczas ukrywania. Jej wartością jest tablica dwóch liczb; określają one odpowiednio współrzędną pionową i poziomą tego punktu.

  Ostatnia z opcji, `scale`, określa, co należy skalować, i może przyjmować następujące wartości: `both`, `box` oraz `content`. Aby na przykład jedynie zmniejszać lub zwiększać zewnętrzne pudełko elementu (jego obramowanie, tło, wysokość i szerokość), należy przypisać jej wartość `box`. W takim przypadku zawartość elementu (taka jak umieszczony w nim tekst) nie będzie skalowana.

- **`slide`**. Efekt powoduje wysunięcie elementu poza obszar strony (w przypadku jego ukrywania) lub wsunięcie na stronę (w przypadku wyświetlania). Udostępnia on jedną właściwość, `direction`, która może przyjmować wartości `left`, `right`, `up` lub `down` i określa kierunek, w jakim element będzie przesuwany.

```
$('#element').show('slide', { direction : 'right' }, 1000 );
```

Uwaga: Więcej informacji na temat efektów jQuery UI można znaleźć na stronie *http://api.jqueryui.com/category/effects/*.

Tempo animacji

Biblioteka jQuery UI udostępnia także zestaw funkcji (ang. *easing functions*) określających, jak zmienia się tempo animacji wraz z upływem czasu. Funkcje te nie mają wpływu na długość trwania animacji, lecz zmieniają szybkość odtwarzania jej poszczególnych etapów. Jeśli na przykład używamy efektu `bounce`, możemy zastosować funkcję, która sprawi, że efekt początkowo będzie odtwarzany wolno, a następnie przed zakończeniem przyspieszy.

Uwaga: Na witrynie jQuery UI dostępna jest lista wszystkich funkcji określających tempo animacji oraz przykłady pozwalające zobaczyć, jakie są efekty ich działania: *http://api.jqueryui.com/easings/*.

Aby zastosować jedną z tych funkcji, należy ją przekazać jako właściwość obiektu opcji. W tym celu trzeba użyć właściwości `easing`, której wartością powinna być nazwa wybranej funkcji, na przykład `easeInOutQuarter`, `easeInSine` i tak dalej (pełna lista dostępnych funkcji dostępna jest na stronie *http://api.jqueryui.com/easings/*).

Załóżmy, że chcemy zastosować efekt `bounce` (patrz strona 463), aby animować element za każdym razem, gdy zostanie kliknięty. Chcemy, by element odbił się 20 razy na maksymalną odległość 100 pikseli (te dwa parametry są opcjami efektu `bounce`). Chcemy także zastosować funkcję, która sprawi, że efekt będzie wyglądał bardziej realistycznie — odbijający się element zwalnia, gdy traci energię.

W poniższym przykładzie pokazano, w jaki sposób można dodać właściwość `easing` do obiektu opcji:

```
$('#theElement').click(function () {
  $(this).effect('bounce', {
      distance : 100,
      times : 20,
      easing : 'easeOutBounce'
    },
    1000
  );
});
```

Nasze możliwości nie ograniczają się do stosowania tych funkcji wyłącznie z efektami jQuery UI. Zgodnie z informacjami podanymi na stronie 221, można ich także używać z animacjami tworzonymi z pomocą jQuery UI.

Animowanie zmiany klas

Biblioteka jQuery UI udostępnia także kilka funkcji pozwalających na animowanie zmian wartości właściwość CSS elementu, podczas gdy są do niego dodawane klasy CSS lub też klasy te są z niego usuwane. Funkcje te są w zasadzie rozszerzeniami istniejących już funkcji jQuery: `addClass()`, `toggleClass()` oraz `removeClass()` (patrz strona 160). Biblioteka jQuery UI jedynie dodaje mechanizm animowania wprowadzanych zmian — w ten sposób, kiedy dodajemy klasę do jakiegoś elementu lub ją z niego usuwamy, jQuery UI animuje zauważalne zmiany wynikające z faktu dodania lub usunięcia właściwości CSS.

Uwaga: Animowanie klas obsługiwane przez jQuery UI może przywodzić na myśl przemiany CSS. I faktycznie, są one dosyć podobne: przemiany CSS zależą od wbudowanego w przeglądarkę mechanizmu wyświetlania, natomiast jQuery animuje zmiany, używając języka JavaScript. Przemiany CSS są lepszym rozwiązaniem: działają bardziej płynnie i nie wymagają angażowania interpretera JavaScript. Jednak z drugiej strony przemiany CSS nie są obsługiwane przez przeglądarkę Internet Explorer 9 i jej wcześniejsze wersje, dlatego, aby uzyskać w nich ten sam efekt, trzeba będzie użyć możliwości animowania zmian klas udostępnianej przez jQuery UI. Jeśli jednak nie musimy obsługiwać starych wersji Internet Explorera, lepiej użyć przemian CSS i zrezygnować z funkcji jQuery UI. Informacje na temat przemian CSS można znaleźć na stronie *https://developer.mozilla.org/en-US/docs/Web/Guide/CSS/Using_CSS_transitions*.

Wszystkie funkcje jQuery UI służące do animowania zmian klas mogą przyjmować do czterech argumentów określających sposób ich działania. Pierwszym z nich jest nazwa dodawanej lub usuwanej klasy. Kolejnymi trzema argumentami są: czas trwania animacji, używana funkcja określająca jej tempo oraz funkcja zwrotna wywoływana po zakończeniu animacji — czyli te same argumenty, które są przekazywane w metodzie `effect()` (patrz strona 462). Aby na przykład wybrać element o identyfikatorze `feature`, dodać do niego klasę o nazwie `highlight` i animować widoczne zmiany wywołane przez dodanie tej klasy, należałoby użyć następującego fragmentu kodu:

```
$('#feature').addClass('highlight', 1000, 'easeOutBack', function () {
  alert('Animacja została zakończona.');
});
```

Powyższy kod dodaje do elementu klasę o nazwie `highlight` i animuje widoczne zmiany w wyglądzie elementu, wprowadzając je w czasie jednej sekundy (1000 milisekund) i używając przy tym funkcji `easeOutBack` (patrz strona 465). Po zakończeniu animacji wywoływana jest funkcja zwrotna, która wyświetla okno dialogowe z komunikatem „Animacja została zakończona.". W celu wykonania animacji zmiany klas konieczne jest jedynie podanie nazwy klasy oraz czasu trwania animacji.

Biblioteka jQuery UI udostępnia cztery metody służące do animowania zmian klas.

- **Metoda `addClass()`** pozwala na dodanie klasy do jednego lub kilku elementów oraz wykonanie animacji widocznych zmian w zadanym okresie czasu.

Uwaga: Jeśli w wywołaniu którejkolwiek z tych metod nie zostanie przekazany czas trwania animacji, jQuery UI zastosuje standardową funkcję jQuery, która nie korzysta z animacji, na przykład:

```
$('#feature').addClass('highlight');
```

Ten kod dodaje do elementu klasę `highlight`, lecz nie wykonuje animacji zmian.

- **Metoda `removeClass()`** działa podobnie jak metoda `addClass()`. W jej wywołaniu podajemy nazwę klasy, czas trwania animacji, opcjonalną nazwę funkcji określającej tempo animacji oraz także opcjonalną funkcję zwrotną. Jednak w odróżnieniu do `addClass()` ta metoda *usuwa* z elementu podaną klasę, animując przy tym wszelkie wizualne zmiany, które to usunięcie wywoła.
- **Metoda `toggleClass()`** dodaje podaną klasę do elementu, jeśli jeszcze nie jest w nim używana, lub usuwa ją, jeśli jest. Można jej używać w połączeniu ze zdarzeniem `click` (patrz strona 179), aby dodać klasę do elementu, gdy zostanie kliknięty po raz pierwszy, a następnie usunąć ją po ponownym kliknięciu — w taki sposób można utworzyć element działający jak przełącznik, który może być włączony lub wyłączony. Oczywiście także ta metoda animuje wszelkie widoczne zmiany wywołane dodaniem lub usunięciem klasy.

Uwaga: Metoda `toggleClass()` jQuery UI udostępnia także bardziej zaawansowane opcje. Wszelkie informacje na ich temat można znaleźć na stronie *http://api.jqueryui.com/toggleClass/*.

- **Metoda `switchClass()`** jest jedyną spośród metod związanych z operacjami na klasach, która nie ma swojego odpowiednika wśród funkcji biblioteki jQuery. Wymaga podania nazw dwóch klas: pierwsza z nich jest klasą, którą jQuery UI usunie z wybranego elementu, a druga — klasą, która zostanie do niego dodana. A zatem metoda ta działa jak połączenie wywołań dwóch metod — `removeClass()` i `addClass()`— lecz pozwala animować zmiany, przyjmując jako punkt początkowy stan elementu sprzed usunięcia pierwszej klasy, a jako punk końcowy — stan elementu po dodaniu drugiej klasy. Jeśli na przykład usuniemy z elementu klasę, która definiuje jego kolor tła jako czerwony, i dodamy klasę, która zmienia to tło na zielone, jQuery UI wykona animację zmiany koloru tła pomiędzy kolorami czerwonym i zielonym:

```
$('#feature').switchClass('defaultStyles','highlight',1000);
```

Uwaga: Więcej informacji na temat metod jQuery UI można znaleźć na stronie *http://api.jqueryui.com/category/effects-core/*.

IV

CZĘŚĆ

Zaawansowane zastosowania jQuery i języka JavaScript

13
ROZDZIAŁ

Wprowadzenie do technologii AJAX

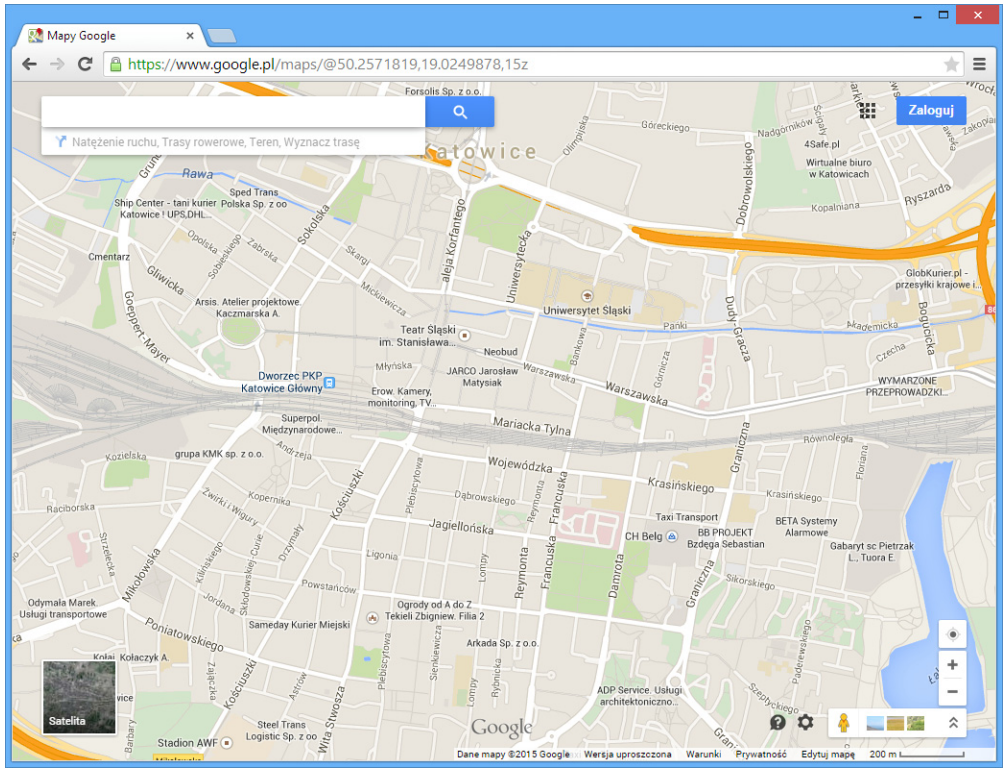
JavaScript to wspaniały język, ale nawet jego możliwości są ograniczone. Jeśli chcesz wyświetlić informacje z bazy, wysłać e-mail z danymi z formularza lub wczytać dodatkowy kod HTML, musisz nawiązać komunikację z serwerem WWW. Zwykle odbywa się to przez pobranie nowej strony WWW. Przykładowo przy wyszukiwaniu informacji w bazie przeglądarka zwykle opuszcza pierwotną stronę i wyświetla wyniki w nowej.

Oczekiwanie na wczytanie nowej strony oczywiście zajmuje czas. Użytkownicy na pewno chcieliby, by witryny sprawiały wrażenie, że działają szybciej i sprawniej reagują na wykonywane czynności, tak jakby były otwierane na lokalnym komputerze, a nie na odległym serwerze. Witryny, takie jak Facebook, Twitter, Google Docs czy też Gmail, sprawiają, że granica pomiędzy witrynami WWW a tradycyjnym programami komputerowymi staje się coraz bardziej niewyraźna. Technologią, która pomaga tworzyć aplikacje internetowe nowej generacji, jest AJAX.

AJAX umożliwia żądanie i pobieranie odpowiedzi od serwera WWW oraz aktualizowanie wyświetlanych materiałów bez konieczności wczytywania całej nowej strony. Efektem są witryny, które dużo szybciej reagują na działania użytkowników. Przykładowo w witrynie Google Maps (patrz rysunek 13.1) możesz przybliżyć widok, przejść na północ, południe, wschód lub zachód, a nawet chwycić i przeciągnąć mapę. Wszystkie te operacje zachodzą bez wczytywania nowych stron.

Czym jest AJAX?

Nazwę „AJAX" wymyślono w roku 2005. Miała ona opisywać istotę nowych witryn udostępnionych przez firmę Google — Google Maps (*http://maps.google.com/*) oraz Gmail (*http://www.gmail.com*). AJAX to akronim od zwrotu *Asynchronous JavaScript and XML*, jednak w przeciwieństwie do języków HTML, JavaScript i CSS



Rysunek 13.1. *Google Maps (http://maps.google.com/) to jedna z pierwszych dużych witryn, w której użyto AJAX-a do odświeżania zawartości strony bez konieczności wczytywania całych dokumentów. Szybkość reagowania strony wynika z tego, że zmieniają się jedynie dane mapy, a pozostałe części strony — logo, pole wyszukiwania, ramka z wynikami i kontrolki mapy — pozostają takie same przy żądaniu nowych obszarów*

nie jest to „oficjalna" technologia. To po prostu określenie, które opisuje współdziałanie kilku technologii — języka JavaScript, mechanizmów przeglądarek i serwerów WWW — przy pobieraniu i wyświetlaniu nowych materiałów bez konieczności wczytywania całych stron WWW.

Oto sposób działania technologii AJAX: umieszczony na stronie skrypt wysyła żądanie z przeglądarki na serwer WWW, który z kolei przesyła z powrotem do przeglądarki jakieś dane (nazywane *odpowiedzią*). Skrypty JavaScript przyjmują te dane i używają ich. Jeśli klikniesz w witrynie Google Maps strzałkę skierowaną w górę, kod JavaScript zażąda od serwera Google nowych danych, a następnie użyje ich do wyświetlenia odpowiedniego fragmentu mapy.

Choć pewnie nie pracujesz nad nową wersją witryny Google Maps, AJAX umożliwia też wykonywanie wielu prostych operacji. Oto przykłady.

- **Wyświetlanie nowych danych HTML bez konieczności odświeżania strony**. Przykładowo na stronie prezentującej kilka nagłówków oraz wyświetlającej treść artykułu po kliknięciu jednego z nagłówków można uchronić użytkowników przed oczekiwaniem na wczytanie nowej strony. Zamiast tego wybrany artykuł może zostać wyświetlony bezpośrednio na dotychczasowej stronie, bez żadnych

banerów reklamowych, bocznych kolumn, stopki oraz całej pozostałej zawartości strony, którą w przeciwnym razie przeglądarka musiałaby wczytywać. Jak utworzyć takie rozwiązanie, dowiesz się na stronie 480.

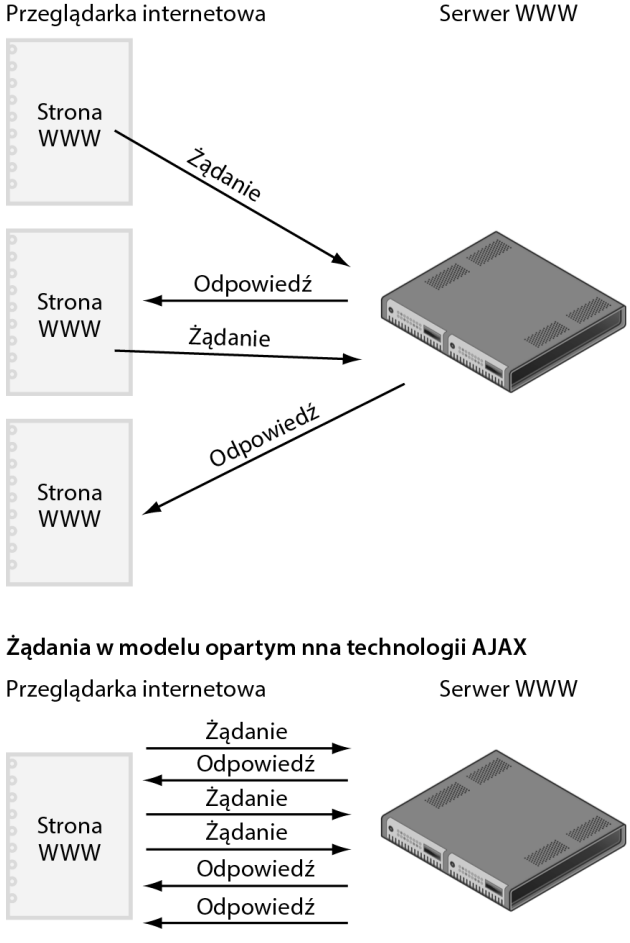
- **Przesyłanie formularzy i natychmiastowe wyświetlanie wyników**. Wyobraź sobie formularz rejestrowania się na liście abonentów biuletynu. Kiedy użytkownik wypełni i prześle taki formularz, jego elementy znikną i natychmiast pojawi się komunikat typu „Zarejestrowałeś się na liście abonentów". Na stronie 495 zobaczysz, jak utworzyć taki formularz za pomocą AJAX-a.
- **Logowanie się bez opuszczania strony**. Strona z małym formularzem logowania to następne zastosowanie języka JavaScript związane z formularzami. Wystarczy wypełnić formularz i wcisnąć przycisk *Zaloguj*, a skrypt nie tylko zaloguje użytkownika, ale też wyświetli jego status, nazwę i inne specyficzne informacje.
- **Kontrolka do oceny materiałów za pomocą liczby gwiazdek**. W witrynach z listami książek, filmów i innych produktów często dostępne są oceny w postaci liczby gwiazdek (zwykle od jednej do pięciu), określające jakość towaru zdaniem klientów. Takie systemy oceniania przeważnie pozwalają wyrazić swoją opinię przez zaznaczenie odpowiedniej liczby gwiazdek. Dzięki AJAX-owi można umożliwić użytkownikom dokonanie oceny bez opuszczania strony. Wystarczy, że klient kliknie właściwą gwiazdkę. Do obsługi tego mechanizmu można użyć wtyczki biblioteki jQuery (*http://www.wbotelhos.com/raty/*).
- **Przeglądanie informacji z bazy danych**. Amazon to typowy przykład internetowej bazy danych, którą można przeglądać. Kiedy klient szuka w witrynie sklepu Amazon książek na temat języka JavaScript, otrzymuje listę dostępnych podręczników. Zwykle nie mieszczą się one wszystkie na jednej stronie, dlatego trzeba przechodzić między kolejnymi fragmentami listy, aby wyświetlić następne 10 pozycji. Za pomocą AJAX-a można poruszać się po rekordach bazy danych bez konieczności przechodzenia do nowej strony. A oto sposób, w jaki AJAX jest używany w serwisie Twitter: gdy przeglądasz swoją stronę na Twitterze, wyświetlana jest lista komunikatów od osób, które „śledzimy". Po przewinięciu tej listy do samego końca serwis wczytuje nową porcję komunikatów. Wystarczy przewinąć trochę dalej, a pojawią się nowe komunikaty. W ten sposób można odnieść wrażenie, że strona jest nieskończona!

W żadnej z tych operacji nie ma nic rewolucyjnego. Podobne efekty można uzyskać za pomocą standardowego kodu HTML i skryptów działających po stronie serwera (są potrzebne na przykład do pobierania danych z formularza lub informacji z bazy). Różnica kryje się w zwrocie „bez konieczności wczytywania nowej strony". AJAX sprawia, że strony reagują szybciej, zatem usprawnia korzystanie z witryny i poprawia doznania użytkowników.

AJAX — podstawy

Technologie, na których oparto AJAX, są dość skomplikowane. Niezbędne jest współdziałanie kodu JavaScript, skryptów działających po stronie serwera i mechanizmów przeglądarek. Jednak podstawowa zasada funkcjonowania tej technologii jest

prosta, jeśli zrozumiesz wszystkie kroki związane z użytkowaniem AJAX-a. Na rysunku 13.2 przedstawiono różnicę między komunikacją serwera WWW z tradycyjnymi stronami HTML i ze stronami opartymi na AJAX-ie.



Rysunek 13.2. *W tradycyjnych, prostych witrynach WWW przeglądarka żąda przesłania strony z serwera i otrzymuje ją od niego (na górze). Proces ten polega na ciągłym wczytywaniu i odświeżaniu stron. Przy korzystaniu z AJAX-a przeglądarka żąda tylko nowych informacji. Serwer zwraca żądane dane, po czym — przy udziale skryptów JavaScript — następuje aktualizacja treści i wyglądu strony (u dołu)*

Elementy układanki

AJAX nie jest niezależną technologią. Składa się z wielu różnych elementów, których współdziałanie poprawia komfort pracy użytkowników. Oto trzy podstawowe składniki AJAX-a.

- **Przeglądarka internetowa**. Jest ona oczywiście niezbędna do przeglądania stron WWW i uruchamiania kodu JavaScript, jednak większość przeglądarek ma też wbudowany tajemny składnik umożliwiający działanie AJAX-a. Jest to *obiekt* `XMLHttpRequest`. Ten element o dziwnej nazwie sprawia, że kod JavaScript może nawiązać komunikację z serwerem WWW i odbierać od niego informacje.

Obiekt `XMLHttpRequest` wprowadzono w przeglądarce Internet Explorer 5 wiele lat temu, jednak stopniowo zaczął się pojawiać we wszystkich najważniejszych przeglądarkach. Więcej o tym obiekcie dowiesz się na stronie 476.

- **Język JavaScript** wykonuje większość skomplikowanych zadań w technologii AJAX. Przesyła żądania na serwer, oczekuje na odpowiedź, przetwarza ją i zazwyczaj aktualizuje stronę przez dodanie nowych materiałów lub zmianę jej wyglądu. W zależności od przeznaczenia programu kod JavaScript może przesyłać informacje z formularza, żądać dodatkowych rekordów z bazy lub wysyłać pojedyncze dane (na przykład ocenę przyznaną książce przez użytkownika). Po przesłaniu danych na serwer skrypt JavaScript jest gotowy na odbiór odpowiedzi, na przykład rekordów z bazy danych lub prostych komunikatów typu „Twój głos został dodany".

  Na podstawie uzyskanych informacji skrypt JavaScript aktualizuje stronę, na przykład wyświetla nowe rekordy lub informuje użytkownika o udanym logowaniu. Aktualizowanie strony obejmuje manipulowanie modelem DOM (ang. *Document Object Model*; patrz strona 145) w celu dodania, zmiany lub usunięcia znaczników HTML i ich zawartości. Większość tej książki opisuje właśnie takie operacje — modyfikowanie treści i wyglądu stron za pomocą języka JavaScript.

- **Serwer WWW** odbiera żądanie od przeglądarki i przesyła odpowiedź z danymi. Serwer może zwracać kod HTML lub zwykły tekst, a także dokumenty XML (patrz ramka na stronie 495) lub dane w formacie JSON (patrz strona 500). Jeśli serwer odbiera informacje z formularza, może dodać je do bazy danych i zwrócić komunikat z potwierdzeniem: „Rekord został dodany". Skrypt JavaScript może też zażądać 10 następnych rekordów z bazy, a serwer powinien wtedy zwrócić informacje zawierające te dane.

  Ten element układanki bywa skomplikowany i zwykle wymaga użycia kilku technologii: serwera WWW, serwera aplikacji i serwera baz danych. Serwer WWW to specyficzna „szafka na akta". Przechowuje dokumenty, a także udostępnia je, kiedy przeglądarka ich zażąda. Do wykonywania bardziej skomplikowanych zadań, na przykład umieszczania danych z formularza w bazie, potrzebne są *serwer aplikacji* i *serwer baz danych*. Serwer aplikacji obsługuje języki programowania używane po stronie serwera, takie jak PHP, Java, C#, Ruby lub Cold Fusion, oraz przetwarza zadania, których nie można wykonać za pomocą samych stron HTML. Umożliwia na przykład wysyłanie listów elektronicznych, sprawdzanie cen książek w witrynie Amazon lub zapisywanie informacji w bazie danych. Serwer baz danych służy do przechowywania informacji, między innymi nazwisk i adresów klientów, szczegółowych informacji o sprzedawanych produktach lub archiwum ulubionych przepisów. Do popularnych serwerów tego typu należą MySQL, PostgreSQL i SQL Server.

Uwaga: Pojęcie „serwer" może oznaczać sprzęt lub oprogramowanie. W tej książce nazwy *serwer aplikacji*, *serwer WWW* i *serwer baz danych* oznaczają oprogramowanie, które może działać na tym samym komputerze (jest to często stosowane rozwiązanie).

Różne serwery WWW, serwery aplikacji i serwery baz danych można łączyć na wiele sposobów. Na przykład można korzystać z serwera WWW IIS Microsoftu, ASP.NET (serwera aplikacji) i narzędzia SQL Server (serwera baz danych). Inny zestaw to Apache (serwer WWW), PHP (serwer aplikacji) i MySQL (serwer baz danych).

WIEDZA W PIGUŁCE

Konfigurowanie serwera WWW

AJAX współpracuje z serwerem WWW. W końcu podstawowym zadaniem tej technologii jest umożliwienie wysyłania i pobierania informacji z serwera za pomocą kodu JavaScript. Wszystkie przykłady przedstawione w tym rozdziale, z wyjątkiem jednego — prezentującego korzystanie z serwisu Flickr — wymagają działającego serwera WWW. Co więcej, prawdopodobnie będziesz chciał go uruchomić, także po to, aby lepiej poznać świat AJAX-a. Jeśli masz już witrynę dostępną w internecie, możesz przetestować programy ajaksowe przez przeniesienie plików na używany serwer WWW. Niestety, ta technika jest niewygodna w użyciu. Musisz utworzyć strony na własnym komputerze, a następnie przenieść je na serwer za pomocą programu do obsługi kont FTP, aby zobaczyć, czy działają.

Lepsze podejście polega na zainstalowaniu *serwera do tworzenia oprogramowania*. W tym celu należy zainstalować serwer WWW na własnym komputerze, aby można było rozwijać i testować na nim programy ajaksowe. Na pozór jest to trudne zadanie, jednak istnieje wiele bezpłatnych programów, które umożliwiają instalację wszystkich potrzebnych komponentów przez dwukrotne kliknięcie pliku.

W systemie Windows możesz zainstalować serwery Apache, PHP i MySQL za pomocą pakietu WAMP (*http://www.wampserver.com/en/*). Jest to bezpłatny pakiet instalacyjny, który konfiguruje wszystkie elementy potrzebne do zasymulowania działania prawdziwych witryn dostępnych w internecie.

Miłośnicy komputerów Mac mogą skorzystać z łatwego w użyciu programu MAMP (*http://www.mamp.info/en/*), który obejmuje serwery Apache, PHP i MySQL. Także ten pakiet jest bezpłatny. (Dostępna jest także wersja MAMP przeznaczona dla systemów Windows).

Przykłady przedstawione na stronach 482 oraz 495 wymagają użycia serwera WWW. Dlatego jeśli chcesz uruchomić wszystkie przykłady, musisz zainstalować zestaw AMP przy użyciu jednego z wymienionych wcześniej pakietów. Jeśli masz już witrynę działającą na innym serwerze (na przykład IIS Microsoftu), prawdopodobnie zechcesz zainstalować go także na własnym komputerze, jeżeli planujesz tworzenie aplikacji ajaksowych i udostępnianie ich w internecie. Serwer IIS jest domyślnie instalowany w systemie Windows 8 — trzeba go jedynie włączyć. Dowiesz się, jak to zrobić, z tego klipu wideo: *https://www.youtube.com/watch?v=mRm9-Xddt2w*.

Uwaga: Zestaw Apache, PHP i MySQL (często nazywany AMP) jest bardzo popularny i dostępny bezpłatnie. Większość dostawców usług hostingowych korzysta z tych serwerów. Także przykłady przedstawione w tej książce oparto na tym zestawie (patrz ramka na stronie 476).

Komunikacja z serwerem WWW

Podstawą każdego programu ajaksowego jest obiekt `XMLHttpRequest` (czasem nazywany też *XHR*). Jest on wbudowany we współczesne przeglądarki, które umożliwiają przesyłanie informacji na serwer WWW i odbieranie ich za pomocą kodu JavaScript. Proces komunikacji obejmuje pięć głównych kroków, a wszystkie je można obsłużyć przy użyciu języka JavaScript.

1. **Tworzenie egzemplarza obiektu `XMLHttpRequest`.**

   Pierwszy krok przygotowuje przeglądarkę na przesłanie przez skrypt informacji na serwer WWW. W najprostszej postaci instrukcja tworząca obiekt `XMLHttpRequest` w kodzie JavaScript wygląda następująco:

   ```
   var xhr = new XMLHttpRequest();
   ```

   Choć powyższa instrukcja jest prosta, to jednak korzystanie z technologii AJAX przy wykorzystaniu wyłącznie zwyczajnych możliwości języka JavaScript może być trudne. Na szczęście biblioteka jQuery znacznie ułatwia wysyłanie informacji i obsługę AJAX-a. Dowiesz się o tym więcej na stronie 479.

2. **Zastosowanie metody `open()` obiektu `XHR` do określenia rodzaju przesyłanych danych i ich docelowej lokalizacji.**

   Dane można przesyłać na kilka sposobów. Zdecydowanie najczęściej używane są jednak dwie, określane jako metody `GET` lub `POST` (są to te same metody, których używasz do wysyłania formularzy HTML). Metoda `GET` przesyła dane na serwer WWW w adresie URL, na przykład *show.php?productID=34*. W tym przypadku dane są przesyłane w formie *łańcucha zapytania* (ang. *query string*) i zostały zapisane po znaku zapytania (`?`); konkretnie rzecz biorąc, jest to fragment *productID=34*. Jak widać, mają one postać pary nazwa – wartość (*productID* to nazwa, a *34* to wartość). Wyobraź sobie, że ten pierwszy element to nazwa pola formularza, a wartość to dane wprowadzone w tym polu przez użytkownika.

   Metoda `POST` przesyła dane niezależnie od adresu URL. Programiści zwykle używają metody `GET` do pobierania danych z serwera, a metody `POST` — do przesyłania informacji, które serwer powinien zapamiętać (czyli informacji służących do dodawania, modyfikowania lub usuwania rekordów w bazie danych). Na stronie 486 dowiesz się, jak korzystać z obu tych metod.

   W metodzie `open()` można też określić stronę na serwerze, do której kierowane są dane. Zwykle jest to strona z kodem w języku działającym po stronie serwera (takim jak PHP), która pobiera dane z bazy lub wykonuje inne zadania. Stronę tę należy wskazać za pomocą adresu URL. Na przykład poniższy kod informuje obiekt `XHR` o tym, której metody ma użyć (`GET`) i do jakiej strony na serwerze skierować żądanie:

```
xhr.open('GET', shop.php?productID=34');
```

Uwaga: Adres URL podany w wywołaniu metody `open()` musi wskazywać zasób umieszczony na tym samym serwerze, z którego pochodzi strona generująca żądanie. Ze względów bezpieczeństwa przeglądarki WWW nie pozwalają na przesyłanie przy użyciu technologii AJAX żądań skierowanych do innych domen. Ograniczenie to można obejść, korzystając z rozwiązania nazywanego JSONP, które zostało opisane na stronie 506.

3. **Tworzenie funkcji obsługującej pobrane dane.**

   Kiedy serwer WWW zwróci odpowiedź, na przykład nowe informacje z bazy, potwierdzenie przetworzenia formularza lub zwykły komunikat tekstowy, zwykle należy użyć odebranych danych. Może to wymagać tylko wyświetlenia tekstu typu „Przesyłanie formularza zakończyło się powodzeniem" lub zastąpienia całej tabeli rekordów bazy danych tabelą z nowymi informacjami. Zawsze jednak trzeba przygotować funkcję JavaScript do obsługi odpowiedzi. Ta funkcja (jest to *funkcja wywoływana zwrotnie*) to często najważniejsza część programu.

   Zwykle takie funkcje manipulują zawartością strony i zmieniają jej model DOM. Usuwają elementy (na przykład przesłany za pomocą AJAX-a formularz), dodają je (na przykład komunikat „Przesyłanie formularza zakończyło się powodzeniem" lub nową tabelę HTML z rekordami z bazy danych) lub modyfikują (na przykład wyróżniają liczbę gwiazdek kliknięta przez użytkownika przy ocenie produktu).

Potrzebne są jeszcze pewne dodatkowe operacje, jednak do zarządzania szczegółami posłuży biblioteka jQuery, dlatego musisz jedynie pamiętać, że wywoływana zwrotnie funkcja zawiera kod JavaScript, który obsługuje odpowiedź zwróconą przez serwer.

4. **Wysyłanie żądania.**

   Aby przesłać informacje na serwer WWW, należy użyć metody `send()` obiektu `XHR`. Wszystkie operacje do tego momentu to faza przygotowawcza. Dopiero *ten* krok informuje przeglądarkę, że wszystko jest gotowe i można wysłać żądanie. Jeśli używasz metody `GET`, ten etap jest bardzo prosty:

   ```
   xhr.send();
   ```

   Metoda `send()` może przyjmować jeden argument — dane, które mają zostać przesłane na serwer. W przypadku żądań przesyłanych metodą `GET` informacje są przesyłane w adresie URL (na przykład *search.php?q=javascript*, gdzie *q=javascript* to dane). Przy korzystaniu z metody `POST` trzeba przekazać dane do metody `send()` w następujący sposób:

   ```
   xhr.send('q=javascript');
   ```

   Także tu nie musisz martwić się szczegółami. W następnym punkcie zobaczysz, jak uprościć tę operację za pomocą biblioteki jQuery.

   Po przesłaniu żądania program JavaScript nie musi wstrzymywać działania. Litera „A" w nazwie AJAX pochodzi od słowa *asynchroniczny*, co oznacza, że po wysłaniu żądania skrypt może wykonywać dalsze operacje. Przeglądarka nie musi bezczynnie oczekiwać na odpowiedź od serwera.

5. **Pobieranie odpowiedzi.**

   Kiedy serwer przetworzy żądanie, przesyła odpowiedź do przeglądarki. Za obsługę odpowiedzi odpowiada wywoływana zwrotnie funkcja, którą utworzyłeś w kroku 3., jednak obiekt `XHR` otrzymuje w tym czasie kilka informacji, w tym *status* żądania, *tekst* odpowiedzi i — w zależności od ustawień — odpowiedź w formacie *XML*.

   Status odpowiedzi to numer określający, jak serwer zareagował na żądanie. Prawdopodobnie znasz status *404*, który oznacza, że nie znaleziono żądanego pliku. Jeśli wszystko poszło zgodnie z planem, serwer zwróci wartość 200 lub 304. Jeśli w czasie przetwarzania strony wystąpił błąd, otrzymasz status 500 (wewnętrzny błąd serwera), a jeśli żądany plik jest zabezpieczony hasłem, pojawi się błąd 403 (dostęp wzbroniony). Kompletną listę kodów statusu można znaleźć na stronie *http://www.w3.org/Protocols/rfc2616/rfc2616-sec10.html*.

Przeważnie serwer zwraca odpowiedź tekstową, która jest zapisywana we właściwości `responseText` obiektu `XHR`. Tą odpowiedzią może być fragment kodu HTML, prosty komunikat tekstowy lub skomplikowany zbiór danych w formacie JSON (patrz strona 500). Jeśli serwer zwróci plik XML, zostanie on zapisany we właściwości `responseXML` obiektu `XHR`. Choć format XML nadal jest używany, strony działające na serwerze częściej zwracają dane jako tekst, kod HTML lub JSON, dlatego może się okazać, że nigdy nie będziesz musiał przetwarzać odpowiedzi w formie kodu XML.

Niezależnie od formatu zwróconych danych są one dostępne dla wywoływanej zwrotnie funkcji, która może ich użyć do zaktualizowania strony. Wykonanie kodu tej funkcji kończy cały cykl obsługi ajaksowego żądania. Warto jednak pamiętać, że w tym samym czasie można zgłosić wiele takich żądań.

AJAX w bibliotece jQuery

Choć podstawowy proces korzystania z obiektu `XMLHttpRequest` nie jest skomplikowany, to przesłanie i obsługa każdego żądania wymagają wykonania sekwencji tych samych czynności. Na szczęście biblioteka jQuery udostępnia kilka funkcji, które znacznie upraszczają ten proces. W końcu, po przyjrzeniu się pięciu krokom obsługi żądań ajaksowych (patrz strona 476) można zauważyć, że fragmenty wykonujące istotne dla skryptu operacje (czyli kod przetwarzający odpowiedź zwróconą przez serwer) trzeba dodać tylko w jednym, 3. kroku (opisanym na stronie 477). Biblioteka jQuery upraszcza wszystkie pozostałe etapy, dlatego można skoncentrować się na pisaniu ciekawego kodu.

WIEDZA W PIGUŁCE

Nauka tworzenia skryptów działających po stronie serwera

Jeśli do wczytywania kodu HTML ze strony zapisanej na serwerze do strony widocznej w przeglądarce nie używasz podstawowej metody `load()` biblioteki jQuery (opisanej powyżej), to aby zastosować AJAX, potrzebujesz skryptów uruchamianych po stronie serwera. Podstawowym zadaniem AJAX-a jest umożliwianie komunikowania się kodu JavaScript z serwerem (i pobieranie w ten sposób informacji). Przeważnie oznacza to, że na serwerze WWW znajduje się inny skrypt, który wykonuje zadania niemożliwe do obsłużenia za pomocą języka JavaScript, na przykład wczytuje informacje z bazy danych, wysyła listy elektroniczne lub loguje użytkowników.

Omawianie tworzenia skryptów działających po stronie serwera wykracza poza zakres tej książki, dlatego musisz nauczyć się używać języków serwerowych, takich jak PHP, Ruby on Rails, .NET, JSP (możesz też skorzystać z usług programisty, który napisze taki kod za Ciebie). Jeśli nie wybrałeś jeszcze języka, którego chcesz używać po stronie serwera, dobrym punktem wyjścia będzie PHP. To jeden z najpopularniejszych języków tego typu; jest bezpłatny i prawie wszystkie firmy hostingowe obsługują go na swych serwerach. Język ten ma duże możliwości, został opracowany specjalnie pod kątem sieci WWW i jest *stosunkowo* łatwy w nauce. Jeśli chcesz rozpocząć poznawanie tego języka, wypróbuj książki *Learning PHP, MySQL & JavaScript: With jQuery, CSS & HTML5, 4th Edition*[1] (wydawnictwo O'Reilly), *Head First PHP & MySQL*[2] (wydawnictwo O'Reilly)

Dostępnych jest też wiele bezpłatnych materiałów do nauki języka PHP. Seria samouczków PHP 101 (*http://devzone.zend.com/6/php-101-php-for-the-absolute-beginner/*) firmy Zend, jednej z głównych organizacji wspomagających rozwój języka PHP, zawiera mnóstwo podstawowych i zaawansowanych informacji. Witryna W3Schools także udostępnia przydatne materiały dla początkujących programistów języka PHP (*http://www.w3schools.com/PHP*).

[1] Wydanie polskie: *PHP, MySQL i JavaScript. Wprowadzenie. Wydanie IV*, Helion, Gliwice 2015 — *przyp. red.*

[2] Wydanie polskie: *Head First PHP & MySQL. Edycja polska*, Helion, Gliwice 2010 — *przyp. red.*

Używanie metody load()

Najprostszą metodą biblioteki jQuery związaną z korzystaniem z AJAX-a jest `load()`. Wczytuje ona plik HTML do określonego elementu strony. Załóżmy, że na stronie znajduje się obszar przeznaczony na krótką listę nagłówków wiadomości. Po wczytaniu strony na liście ma znaleźć się pięć najnowszych informacji. Warto też udostępnić kilka odnośników, które pozwolą użytkownikom wybrać rodzaj wyświetlanych artykułów (na przykład wczorajsze wydarzenia, informacje lokalne, wiadomości sportowe i tak dalej). Odsyłacze te mogą prowadzić do odrębnych stron, z których każda zawiera odpowiednie teksty, jednak zmusza to czytelników do pobierania nowych dokumentów (i w ogóle nie wymaga użycia AJAX-a!).

Inne podejście polega na wczytaniu wybranych wiadomości do pola z artykułami na aktualnie widocznej stronie. Oznacza to, że kiedy użytkownik wybierze nową kategorię informacji, przeglądarka zażąda z serwera nowego pliku HTML, a następnie umieści go w obszarze przeznaczonym na wiadomości, nie przechodząc przy tym do następnej strony (patrz rysunek 13.3).

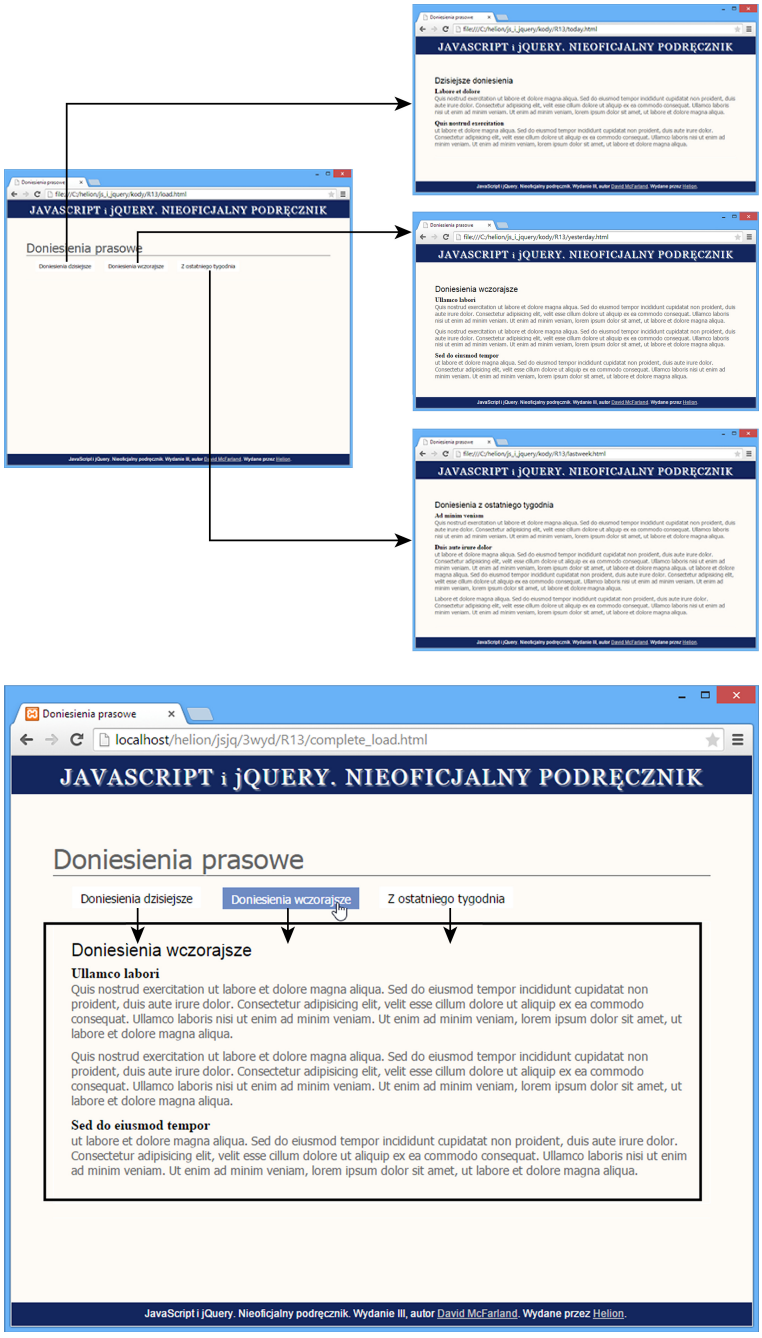
Aby wywołać metodę `load()`, najpierw należy użyć selektora jQuery do pobrania elementu strony, w którym ma znaleźć się żądany kod HTML. Następnie można wywołać tę funkcję i przekazać do niej adres URL pobieranej strony. Załóżmy, że na stronie znajduje się znacznik `<div>` o identyfikatorze `headlines` i chcesz zapisać w tym elemencie kod HTML z pliku *todays_news.html*. Można to zrobić w następujący sposób:

```
$('#headlines').load('todays_news.html');
```

Kiedy skrypt uruchomi ten kod, przeglądarka zażąda pliku *todays_news.html* z serwera WWW. Po pobraniu go przeglądarka zastąpi bieżącą zawartość znacznika `<div>` o identyfikatorze `headlines` kodem nowego pliku. W żądanym pliku HTML może znajdować się kompletna strona HTML (wraz ze znacznikami `<html>`, `<head>` i `<body>`) lub tylko fragment kodu, na przykład jeden znacznik `<h1>` i akapit tekstu. Plik ten nie musi zawierać całej strony, ponieważ metoda `load()` tylko dołącza jego kod do aktualnej (kompletnej) strony WWW.

Uwaga: Można wczytywać pliki HTML pochodzące tylko z tej samej witryny, w której działa bieżąca strona. Nie możesz użyć metody `load()` na przykład do wczytania strony głównej witryny Google do elementu `<div>` strony z własnej witryny.

Kiedy używasz metody `load()`, musisz zwrócić uwagę na ścieżki prowadzące do plików. Adres URL przekazywany do tej metody należy podać względem bieżącej strony. Oznacza to, że musisz użyć takiej samej ścieżki jak w odnośniku prowadzącym z bieżącej strony do pobieranego pliku HTML. Ponadto ścieżki w kodzie HTML nie są aktualizowane po wczytaniu tego kodu do dokumentu, dlatego jeśli pobierany plik zawiera odnośniki lub rysunki, ich adresy URL muszą być poprawne na stronie wywołującej metodę `load()`. Jeśli używasz ścieżek podawanych względem dokumentu (patrz ramka na stronie 45), a pobierany plik HTML znajduje się w innym katalogu witryny, rysunki i odnośniki mogą nie działać po wczytaniu kodu HTML do bieżącej strony. Jest jednak proste rozwiązanie tego problemu — wystarczy używać ścieżek podawanych względem katalogu głównego lub upewnić się, że wczytywany plik znajduje się w tym samym katalogu co strona wywołująca metodę `load()`.



Rysunek 13.3. *Górna część rysunku przedstawia standardową, opartą na odnośnikach metodę dostępu do dodatkowego kodu HTML. Kliknięcie odsyłacza na stronie (po lewej) powoduje wczytanie zupełnie nowego dokumentu (po prawej). Jednak przy użyciu AJAX-a i funkcji load() biblioteki jQuery można wyświetlić ten sam kod HTML bez opuszczania bieżącej strony (u dołu). Kliknięcie odnośnika prowadzi do wczytania kodu HTML do znacznika *

Metoda `load()` pozwala nawet określić, która część pobranego pliku HTML ma znaleźć się na stronie. Załóżmy, że żądany plik to zwykła strona witryny. Obejmuje ona wszystkie standardowe elementy, między innymi baner, pasek nawigacji i stopkę. Możliwe, że potrzebny jest tylko fragment tej strony, na przykład konkretny element `<div>` i jego zawartość. Aby określić, którą część strony chcesz wczytać, po adresie URL dodaj odstęp i selektor jQuery. Przykładowo załóżmy, że w poprzednim przykładzie chcesz wstawić tylko zawartość elementu `<div>` o identyfikatorze `news` z pliku *todays_news.html*; możesz to zrobić, używając następującego kodu:

```
$('#headlines').load('todays_news.html #news');
```

Przeglądarka pobierze stronę *todays_news.html*, ale zamiast wstawiać cały kod z tego pliku do znacznika `<div>` o identyfikatorze `headlines`, doda wyłącznie tag `<div>` o identyfikatorze `news` (i jego zawartość). W następnym przykładzie zobaczysz, jak zastosować tę technikę.

Przykład — korzystanie z metody load()

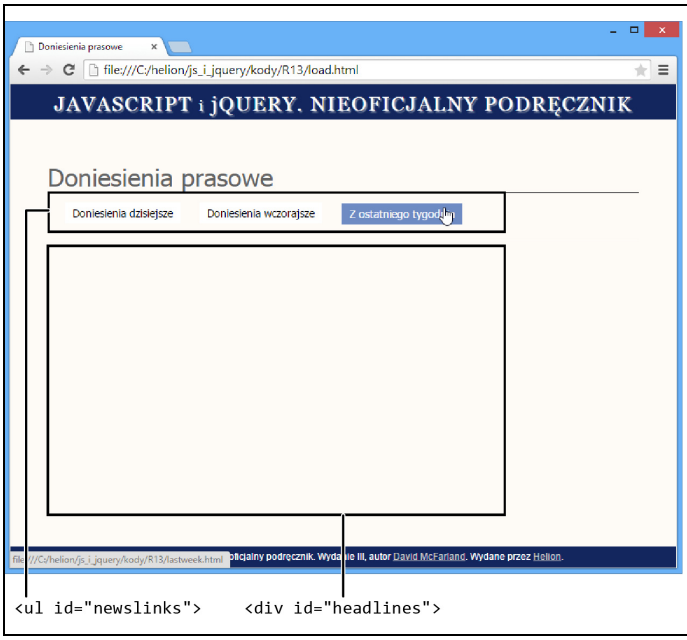
W tym przykładzie użyjesz biblioteki jQuery, aby zamiast tradycyjnej metody otwierania stron HTML techniką „kliknij i wczytaj" (patrz rysunek 13.3, u góry) zastosować bardziej interaktywnym podejście, które zastępuje treść bieżącej strony nowym kodem HTML (patrz rysunek 13.3 — na dole).

Omówienie przykładu

Aby zrozumieć, jak ma działać ten przykład, trzeba najpierw poznać kod HTML strony, na której chcesz zastosować AJAX. Przyjrzyj się rysunkowi 13.4. Strona zawiera listę wypunktowaną odnośników, z których każdy wskazuje na inną stronę z różnymi wiadomościami. Lista ta znajduje się w znaczniku `<ul>` o identyfikatorze `newslinks`. Ponadto w ramce w prawej części strony (pod napisem „Doniesienia") znajduje się pusty znacznik `<div>` o identyfikatorze `headlines`. Na tym etapie jest on tylko pustym kontenerem na dane. Kiedy użyjesz metody `load()` biblioteki jQuery, kliknięcie jednego z odnośników spowoduje umieszczenie informacji w tym elemencie `<div>`.

Obecnie kliknięcie odnośnika jedynie otwiera stronę WWW z wiadomościami. Oznacza to, że strona działa w tradycyjny sposób — zawiera odsyłacze, które prowadzą do innych plików. W rzeczywistości nawet bez zmyślnego kodu JavaScript, który wkrótce dodasz, strona działa zupełnie dobrze i doprowadzi użytkowników do szukanych informacji. Jest to korzystne, ponieważ nie wszystkie przeglądarki obsługują język JavaScript. Ponadto jeśli jedynym sposobem na dotarcie do wiadomości będzie kod JavaScript, wyszukiwarki pominą te wartościowe informacje.

Uwaga: Większość przeglądarek nie pozwala na stosowanie metody `load()` do bezpośredniego pobierania plików z własnego dysku twardego, bez wykorzystania serwera WWW. Dlatego też, aby wykonać ten przykład, będziesz musiał taki serwer zainstalować (patrz ramka na stronie 476). W czasie pisania tej książki przeglądarka Safari dla systemu Mac OS *pozwalała* na stosowanie metody load() bez użycia serwera WWW.



Rysunek 13.4. *Przy używaniu języka JavaScript do dodawania treści do strony programiści często dodają pusty znacznik o określonym identyfikatorze. Następnie można w dowolnym momencie pobrać ten element i umieścić w nim dane. W ramce w prawej części widocznej strony znajduje się pusty tag ( ). Przy użyciu AJAX-a można w łatwy sposób umieścić w nim zawartość dowolnego pliku, do którego prowadzą odnośniki ze środkowej części strony*

Ten przykład ilustruje technikę *stopniowego wzbogacania*. Strona działa prawidłowo także bez kodu JavaScript, jednak użycie tego języka pozwala ją usprawnić. Oznacza to, że każdy użytkownik może uzyskać dostęp do danych i nikt nie będzie poszkodowany.

Uwaga: Informacje o pobieraniu przykładowych plików znajdziesz na stronie 46.

Tworzenie kodu

Aby zaimplementować stopniowe wzbogacenie strony, dodasz do niej kod JavaScript, który „przechwyci" standardowy sposób działania odnośników, następnie odczyta adres URL klikniętego odnośnika, wczyta odpowiednią stronę i umieści jej zawartość wewnątrz pustego elementu . W tym celu wykonaj następujące, proste czynności.

1. **Otwórz w edytorze tekstu plik *load.html* z katalogu *R13*.**

   Należy zacząć od przypisania zdarzenia `click` do każdego odnośnika z listy wypunktowanej z głównej części strony. Ta lista (znacznik `<ul>`) ma identyfikator `newslinks`, dlatego można użyć jQuery do łatwego pobrania wszystkich odsyłaczy i wywołania dla nich funkcji `click()`.

   W pliku *load.html* znajduje się już element dołączający bibliotekę jQuery, a także element `<script>` z funkcją `$(document).ready()`.

2. **Kliknij pusty wiersz pod funkcją `$(document).ready()` i wpisz poniższy kod:**

```
$('#newslinks a').click(function() {

});
```

Wyrażenie `$('#newslinks a')` pozwala pobrać odnośniki za pomocą jQuery, a funkcja `.click()` umożliwia określenie funkcji obsługującej zdarzenie `click` (omówienie zdarzeń znajdziesz na stronie 182).

Następny krok polega na pobraniu adresu URL każdego odnośnika.

3. **W funkcji `click()` (pusty wiersz w kroku 2. powyżej) wpisz instrukcję `var url=$(this).attr('href');` i wciśnij klawisz *Enter*, aby utworzyć pusty wiersz.**

   Ten wiersz kodu tworzy nową zmienną (`url`) i przypisuje do niej wartość atrybutu `href` odnośnika. Na stronie 155 dowiedziałeś się, że jeśli dołączysz funkcję (na przykład `click()`) do znaczników pobranych za pomocą jQuery, `$('#newslinks a')`, biblioteka przejdzie po każdym znalezionym elemencie (tu są to odnośniki) i wywoła dla niego podaną funkcję. Konstrukcja `$(this)` to sposób na uzyskanie dostępu do aktualnie przetwarzanego elementu. Kiedy jQuery przechodzi w pętli po kolekcji elementów, konstrukcja ta wskazuje na kolejne odnośniki. Metoda `attr()` (patrz strona 166) służy do pobierania i ustawiania atrybutów tych odnośników. Tu metoda ta pobiera wartość atrybutu `href`, co pozwala ustalić adres URL strony, do której prowadzi dany odsyłacz. W następnym kroku użyjesz tego adresu wraz z metodą `load()` do pobrania zawartości pliku i wyświetlenia jej w elemencie `<div>` na bieżącej stronie.

4. **Dodaj instrukcję `$('#headlines').load(url);`, aby skrypt wyglądał następująco:**

```
$('#newslinks a').click(function() {
  var url=$(this).attr('href');
  $('#headlines').load(url);
});
```

   Pamiętaj, że pusty znacznik `<div>` przeznaczony na pobrany kod HTML ma identyfikator `headlines`, dlatego wyrażenie `$('#headlines')` pobiera ten element. Funkcja `load()` wczytuje plik HTML przy użyciu adresu URL pobranego w poprzednim wierszu, a następnie umieszcza zawartość tego pliku w pustym znaczniku `<div>`. To prawda, na zapleczu zachodzi *mnóstwo* operacji, aby można było uzyskać ten efekt, jednak dzięki jQuery nie musisz samodzielnie ich programować.

   Strona nie jest jeszcze gotowa. Jeśli zapiszesz plik i wyświetlisz go w przeglądarce, zauważysz, że kliknięcie odnośników nie powoduje wczytania nowych informacji na stronę. Przeglądarka opuszcza bieżący dokument i otwiera stronę, do której prowadzi odsyłacz. Co się stało z ajaksowym kodem? Wciąż znajduje się w pliku, jednak przeglądarka wykonuje domyślne operacje związane z kliknięciem odnośnika i wczytuje nową stronę. Należy zablokować ten proces.

5. **Dodaj nowy, pusty wiersz pod kodem wprowadzonym w poprzednim kroku i wpisz instrukcję `return false;`. Skrypt powinien wyglądać jak poniżej:**

```
$('#newslinks a').click(function() {
  var url=$(this).attr('href');
  $('#headlines').load(url);
  return false;
});
```

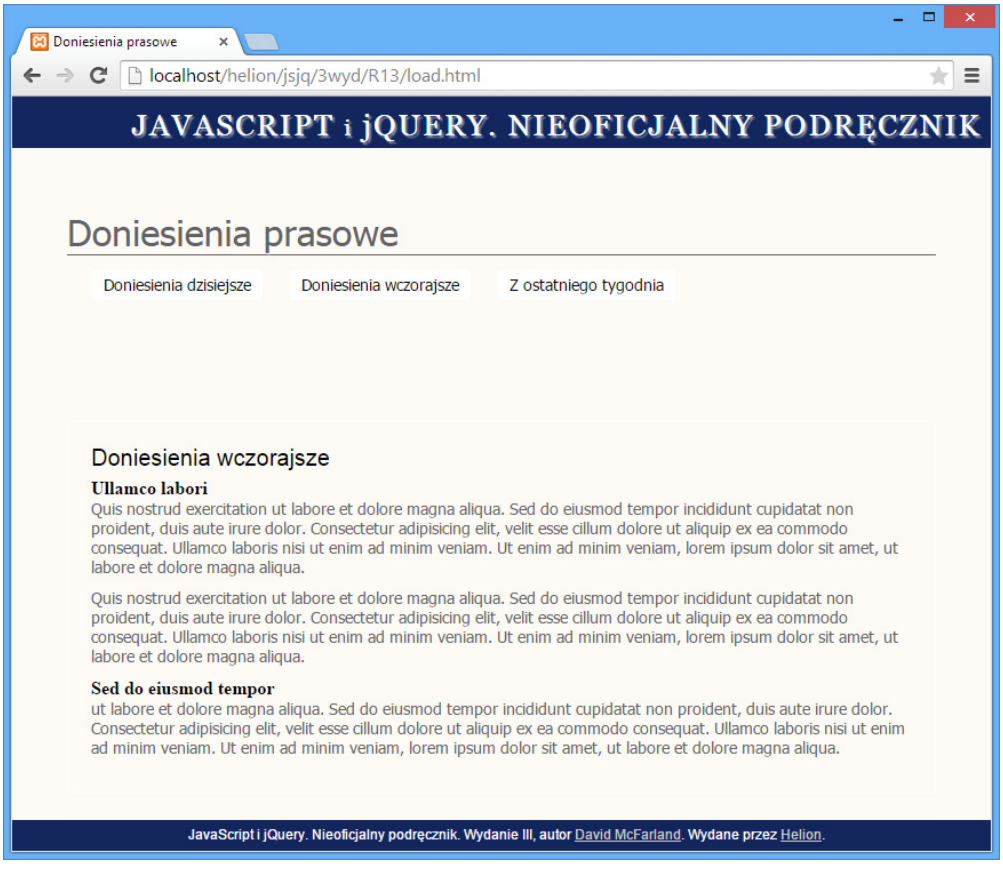
   Ten prosty kod informuje przeglądarkę o tym, że ma nie przechodzić do strony wskazanej w odnośniku. Jest to jeden ze sposobów na zablokowanie domyślnej

reakcji przeglądarki na zdarzenie. Ten sam efekt można uzyskać za pomocą funkcji `preventDefault()` biblioteki jQuery (patrz strona 195).

6. **Zapisz plik.**

   Jeśli zainstalowałeś na swoim komputerze serwer WWW, będziesz mógł wyświetlić stronę w przeglądarce. Kliknij jakiś odnośnik, aby sprawdzić jego działanie.

   Teraz pojawia się następny problem, widoczny na rysunku 13.5. Metoda `load()` działa, jednak strona nie wygląda prawidłowo. Metoda ta pobrała całą stronę WWW i wstawiła jej zawartość do elementu `<div>`. Spowodowało to pojawienie się na stronie dziwnych odstępów i zupełnie niepotrzebne powtórzenie nagłówka. A Ty chciałeś wstawić do elementu `<div>` tylko fragment strony — ten, który zawiera nowe elementy. Na szczęście także tu można skorzystać z metody `load()`.



Rysunek 13.5. *Metoda load() biblioteki jQuery pobiera cały kod HTML wskazanego pliku i umieszcza go w elemencie na bieżącej stronie. Jeśli wczytany plik zawiera niepotrzebne fragmenty witryny, na przykład powtórzony nagłówek, ramkę boczną lub stopkę, będzie wyglądał jak strona wewnątrz innej strony. W tym przypadku pojawiło się sporo miejsca powyżej nagłówka „Doniesienia wczorajsze", a nagłówek u góry strony został przesunięty w bok; co więcej, nad początkowym nagłówkiem pojawił się drugi!*

7. **Znajdź wiersz z funkcją `load()` i dodaj fragment `+ ' #newsItem'` po argumencie `url`. Gotowy kod powinien wyglądać następująco:**

```
$('#newslinks a').click(function() {
  var url=$(this).attr('href');
  $('#headlines').load(url + ' #newsItem');
  return false;
});
```

   Na stronie 480 przeczytałeś, że można określić, który fragment pobranego pliku funkcja `load()` ma dodać do strony. W tym celu należy dodać odstęp po adresie URL, a następnie podać selektor wskazujący obszar, który chcesz wyświetlić.

   Oto opis użytego kodu. Na każdej z pobieranych stron znajduje się znacznik `<div>` o identyfikatorze `newsItem`. Zawiera on potrzebny kod HTML z informacjami. Dlatego należy nakazać funkcji `load()` wstawienie tylko tej części wczytanego kodu HTML, dodając do adresu URL przekazanego do tej funkcji odstęp i selektor `#newsItem`. Na przykład jeśli zechcesz pobrać plik *today.html* i umieścić w znaczniku `<div>` o identyfikatorze `headlines` tylko tag `<div>` o identyfikatorze `newsItem`, powinieneś użyć następującego kodu:

```
$('#headlines').load('today.html #newsItem');
```

   Tu trzeba połączyć dwa łańcuchy znaków — zawartość zmiennej `url` i selektor `'#newsItem'`, określający potrzebny kod. Dlatego w implementacji `load(url + '#newsItem')` użyto operatora łączenia łańcuchów znaków (znaku +). (Jeśli chcesz przypomnieć sobie informacje o scalaniu łańcuchów znaków, zajrzyj na stronę 69.)

8. **Zapisz plik.**

   Jeśli zainstalowałeś na swoim komputerze serwer WWW, będziesz mógł wyświetlić stronę w przeglądarce. Kliknij jakiś odnośnik, aby sprawdzić jego działanie. Teraz w ramce widocznej w środkowej części strony powinny pojawić się wiadomości — i tylko one — z każdego pliku wskazanego w odnośnikach. Dodałeś AJAX za pomocą tylko kilku wierszy kodu! (Gotową wersję tego przykładu znajdziesz w pliku *complete_load.html* w katalogu *R13*.)

Metody get() i post()

Medota `load()` (opisana w poprzednim punkcie rozdziału) to prosty sposób na pobranie z serwera WWW kodu HTML i dodanie go do strony. Jednak serwer nie zawsze zwraca zwykły kod HTML. Może też przesłać komunikat tekstowy, numer kodowy lub dane, które trzeba przetworzyć za pomocą kodu JavaScript. Jeśli chcesz użyć AJAX-a do pobrania rekordów z bazy danych, serwer może zwrócić plik XML z tymi rekordami (patrz ramka na stronie 495) lub obiekt w formacie JSON (patrz strona 500). Nie należy po prostu dodawać takich danych do strony. Najpierw trzeba je przetworzyć i wygenerować potrzebny kod HTML.

Metody `get()` i `post()` biblioteki jQuery to proste narzędzia do przesyłania danych na serwer oraz pobierania z niego informacji. Jak wspomniano w kroku 2. na stronie 477, obiektem `XMLHttpRequest` należy zarządzać nieco inaczej przy korzystaniu z metod `GET` i `POST`. Jednak biblioteka jQuery automatycznie obsługuje różnice

między nimi, dlatego funkcje `get()` i `post()` działają identycznie. Której z nich powinieneś używać? Odpowiedź znajdziesz w ramce na stronie 488.

Podstawowa składnia tych funkcji wygląda następująco:

```
$.get(url, data, callback);
```

lub:

```
$.post(url, data, callback);
```

W odróżnieniu od większości innych funkcji biblioteki jQuery przed funkcjami `get()` i `post()` nie należy dodawać selektora. Oznacza to, że nie wolno używać instrukcji typu `$('#mainContent').get('products.php')`. Te dwie funkcje są samodzielne i nie łączą się z żadnym elementem strony, dlatego wystarczy użyć symbolu `$`, kropki i wywołania `get` lub `post` (na przykład `$.get()`).

Funkcje `get()` i `post()` przyjmują trzy argumenty: `url` to łańcuch znaków zawierający ścieżkę do skryptu przetwarzającego dane po stronie serwera (na przykład `'processForm.php'`); argument `data` to albo łańcuch znaków, albo literał obiektowy języka JavaScript z danymi przesyłanymi na serwer (w następnym punkcie dowiesz się, jak przygotować takie dane); argument ostatni, `callback`, to funkcja przetwarzająca instrukcje zwrócone przez serwer (szczegółowe informacje o funkcjach zwrotnych znajdziesz na stronie 223).

W momencie wywołania funkcji `get()` lub `post()` przeglądarka wysyła określone dane pod wskazany adres URL. Kiedy serwer prześle dane z powrotem do przeglądarki, ta przekazuje je do funkcji zwrotnej, która z kolei przetwarza zawarte w nich informacje i zazwyczaj w jakiś sposób aktualizuje zawartość strony. Będziesz miał okazję zobaczyć takie rozwiązanie w działaniu na stronie 495.

Formatowanie danych przesyłanych na serwer

Programy JavaScript używające AJAX-a zwykle przesyłają dane na serwer. Aby na przykład pobrać informacje na temat produktu zapisanego w bazie danych, trzeba przesłać numer reprezentujący dany towar. Kiedy serwer otrzyma liczbę w żądaniu od obiektu `XHR`, wyszuka w bazie pasujący do tego numeru produkt, pobierze informacje i prześle je z powrotem do przeglądarki. Możesz też użyć AJAX-a do przesłania informacji z całego formularza z zamówieniem lub danymi abonenta biuletynu rozsyłanego pocztą elektroniczną.

Dane przesyłane w żądaniu trzeba tak sformatować, aby były zrozumiałe dla funkcji `get()` i `post()`. Drugi argument przekazywany do każdej z tych funkcji zawiera wysyłane dane. Mogą one mieć format łańcucha znaków z zapytaniem lub literału obiektowego języka JavaScript. Możliwości te opisano w dwóch następnych podpunktach.

Łańcuch znaków z zapytaniem

Prawdopodobnie widziałeś już wiele łańcuchów znaków z zapytaniem. Często pojawiają się one na końcu adresu URL, po symbolu `?`, na przykład w adresie *www.chia-vet.com/products.php?prodID=18&sessID=1234*. Ten łańcuch znaków

z zapytaniem zawiera dwie pary nazwa – wartość: `prodID=18` i `sessID=1234`. Jest to odpowiednik utworzenia dwóch zmiennych, `prodID` i `sessID`, oraz zapisania w nich wartości. Łańcuch znaków z zapytaniem to standardowa technika przekazywania informacji w adresach URL.

W ten sposób można przesyłać dane na serwer także za pomocą AJAX-a. Przyjmijmy, że utworzyłeś stronę, na której użytkownicy mogą ocenić film przez zaznaczenie określonej liczby gwiazdek. Kliknięcie pięciu gwiazdek powoduje przesłanie na serwer oceny „pięć". Wysyłane dane mogą wyglądać następująco: `rating=5`. Jeśli nazwa strony przetwarzającej oceny to *rateMovie.php*, kod przesyłający dane na serwer za pomocą AJAX-a powinien wyglądać następująco:

```
$.get('rateMovie.php','rating=5');
```

Jeśli używasz metody `post`, skorzystaj z poniższego kodu:

```
$.post('rateMovie.php','rating=5');
```

Uwaga: Funkcje `get()` i `post()` biblioteki jQuery nie wymagają definiowania danych ani funkcji wywoływanych zwrotnie. Wystarczy przekazać adres URL strony działającej po stronie serwera, jednak prawie zawsze podawane są także dane. Na przykład w kodzie `$.get('rankMovie.php','rating=5');` podano tylko adres URL i dane. Nie ma tu wywoływanej zwrotnie funkcji. Użytkownik jedynie ocenia film, dlatego serwer nie musi zwracać odpowiedzi, a wywoływana zwrotnie funkcja — wykonywać żadnych operacji.

CZĘSTO ZADAWANE PYTANIA

Metoda GET czy POST?

Dwie metody przesyłania danych na serwer WWW, GET i POST, wyglądają bardzo podobnie. Której z nich powinienem używać?

Trudno udzielić jednoznacznej odpowiedzi na to pytanie. W niektórych sytuacjach programista nie ma wyboru. Załóżmy, że przesyłasz informacje do gotowego skryptu działającego po stronie serwera. Oznacza to, że wystarczy użyć kodu JavaScript, aby nawiązać komunikację z utworzonym wcześniej skryptem. Wtedy trzeba użyć metody oczekiwanej przez gotowy program. Zwykle programiści tak tworzą skrypty, aby przyjmowały dane przesłane albo metodą `GET`, albo metodą `POST`. Dlatego należy porozmawiać z autorem danego skryptu lub zajrzeć do kodu i sprawdzić, której metody używa. Następnie trzeba użyć odpowiedniej funkcji biblioteki jQuery — `get()` lub `post()`.

Jeśli skrypt działający po stronie serwera nie jest jeszcze gotowy, możesz wybrać sposób komunikacji. Metoda `GET` jest przeznaczona głównie do przesyłania żądań, które nie zmieniają stanu bazy danych i plików na serwerze. Oznacza to, że służy do *pobierania* informacji, na przykład żądania ceny danego produktu lub listy najpopularniejszych towarów. Metoda `POST` jest przeznaczona do wysyłania danych modyfikujących informacje po stronie serwera. W ten sposób można zażądać usunięcia pliku, zaktualizowania bazy danych lub wstawienia do niej nowych informacji.

W praktyce można wymiennie używać obu metod, dlatego programiści często stosują metodę `GET` do usuwania danych z bazy, a metody `POST` — do pobierania informacji z serwera. Jednak w pewnej konkretnej sytuacji metoda `POST` jest niezbędna. Jeśli przesyłasz na serwer duży zbiór danych z formularza (na przykład składający się z setek słów artykuł w blogu), użyj właśnie tego sposobu. Metoda `GET` ma wbudowane ograniczenie ilości przesyłanych danych. Jest ono różne w poszczególnych przeglądarkach, jednak limit narzucany przez Internet Explorera wynosi kilka tysięcy znaków. Do przesyłania danych z formularzy zawierających więcej niż kilka pól programiści zazwyczaj używają metody `POST`.

Jeśli chcesz przesłać na serwer więcej niż jedną parę nazwa – wartość, dodaj między parami znak &:

```
$.post('rateMovie.php','rating=5&user=Robert');
```

Musisz jednak zachować staranność przy korzystaniu z tej metody, ponieważ niektóre znaki w łańcuchach zapytania mają specjalne znaczenie. Na przykład symbol & pozwala dołączyć następną parę nazwa – wartość, a znak = przypisuje wartość do nazwy. Na przykład poniższy łańcuch jest nieprawidłowy:

```
'favFood=Mac & Cheese' // Błąd.
```

Symbol „&" miał tu być częścią wartości „Mac & Cheese", jednak zostanie potraktowany jak początek drugiej pary nazwa – wartość. Jeśli chcesz użyć znaków specjalnych w nazwie lub wartości, musisz użyć *sekwencji ucieczki*, czyli *zakodować* dany symbol, aby nie został uznany za znak o specjalnym znaczeniu. Na przykład odstępowi odpowiada sekwencja %20, symbol & ma kod %26, a znak = to %3D. Dlatego parę z wartością „Mac & Cheese" należy przepisać w następujący sposób:

```
'favFood=Mac%20%26%20Cheese' // Prawidłowo zakodowane.
```

JavaScript udostępnia metodę encodeURIComponent(), która służy do kodowania znaków w łańcuchach. Należy przekazać do niej łańcuch, a metoda zwróci jego poprawnie zakodowaną wersję, na przykład:

```
var queryString = 'favFood=' + encodeURIComponent('Mac & Cheese');
$.post('foodChoice.php', queryString);
```

Literały obiektowe

Łańcuchy znaków z zapytaniem dobrze nadają się do przesyłania krótkich i prostych fragmentów danych, które nie zawierają żadnych znaków specjalnych. Jednak bezpieczniejsza metoda obsługiwana przez funkcje get() i post() biblioteki jQuery polega na zapisywaniu danych w literałach obiektowych. Na stronie 165 dowiedziałeś się, że literały obiektowe języka JavaScript umożliwiają przechowywanie par nazwa – wartość. Podstawowa struktura takiego literału wygląda następująco:

```
{
  nazwa1: 'wartość1',
  nazwa2: 'wartość2'
}
```

Literał obiektowy można przekazać bezpośrednio do funkcji get() lub post(). W poniższym kodzie użyto łańcucha znaków z zapytaniem:

```
$.post('rateMovie.php','rating=5');
```

Aby użyć literału obiektowego, należy wprowadzić następujące zmiany:

```
$.post('rateMovie.php', { rating: 5 });
```

Literały obiektowe można przekazać bezpośrednio do funkcji get() lub post() albo najpierw zapisać w zmiennej, a następnie użyć w jednej z omawianych metod:

```
var data = { rating: 5 };
$.post('rankMovie.php', data);
```

W obiekcie przekazywanym do funkcji get() lub post() można oczywiście umieścić dowolną liczbę par nazwa – wartość:

```
var data = {
  rating: 5,
  user: 'Robert'
}
$.post('rankMovie.php', data);
```

Literał obiektowy można także przekazać bezpośrednio w wywołaniu funkcji `post()`:

```
$.post('rankMovie.php',
  {
    rating: 5,
    user: 'Robert'
  }
); // koniec post
```

Funkcja serialize() biblioteki jQuery

Tworzenie łańcucha znaków z zapytaniem lub literału obiektowego z parami nazwa – wartość dla wszystkich pól formularza bywa pracochłonne. Trzeba pobrać nazwę i wartość każdego elementu formularza, a następnie połączyć je w długi łańcuch znaków lub duży literał obiektowy języka JavaScript. Na szczęście jQuery udostępnia funkcję, która ułatwia przekształcanie informacji z formularza na dane zrozumiałe dla funkcji `get()` i `post()`.

Funkcję `serialize()` możesz zastosować do dowolnego formularza, a nawet do wybranych pól, aby utworzyć potrzebny łańcuch znaków z zapytaniem. W celu użycia jej najpierw pobierz formularz za pomocą biblioteki jQuery, a następnie wywołaj dla niego funkcję `serialize()`. Załóżmy, że strona zawiera formularz o identyfikatorze `login`. Jeśli zechcesz utworzyć łańcuch znaków z zapytaniem obejmujący dane z tego formularza, użyj następującego kodu:

```
var formData = $('#login').serialize();
```

Fragment `var formData` tworzy nową zmienną, wyrażenie `$('#login')` znajduje formularz za pomocą biblioteki jQuery, a wywołanie `.serialize()` pobiera nazwy i aktualne wartości pól formularza, po czym tworzy pojedynczy łańcuch znaków z zapytaniem.

Aby użyć tego łańcucha w funkcji `get()` lub `post()`, należy przekazać go do wybranej funkcji jako drugi argument — po adresie URL. Jeśli chcesz wysłać zawartość formularza logowania do strony *login.php*, możesz to zrobić za pomocą poniższego kodu:

```
var formData = $('#login').serialize();
$.get('login.php', formData,loginResults);
```

Ten kod przesyła dane wprowadzone przez użytkownika w formularzu do pliku *login.php* za pomocą metody `GET`. Ostatni argument tej metody, `loginResults`, to wywoływana zwrotnie funkcja. Przyjmuje ona dane zwrócone przez serwer i używa ich do wykonania odpowiednich operacji. Wkrótce dowiesz się, jak tworzyć takie funkcje.

Przetwarzanie danych zwróconych z serwera

AJAX to technologia dwustronna. Program JavaScript przesyła dane na serwer, który z kolei zwraca informacje do programu. Wtedy skrypt może użyć zwróconych danych do zaktualizowania strony. W poprzednich punktach zobaczyłeś, jak sformatować

dane i przesłać je na serwer za pomocą funkcji `get()` i `post()`. Teraz dowiesz się, jak odbierać i przetwarzać odpowiedzi zwrócone przez serwer.

Kiedy przeglądarka wysyła żądanie na serwer za pomocą obiektu `XMLHttpRequest`, oczekuje na odpowiedź. Jak serwer ją prześle, wywoływana jest funkcja zwrotna, która obsługuje pobrane dane. Funkcja ta przyjmuje kilka argumentów. Pierwszy i najważniejszy z nich to informacje zwrócone przez serwer.

Odpowiedź przesyłaną przez serwer można sformatować na wiele sposobów. Skrypt działający po stronie serwera może zwrócić liczbę, słowo, akapit tekstu lub kompletną stronę WWW. Jeśli serwer przesyła dużo informacji (na przykład zbiór rekordów z bazy danych), często używany jest format XML lub JSON (XML opisano w ramce na stronie 495, a omówienie formatu JSON znajdziesz na stronie 500).

Drugi argument funkcji zwrotnej to łańcuch znaków określający status odpowiedzi. Przeważnie informuje on o udanym przetworzeniu żądania i zwróceniu danych. Jednak czasem obsługa żądania kończy się niepowodzeniem. Wynika to z różnych przyczyn. Możliwe, że żądany plik nie istnieje lub wystąpił błąd w skrypcie działającym po stronie serwera. Jeśli tak się stanie, wywoływana zwrotnie funkcja otrzyma jako status komunikat o błędzie.

Funkcja zwrotna przetwarza pobrane informacje i zazwyczaj aktualizuje stronę WWW, na przykład zastępuje przesłany formularz danymi z serwera lub wyświetla komunikat typu „Przetwarzanie żądania zakończyło się powodzeniem". Aktualizowanie zawartości strony jest proste — wystarczy użyć funkcji `html()` i `text()` biblioteki jQuery (patrz strona 157). Inne metody manipulowania modelem DOM strony opisano w rozdziale 4.

Aby zrozumieć cały cykl zgłaszania żądania i przetwarzania odpowiedzi, przyjrzyj się prostemu przykładowi oceniania filmu (patrz rysunek 13.6). Użytkownik może dokonać oceny przez kliknięcie jednego z pięciu odnośników. Każdy z nich oznacza inną liczbę punktów. Kiedy użytkownik wybierze odsyłacz, skrypt prześle ocenę i identyfikator filmu do programu działającego po stronie serwera. Program ten dodaje liczbę punktów do bazy, a następnie zwraca średnią ocenę danego filmu, która jest wyświetlana na stronie.

Aby opisane rozwiązanie funkcjonowało bez kodu JavaScript, każdy odnośnik musi prowadzić do działającej na serwerze strony, która potrafi przetworzyć ocenę. Na przykład w odsyłaczu do pięciogwiazdkowej oceny (patrz rysunek 11.6) może to być strona *rate.php?rate=5&movie=123*. Nazwa pliku przetwarzającego oceny to *rate.php*, a łańcuch znaków z zapytaniem (*?rate=5&movie=123*) obejmuje dwie porcje informacji dla serwera — ocenę (*rate=5*) i liczbę, która określa oceniany firm (*movie=123*). Można użyć kodu JavaScript do przechwytywania kliknięć tych odnośników i przekształcania ich na ajaksowe wywołania kierowane na serwer:

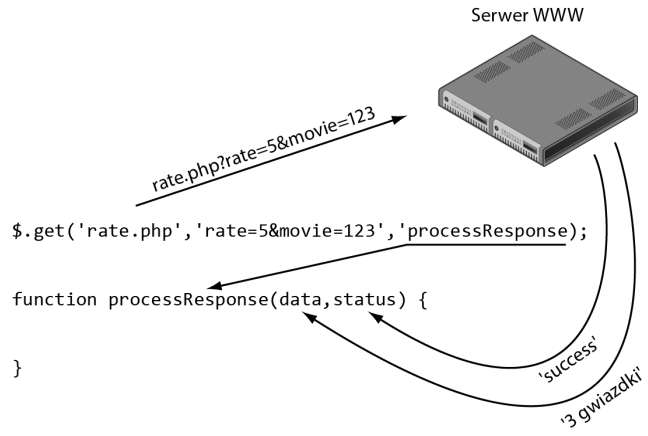
```
1 $('#message a').click(function() {
2   var href=$(this).attr('href');
3   var querystring=href.slice(href.indexOf('?')+1);
4   $.get('rate.php', querystring, processResponse);
5   return false; // Blokowanie działania odnośnika.
6 });
```

Rysunek 13.6. *Na tej stronie użytkownik może kliknąć odnośnik, aby ocenić film (na górze). Przy użyciu AJAX-a można przesłać ocenę na serwer bez opuszczania strony. Na podstawie odpowiedzi zwróconej przez serwer można następnie zaktualizować zawartość strony (na dole)*

Wiersz 1. pobiera wszystkie odnośniki (znaczniki `<a>`) z tagu o identyfikatorze `message` (tu każdy odsyłacz służący do oceny filmu znajduje się w znaczniku `<div>` o takim identyfikatorze). Następnie skrypt przypisuje funkcję do zdarzenia `click` pobranych odnośników.

Wiersz 2. pobiera atrybut `HREF` odnośnika i przypisuje do zmiennej `href` adresy URL typu `rate.php?rate=5&movie=123`. Wiersz 3. zapisuje fragment tego adresu znajdujący się po znaku `?`. Służy do tego metoda `slice()` (patrz strona 569), która pobiera fragment łańcucha znaków, i metoda `indexOf()` (patrz strona 567), określająca pozycję znaku `?` (metoda `slice()` używa tej informacji do ustalenia miejsca rozpoczęcia pobierania łańcucha).

Wiersz 4. to ajaksowe żądanie. Skrypt przesyła je do strony *rate.php* z serwera (patrz rysunek 13.7), używając metody `GET` i łańcucha znaków z zapytaniem. Zwrócone dane trafiają do wywoływanej zwrotnie funkcji `processResponse()`. Wiersz 5. blokuje domyślne działanie odnośników i zapobiega przejściu przeglądarki do strony wskazanej w odnośniku.



Rysunek 13.7. *Ten rysunek ilustruje interakcję między kodem JavaScript a serwerem WWW. Funkcja get() przesyła informacje na serwer, natomiast funkcja zwrotna, processResponse(), obsługuje dane zwrócone przez serwer*

Uwaga: Jeśli chcesz przypomnieć sobie działanie funkcji i sposoby ich tworzenia, wróć do strony 115.

Pora utworzyć funkcję zwrotną. Przyjmuje ona dane i łańcuch znaków ze statusem odpowiedzi (jeśli serwer zwrócił informacje, ma ona wartość `'success'`). Pamiętaj, że nazwę funkcji zwrotnej należy określić w żądaniu (wiersz 4. kodu ze strony 491). Tu ta nazwa to `processResponse`. Kod do obsługi odpowiedzi zwróconej przez serwer może wyglądać następująco:

```
1  function processResponse(data) {
2    var newHTML;
3    newHTML = '<h2>Twój głos został dodany.</h2>';
4    newHTML += '<p>Średnia ocena filmu to ';
5    newHTML += data + '.</p>';
6    $('#message').html(newHTML);
7  }
```

Funkcja ta przyjmuje argument `data`, zawierający informacje zwrócone przez serwer. Informacje te mogą być zapisane w formie zwyczajnego tekstu, kodu HTML, XML lub w formacie JSON. Wiersz 2. tworzy nową zmienną, która przechowuje kod HTML wyświetlany na stronie (na przykład „Twój głos został zapisany."). W wierszach 3. i 4. w zmiennej `newHTML` zapisywany jest kod HTML, zawierający znaczniki `<h2>` i `<p>`. Odpowiedź serwera (zapisana w zmiennej `data`) jest używana dopiero w wierszu 5., gdzie skrypt dodaje ją do zmiennej `newHTML`. Tu serwer zwraca łańcuch znaków ze średnią oceną filmu, na przykład `'3 gwiazdki'`.

Uwaga: Jeśli chcesz dodać do witryny system oceny za pomocą gwiazdek, możesz użyć do tego doskonałej wtyczki biblioteki jQuery, która obsługuje większość szczegółowych operacji (*http://www.wbotelhos.com/raty/*).

Wiersz 6. modyfikuje kod HTML strony za pomocą funkcji `html()` biblioteki jQuery (patrz strona 157). Skrypt zastępuje zawartość znacznika `<div>` o identyfikatorze `message` nowym kodem HTML. Przykładowy efekt przedstawia dolna część rysunku 13.6.

W tym przykładzie funkcję zwrotną zdefiniowano poza funkcją `get()`. Jednak jeśli chcesz umieścić cały kod związany z AJAX-em w jednym miejscu, możesz użyć funkcji anonimowej (patrz strona 168):

```
$.get('file.php', data, function(data,status) {
  // Tu kod wpisz funkcji zwrotnej.
});
```

Niżej pokazałem, w jaki sposób można zmienić 4. wiersz kodu ze strony 491, tak by korzystał z funkcji anonimowej:

```
$.get('rate.php', querystring, function(data) {
  var newHTML;
  newHTML = '<h2>Twój głos został dodany.</h2>';
  newHTML += '<p>Średnia ocena filmu to ';
  newHTML += data + '.</p>';
  $('#message').html(newHTML);
}); // koniec get
```

Obsługa błędów

Niestety nie wszystko zawsze idzie zgodnie z planem. Podczas korzystania z technologii AJAX w celu prowadzenia wymiany danych z serwerem mogą pojawić się problemy. Może się zdarzyć, że w danej chwili serwer będzie niedostępny bądź połączenie komputera użytkownika z internetem zostanie przerwane. W takich przypadkach wywołanie metod `$.get()` i `$.post()` zakończy się niepowodzeniem, a użytkownik się o tym nie dowie. Choć problemy tego typu pojawiają się sporadycznie, jednak warto się na nie przygotować i informować użytkowników o chwilowych problemach, gdyż to może im pomóc w ustaleniu, co mają zrobić (na przykład odświeżyć stronę, podjąć próbę wykonania operacji jeszcze raz bądź wrócić na stronę po jakimś czasie).

Aby reagować na błędy, wystarczy za wywołaniem funkcji `$.get()` lub `$.post()` umieścić wywołanie funkcji `.error()`. Podstawowa struktura takiego kodu powinna wyglądać tak:

```
$.get(url, dane, funObslugiPowodzenia).error(funObslugiBledow)
```

Przykładowo 4. wiersz przykładu zamieszczonego na stronie 491 można by zmodyfikować w następujący sposób:

```
$.get('rate.php', querystring, processResponse).error(errorResponse);
```

Następnie należałoby zdefiniować funkcję o nazwie `errorResponse()`, która informowałaby użytkownika o zaistniałych problemach. Oto przykład takiej funkcji:

```
function errorResponse() {
  var errorMsg = "Nie można było przetworzyć Twojej oceny. ";
  errorMsg += "Spróbuj ponownie później.";
  $('#message').html(errorMsg);
}
```

W tym przypadku funkcja `errorResponse()` zostanie wywołana wyłącznie w przypadku wystąpienia jakiegoś problemu z serwerem lub połączeniem z internetem.

Uwaga: Metoda `.error()` nie działa z metodą `.load()` ani z żądaniami przesyłanymi na inne witryny przy wykorzystaniu JSONP (patrz strona 506).

PORADNIA DLA ZAAWANSOWANYCH

Pobieranie kodu XML z serwera

XML to popularny format do przesyłania danych między komputerami. W języku XML, podobnie jak HTML-u, informacje są zapisane w znacznikach. Różnica polega na tym, że w XML-u można samodzielnie tworzyć tagi, które dokładnie odzwierciedlają treść danych. Na przykład prosty plik XML może wyglądać następująco:

```
<?xml version="1.0"?>
<message id="234">
  <from>Robert</from>
  <to>Żaneta</to>
  <subject>Witaj, Żaneto</subject>
  <content>Żaneto, wyskoczmy dziś
    na lunch.</content>
</messages>
```

Główny znacznik (tak zwany *element główny*; to odpowiednik znacznika `<html>` z kodu HTML), `<message>`, i kilka dodatkowych tagów określają znaczenie zapisanych w nich danych.

Działający na serwerze program może zwracać do skryptu ajaksowego plik w formacie XML. Biblioteka jQuery ułatwia odczyt i pobieranie danych z takich plików. Jeśli używasz metod `$.get()` lub `$.post()`, a serwer zwraca informacje w formacie XML, argument `data` przekazywany do funkcji zwrotnej będzie zawierał model DOM pliku XML. Oznacza to, że jQuery wczyta plik XML i potraktuje go jak dowolny inny dokument. Następnie można użyć selektora jQuery, aby uzyskać dostęp do informacji z tego pliku.

Załóżmy, że działający na serwerze plik *xml.php* zwraca przedstawione wcześniej dane w formacie XML, a skrypt ma pobierać tekst ze znacznika `<content>`. Plik XML to zwracane dane, dlatego można go przetworzyć w funkcji zwrotnej. Za pomocą funkcji `find()` i standardowych selektorów biblioteki jQuery należy znaleźć odpowiednie dane z tego pliku. Możesz użyć do tego selektorów elementów, klas, identyfikatorów i potomków (patrz strona 151), a także filtrów biblioteki jQuery (patrz strona 153).

Oto przykład:

```
$.get('xml.php','id=234',processXML);
function processXML(data){
  var messageContent=$(data).
    find('content').text();
}
```

Kluczowy jest tu fragment `$(data).find('content')`, który nakazuje bibliotece jQuery pobranie wszystkich znaczników `<content>` ze zmiennej `data`. Zmienna ta zawiera plik XML, zatem kod sprawia, że jQuery znajdzie znacznik `<content>` w danych XML.

Aby lepiej poznać format XML, odwiedź stronę *http://www.learn-xml-tutorial.com/xml-basics.cfm*. Dodatkowe informacje o funkcji `find()` biblioteki jQuery znajdziesz na stronie *http://api.jquery.com/find/*.

Przykład — korzystanie z metody $.get()

W tym przykładzie użyjesz AJAX-a do przesyłania danych z formularza logowania. Kiedy użytkownik poda odpowiednią nazwę i właściwe hasło, pojawi się komunikat informujący o udanym logowaniu. Jeśli dane uwierzytelniające są nieprawidłowe, na tej samej stronie (bez wczytywania nowego dokumentu) znajdzie się komunikat o błędzie.

Uwaga: Aby uruchomić ten przykład, musisz użyć serwera AMP (Apache, MySQL i PHP), w którym będziesz mógł przetestować strony. W ramce na stronie 476 znajdziesz informacje o instalowaniu serwera na potrzeby testów.

Omówienie przykładu

Rozpoczniesz pracę od formularza widocznego na rysunku 13.8. Zawiera on pola na nazwę użytkownika i hasło przesyłane na serwer. Kiedy internauta prześle formularz, serwer sprawdzi, czy określony użytkownik istnieje i czy podano prawidłowe hasło. Jeśli dane uwierzytelniające są poprawne, serwer zaloguje użytkownika.

Rysunek 13.8. *Strona logowania jest całkiem prosta — zawiera kilka pól i przycisk Wyślij. Nie ma powodu, aby opuszczać tę stronę po zalogowaniu się użytkownika. Za pomocą AJAX-a można przesłać dane uwierzytelniające, a następnie poinformować internautę o tym, czy logowanie zakończyło się powodzeniem, czy porażką*

Aby obsługiwać formularz przy użyciu AJAX-a, należy przesłać dane uwierzytelniające za pomocą obiektu `XMLHttpRequest`. Serwer zwróci następnie komunikat do wywoływanej zwrotnie funkcji, która usunie formularz i wyświetli informację o udanym logowaniu, jeśli dane były poprawne, lub komunikat o błędzie, jeśli wystąpiły problemy.

Tworzenie kodu

W uwadze na stronie 46 znajdziesz informacje o pobieraniu przykładowych plików. Wyjściowy dokument zawiera kod HTML formularza. Należy do niego dodać kod oparty na jQuery i technologii AJAX.

1. **Otwórz w edytorze tekstu plik *login.html* z katalogu *R13*.**

   Dokument zawiera już kod dołączający bibliotekę jQuery i funkcję `$(document).ready()`. Najpierw należy pobrać formularz i dodać do niego zdarzenie `submit`.

2. **Kliknij pusty wiersz w funkcji `$(document).ready()` i wpisz poniższy kod:**

   ```
   $('#login').submit(function() {

   }); // Koniec funkcji submit
   ```

   Znacznik `<form>` ma identyfikator `login`, dlatego selektor `$('#login')` biblioteki jQuery pobierze formularz, a funkcja `submit()` doda do niego uchwyt zdarzenia `submit`. Oznacza to, że przy próbie przesłania formularza skrypt uruchomi funkcję, którą zaraz utworzysz.

Następny krok wymaga pobrania informacji z formularza i przekształcenia ich na łańcuch znaków z zapytaniem, który będzie można przesłać na serwer. Można to zrobić przez znalezienie każdego pola, określenie wartości wpisanej przez użytkownika i utworzenie łańcucha przez połączenie poszczególnych informacji. Na szczęście metoda `serialize()` biblioteki jQuery pozwala wykonać wszystkie te operacje w jednym kroku.

3. **Umieść kursor w pustym wierszu wewnątrz funkcji dodanej w kroku 2. Wpisz w nim poniższy kod:**

```
var formData = $(this).serialize();
```

Ten wiersz tworzy nową zmienną na dane z formularza, a następnie wywołuje dla tego formularza metodę `serialize()`. Konstrukcja `$(this)` wskazuje na przetwarzany element, którym tu jest formularz logowania (konstrukcja ta oznacza to samo co wyrażenie `$('#login')`; więcej informacji o niej znajdziesz na stronie 169). Metoda `serialize()` (patrz strona 490) pobiera nazwy i wartości pól formularza, a następnie przekształca je przed przesłaniem na serwer na odpowiedni format.

Teraz należy użyć funkcji `$.get()` do skonfigurowania obiektu `XMLHttpRequest` i wysłania żądania.

4. **Wciśnij klawisz *Enter*, aby utworzyć następny pusty wiersz. Wpisz w nim następujący kod:**

```
$.post('login.php',formData,processData);
```

Ta instrukcja przekazuje do metody `$.get()` trzy argumenty. Pierwszy, `'login.php'`, to łańcuch znaków określający lokalizację docelową wysyłanych danych. Tu informacje trafiają do zapisanego na serwerze pliku *login.php*. Drugi argument to łańcuch znaków z zapytaniem zawierający dane uwierzytelniające przesyłane na serwer. Ostatni argument, `processData`, to nazwa funkcji zwrotnej, która będzie przetwarzać odpowiedź serwera. Teraz należy przygotować tę funkcję.

5. **Dodaj następny pusty wiersz i wpisz w nim poniższy kod:**

```
function processData(data) {

} // Koniec funkcji processData.
```

Te wiersze tworzą szkielet funkcji zwrotnej. Na razie nie zawiera ona żadnego kodu. Zauważ, że funkcja ma przyjmować jeden argument (`data`), którym jest odpowiedź serwera. Strona działająca na serwerze zwraca jedno słowo. Jeśli logowanie zakończyło się powodzeniem, jest to łańcuch `pass`. Jeśli wystąpił błąd, strona zwraca słowo `fail`.

Uwaga: Przykładowa strona działająca na serwerze nie jest kompletnym skryptem do obsługi logowania. Reaguje na podanie prawidłowych danych uwierzytelniających, ale nie należy jej stosować w witrynach chronionych hasłem. Istnieje wiele technik zabezpieczania witryn w podobny sposób, jednak większość z nich wymaga przygotowania bazy danych lub skonfigurowania różnych ustawień serwera WWW. Opis tych zagadnień wykracza poza zakres tej książki. Kompletny, oparty na języku PHP skrypt logowania znajdziesz na stronie *http://www.wikihow.com/Create-a-Secure-Login-Script-in-PHP-and-MySQL*.

Skrypt na podstawie odpowiedzi serwera wyświetla odpowiedź z informacją o udanym lub nieudanym logowaniu. Do obsługi takiego rozwiązania doskonale nadaje się instrukcja warunkowa.

6. **W funkcji `processData()` wpisz poniższy kod:**

```
1 function processData(data) {
2   if (data === 'pass') {
3     $('.main').html('<p>Logowanie zakończyło się
powodzeniem!</p>');
4   }
5 } // koniec funkcji processData.
```

Wiersz 2. sprawdza, czy serwer zwrócił łańcuch znaków `'pass'`. Jeśli tak, logowanie zakończyło się powodzeniem i skrypt wyświetli informujący o tym komunikat (wiersz 3.). Formularz znajduje się w znaczniku `<div>` klasy `main`, dlatego instrukcja `$('.main').html('<p> Logowanie zakończyło się powodzeniem!</p>')` zastąpi zawartość tego elementu `<div>` nowym akapitem. Oznacza to, że formularz zniknie, a pojawi się komunikat o udanym logowaniu.

Aby ukończyć przykład, należy dodać klauzulę `else` i poinformować użytkownika, że podał nieprawidłowe dane uwierzytelniające.

7. **Dodaj do funkcji `processData()` klauzulę `else`, aby kod wyglądał następująco (zmiany wyróżniono pogrubieniem):**

```
1 function processData(data) {
2   if (data=='pass') {
3     $('.main').html('<p>Logowanie zakończyło się
powodzeniem!</p>');
4   } else {
5     $('#formwrapper').prepend('<p id="fail">Nieprawidłowe dane
      ↪uwierzytelniające. Spróbuj ponownie.</p>');
6   }
7 } // Koniec funkcji processData.
```

Wiersz 5. wyświetla informację o nieudanym logowaniu. Zauważ, że użyto tu funkcji `prepend()` (patrz strona 159). Umożliwia ona dołączanie kodu na początek elementu. Funkcja ta nie usuwa obecnej zawartości znacznika, ale dodaje nowy kod. Skrypt nie powinien usuwać w tym miejscu formularza, aby użytkownik mógł ponownie spróbować się zalogować.

8. **Zapisz plik i wyświetl go w przeglądarce.**

Aby uruchomić przykład, musisz otworzyć tę stronę w przeglądarce za pomocą adresu URL, na przykład *http://localhost/R11/login.html*. Informacje o instalowaniu serwera WWW znajdziesz na stronie 476.

9. **Spróbuj zalogować się w witrynie.**

Prawdopodobnie pomyślisz sobie: „Jak mam to zrobić, skoro nie otrzymałem nazwy użytkownika ani hasła?". I o to chodzi. Zobacz, co się stanie, jeśli wprowadzisz nieprawidłowe dane. Spróbuj zalogować się po raz drugi. Na stronie pojawi się druga wiadomość „Nieprawidłowe dane uwierzytelniające" (patrz rysunek 13.9). Funkcja `prepend()` nie usuwa pierwszego komunikatu o błędzie, a jedynie dodaje po raz wtóry ten sam tekst. Nie jest to dobre rozwiązanie.

Rysunek 13.9. *Funkcja prepend() biblioteki jQuery dodaje kod HTML do istniejącego elementu. Metoda ta nie usuwa żadnych danych, dlatego skrypt może wielokrotnie dodawać ten sam komunikat*

Problem ten można rozwiązać na kilka sposobów. Można na przykład umieścić pod formularzem nowy, pusty znacznik `<div>` — `<p id= failMessage >`. Następnie, w przypadku nieudanego logowania można po prostu podmieniać umieszczony w nim kod HTML. Jednak w naszym przykładzie na stronie nie ma żadnego pustego znacznika `<div>`. Zamiast niego użyjemy zwyczajnej instrukcji warunkowej, która będzie sprawdzać, czy na stronie został już wyświetlony jakiś komunikat o błędzie — jeśli taki komunikat faktycznie został już wyświetlony, dodawanie go po raz wtóry nie będzie konieczne.

10. **Dodaj następną instrukcję warunkową (wiersze 5. i 7.):**

```
1 function processData(data) {
2   if (data=='pass') {
3     $('.main').html('<p>Logowanie zakończyło się powodzeniem!</p>');
4   } else {
5     if ($('#fail').length==0) {
6       $('#formwrapper').prepend('<p id="fail">Nieprawidłowe dane

        ↪uwierzytelniające. Spróbuj ponownie.</p>');
7     }
8   }
9 } // Koniec funkcji processData.
```

Zauważ, że akapit z komunikatem o błędzie ma identyfikator, `fail`, dlatego można użyć jQuery do sprawdzenia, czy element o takim identyfikatorze znajduje się na stronie. Jeśli nie, skrypt dodaje do dokumentu odpowiedni komunikat. Aby sprawdzić, czy element znajduje się na stronie, można spróbować pobrać go za pomocą biblioteki jQuery, a następnie sprawdzić wartość atrybutu `length` kolekcji znalezionych znaczników. Jeśli jQuery nie znajdzie żadnych

elementów, wartość tego atrybutu będzie równa `0`. Wyrażenie `$('#fail')` pobiera element o identyfikatorze `fail`. Jeśli jQuery go nie znajdzie (czyli komunikatu o błędzie nie ma na stronie), atrybut `length` będzie miał wartość `0`, warunek będzie prawdziwy, a program wyświetli informacje o nieudanym logowaniu. Po dodaniu komunikatu o błędzie warunek będzie nieprawdziwy, a skrypt nie doda następnej wiadomości.

Teraz trzeba poinformować przeglądarkę o tym, że nie powinna przesyłać formularza z danymi, ponieważ skrypt zrobił to już za pomocą AJAX-a.

11. **Dodaj instrukcję `return false;` na końcu funkcji obsługującej zdarzenie `submit` (wiersz 14. poniżej). Gotowy skrypt powinien wyglądać następująco:**

```
1  $(document).ready(function() {
2  $('#login').submit(function() {
3    var formData = $(this).serialize();
4    $.post('login.php',formData,processData);
5    function processData(data) {
6      if (data=='pass') {
7        $('.main').html('<p>Logowanie zakończyło się
         ➥powodzeniem!</p>');
8      } else {
9        if ($('#fail').length==0) {
10         $('#formwrapper').prepend('<p id="fail">Nieprawidłowe dane
           ➥uwierzytelniające. Spróbuj ponownie.</p>');
11       }
12     }
13   } // Koniec funkcji processData.
14   return false;
15 }); // Koniec funkcji submit
16 }); // Koniec funkcji ready
```

12. **Zapisz plik i ponownie wyświetl stronę.**

    Ponownie spróbuj się zalogować. Prawidłowe dane to nazwa `007` i hasło `secret`. Gotową wersję tego przykładu zawiera plik *kompletny_login.html* z katalogu *R13*.

Uwaga: Jak wspomniano na stronie 486, funkcje `$.post()` i `$.get()` biblioteki jQuery działają identycznie, choć na zapleczu jQuery wykonuje dwa różne zestawy operacji, żeby ajaksowe żądania funkcjonowały prawidłowo. Aby się o tym przekonać, zmień w skrypcie funkcję `post` na `get` (wiersz 4. w kroku 11.). Program działający po stronie serwera obsługuje zarówno żądania `GET`, jak i `POST`.

Format JSON

Inny popularny format do przesyłania danych z serwera to JSON (ang. *JavaScript Object Notation*, czyli notacja obiektowa języka JavaScript). Struktura formatu JSON odpowiada obiektom języka JavaScript. Format ten — podobnie jak XML (patrz ramka na stronie 495) — służy do przesyłania danych. Łańcuch znaków zapisany w formacie JSON przypomina obiekty JavaScript: przeglądarki mogą szybko i łatwo konwertować je na prawdziwe obiekty JavaScript. W odróżnieniu od formatu JSON, dane zapisane w języku XML muszą być analizowane jako seria „węzłów" — kod JavaScript musi je kolejno przejść, aby odczytać umieszczone w nich dane. Ogólnie rzecz biorąc, ten proces trwa dłużej i wymaga napisania

znacznie bardziej rozbudowanego kodu. Właśnie z tego powodu podczas przekazywania danych przy użyciu technologii AJAX znacznie częściej wybieranym formatem jest JSON.

Format JSON przypomina literały obiektowe języka JavaScript bądź pary nazwa – wartość. Oto przykładowe dane w tym formacie:

```
{
  "firstName": "Franciszek",
  "lastName": "Nowak",
  "phone": "503-555-121"
}
```

Symbol { oznacza początek obiektu JSON, a znak } określa jego koniec. Między tymi symbolami znajdują się pary nazwa – wartość, na przykład "firstName": "Franciszek". Nazwa właściwości jest oddzielona do jej wartości znakiem dwukropka. W odróżnieniu od zwyczajnych literałów obiektowych JavaScriptu, w formacie JSON nazwy właściwości muszą być zapisywane w cudzysłowach; w ten sam sposób muszą być zapisywane także wszystkie łańcuchy znaków. Innymi słowy, poniższy zapis nie jest prawidłowy:

```
firstName: 'Franciszek'
```

Natomiast ten jest:

```
"firstName": "Franciszek"
```

Podobnie jak w zwyczajnych literałach obiektowych JavaScriptu, także i w formacie JSON poszczególne pary należy rozdzielać przecinkami, nie należy jednak umieszczać tego znaku po ostatniej parze (bo niektóre wersje Internet Explorera zgłoszą błąd).

Pary nazwa – wartość możesz traktować jak zmienne. Nazwa to nazwa zmiennej, a wartość to przechowywane w niej dane. We wcześniejszym fragmencie `lastName` to odpowiednik zmiennej, w której zapisano łańcuch znaków `'Nowak'`.

Kiedy serwer WWW reaguje na żądania ajaksowe, nie przesyła kodu JavaScript, a jedynie tekst sformatowany w odpowiedni sposób. Do momentu przekształcenia go na rzeczywisty obiekt JSON nie jest to gotowa do użytku jednostka kodu JavaScript. Na szczęście jQuery udostępnia specjalną funkcję, `$.getJSON()`, która obsługuje wszystkie szczegółowe operacje. Funkcja `$.getJSON()` wygląda i działa bardzo podobnie jak funkcje `$.get()` i `$.post()`. Jej składnia przedstawia się następująco:

```
$.getJSON(url, data, callback);
```

Funkcja ta przyjmuje te same trzy argumenty co funkcje `$.post()` i `$.get()` — adres URL strony działającej po stronie serwera, dane przekazywane do tej strony i nazwę wywoływanej zwrotnie funkcji. Różnica polega na tym, że funkcja `$.getJSON()` przetwarza odpowiedź serwera (łańcuch znaków zapisany w formacie JSON) i przekształca ją za pomocą skomplikowanego kodu JavaScript na gotowy do użytku obiekt JavaScript.

Funkcja `$.getJSON()` działa podobnie jak `$.post()` i `$.get()`, ale dane przekazane do funkcji zwrotnej są obiektem JavaScript. Aby użyć metody `$.getJSON()`, trzeba tylko zrozumieć, jak przetwarzać takie dane w funkcjach zwrotnych. Załóżmy, że chcesz użyć AJAX-a do zażądania informacji o danej osobie z zapisanego

na serwerze pliku *contacts.php*. Plik ten zwraca dane kontaktowe w formacie JSON (patrz przykładowy obiekt JSON na poprzedniej stronie). Proste żądanie może wyglądać następująco:

```
$.getJSON('contacts.php','contact=123',processContacts);
```

Ten kod przesyła łańcuch znaków z zapytaniem (`contact=123`) do strony *contacts.php*. Plik *contacts.php* korzysta z tych informacji do znalezienia danych kontaktowych w bazie i pobrania ich, a następnie przesyła je do przeglądarki, gdzie dane trafiają do funkcji zwrotnej `processContacts`. Podstawowa struktura takiej funkcji wygląda następująco:

```
function processContacts(data) {

}
```

Funkcja `processContacts()` przyjmuje jeden argument, `data`, który zawiera obiekt JavaScript zwrócony przez serwer. Zobaczmy, jak funkcja ta może uzyskać dostęp do informacji z tego obiektu.

Dostęp do danych z obiektów JSON

Są dwa sposoby na uzyskanie dostępu do obiektu JSON: *składnia z kropką* i *notacja tablicowa*. Składnia z kropką (patrz strona 87) służy do wskazywania właściwości obiektu. Kropkę należy umieścić między nazwą danego obiektu a potrzebną właściwością. Używałeś tej techniki do pobierania właściwości wielu różnych obiektów języka JavaScript, między innymi łańcuchów znaków i tablic. Na przykład instrukcja `'abc'.length` sprawdza właściwość `length` łańcucha znaków i zwraca liczbę liter w tekście `'abc'`, czyli wartość `3`.

Poniższy kod tworzy zmienną i zapisuje w niej literał obiektowy:

```
var bdate = {
  "person": "Roman",
  "date": "10/27/1980"
};
```

Zmienna `bdate` zawiera literał obiektowy, dlatego aby pobrać wartość właściwości `person` tego obiektu, należy użyć składni z kropką:

```
bdate.person // "Roman"
```

Poniższy kod pobiera datę urodzenia:

```
bdate.date // "10/27/1980"
```

Tak samo można korzystać z obiektów JSON zwracanych przez serwer WWW. Przyjrzyj się poniższej instrukcji z metodą `$.getJSON()` i funkcją zwrotną:

```
$.getJSON('contacts.php','contact=123',processContacts);
function processContacts(data) {

}
```

Jeśli serwer zwróci dane JSON ze strony 500, obiekt JSON zostanie przypisany do zmiennej `data` (to argument funkcji zwrotnej `processContacts()`), co odpowiada uruchomieniu poniższego kodu:

```
var data = {
  "firstName": "Franciszek",
  "lastName": "Nowak",
  "phone": "503-555-1212"
};
```

W wywoływanej zwrotnie funkcji można pobrać wartość właściwości `firstName` w następujący sposób:

```
data.firstName // "Franciszek"
```

Aby pobrać nazwisko (właściwość `lastName`), użyj poniższej instrukcji:

```
data.lastName // "Nowak"
```

Załóżmy, że całe zadanie tego krótkiego ajaksowego skryptu polega na pobieraniu danych kontaktowych i wyświetlaniu ich w znaczniku `<div>` o identyfikatorze `info`. Kod tego programu może wyglądać następująco:

```
$.getJSON('contacts.php','contact=123',processContacts);
function processContacts(data) {
  var infoHTML='<p>Imię i nazwisko: ' + data.firstName;
  infoHTML+=' ' + data.lastName + '</br>';
  infoHTML+='Telefon: ' + data.phone + '</p>';
  $('#info').html(infoHTML);
}
```

Ostateczny efekt to dodany do strony akapit:

```
Imię i nazwisko: Franciszek Nowak
Telefon: 503-555-1212
```

Złożone dane JSON

Aby tworzyć także bardziej złożone kolekcje informacji, można używać literałów obiektowych jako wartości w łańcuchach znaków zapisanych w formacie JSON. Technika ta polega na zagnieżdżaniu literałów obiektowych w innych strukturach tego typu (nie odkładaj jeszcze książki!).

Oto przykład. Załóżmy, że chcesz, aby serwer zwracał dane kontaktowe kilku osób w formacie JSON. Skrypt przesyła do pliku *contacts.php* żądanie w postaci łańcucha znaków z zapytaniem, który określa liczbę pobieranych zbiorów danych kontaktowych. Potrzebna instrukcja może wyglądać następująco:

```
$.getJSON('contacts.php','limit=2',processContacts);
```

Fragment `limit=2` to przesyłana na serwer informacja, która określa liczbę zwracanych zbiorów danych. Przy tych ustawieniach serwer prześle dane dwóch osób. Załóżmy, że pierwszą z nich będzie Franciszek Nowak z poprzedniego przykładu. Drugi zbiór danych kontaktowych należy zapisać w następnym obiekcie JSON:

```
{
  "firstName": "Małgorzata",
  "lastName": "Kowal",
  "phone": "415-555-5235"
}
```

Serwer WWW może zwrócić łańcuch znaków z pojedynczym obiektem JSON, zawierającym oba opisane zbiory danych:

```
{
  "contact1": {
    "firstName": "Franciszek",
    "lastName": "Nowak",
    "phone": "503-555-1212"
  },
  "contact2": {
    "firstName": "Małgorzata",
    "lastName": "Kowal",
    "phone": "415-555-5235"
  }
}
```

Funkcja zwrotna ma jeden parametr o nazwie `data` (`function processContacts ↪(data)`). W momencie wywołania tej funkcji do zmiennej `data` zostanie przypisany obiekt JSON, co odpowiada uruchomieniu poniższego kodu:

```
var data = {
  "contact1": {
    "firstName": "Franciszek",
    "lastName": "Nowak",
    "phone": "503-555-1212"
  },
  "contact2": {
    "firstName": "Małgorzata",
    "lastName": "Kowal",
    "phone": "415-555-5235"
  }
};
```

W funkcji zwrotnej dostęp do pierwszego obiektu z danymi kontaktowymi można uzyskać w następujący sposób:

```
data.contact1
```

Aby pobrać imię pierwszej osoby, użyj poniższego kodu:

```
data.contact1.firstName
```

Jednak ponieważ chcesz przetworzyć dane kontaktowe wielu osób, możesz użyć funkcji biblioteki jQuery, która umożliwia przejście w pętli po wszystkich obiektach JSON. Służy do tego funkcja `$.each()`. Jej podstawowa struktura wygląda następująco:

```
$.each(JSON,function(name,value) {

});
```

Do metody `each()` należy przekazać obiekt JSON i funkcję. Ta funkcja przyjmuje nazwę i wartość każdego elementu z obiektu JSON. Poniższy kod używa przedstawionego wcześniej obiektu JSON:

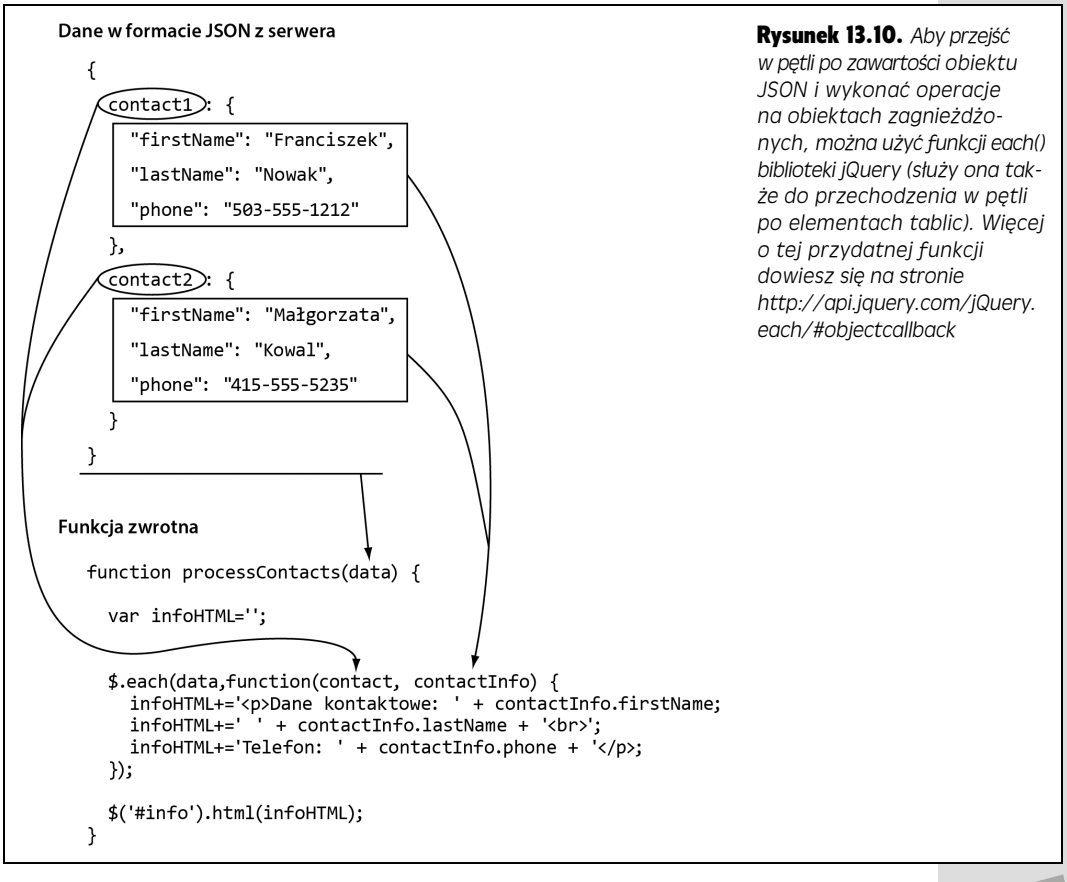
```
1  $.getJSON('contacts.php','limit=2',processContacts);
2  function processContacts(data) {
3    // Tworzenie zmiennej z pustym łańcuchem znaków.
4    var infoHTML='';
5
6    // Przejście w pętli po wszystkich obiektach z danych JSON.
7    $.each(data,function(contact, contactInfo) {
8      infoHTML+='<p>Dane kontaktowe: ' + contactInfo.firstName;
9      infoHTML+=' ' + contactInfo.lastName + '<br>';
10     infoHTML+='Telefon: ' + contactInfo.phone + '</p>';
11   }); // Koniec funkcji each
12
```

```
13    // Dodaje gotowy kod HTML do strony.
14    $('#info').html(infoHTML);
15 }
```

Oto analiza tego kodu:

1. **Wiersz 1. tworzy żądanie ajaksowe (wraz z daną `limit=2`) i określa funkcję zwrotną (`processContacts`).**
2. **Wiersz 2. tworzy funkcję zwrotną, która przyjmuje obiekt JSON zwrócony przez serwer i zapisuje go w zmiennej `data`.**
3. **Wiersz 4. tworzy pusty łańcuch znaków. Skrypt później zapisze w nim kod HTML dołączany do strony.**
4. **Wiersz 7. to metoda `$.each()`, która przechodzi po obiektach w danych w formacie JSON.**

   Metoda `$.each()` przyjmuje obiekt JSON jako pierwszy argument (`data`) i funkcję anonimową jako argument drugi. Ilustruje to rysunek 13.10. Funkcja anonimowa przyjmuje główne obiekty JSON (tu są to `contact1` i `contact2`), które obejmują łańcuch znaków z nazwą obiektu (argument `contact` w wierszu 7.) i jego wartość (argument `contactInfo`). W tym przykładzie zmienna `contactInfo` przechowuje literał obiektowy z danymi kontaktowymi.



Rysunek 13.10. *Aby przejść w pętli po zawartości obiektu JSON i wykonać operacje na obiektach zagnieżdżonych, można użyć funkcji each() biblioteki jQuery (służy ona także do przechodzenia w pętli po elementach tablic). Więcej o tej przydatnej funkcji dowiesz się na stronie http://api.jquery.com/jQuery.each/#objectcallback*

5. **Wiersze od 8. do 10. pobierają dane kontaktowe jednej osoby.**

   Pamiętaj, że metoda `$.each()` działa jak pętla, dlatego wiersze od 8. do 10. zostaną uruchomione dwukrotnie — jeden raz dla każdej osoby.

6. **Wiersz 14. aktualizuje stronę — dodaje do niej kod HTML:**

   Ostateczny efekt to dodanie do strony poniższego kodu HTML:

```
<p>Imię i nazwisko: Franciszek Nowak<br>
Telefon: 503-555-1212</p>
<p>Imię i nazwisko: Małgorzata Kowal<br>
Telefon: 415-555-5235</p>
```

Prezentacja JSONP

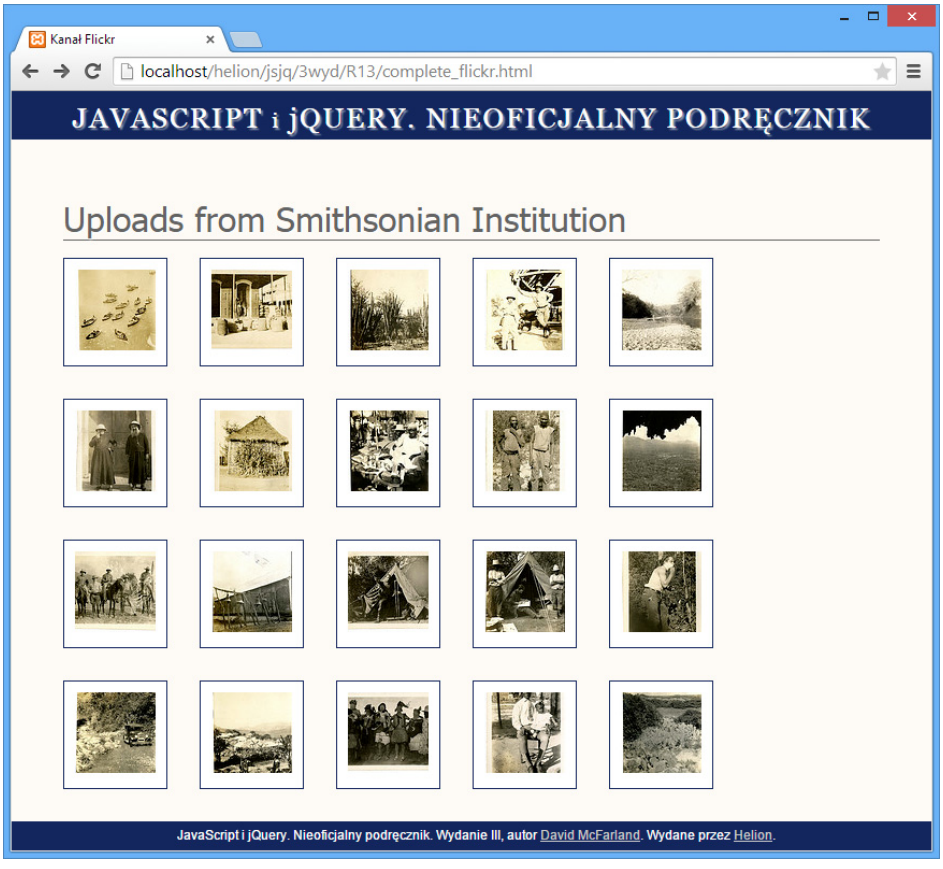
Ze względów bezpieczeństwa AJAX pozwala na przesyłanie żądań tylko do tej samej domeny. Oznacza to, że strona generująca żądania musi pochodzić z tego samego serwera, na którym działa skrypt obsługujący te żądania. Politykę tę wymuszają przeglądarki WWW, by uniemożliwić jednej witrynie kontaktowanie się z inną witryną (potencjalnie w złych zamiarach) — na przykład witryną naszego banku. Istnieje jednak pewien sposób obejścia tego ograniczenia. Choć przeglądarka nie jest w stanie przesłać żądania XML HTTP skierowanego do innej witryny, to jednak może pobierać przechowywane na niej zasoby, takie jak obrazki, arkusze stylów oraz zewnętrzne pliki JavaScript.

Technika JSONP (co stanowi skrót od słów *JSON with padding* — JSON z wypełnieniem) zapewnia możliwość pobierania informacji z innych witryn. Ogólnie rzecz biorąc, zamiast kierować do innej witryny normalne żądanie ajaksowe, pobieramy z niej skrypt zawierający kod zapisany w formacie JSON. Rozwiązanie to przypomina nieco dołączanie do stron zewnętrznych plików JavaScript z innych serwerów.

Technika ta nie pozwala jednak na pobieranie całkowicie dowolnych danych. Aby można było z niej skorzystać, witryna odpowiadająca na żądania musi zostać odpowiednio do tego przygotowana. Większość witryn nie zapewnia niezbędnych możliwości, jednak wiele dużych witryn, na przykład Google Maps, Twitter, Flickr, Facebook, Netflix czy też YouTube, udostępnia API (ang. *Application Programming Interface*), czyli interfejs programowania aplikacji pozwalający na pobieranie danych, takich jak mapy, zdjęcia, opisy filmów i tak dalej, przy użyciu funkcji `$.getJSON()` biblioteki jQuery (patrz rysunek 13.11).

Dodawanie do witryny kanału Flickr

Flickr jest popularnym serwisem umożliwiającym publikowanie zdjęć i dzielenie się nimi. Istnieje już od wielu lat i gromadzi miliony fotografii. Na wielu witrynach wyświetlane są zdjęcia zrobione przez ich autorów i opublikowane w serwisie Flickr bądź też pochodzące z grupy Flickr (grupa jest kolekcją zdjęć przesłanych przez wiele różnych osób i dotyczących konkretnego zagadnienia, takiego jak projektowanie stron WWW, widoki i tym podobne).



Rysunek 13.11. *Choć technologia AJAX zapewnia możliwość pobierania danych wyłącznie z tej samej domeny, jednak technika określana jako JSONP pozwala na pobieranie danych w formacie JSON, poprzez pobranie z odpowiednio przygotowanej witryny zewnętrznego pliku JavaScript. Za pomocą tej techniki możemy pobierać komunikaty publikowane na witrynie Twitter, mapy z aplikacji Google Maps czy zdjęcia z serwisu Flickr (przedstawione na tym rysunku) i umieszczać je bezpośrednio na swojej stronie*

Flickr udostępnia kilka sposobów pobierania zdjęć oraz informacji na ich temat. Największe możliwości zapewnia Flickr API, choć jednocześnie jest najtrudniejszy w użyciu. Pozwala na wyszukiwanie zdjęć. Aby z niego skorzystać, konieczne jest zarejestrowanie się w serwisie Flickr i pobranie specjalnego **klucza** (ang. *API key*; jest to łańcuch złożony z liter i cyfr, który określa naszą tożsamość). Dodatkowo konieczne jest także tworzenie złożonego kodu. Z kolei najprostszym sposobem jest skorzystanie z Flickr Feed Service — kanałów Flickra. Kanały są sposobem pozwalającym użytkownikom na dostęp do aktualnych informacji zgromadzonych na serwisie. Zapewne spotkałeś się z witrynami udostępniającymi kanały RSS, pozwalającymi na pobieranie najnowszych artykułów i informacji opublikowanych na witrynie. Flickr udostępnia podobne usługi udostępniające publikowane zdjęcia — można pobrać listę 20 najnowszych zdjęć konkretnego użytkownika lub zdjęć opublikowanych w określonej grupie.

W tym podrozdziale wykorzystamy kanały, by pobrać kolekcję zdjęć z serwisu Flickr i wyświetlić je na swojej stronie, a przy okazji dowiesz się, jak używać metody `$.getJSON()` biblioteki jQuery, by pobierać dane JSONP z innej witryny.

Tworzenie adresu URL

Flickr udostępnia kilka różnych adresów URL pozwalających na pobieranie odmiennych rodzajów zdjęć (patrz strona *https://www.flickr.com/services/feeds/*). Przykładowo adresu *http://api.flickr.com/services/feeds/photos_public.gne* można używać, by pobierać zdjęcia publiczne z określonych kont serwisu Flickr (na przykład ze swojego własnego konta, jeśli je posiadamy), jak również, by pobierać zdjęcia z określonej grupy (takiej jak *Web Design*, zawierającej zdjęcia i obrazki mające inspirować projektantów do tworzenia pięknych stron WWW).

Kiedy już określimy, jaki rodzaj kanał nas interesuje oraz jaki jest jego podstawowy adres URL, trzeba go będzie uzupełnić o pewne dodatkowe informacje. W tym celu do adresu URL dodaje się łańcuch zapytania zawierający kilka informacji. (Jak sobie zapewne przypominasz, była o tym mowa na stronie 487, łańcuch zapytania jest umieszczany na końcu adresu URL i składa się ze znaku zapytania (`?`) oraz kilku par nazwa – wartość; oto przykład: *https://api.flickr.com/services/feeds/groups_discuss.gne?id=1003995@N21&lang=en-us&format=rss_200*).

- **Dodanie jednego lub kilku identyfikatorów**. Aby pobrać zdjęcia z konkretnej grupy albo jednego bądź kilku kont indywidualnych, należy dodać ciąg znaków `id`, znak równości (`=`) oraz numer konta danej osoby lub grupy. Aby na przykład pobrać kanał ze zdjęciami z grupy *Web Design*, należy użyć ogólnego adresu URL kanału grupy i dodać do niego identyfikator tej grupy, która nas interesuje; oto przykład:

  https://api.flickr.com/services/feeds/groups_pool.gne?id=37996591093@N01

  W przypadku kanałów ze zdjęciami pochodzącymi z kont konkretnych użytkowników serwisu Flickr konieczne jest użycie adresu URL kanału publicznego i uzupełnienie go o identyfikator lub identyfikatory wybranych użytkowników. Aby za jednym zamachem pobrać zdjęcia z kanału Instytutu Smithsona (który posiada swoje własne konto w serwisie Flickr) oraz kanału Biblioteki Kongresu, konieczne byłoby dodanie do adresu URL kanału publicznego następujących dwóch identyfikatorów:

  *https://api.flickr.com/services/feeds/photos_*
  public.gne?ids=8623220@N02,25053835@N03

  Gdy trzeba podać większą liczbę identyfikatorów, należy oddzielić je od siebie przecinkami. Trzeba przy tym pamiętać, że takie podawanie większej liczby identyfikatorów jest możliwe wyłącznie w przypadku pobierania zdjęć z kont indywidualnych. Pobieranie zdjęć z większej liczby grup w podobny sposób nie jest możliwe.

Wskazówka: Jeśli znasz nazwę użytkownika — jakiejś osoby posiadającej konto w serwisie Flickr, możesz znaleźć jej identyfikator na stronie *http://idgettr.com/*.

- **Dodanie formatu JSON**. Usługi kanałów zdjęć udostępniane przez Flickr są bardzo elastyczne i mogą zwracać informacje dotyczące zdjęć, zapisane w wielu różnych formatach, takich jak RSS, Atom, CSV czy też JSON. Aby poinformować je, że interesuje nas format JSON, do łańcucha zapytania należy dodać ciąg znaków `&format=json`. Aby na przykład pobrać dane z kanału Flickr Instytutu Smithsona w formacie JSON, należałoby użyć następującego adresu URL:

  *https://api.flickr.com/services/feeds/photos_*
  public.gne?ids=25053835@N03&format=json

  Spróbuj teraz wpisać powyższy adres w przeglądarce (jeśli jesteś na to zbyt leniwy, możesz skopiować i wkleić adres URL podany w pliku *flickr_json.txt* dostępnym w przykładach do książki, w katalogu *R13*). W rezultacie w przeglądarce zostaną wyświetlone dane, a konkretnie literał obiektowy zawierający dane z kanału Flickr. To są właśnie te dane, które pobierzesz z serwisu przy użyciu metody `$.getJSON()` (opisanej na stronie 501). Teraz konieczne będzie napisanie kodu JavaScript, który przetworzy ten obiekt i wykorzysta zapisane w nim dane do wygenerowania małej, efektownej galerii zdjęć. (Struktura danych kanałów Flickr zapisanych w formacie JSON została opisana na stronie 511, natomiast na stronie 512 pokazano, jak można pobrać z niego wybrane informacje i użyć ich w skrypcie).
- **Dodanie do adresu URL odwołania zwrotnego JSONP**. I w końcu, aby strona należąca do naszej witryny mogła pobrać dane kanału Flickr, konieczne jest dodanie do adresu URL jeszcze jednego parametru: `&jsoncallback=?`. Przypomnij sobie, że ze względów bezpieczeństwa nie można tak po prostu przesłać żądania XMLHTTP na adres należący do innej domeny. Aby ominąć ten problem, używany jest właśnie parametr `&jsoncallback=?`, który informuje serwis Flickr, że interesują nas dane JSONP, i pozwala metodzie `$.getJSON()` potraktować żądanie w taki sposób, jakby było ono skierowane do zwyczajnego, zewnętrznego pliku JavaScript. Innymi słowy, aby pobrać kanał z najnowszymi zdjęciami opublikowanymi przez Instytut Smitshona, w wywołaniu metody `$.getJSON()` należy podać następujący adres URL:

  *https://api.flickr.com/services/feeds/photos_*
  public.gne?ids=25053835@N03&format=json&jsoncallback=?

Kilka innych opcji publicznych kanałów Flickr

Serwis Flickr daje możliwość dodawania do każdego publikowanego zdjęcia znaczników (ang. *tags*), czyli skojarzenia z nim słów lub krótkich zwrotów opisujących dane zdjęcie. Przykładowo do zdjęcia zachodu słońca można dodać znacznik `sunset`. Do każdego zdjęcia można dodać większą liczbę takich znaczników, a zatem nasze zdjęcie zachodu słońca może mieć znaczniki `sunset, orange, beach`.

Usługi obsługi kanałów serwisu Flickr udostępniają opcje pozwalające przeglądać kanały w poszukiwaniu zdjęć zawierających określone znaczniki. Oto one.

- **`tags`**. Parametr `tag` pozwala dodać do adresu URL kanału jeden lub kilka, oddzielonych od siebie przecinkami znaczników, na przykład `&tags=fireworks,` `↪night`. Aby odszukać zdjęcia ogni sztucznych w nocy, należy użyć następującego adresu URL:

https://api.flickr.com/services/feeds/photos_public.gne?tags=fireworks,night&format=json&jsoncallback=?

- **`tagmode`**. Zazwyczaj podczas poszukiwania grupy znaczników Filckr zwraca tylko te zdjęcia, w których zostały zastosowane wszystkie podane znaczniki. Przykładowo załóżmy, że do adresu dodaliśmy parametr `?tags=chipmunk,baseball,winter`. W takim przypadku znajdziemy wyłącznie zdjęcia burunduków grających w baseball zimą. Gdybyśmy jednak chcieli wyszukać zdjęcia burunduków lub gry w baseball lub zdjęcia zimowe (innymi słowy, zdjęcia, w których użyto przynajmniej jednego z interesujących nas znaczników), musielibyśmy dodać do adresu URL jeszcze jeden parametr — `&tagmode=any`. Oto przykład:

  https://api.flickr.com/services/feeds/photos_public.gne?tags=chipmunk,baseball,winter&tagmode=any&format=json&jsoncallback=?

Łączenie opcji

Wyszukiwanie zdjęć można dodatkowo usprawnić, łącząc opcje. Przykładowo do parametru `tag` można także dodać parametr `ID`, dzięki czemu odszukane zostaną wyłącznie odpowiednie zdjęcia nadesłane przez konkretnego użytkownika serwisu. Załóżmy, że wraz z grupą przyjaciół lubicie publikować w serwisie Flickr zdjęcia burunduków i właśnie chciałbyś pobrać ostatnie 20 opublikowanych zdjęć tych zwierząt. Możesz to zrobić, filtrując je przy użyciu jednego lub większej liczby parametrów:

https://api.flickr.com/services/feeds/photos_public.gne?ids=25053835@N03,8623220@N02&tags=chipmunk&format=json&jsoncallback=?

Stosowanie metody $.getJSON()

Skorzystanie z funkcji `$.getJSON()` w celu pobierania danych kanału z serwisu Flickr daje dokładnie te same efekty jak pobieranie danych w formacie JSON z własnej witryny. Podstawowy sposób użycia tej funkcji jest taki sam. Poniżej przedstawione zostały czynności wstępne, niezbędne do pobrania kanału Instytutu Smithsona:

```
1   var flickrURL = "https://api.flickr.com/services/feeds/
        ↪photos_public.gne?ids=25053835@N03&format=json&jsoncallback=?"
2   $.getJSON(flickrURL, function(data) {
3     // przetwarzamy pobrane dane w formacie JSON
4   }); // koniec funkcji get
```

W 1. wierszu powyższego przykładu tworzona jest zmienna o nazwie `flickrURL`, zawierająca adres URL (utworzony zgodnie z opisanymi wcześniej regułami). W wierszu 2. generujemy żądanie, przesyłając je na przygotowany wcześniej adres URL, i określamy funkcję anonimową, służącą do przetwarzania pobranych danych. Po przesłaniu żądania kod pobiera dane wysłane z serwera — w tym przypadku zostaną one przekazane do funkcji anonimowej i zapisane w zmiennej `data`. Już niebawem nauczysz się, jak można takie dane przetwarzać, jednak najpierw musisz dowiedzieć się, jak wyglądają.

Prezentacja danych kanału Flickr w formacie JSON

Zgodnie z informacjami podanymi na stronie 500, format JSON jest tekstowym sposobem zapisu literałów obiektowych języka JavaScript. Czasami postać takich literałów może być bardzo prosta, oto przykład:

```
{
  "firstName" : "Jan",
  "lastName" : "Kowalski"
}
```

W tym przykładzie `firstName` można porównać do klucza, skojarzonego z wartością `'Jan'` — zwyczajnym łańcuchem znaków. Jednak taką wartością może być także kolejny obiekt (patrz rysunek 13.10, na stronie 505), więc często można się spotkać ze złożonymi, zagnieżdżonymi strukturami danych, przypominającymi nieco rosyjskie „matrioszki", lalki ukryte wewnątrz innych lalek. Właśnie w taki sposób wygląda kanał Flickr z danymi zdjęć. Oto niewielki fragment takich danych, przedstawia dane dotyczące dwóch zdjęć:

```
1  {
2    "title": "Uploads from Smithsonian Institution",
3    "link": "http://www.flickr.com/photos/smithsonian/",
4    "description": "",
5    "modified": "2011-08-11T13:16:37Z",
6    "generator": "http://www.flickr.com/",
7    "items": [
8      {
9        "title": "East Island, June 12, 1966.",
10       "link": "http://www.flickr.com/photos/smithsonian/5988083516/",
11       "media": {"m":"http://farm7.static.flickr.com/6029/5988083516_
         ↪bfc9f41286_m.jpg"},
12       "date_taken": "2011-07-29T11:45:50-08:00",
13       "description": "Short description here",
14       "published": "2011-08-11T13:16:37Z",
15       "author": "nobody@flickr.com (Smithsonian Institution)",
16       "author_id": "25053835@N03",
17       "tags": "ocean birds redfootedbooby"
18     },
19     {
20       "title": "Phoenix Island, April 15, 1966.",
21       "link": "http://www.flickr.com/photos/smithsonian/5988083472/",
22       "media": {"m":"http://farm7.static.flickr.com/6015/5988083472_
         ↪c646ef2778_m.jpg"},
23       "date_taken": "2011-07-29T11:45:48-08:00",
24       "description": "Another short description",
25       "published": "2011-08-11T13:16:37Z",
26       "author": "nobody@flickr.com (Smithsonian Institution)",
27       "author_id": "25053835@N03",
28       "tags": ""
29     }  // ...
30   ]
31 }
```

Obiekt JSON generowany przez serwis Flickr zawiera nieco informacji o samym kanale: są one umieszczone na samym początku i obejmują takie dane jak `title`, `link` i tak dalej. Element `title` (umieszczony w wierszu 2.) zawiera nazwę kanału. W tym przypadku jest to `Uploads from Smithsonian Institution`; z kolei element `link` zawiera adres URL głównej strony Instytutu Smithsona na serwisie Flickr. Informacji tych można użyć na przykład jako nagłówka wyświetlanego nad galerią zdjęć.

Aby uzyskać dostęp do tych informacji, konieczne jest skorzystanie z zapisu z kropką, opisanego na stronie 87. Załóżmy, że użyliśmy kodu przedstawionego w poprzednim punkcie rozdziału (na stronie 510): funkcji anonimowej, przetwarzającej dane JSON przekazane do niej w zmiennej `data` (patrz 2. wiersz kodu na stronie 510). Aby w takim przypadku pobrać wartość właściwości `title` obiektu `data`, musimy użyć wyrażenia w postaci:

```
data.title
```

Najważniejszym elementem danych kanału Flickr jest właściwość `items` (wiersz 7.), zawierająca dodatkowe obiekty, z których każdy przechowuje informacje dotyczące jednego zdjęcia. Przykładowo wiersze od 8. do 18. zawierają informacje o pierwszym zdjęciu, natomiast wiersze od 19. do 29. — o drugim. Wewnątrz każdego z tych obiektów można znaleźć kolejne właściwości, takie jak tytuł zdjęcia (wiersz 9.), odnośnik do strony tego zdjęcia (wiersz 10.), datę zrobienia zdjęcia (wiersz 12.), jego opis (wiersz 13., w tym przypadku jest to `Short description here`, czyli: Tu jest umieszczony krótki opis — pracownicy Instytutu Smitshona musieli być tego dnia trochę leniwi) i tak dalej.

Kolejną ważną informacją dotyczącą każdego zdjęcia jest element `media` — zawiera on kolejny obiekt. Oto przykład:

```
{
  "m":"http://farm7.static.flickr.com/6029/5988083516_bfc9f41286_m.jpg"
}
```

Litera `m` na początku jest skrótem od angielskiego słowa *medium* — średni. Właściwość ta zawiera adres URL zdjęcia. Zdjęcia na serwisie Flickr są zazwyczaj dostępne w kilku różnych rozmiarach, takich jak średni (ang. *medium*), miniaturka (ang. *thumbnail*) lub mały (ang. *small*; w tym przypadku jest to mały, kwadratowy obrazek). Jeśli chcemy wyświetlać zdjęcia z serwisu Flickr na własnej stronie, to właśnie ten adres URL jest informacją, której potrzebowaliśmy. Tego adresu możemy użyć w znaczniku `<img>`, by wskazać położenie zdjęcia na serwerze Flickr. Zobaczysz, jak to należy zrobić, w przykładzie przedstawionym w kolejnym podrozdziale.

Przykład — dodawanie zdjęć z Flickr na własnej stronie

W tym przykładzie dowiesz się, jak trzeba połączyć w jedną całość wszystkie czynności związane z pobieraniem kanału zdjęć Instytutu Smithsona z serwisu Flickr, wyświetlaniem na własnej stronie miniaturek pobranych zdjęć i dodaniem do każdej z nich odnośnika, który pozwoli użytkownikowi przejść na stronę danego zdjęcia.

Uwaga: Informacje na temat pobierania przykładów dołączonych do tej książki można znaleźć na stronie 46.

1. **W edytorze tekstów otwórz plik *flickr.html* umieszczony w katalogu *R13*.**

   Zaczniesz od utworzenia kilku zmiennych, w których będą zapisane komponenty adresu URL koniecznego do pobrania danych kanału z serwisu Flickr.

2. **Kliknij pusty wiersz wewnątrz funkcji `$(document).ready()` i wpisz:**

```
var URL = "https://api.flickr.com/services/feeds/photos_public.
➥gne?jsoncallback=?";
```

Ten adres odwołuje się do publicznego kanału i zawiera jeden magiczny fragment — `?jsoncallback=?` — informujący serwis Flickr, że w odpowiedzi powinien przesłać dane JSONP. Innymi słowy, powyższy kod nakazuje serwisowi Flickr przesłanie pliku JavaScript zawierającego informacje o zdjęciach.

Teraz utworzysz obiekt, który będzie zawierał dodatkowe informacje przekazywane do serwisu Flickr.

Każda z informacji zapisanych w tym obiekcie będzie jednym z elementów długiego adresu URL, opisanego na stronie 509. Zapisanie każdego z elementów tego adresu w osobnej właściwości obiektu ułatwia wprowadzanie ewentualnych zmian w kodzie.

Wskazówka: Jeśli dysponujesz kontem w serwisie Flickr, do powyższego adresu możesz dodać jego identyfikator. Jeśli go nie znasz, możesz to sprawdzić na witrynie *http://idgettr.com/*.

W następnym kroku przygotujesz obiekt z dodatkowymi informacjami.

3. **Dodaj kolejny wiersz poniżej tego, który wpisałeś w poprzednim kroku, i napisz w nim:**

```
var searchInfo = {


};
```

Te trzy wiersze kodu zawierają pusty literał obiektowy języka JavaScript. Zgodnie z informacjami podanymi na stronie 500, w żądaniach AJAX można przesyłać dodatkowe dane, których serwer może użyć do modyfikowania postaci odpowiedzi. W tym przypadku chcesz podać numer użytkownika serwisu Flickr, aby odszukać wyłącznie jego zdjęcia.

4. **Wewnątrz obiektu dodaj jedną parę nazwa – wartość (wyróżnioną pogrubieniem):**

```
var searchInfo = {
  id: "25053835@N03"
};
```

API serwisu Flickr oczekuje identyfikatora użytkownika. W tym przypadku identyfikator ten należy do Instytutu Smithsona. Zwróć uwagę, że nazwa właściwości musi być zapisana małymi literami — `id`. Aby pobrać kanał zdjęć innego użytkownika, wystarczy zmienić wartość właściwości `id` (jeśli masz swoje konto w serwisie Flickr, możesz tu podać własny identyfikator).

Dodatkowo musisz poinformować serwis, by przesyłał dane w formacie JSON.

Wskazówka: Jeśli chcesz pobrać zdjęcia z jakieś grupy serwisu Flickr, takiej jak grupa Web Design, zmień adres URL podany w kroku 2. na następujący: *https://api.flickr.com/services/feeds/groups_pool.gne?jsoncallback=?;*, a w kroku 4. podaj identyfikator grupy.

5. **Na końcu wiersza zawierającego właściwość `id` wpisz przecinek, naciśnij klawisz Enter i wpisz: `format : "json"`:**

```
var searchInfo = {
  id: "25053835@N03",
  format : "json"
};
```

Jeśli nie dodasz właściwości `format`, serwis prześle w odpowiedzi dane w formacie XML.

W końcu nadszedł czas, aby zająć się kodem JavaScript i skorzystać z metody `$.getJSON()`.

6. **Dodaj następny wiersz i wpisz w nim:**

```
$.getJSON(URL,searchInfo,function(data) {

}); // koniec funkcji getJSON
```

To ogólna struktura wywołania funkcji `$.getJSON()`: prześle ona żądanie na adres URL podany w kroku 2., dodając do niego kryteria wyszukiwania określone w krokach od 3. do 5. oraz pobierze zwrócone dane. Dane te zostaną następnie przekazane do funkcji anonimowej i zapisane w zmiennej `data`. Wewnątrz tej funkcji będziesz już mógł odwoływać się do pobranych danych i używać ich do tworzenia strony. Zaczniemy od pobrania tytułu kanału i umieszczenia go w znaczniku `<h1>`, który już jest dostępny w kodzie strony.

7. **Do kodu z poprzedniego kroku dodaj wiersz wyróżniony pogrubioną czcionką:**

```
$.getJSON(URL,searchInfo,function(data) {

  $('h1').text(data.title);
}); // koniec funkcji getJSON
```

Zastosowaliśmy w nim prosty selektor jQuery — `$('h1')` — oraz funkcję `text()`, aby pobrać znacznik `<h1>` dostępny na stronie i zastąpić tekst umieszczony wewnątrz niego. Dane kanału w formacie JSON są zapisane w zmiennej `data`. Aby uzyskać dostęp do ich elementów, konieczne jest zastosowanie zapisu z kropką (patrz strona 87), a zatem wyrażenie `data.title` pobiera tytuł kanału. Gdybyś już teraz zapisał stronę i wyświetlił ją w przeglądarce, zobaczyłbyś pogrubiony nagłówek o treści *Uploads from Smithsonian Institution.*

W kolejnym punkcie utworzysz nową zmienną, w której zostanie zapisany łańcuch znaków zawierający kod HTML, jaki później dodasz do strony.

8. **Dodaj kolejny wiersz wyróżniony pogrubioną czcionką, umieszczając go poniżej kodu wpisanego w poprzednim kroku:**

```
$.getJSON(URL,searchInfo,function(data) {
  $('h1').text(data.title);
  var photoHTML = '';
}); // koniec funkcji getJSON
```

Obecnie łańcuch ten jest pusty, jednak już wkrótce dodasz do niego kod HTML niezbędny do wyświetlenia na stronie zdjęć z serwisu Flickr. W celu utworzenia tego kodu HTML przeglądniesz w pętli tablicę `items`, zawierającą obiekty zwrócone z kanału Flickr. Dla każdego zwróconego zdjęcia dodasz do zmiennej `photoHTML` kolejny fragment kodu HTML.

9. **Poniżej wiersza kodu dodanego w poprzednim kroku dodaj fragment wyróżniony pogrubioną czcionką:**

```
$.getJSON(ajaxURL,searchInfo,function(data) {
  $('h1').text(data.title);
  var photoHTML = '';
  $.each(data.items,function(i,photo) {

  }); // koniec funkcji each
}); // koniec funkcji getJSON
```

Funkcja `.each()`(opisana na stronie 167) służy do przeglądnięcia całej zawartości kolekcji jQuery. Metoda `$.each()` jest podobna, choć działa nieco inaczej. Jest to ogólna pętla pozwalająca przejrzeć całą zawartość tablicy (patrz strona 77) lub grupy obiektów. W jej wywołaniu przekazywana jest tablica lub literał obiektowy oraz funkcja anonimowa. Metoda `$.each()` pobiera kolejno każdy element tablicy lub obiektu i dla każdego z nich wywołuje funkcję anonimową. Z kolei do tej funkcji anonimowej przekazywane są dwa argumenty (w naszym przypadku są to `i` oraz `photo`), zawierające odpowiednio indeks elementu oraz sam element. Indeks jest numerem elementu przetwarzanego w pętli i jest liczony tak samo jak indeksy tablic (patrz strona 80), czyli pierwszy element ma indeks o wartości `0`. Drugim argumentem wywołania funkcji anonimowej (w naszym przykładzie nosi on nazwę `photo`) jest obiekt zawierający faktyczne dane zdjęcia, takie jak jego nazwa, opis, adres URL i tak dalej (zgodnie z informacjami podanymi na stronie 512).

W kanałach Flickr element `data.items` reprezentuje tablicę obiektów z danymi o poszczególnych zdjęciach, a zatem funkcja `$.each()` będzie po kolei przekazywać każdy z tych obiektów do funkcji anonimowej, przy czym każdy z obiektów zostanie umieszczony w zmiennej `photo`. Innymi słowy, powyższy kod przeglądnie w pętli wszystkie zdjęcia pobrane w kanale i coś z nimi zrobi. W naszym przypadku wykonywane operacje sprowadzą się do utworzenia sekwencji miniaturek będących jednocześnie odnośnikami do stron poszczególnych zdjęć w serwisie Flickr. Naszym celem jest wygenerowanie prostego kodu HTML, który pozwoli wyświetlić każde ze zdjęć i dodać do niego odnośnik. Oto przykład:

```
<span class="image">
<a href="http://www.flickr.com/photos/smithsonian/5988083516/">
<img src="http://farm7.static.flickr.com/6029/5988083516_bfc9f41286_s.jpg">
</a>
</span>
```

Do wygenerowania takiego kodu potrzebujemy tylko dwóch informacji — adresu URL strony zdjęcia w serwisie Flickr oraz ścieżki do pliku zdjęcia. W kolejnym kroku wygenerujesz długi łańcuch znaków, dokładnie przypominający powyższy kod HTML, w którym dla każdego zdjęcia zmieniać się będzie jedynie adres URL strony oraz miniaturki zdjęcia.

10. **Wewnątrz metody `$.each()` dodaj poniższy kod wyróżniony pogrubioną czcionką:**

```
$.each(data.items,function(i,photo) {
  photoHTML += '<span class="image">';
  photoHTML += '<a href="' + photo.link + '">';
  photoHTML += '<img src="' + photo.media.m + '"></a></span>';
}); // koniec funkcji each
```

Powyższy kod rozpoczyna się od dodania do zmiennej `photoHTML` łańcucha zawierającego otwierający znacznik `<span>`. Każda kolejna instrukcja dodaje do tego łańcucha kolejny fragment (aby sobie przypomnieć znaczenie operatora +=, zajrzyj na stronę 72). Kluczowymi elementami tych instrukcji są odwołania `photo.link` oraz `photo.media.m`. Jeśli zajrzysz do danych zapisanych w formacie JSON i przedstawionych na stronie 511, przekonasz się, że dane każdego ze zdjęć zawierają różne właściwości, takie jak `title` (nazwa zdjęcia) bądź `description` (krótki opis danego zdjęcia). Właściwość `link` zawiera adres URL strony danego zdjęcia w serwisie Flickr, natomiast właściwość `media` — obiekt posiadający właściwość `m`, zawierającą ścieżkę dostępu do pliku zdjęcia w wersji o średniej wielkości. Cały ten kod generuje HTML w dokładnie takiej postaci, jaka została przedstawiona w kroku 6. Teraz trzeba wyświetlić ten kod na stronie.

11. **Dodaj kolejny fragment kodu wyróżniony pogrubioną czcionką:**

```
$.each(data.items,function(i,photo) {
  photoHTML += '<span class="image">';
  photoHTML += '<a href="' + photo.link + '">';
  photoHTML += '<img src="' + photo.media.m + '"></a></span>';
}); // koniec funkcji each
$('#photos').append(photoHTML);
```

Zwróć uwagę, że ten wiersz kodu jest umieszczony *poza* pętlą. Przecież nie chcesz dodawać kodu HTML do strony aż do momentu, gdy zostanie on w całości wygenerowany wewnątrz pętli. Wywołanie `$('#photos')` pobiera istniejący na stronie znacznik `<div>`, natomiast funkcja `append()` (opisana dokładniej na stronie 158) dodaje przekazany kod HTML na samym końcu tego znacznika.

12. **Zapisz stronę i wyświetl ją w przeglądarce.**

Na stronie powinno pojawić się 20 miniaturek. (Jeśli nic na niej nie zobaczysz, sprawdź dokładnie kod i skorzystaj z konsoli JavaScript przeglądarki, zgodnie z informacjami podanymi na stronie 51, by odszukać wszelkie błędy składniowe). Nasz problem polega na tym, że miniaturki zdjęć są bardzo małe i nie są wyświetlane na stronie w formie estetycznej tabelki. Dzieje się tak dlatego, że w kanałach Flickr podawane są jedynie odnośniki do zdjęć średniej wielkości.

Flickr przechowuje jednak także ładne, kwadratowe miniaturki wszystkich zgromadzonych zdjęć. Wyświetlenie na stronie takich miniaturek o tym samym kształcie i wymiarach pozwoliłoby utworzyć estetyczną, uporządkowaną galerię. Na szczęście, pobranie tych miniaturek nie przysparza większych problemów. Flickr korzysta ze spójnego sposobu określania nazw zdjęć — zdjęcie o średniej wielkości ma na przykład adres *http://farm7.static.flickr.com/6029/5988083516_bfc9f41286_m.jpg*, natomiast miniaturka tego samego zdjęcia — *http://farm7.static.flickr.com/6029/5988083516_bfc9f41286_s.jpg*. Jak widać, jedyną różnicą jest inna końcówka nazwy pliku: w przypadku zdjęć średniej wielkości jest to *_m*, w małych, kwadratowych miniaturkach (o wymiarach 75×75 pikseli) jest to *_s*, małe zdjęcia, których dłuższa krawędź ma prawie 100 pikseli mają końcówkę *_t*, końcówka *_o* oznacza oryginalne zdjęcie (czyli takie, jakie zostało przesłane na serwer Flickr przez użytkownika), natomiast duże zdjęcia (których dłuższa krawędź ma co najwyżej 1024 piksele długości) mają końcówkę *_b*. Oznacza to, że gdy zmienimy nieznacznie nazwę pliku (na

przykład zamieniając ciąg znaków `_m` na ciąg `_s`), możemy wyświetlać obrazki innej wielkości. Tak się składa, że JavaScript udostępnia wygodną metodę pozwalającą na zamienianie fragmentów łańcuchów znaków.

13. **W kodzie skryptu zmień `photo.media.m` na `photo.media.m.replace('_m', '_s')`. Końcowa postać kodu powinna wyglądać tak, jak na poniższym przykładzie**:

```
$(document).ready(function() {
  var URL = "https://api.flickr.com/services/feeds/
  ↪photos_public.gne?jsoncallback=?";
  var searchInfo = {
    id : "25053835@N03",
    format : "json"
  };
  $.getJSON(URL,searchInfo,function(data) {
    $('h1').text(data.title);
    var photoHTML = '';
    $.each(data.items,function(i,photo) {
      photoHTML += '<span class="image">';
      photoHTML += '<a href="' + photo.link + '">';
      photoHTML += '<img src="'
      ↪+ photo.media.m.replace('_m','_s') + '"></a></span>';
    }); // koniec funkcji each
    $('#photos').append(photoHTML);
    }); // koniec funkcji get JSON
}); // koniec funkcji ready
```

Metoda `replace()` języka JavaScript (opisana na stronie 585) operuje na łańcuchach znaków; wymaga ona przekazania dwóch argumentów — poszukiwanego łańcucha znaków (w naszym przypadku jest to `'_m'`) oraz zamiennika (u nas jest to `'_s'`).

14. **Zapisz stronę i wyświetl ją w przeglądarce.**

Teraz powinieneś zobaczyć na stronie 20 równo rozmieszczonych, kwadratowych miniaturek (patrz rysunek 13.11). Możesz kliknąć jedną z nich, by wyświetlić stronę danego zdjęcia. Działającą wersję tego przykładu można znaleźć w pliku *complete_flickr.html*, w katalogu *R13*.

Uwaga: Kanały Flickr udostępniają co najwyżej 20 zdjęć. Z żadnego kanału nie można pobrać więcej niż 20 zdjęć. A co zrobić, jeśli będziemy chcieli pobrać z kanału 10 zdjęć? Przykład rozwiązania takiego problemu możesz znaleźć w pliku *complete_flickr_limit_photos.html* dostępnym w katalogu *R13*.

14

ROZDZIAŁ

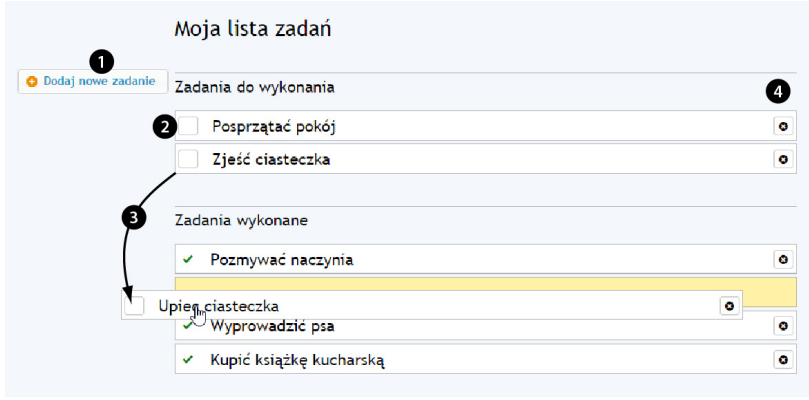
Tworzenie aplikacji do obsługi listy zadań

Biblioteki jQuery i jQuery UI udostępniają narzędzia do tworzenia zaledwie w kilku prostych krokach profesjonalnie wyglądających aplikacji internetowych. Pierwsza z nich może zadbać o wybieranie elementów strony, dodawanie do niej nowych elementów oraz aktualizowanie DOM. Z kolei biblioteka jQuery UI udostępnia wspaniale wyglądające widżety, sposoby interakcji z użytkownikiem oraz efekty animacji, rozwiązując tym samym wiele najczęściej występujących problemów związanych z opracowaniem interfejsu użytkownika. Korzystając z obu tych bibliotek, można uniknąć żmudnych oraz czasochłonnych zadań i skoncentrować się wyłącznie na tworzeniu dynamicznej, interaktywnej aplikacji. W tym rozdziale przedstawiony został proces pisania prostej aplikacji do zarządzania listą zadań.

Przegląd aplikacji

Lista zadań, którą napiszemy w tym rozdziale, będzie pozwalała użytkownikom na wykonywanie następujących operacji.

- **Dodawanie nowych zadań do listy.** W tym celu dodamy do niej widżet przycisku jQuery UI (nr 1 na rysunku 14.1), a następnie po kliknięciu wyświetlimy okno dialogowe jQuery UI.
- **Oznaczanie zadania jako wykonanego.** Każde zadanie na liście będzie mieć pole wyboru z lewej strony (numer 2 na rysunku 14.1). Kliknięcie tego pola będzie automatycznie usuwać zadanie z listy *Zadania do wykonania* i przenosić na listę *Zadania wykonane*.
- **Przeciąganie zadań z listy *Zadania do wykonania* na listę *Zadania wykonane* i na odwrót** (numer 3 na rysunku 14.1)**.** Choć klikanie pola wyboru niewątpliwie spełni swoją rolę, to niby czemu mamy się ograniczać do tylko jednego sposobu oznaczania zadań jako wykonanych? Korzystając z widżetu



Rysunek 14.1. *Biblioteki jQuery oraz jQuery UI sprawiają, że napisanie prostej listy zadań do zrobienia jest relatywnie łatwe. Pozwalają one na rozwiązywanie złożonych problemów związanych z tworzeniem interfejsów użytkownika, takich jak opracowanie okien dialogowych, list z możliwością sortowania oraz animowanie wybranych elementów stron. Dzięki nim będziemy mogli skoncentrować się wyłącznie na zaimplementowaniu możliwości funkcjonalnych aplikacji*

Sortable jQuery UI, możemy także pozwolić użytkownikom na oznaczanie zdań jako wykonanych poprzez przeciąganie ich na listę *Zadania wykonane*. Co więcej, możemy także pozwolić na przeciąganie zadań *z powrotem* na listę *Zadania do wykonania*.

- **Usuwanie zadań z listy po kliknięciu przycisku *Usuń*** (numer 4 na rysunku 14.1)**.** Aby zapewnić użytkownikom możliwość całkowitego usunięcia zadania, dodamy przycisk *Usuń*, którego kliknięcie całkowicie usunie zadanie ze strony. Zastosujemy przy tym jeden z efektów jQuery UI, by cała operacja wyglądała interesująco.

Ponieważ biblioteka jQuery UI udostępnia większość widżetów i mechanizmów interakcji, pozostaje jedynie zaimplementowanie podstawowej logiki działania aplikacji. Zadanie to wykonamy krok po kroku, zgodnie z przedstawionym powyżej planem. Zaczniemy od dodania przycisku, a następnie dokończymy pozostałą część aplikacji. Zadanie programistyczne podczas rozwiązywania warto podzielić na mniejsze elementy, które można łatwo przetestować. Podobnie będzie w tym przykładzie — najpierw zrobimy jedną rzecz, upewnimy się, że działa prawidłowo, i dopiero potem przejdziemy do kolejnej.

Dodanie przycisku

Najpierw dodamy do strony przycisk i sformatujemy go przy użyciu jQuery UI. W tym przykładzie będziemy pracować nad dwoma plikami dostępnymi w katalogu *R14*; są to *index.html* oraz *todo.js*. Cały kod JavaScript będzie umieszczany w pliku *todo.js*, natomiast kod HTML niezbędny do utworzenia przycisku oraz okna dialogowego trafi do pliku *index.html*.

Uwaga: Informacje na temat pobierania przykładów do książki można znaleźć na stronie 46.

1. **Otwórz w edytorze tekstu plik *index.html* dostępny w katalogu *R14.***

   Plik zawiera już odwołania do potrzebnych plików CSS, jQuery oraz jQuery UI. Jednak nie ma w nim jeszcze odwołania do skryptu *todo.js* — czyli pliku, w którym zapiszesz kod JavaScript tworzonej aplikacji.

2. **W wierszu bezpośrednio poniżej `<script src="js/jqeury-ui.min.js">↪</script>` dodaj następujący wiersz kodu :**

   ```
   <script src="todo.js"></script>
   ```

   Bardzo ważne jest to, by plik *todo.js* został dołączony do strony jako ostatni, gdyż wymaga on zarówno biblioteki jQuery, jak i jQuery UI. Jeśli wczytasz go przed którąkolwiek z tych bibliotek, podczas wyświetlania strony przeglądarka zgłosi błąd syntaktyczny.

   Teraz zajmiesz się dodaniem przycisku. Użytkownik będzie mógł go kliknąć w celu dodania do listy nowego zadania.

3. **W treści pliku odszukaj komentarz `<!-- Tu dodaj przycisk. -->` i zastąp go poniższym wierszem kodu HTML:**

   ```
   <button id="add-todo">Dodaj nowe zadanie</button>
   ```

   W tym kodzie nie ma nic szczególnego — dodajesz do strony zwyczajny element `<button>`. Gdybyś teraz wyświetlił stronę w przeglądarce, przekonałbyś się, że wygląda on jak zwyczajny przycisk do wysyłania formularza — bezbarwny i nieinteresujący. Już zaraz przekształcisz go w przycisk jQuery UI.

4. **W edytorze tekstów otwórz plik *todo.js*, a następnie wewnątrz funkcji `$(document).ready(function(e) {` wpisz następujący wiersz kodu:**

   ```
   $("#add-todo").button();
   ```

   Powyższy kod spowoduje zastosowanie w przycisku formatowania określonego przez aktualnie używamy temat graficzny jQuery UI (patrz strona 407). Możesz go dodatkowo uatrakcyjnić, dodając do niego ikonę jQuery UI.

5. **Wewnątrz funkcji `button()` dodaj literał obiektowy definiujący ikonę, która będzie wyświetlona na przycisku (kod literału został wyróżniony pogrubioną czcionką):**

   ```
   $("#add-todo").button({
     icons: {
       primary: "ui-icon-circle-plus"
     }
   });
   ```

   Powyższy kod umieści z lewej strony napisu widocznego na przycisku niewielką ikonę ze znakiem „+". Zgodnie z informacjami podanymi na stronie 390, jQuery UI pozwala na wyświetlanie na przyciskach dwóch ikon: jednej po lewej stronie (ikona główna — `primary`), a drugiej po prawej (ikona pomocnicza — `secondary`). W przypadku tego przycisku jedna ikona w zupełności wystarczy.

6. **Zapisz oba pliki, *todo.js* oraz *index.html*, a następnie wyświetl stronę *index.html* w przeglądarce.**

   Teraz przycisk powinien już wyglądać tak samo jak widżet przycisku jQuery UI widoczny na rysunku 14.1. Jeśli tak nie wygląda, oznacza to, że przygotowany kod JavaScript nie działa. Dokładnie go sprawdź, używając konsoli JavaScript, by upewnić się, czy nie ma komunikatów o błędach.

Dodanie okna dialogowego

A zatem mamy już przycisk, który jednak nic nie robi. Docelowo kliknięcie tego przycisku powinno powodować wyświetlenie okna dialogowego — jego zaimplementowanie będzie Twoim kolejnym zadaniem. Najpierw dodasz kod HTML okna dialogowego.

1. **W pliku *index.html* odszukaj komentarz `<!-- Tutaj dodaj okienko dialogowe. -->` i zastąp go poniższym kodem HTML:**

```
<div id="new-todo" title="Dodaj zadanie">

  <form>
    <p>
      <label for="task">Nazwa zadania:</label>
      <input type="text" name="task" id="task">
    </p>
  </form>
</div>
```

   Zgodnie z tym, czego dowiedziałeś się na stronie 331, w okno dialogowe możesz zamienić dowolny fragment kodu HTML. Tu użyjemy elementu `<div>`, wewnątrz którego znajduje się formularz z jednym polem tekstowym. Podczas dodawania zadania użytkownicy będą wpisywali w tym polu nazwę zadania.

   Aby przekształcić ten kod HTML w okno dialogowe, będziesz musiał użyć kodu JavaScript.

2. **Wróć do pliku *todo.js*. Poniżej kodu, który dodałeś w kroku 5. na stronie 521, wpisz:**

```
$("#new-todo").dialog();
```

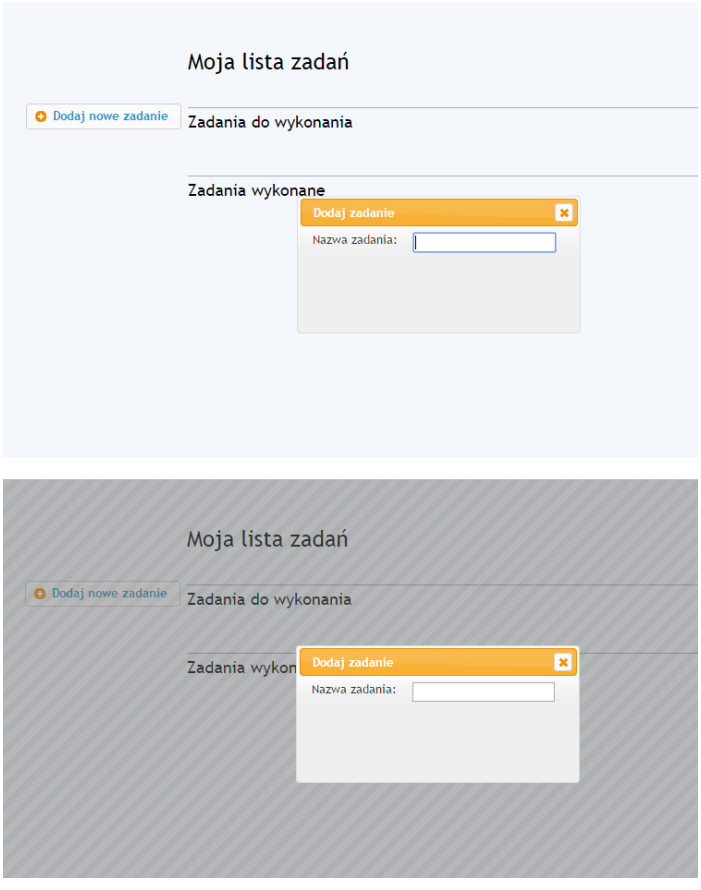
   Zapisz plik *index.html* i odśwież stronę w przeglądarce. Powinno się na niej pojawić okno dialogowe (patrz górny zrzut na rysunku 14.2). Jednak jest ono wyświetlane od razu, a nie po kliknięciu przycisku. To standardowe zachowanie okien dialogowych tworzonych przy użyciu biblioteki jQuery UI. Aby ukryć to okno, będziesz musiał przekazać w wywołaniu funkcji `dialog()` odpowiednie opcje.

3. **W wywołaniu funkcji `dialog()` umieść obiekt zawierający dwie właściwości:**

```
$("#new-todo").dialog({
  modal : true,
  autoOpen : false
});
```

   Zgodnie z tym, czego się dowiedziałeś na stronie 335, opcja `modal` zmusza użytkownika do zamknięcia okna dialogowego, zanim będzie mógł wykonać na stronie jakąkolwiek inną operację. I właśnie o to chodzi — kiedy użytkownik kliknie przycisk *Dodaj nowe zadanie*, wypełnienie okna dialogowego powinno być jedyną rzeczą, na której użytkownik ma się skoncentrować.

   Z kolei przypisanie właściwości `autoOpen` wartości `false` sprawi, że okno dialogowe nie będzie już wyświetlane natychmiast po wczytaniu strony. Teraz będzie początkowo ukryte i trzeba wyświetlić je programowo, w odpowiedzi na kliknięcie przycisku!



Rysunek 14.2. *Okna dialogowe jQuery UI są automatycznie wyświetlane zaraz po wczytaniu strony. To doskonałe rozwiązanie w przypadku prezentowania ważnych informacji, które użytkownicy muszą zobaczyć bezpośrednio po wejściu na stronę. Jednak w większości innych sytuacji, takich jak dodawanie elementów do listy zadań, nie jest już takie dobre. Zwykle będziesz chciał, by okno dialogowe było ukryte, aż do momentu wykonania jakiejś akcji przez użytkownika — na przykład takiej jak kliknięcie przycisku „Dodaj nowe zadanie"*

4. **Dodaj funkcję obsługującą zdarzenia `click`, dopisując do wywołania funkcji `.button()` wywołanie `.click()`:**

```
$("#add-todo").button({
  icons: {
    primary: "ui-icon-circle-plus"
  }
}).click(function() {
  $('#new-todo').dialog('open');
});
```

Wywołania funkcji jQuery można łączyć w sekwencje — w tym celu na końcu jednej z nich wystarczy dopisać kropkę, a za nią umieścić wywołanie kolejnej funkcji. W tym przypadku kod najpierw wybiera element o identyfikatorze `add-todo` (czyli przycisk), potem wywołuje funkcję jQuery UI `button()` i w końcu dodaje do przycisku funkcję, która będzie obsługiwać zdarzenia `click`. Dodana funkcja obsługująca zdarzenia wywołuje z kolei funkcję `open()` okna dialogowego (patrz strona 338). Innymi słowy, kliknięcie przycisku powinno teraz powodować wyświetlenie okna dialogowego.

Nadszedł czas, żeby przetestować działanie kodu.

5. **Zapisz plik *todo.js*, po czym wyświetl w przeglądarce stronę *index.html*. Kliknij przycisk *Dodaj nowe zadanie*.**

   Teraz okno dialogowe powinno być wyświetlane wyłącznie po kliknięciu przycisku (u dołu rysunku 14.2). Co więcej, cały obszar poniżej okna dialogowego powinien zostać nieco zaciemniony — przykryty przez prążkowaną, półprzezroczystą warstwę. To wizualne oznaczenie informuje użytkownika, że zostało wyświetlone *modalne* okno dialogowe, co z kolei znaczy, że dopóki go nie zamknie, nie będzie mógł zrobić nic innego. Jednak aktualnie nie ma jak zamknąć tego okna dialogowego!

   Na szczęście za pomocą jQuery UI dodawanie przycisków do okien dialogowych jest bardzo proste.

6. **Wróć do pliku *todo.js*. Do listy opcji okna dialogowego dodaj kolejną — `buttons`, jej wartością ma być pusty literał obiektowy:**

```
$("#new-todo").dialog({
  modal : true,
  autoOpen : false,
  buttons : {

  }
});
```

   Opcja `buttons` pozwala określać przyciski, które jQuery UI dynamicznie doda do okna dialogowego. Oprócz tego, do każdego z nich można dodać funkcję, dzięki czemu coś się stanie, kiedy użytkownik kliknie dany przycisk. Tworzymy kod aplikacji wolno, krok po kroku, ponieważ dzieje się w nim całkiem dużo. Teraz na przykład dysponujesz nowym obiektem (właściwością `buttons`) umieszczonym wewnątrz innego obiektu (literału obiektowego z opcjami, przekazywanego w wywołaniu funkcji `dialog()`).

   Najpierw dodaj przycisk z pustą funkcją.

7. **Do literału obiektowego wpisanego w poprzednim kroku dodaj nową właściwość (wyróżnioną pogrubioną czcionką):**

```
$("#new-todo").dialog({
  modal : true,
  autoOpen : false,
  buttons : {
    "Dodaj zadanie" : function () {

    }
  }
});
```

   Ten kod powoduje dodanie do okna dialogowego przycisku z napisem *Dodaj zadanie*. Kiedy użytkownik kliknie ten przycisk, zostanie wykonana funkcja. Obecnie funkcja jest pusta, więc nic się nie stanie. Funkcję tę zaimplementujesz do końca w następnej części przykładu. A teraz dodasz do okna dialogowego drugi przycisk.

8. **Za zamykającym nawiasem klamrowym funkcji umieszczonej we właściwości `"Dodaj zadanie"` wpisz przecinek i dodaj kolejną właściwość z funkcją anonimową:**

```
$("#new-todo").dialog({
  modal : true,
  autoOpen : false,
```

```
      buttons : {
        "Dodaj zadanie" : function () {

        },
        "Anuluj" : function () {
          $(this).dialog('close');
        }
      }
    });
```

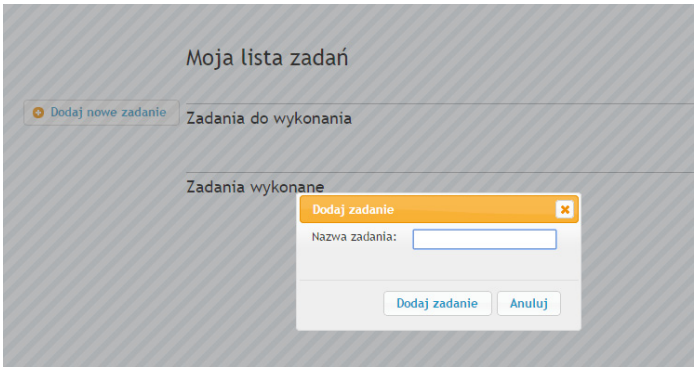
Ten nowy fragment kodu dodaje do okna dialogowego przycisk *Anuluj*. Co więcej, przycisk już coś robi. Wyrażenie `$(this)` odwołuje się do elementu, który wywołał funkcję, czyli do samego okna dialogowego. Kiedy użytkownik kliknie przycisk *Anuluj*, zostaje wywołana funkcja `close()` widżetu okna dialogowego (patrz strona 340). Powoduje ona natychmiastowe zamknięcie tego okna.

Nadszedł czas, by zobaczyć efekt wykonanej pracy.

9. **Zapisz plik *todo.js* i odśwież stronę index.html w przeglądarce. Kliknij przycisk *Dodaj nowe zadanie*.**

   Okno dialogowe powinno teraz zawierać dwa przyciski (patrz rysunek 14.3). Kliknij przycisk *Dodaj zadanie* — oczywiście nic się nie powinno stać, gdyż obsługa przycisku nie została jeszcze zaimplementowana. Kiedy jednak klikniesz drugi przycisk, czyli *Anuluj*, okno powinno zostać zamknięte.

   W następnej części przykładu zajmiesz się pisaniem kodu, który pozwoli na dodawanie zadań do listy.



Rysunek 14.3. *Wystarczyło kilka wierszy kodu, aby przygotować wszystkie komponenty prostego interfejsu użytkownika aplikacji do zarządzania listą zadań*

Dodawanie zadań

Nasza aplikacja do zarządzania listą zadań wygląda już całkiem dobrze, jednak nie dzieje się w niej zbyt dużo. Nadszedł zatem czas, byś zajął się kodem umożliwiającym dodawanie zadań. Zostanie on umieszczony w funkcji skojarzonej z przyciskiem *Dodaj zadanie*, czyli w pustej funkcji anonimowej, którą dopisałeś w kroku 5. na stronie 524. Implementację tego kodu wykonasz w opisanych poniżej czterech krokach.

1. **Pobierz nazwę zadania, którą użytkownik wpisał w polu tekstowym w oknie dialogowym.**

   Wartość ta jest przechowywana w polu tekstowym, a to oznacza, że możesz ją łatwo pobrać, używając funkcji `val()` jQuery (patrz strona 283).

2. **Utwórz element `<li>`, który następnie dodasz do treści strony.**

   Każde zadanie będzie reprezentowane przez jeden element listy wypunktowanej. Poniżej przedstawiona została struktura kodu HTML takiego elementu:

```
<li>
  <span class="done">%</span>
  <span class="delete">x</span>
  <span class="task">Upiec ciasteczka</span>
</li>
```

   Pierwszy element `<span>` reprezentuje obszar, który użytkownik może kliknąć, by oznaczyć zadanie jako wykonane. Drugi element `<span>` reprezentuje przycisk do usunięcie zadania. I w końcu ostatni element `<span>` zawiera nazwę zadania podaną przez użytkownika w oknie dialogowym. Taki kod HTML możesz skonstruować, łącząc ze sobą literały łańcuchowe zawierające odpowiednie znaczniki i dodając do nich wpisaną przez użytkownika nazwę zadania.

Uwaga: Postać obu przycisków określisz, stosując odpowiednie style CSS. Dzięki temu znak procenta (`%`) oraz `x` umieszczone w kodzie po wyświetleniu strony zostaną zamienione odpowiednio na znacznik charakterystyczny dla pól wyboru oraz przycisk z ikoną krzyżyka. Tajemnica tych przycisków tkwi w zastosowaniu specjalnej czcionki, w której zamiast liter są ikony.

---

3. **Dodaj do listy zadań do wykonania nowy element listy, znacznik `<li>`.**

   Biblioteka jQuery sprawia, że bardzo łatwo można zamienić łańcuch znaków na element DOM, a następnie dodać go do strony (szczegółowe informacje na ten temat można znaleźć w rozdziale 4.). Ponieważ dysponujemy potężnym zestawem efektów jQuery UI, możesz skorzystać z nich, by dodawać nowe zadania w sposób atrakcyjny wizualnie.

4. **Zamknij okno dialogowe.**

   Ta operacja jest bardzo prosta. Wykonałeś ją już wcześniej, przy okazji implementowania przycisku do zamykania okna dialogowego, w kroku 8. opisanym na stronie 524.

Skoro już znasz podstawowe kroki, jakie należy zaimplementować, czas zabrać się za pisanie kodu. Najpierw pobierzesz dane wpisane przez użytkownika, czyli nazwę zadania wpisaną w oknie dialogowym.

1. **Odszukaj funkcję skojarzoną z przyciskiem *Dodaj zadanie* okna dialogowego (została ona podana we właściwości `buttons` literału obiektowego przekazywanego w wywołaniu funkcji `dialog()`), a następnie wpisz w niej poniższy kod:**

```
"Dodaj zadanie" : function () {
  var taskName = $('#task').val();
},
```

   Powyższa instrukcja tworzy nową zmienną, `taksName`, i zapisuje w niej wartość pola tekstowego umieszczonego w oknie dialogowym. Pamiętasz zapewne strukturę kodu tworzącego okno dialogowe, została ona przedstawiona w kroku 1.

na stronie 522. Kod ten zawiera między innymi pole tekstowe o określonym identyfikatorze. A zatem wywołanie `$('#task').val()` pobiera wartość tego pola — czyli to, co użytkownik wpisał jako nazwę zadania.

Istnieje także możliwość, że użytkownik kliknie przycisk *Dodaj zadanie*, mimo że pole tekstowe będzie puste. Ponieważ nie chcemy dodawać do naszej listy pustych zadań, zatem powinieneś się upewnić, że wartość zmiennej `taskName` będzie inna od pustego łańcucha znaków.

2. **Poniżej wiersza kodu dodanego w poprzednim kroku wpisz trzy kolejne (wyróżnione pogrubioną czcionką):**

```
"Dodaj zadanie" : function () {
  var taskName = $('#task').val();
  if (taskName === '') {
    return false;
  }
},
```

Ten kod jedynie upewnia się, że zmienna `taskName` zawiera coś innego niż pusty łańcuch znaków (został on zapisany przy użyciu dwóch znaków apostrofu, z których pierwszy oznacza początek łańcucha, a drugi jego koniec), a użytkownik nie kliknął przycisku *Dodaj zadanie* bez podania tytułu zadania. Instrukcja `return false` powoduje zakończenie funkcji, co z kolei sprawia, że okno dialogowe pozostanie widoczne. A zatem użytkownik musi wpisać nazwę zadania lub zamknąć okno dialogowe (co może także zrobić, klikając niewielki przycisk „X" widoczny w jego prawym, górnym rogu).

Teraz zajmiesz się tworzeniem kodu HTML reprezentującego zadanie.

3. **Dodaj kolejną zmienną, a następnie skonstruuj i zapisz w niej łańcuch znaków; do tego celu posłużą trzy kolejne wiersze kodu (wyróżnione pogrubioną czcionką):**

```
"Dodaj zadanie" : function () {
  var taskName = $('#task').val();
  if (taskName === '') {
    return false;
  }
  var taskHTML = '<li><span class="done">%</span>';
  taskHTML += '<span class="delete">x</span>';
  taskHTML += '<span class="task"></span></li>';
},
```

Do utworzenia zmiennej `taskHTML` i zapisania w niej całego, długiego łańcucha znaków wystarczyłby jeden wiersz kodu. Jednak w ten sposób powstałaby instrukcja, którą trudno przeanalizować. Podzielenie długiego łańcucha znaków na części i zapisanie go w kilku kolejnych wierszach pozwala poprawić czytelność kodu. Operator += służy do dołączenia podanego łańcucha znaków do łańcucha, który już jest zapisany w zmiennej (patrz strona 72).

Zauważ, że jeszcze nie dodałeś do tego łańcucha nazwy zadania podanej przez użytkownika i przechowywanej w zmiennej `taskName`. Zrobisz to już za chwilę.

4. **Dodaj kolejną zmienną i zapisz w niej obiekt jQuery zawierający obiekt utworzony na podstawie kodu HTML (nowy kod został wyróżniony pogrubioną czcionką):**

```
"Dodaj zadanie" : function () {
  var taskName = $('#task').val();
  if (taskName === '') {
```

```
      return false;
    }
    var taskHTML = '<li><span class="done">%</span>';
    taskHTML += '<span class="delete">x</span>';
    taskHTML += '<span class="task"></span></li>';
    var $newTask = $(taskHTML);
  },
```

Biblioteka jQuery umożliwia przekształcenie łańcucha znaków, takiego jak `<h1>To jest nagłówek</h1>` , na element DOM. Innymi słowy, `taskHTML` jest zmienną zawierającą łańcuch znaków. Łańcuch ten nie jest jednak „prawdziwym" kodem HTML; natomiast przekazanie łańcucha zawierającego znaczniki HTML w wywołaniu funkcji `$()` spowoduje przekształcenie go na element DOM. A nawet jeszcze lepiej — funkcja `$()` przekształca łańcuch znaków na obiekt jQuery, dzięki czemu można na nim wywoływać standardowe funkcje tej biblioteki. Choć znak `$` na początku zmiennej `$newTask` nie jest konieczny, jednak umieszczanie go na początku nazw zmiennych stanowi praktykę powszechnie stosowaną przez programistów używających jQuery. Ten znak dolara stanowi wizualną podpowiedź informującą, że dana zmienna zawiera obiekt jQuery i można jej używać do wywoływania wszystkich metod biblioteki, takich jak `.show()` czy też `.addClass()`.

W następnym kroku dodasz nazwę zadania podaną przez użytkownika.

5. **Do kodu funkcji obsługującej przycisk *Dodaj zadanie* dopisz kolejny wiersz:**

```
"Dodaj zadanie" : function () {
  var taskName = $('#task').val();
  if (taskName === '') {
    return false;
  }
  var taskHTML = '<li><span class="done">%</span>';
  taskHTML += '<span class="delete">x</span>';
  taskHTML += '<span class="task"></span></li>';
  var $newTask = $(taskHTML);
  $newTask.find('.task').text(taskName);
},
```

Metoda `find()` poszukuje elementu odpowiadającego podanemu selektorowi *wewnątrz* aktualnie wybranego elementu (patrz strona 555). W tym przypadku metoda operuje na elemencie `<li>` przechowywanym w zmiennej `$newTask` i poszukuje w nim innego elementu, należącego do klasy `task` — czyli elementu `<span>`, w którym powinna zostać zapisana nazwa zadania (patrz krok 3.). Następnie wywołanie funkcji `text()` zapisuje zawartość zmiennej `taskName` w odnalezionym elemencie.

Ale dlaczego masz zadawać sobie tyle trudu, zamiast po prostu dodać wartość zmiennej `taskName` do łańcucha znaków tworzonego w kroku 4. w następujący sposób:

```
taskHTML += '<span class="task">' + taskName + '</span></li>';
```

Gdybyś postąpił w ten sposób, złośliwy użytkownik mógłby utworzyć zadanie o następującej nazwie: `<script>alert('ha, ha, ha... włamałem się do tej listy');</script>`. Ten złośliwy kod zostałby dodany bezpośrednio do strony i wykonany. Natomiast funkcja `text()` jQuery zamienia wszystkie znaczniki HTML na ich bezpieczne odpowiedniki, czyli zamienia `<script>` na `<script>`.

Jeśli nawet użytkownik nie ma żadnych złych intencji, takie rozwiązanie pozwoli mu wpisać w nazwie zadania całkowicie prawidłowy tekst, taki jak „Dodać do strony głównej znacznik <h1>", bez spowodowania awarii aplikacji.

W końcu nadszedł czas, by umieścić nowe zadanie na stronie.

6. **Do kodu funkcji dodaj kolejne dwa wiersze (wyróżnione pogrubioną czcionką):**

```
"Dodaj zadanie" : function () {
  var taskName = $('#task').val();
  if (taskName === '') {
    return false;
  }
  var taskHTML = '<li><span class="done">%</span>';
  taskHTML += '<span class="delete">x</span>';
  taskHTML += '<span class="task"></span></li>';
  var $newTask = $(taskHTML);
  $newTask.find('.task').text(taskName);
  $newTask.hide();
  $('#todo-list').prepend($newTask);
},
```

Ponieważ `$newTask` jest obiektem jQuery, można wywoływać na jego rzecz funkcje biblioteki. W dodanym fragmencie kodu najpierw ukrywasz nowe zadanie, dzięki czemu później będziesz mógł je wyświetlić, używając efektu animacji. Po ukryciu elementu kolejna instrukcja najpierw wybiera element listy — jest to lista wypunktowana o identyfikatorze `todo-list` — a następnie dodaje na jej początku nowy (wciąż ukryty) element (więcej informacji na temat działania metody `prepend()` można znaleźć na stronie 159).

W kolejnym kroku wyświetlisz nowy element listy i ukryjesz okno dialogowe.

7. **Do kodu funkcji dodaj kolejne dwa wiersze (zmiany zostały wyróżnione pogrubioną czcionką):**

```
"Dodaj zadanie" : function () {
  var taskName = $('#task').val();
  if (taskName === '') {
    return false;
  }
  var taskHTML = '<li><span class="done">%</span>';
  taskHTML += '<span class="delete">x</span>';
  taskHTML += '<span class="task"></span></li>';
  var $newTask = $(taskHTML);
  $newTask.find('.task').text(taskName);
  $newTask.hide();
  $('#todo-list').prepend($newTask);
  $newTask.show('clip',250).effect('highlight',1000);
  $(this).dialog('close');
},
```

Pierwsza z dodanych instrukcji operuje na dodanym do listy ukrytym elemencie i wyświetla go przy użyciu metody `show()`. Dla dodatkowego poprawienia atrakcyjności strony zastosowane zostały dwa efekty wizualne jQuery UI. Pierwszy z nich, `clip` (patrz strona 463), sprawia, że element wydaje się powiększać. Kiedy element będzie już widoczny, wywołujemy metodę `effect()`, każąc jej odtworzyć efekt `highlight`, który na chwilkę wyświetla element na żółto, przyciągając uwagę użytkownika (i nieodmiennie powodując zachwyt osób korzystających z aplikacji).

Ostatni wiersz kodu zamyka okno dialogowe (ten sam kod zastosowałeś już wcześniej, w kroku 8. na stronie 524, dodając go do przycisku *Anuluj*). Teraz trzeba sprawdzić, jak to wszystko działa.

8. **Zapisz plik todo.js, a następnie odśwież w przeglądarce stronę index.html. Kliknij przycisk *Dodaj nowe zadanie*, wpisz nazwę zadania i kliknij przycisk *Dodaj zadanie*.**

   Nowe zadanie powinno się pojawić poniżej nagłówka *Zadania do wykonania*. Jeśli jednak nie widać go, sprawdź kod i zajrzyj do konsoli JavaScript przeglądarki, by zobaczyć, czy nie ma w niej komunikatów o błędach.

   Spróbuj dodać kolejne zadanie. Przy tej okazji zauważysz zapewne kolejny, niewielki problem — w polu tekstowym w oknie dialogowym pozostała nazwa poprzedniego zadania. Aby dodać nowe, musisz najpierw zaznaczyć ten tekst i go usunąć. Na szczęście ten problem można rozwiązać bardzo łatwo.

9. **W literale obiektowym przekazywanym w wywołaniu funkcji `dialog()`, za właściwością `buttons` wpisz przecinek i dodaj poniższy fragment kodu:**

```
close: function() {
  $('#new-todo input').val('');
}
```

   Pełny kod funkcji `dialog()` powinien mieć następującą postać:

```
$("#new-todo").dialog({
  modal: true,
  autoOpen: false,
  buttons : {
    "Dodaj zadanie" : function() {
      var taskName = $('#task').val();
      if (taskName === '') {
        return false;
      }
      var taskHTML = '<li><span class="done">%</span>';
      taskHTML += '<span class="delete">x</span>';
      taskHTML += '<span class="task"></span></li>';
      var $newTask = $(taskHTML);
      $newTask.find('.task').text(taskName);
      $newTask.hide();
      $('#todo-list').prepend($newTask);
      $newTask.show('clip',250).effect('highlight',1000);
      $(this).dialog('close');
    },
    "Anuluj" : function() {
      $(this).dialog('close');
    }
  },
  close: function() {
    $('#new-todo input').val('');
  }
});
```

   To naprawdę całkiem sporo kodu. Upewnij się, że ten ostatni fragment umieściłeś poza obiektem `buttons`. Opcja `close` widżetu okna dialogowego jQuery UI pozwala na wykonanie podanej funkcji w momencie, gdy użytkownik zamknie okno dialogowe. W tym przypadku wykonywane operacje ograniczą się do wyczyszczenia pola tekstowego, kiedy zatem użytkownik następnym razem otworzy okno dialogowe, pole będzie puste i gotowe do wpisania nazwy kolejnego zadania. (Zawartość pola można by także usunąć w ramach obsługi kliknięcia przycisku *Dodaj zadanie*, jednak chodziło tu o zaprezentowanie opcji `close`).

Oznaczanie zadania jako wykonanego

Jedną z czynności, które podczas korzystania z listy zadań dają najwięcej satysfakcji, jest oznaczanie zadań jako wykonanych. Niestety nasza aplikacja jeszcze nie udostępnia tej jakże przyjemnej operacji. W tym podrozdziale naprawisz to niedopatrzenie. Ogólnie rzecz biorąc, zadanie jest całkiem proste: użytkownik zaznacza puste pole wyboru z lewej strony nazwy i zadanie — czyli element listy — jest przenoszone z jednej listy na drugą.

Delegowanie zdarzeń

Aby oznaczyć zadanie jako wykonane, użytkownik musi kliknąć pole wyboru umieszczone w danym elemencie listy. Zazwyczaj funkcje obsługujące zdarzenia są określane poprzez wybranie odpowiedniego elementu i skojarzenie z nim funkcji; operację tę można wykonać w następujący sposób:

```
$('.done').click(function () {
  // Operacja wykonywana po kliknięciu elementu.
});
```

Jednak sposób obsługi zdarzeń w tworzonej liście zadań będzie nieco inny. Bezpośrednio po wyświetleniu strony w przeglądarce lista zadań jest pusta — nie ma żadnych zadań ani żadnych pól wyboru, które użytkownik mógłby kliknąć. Gdybyśmy spróbowali określić funkcję obsługującą zdarzenia `click` bezpośrednio po wczytaniu strony, nic by się nie stało. Ponieważ w tym momencie na stronie nie ma żadnych zadań ani żadnych pól wyboru, nie byłoby na niej żadnych elementów, w których można by zastosować tę funkcję. Oczywiście można by wywoływać metodę `click()` podczas tworzenia każdego nowego zadania, jednak biblioteka jQuery udostępnia znacznie lepsze rozwiązanie, nazywane delegowaniem zdarzeń (patrz strona 200).

Ogólnie rzecz biorąc, mechanizm delegowania zdarzeń pozwala wybrać jakiś inny element strony — element już istniejący, w którym będą umieszczane elementy dynamicznie dodawane do strony po jej wczytaniu. To właśnie ten element kontenera będzie odbierał zdarzenia skierowane do elementów umieszczonych wewnątrz niego, w naszym przypadku będą to zdarzenia `click`. Kiedy takie zdarzenie zostanie odebrane, kontener sprawdzi, czy faktycznie było ono skierowane do odpowiedniego elementu (w naszym przypadku do pola wyboru w którymś z zadań) i jeśli było, wykona odpowiednią funkcję.

W tym przypadku pusta lista wypunktowana znajduje się na stronie od momentu jej pobrania:

```
<ul id="todo-list">
</ul>
```

Kiedy użytkownik tworzy nowe zadanie, aplikacja dynamicznie dodaje je do tej listy:

```
<ul id="todo-list">
  <li>
    <span class="done">%</span>
    <span class="delete">x</span>
    <span class="task">Upiec ciasto</span>
  </li>
</ul>
```

Aby zareagować na kliknięcie elementu należącego do klasy done, należy delegować funkcję obsługującą zdarzenia do istniejącej już listy wypunktowanej.

1. **W pliku *todo.js* za kodem funkcji `dialog()`, lecz wciąż wewnątrz wywołania funkcji `$(document).ready()`, wpisz następujące wywołanie:**

```
$("#todo-list").on('click', '.done', function() {

});
```

Tak wygląda ogólna struktura delegowanej funkcji obsługi zdarzeń. Najpierw wybierasz element listy wypunktowanej. Następnie wywoływana jest metoda on() jQuery, do której przekazujesz trzy argumenty. Pierwszym z nich jest łańcuch znaków zawierający nazwę zdarzenia, 'click'. Drugim argumentem jest selektor określający element wewnątrz listy wypunktowanej, który musi zostać kliknięty. W tym przypadku chodzi o element klasy done. I w końcu ostatnim argumentem wywołania jest funkcja — kod, który ma zostać wykonany, kiedy użytkownik kliknie pole wyboru oznaczające zadanie jako wykonane.

2. **Wewnątrz funkcji obsługującej zdarzenia dodaj poniższy wiersz kodu (wyróżniony pogrubioną czcionką):**

```
$("#todo-list").on('click', '.done', function() {
  var $taskItem = $(this).parent('li');
});
```

Pamiętaj, że zadaniem tej funkcji jest przeniesienie zadania z listy *Zadania do wykonania* na listę *Zadania wykonane*. W chwili, gdy użytkownik kliknie pole wyboru, dla jQuery istnieje jedynie znacznik umieszczony gdzieś na liście, jednak nas interesuje cały znacznik <li>. Możesz go pobrać, korzystając z metody parent() jQuery, która umożliwia pobranie elementu rodzica aktualnie wybranego elementu.

W kodzie dodanym w tym kroku wyrażenie $(this) odwołuje się do elementu klikniętego przez użytkownika, czyli: %. A zatem całe wywołanie $(this).parent('li') pobiera najbliższego przodka znacznika , którym jest znacznik <li>. Innymi słowy, wybiera ono dokładnie ten element, o który chodziło.

Element ten jest następnie zapisywany w zmiennej $taskItem, a w kolejnym kroku go ukryjesz.

Uwaga: Wyczerpujące informacje na temat różnic pomiędzy this i $(this) można znaleźć na stronie 169.

---

3. **Wewnątrz tej samej funkcji dodaj poniższy fragment kodu (wyróżniony pogrubioną czcionką):**

```
$("#todo-list").on('click', '.done', function() {
  var $taskItem = $(this).parent('li');
  $taskItem.slideUp(250, function() {

  });
});
```

Metoda slideUp() jest zabawnym sposobem ukrycia elementu (patrz strona 216). Jednak jej wywołanie nie usuwa elementu ze strony. Po zakończeniu jej wywołania element wciąż jest dostępny na stronie, choć został ukryty przy użyciu

odpowiednich stylów CSS. Zgodnie z informacjami zamieszczonymi na stronie 211, wszystkie funkcje jQuery tworzące efekty animacji (takie jak `hide()`, `show()` czy też `slideUp()`) pozwalają na podawanie argumentów. W tym przypadku pierwszym z argumentów jest liczba — `250` — określająca czas trwania animacji; w naszym przypadku zajmie ona 250 milisekund.

Drugim argumentem jest funkcja zwrotna. Jest to funkcja, która zostanie wywołana *po* zakończeniu odtwarzania animacji. Pamiętasz zapewne, że po zakończeniu wywołania funkcji `slideUp()` znacznik `<li>` reprezentujący zakończone zadanie wciąż znajduje się na liście *Zadania do wykonania*, tyle że jest niewidoczny. Musisz zatem przenieść go na drugą listę. Kod, który to zrobi, umieścisz właśnie w funkcji zwrotnej.

4. **Wewnątrz funkcji zwrotnej dodaj poniższy fragment kodu (wyróżniony pogrubioną czcionką):**

```
$("#todo-list").on('click', '.done', function() {
  var $taskItem = $(this).parent('li');
  $taskItem.slideUp(250, function() {
    var $this = $(this);
    $this.detach();
  });
});
```

Pierwszy z dwóch nowych wierszy kodu jedynie pobiera element listy — `$(this)` — i zapisuje go w kolejnej zmiennej. Robisz to dlatego, że każde wywołanie funkcji jQuery — `$( )` — zmusza przeglądarkę do wykonania pewnej pracy. Ponieważ na tym elemencie będziesz musiał wykonać kilka operacji, zatem zamiast niepotrzebnie za każdym razem wywoływać funkcję jQuery, lepiej będzie zapisać wynik jej wywołania w zmiennej. (Rozwiązanie to, opisane bardziej szczegółowo na stronie 544, stanowi jedną z ogólnie przyjętych najlepszych praktyk związanych ze stosowaniem biblioteki jQuery).

Drugi z dodanych wierszy kodu wywołuje metodę `detach()`, która usuwa wybrany element lub elementy z drzewa DOM, choć pozostawia je na stronie. Innymi słowy, wybrany element jest usuwany z listy, lecz wciąż znajduje się w pamięci przeglądarki. Co więcej, wciąż jest zapisany w zmiennej `$this`. Dzięki temu w następnym kroku będziesz mógł przenieść go w inne miejsce strony — a konkretnie, na drugą listę!

Uwaga: Wyczerpujące informacje na temat metody `detach()` można znaleźć na stronie *http://api.jquery.com/detach/*.

---

5. **Dokończ kod funkcji zwrotnej, dopisując wewnątrz niej kolejne dwa wiersze kodu (wyróżnione pogrubioną czcionką):**

```
$("#todo-list").on('click', '.done', function() {
  var $taskItem = $(this).parent('li');
  $taskItem.slideUp(250, function() {
    var $this = $(this);
    $this.detach();
    $('#completed-list').prepend($this);
    $this.slideDown();
  });
});
```

Metodę `prepend()` poznałeś już wcześniej (w kroku 6. na stronie 529). Za jej pomocą wstawisz kod HTML wewnątrz innego elementu. W tym przypadku odłączony element listy — `$this` — zostaje zapisany na początku listy *Zadania wykonane* (listy wypunktowanej o identyfikatorze `completed-list`). I w końcu, ponieważ przenoszony element został ukryty przez wywołanie metody `slideUp()`, zatem teraz możesz go wyświetlić w nowym położeniu — w tym celu wywoływana jest metoda `slideDown()` (patrz strona 216).

6. **Zapisz plik *todo.js*, a następnie odśwież stronę index.html w przeglądarce. Aby ją przetestować, najpierw dodaj kilka zadań, a następnie kliknij pole wyboru umieszczone z lewej strony nazwy zadania, aby oznaczyć je jako wykonane.**

   Teraz powinieneś już bez problemu dodawać zadania i oznaczać je jako wykonane (patrz rysunek 14.4). Jeśli przykładowa aplikacja nie działa, dokładnie sprawdź kod i zajrzyj do konsoli JavaScript, by zobaczyć, czy nie zostały w niej wyświetlone jakieś komunikaty o błędach.

Rysunek 14.4. *Pole wyboru oraz przycisk, który pozwala oznaczyć zadanie jako wykonane oraz je usunąć, zmieniają wygląd po wskazaniu ich myszą. Możliwości tej nie zapewnia jednak kod JavaScript, lecz style CSS. A konkretnie, pseudoklasa :hover pozwalająca na zmianę wyglądu elementu, na którym znajduje się wskaźnik myszy. Aby sprawdzić, jak tworzony jest ten efekt, zajrzyj do pliku todo.css umieszczonego w podkatalogu css katalogu R14*

A teraz kolejna sprawa — może warto byłoby mieć możliwość zmiany kolejności zadań na liście? Dzięki temu można by dodać kilka zadań, a następnie uporządkować je w takiej kolejności, w jakiej powinny zostać wykonane, na przykład: „Kupić książkę kucharską", „Upiec ciasteczka", „Zjeść ciasteczka". Przy użyciu widżetu Sortable jQuery UI takie możliwości funkcjonalne można uzyskać bardzo szybko i łatwo. A skoro już przy tym jesteśmy, równie łatwo możesz zapewnić użytkownikom możliwość dowolnego przenoszenia zadań między oboma listami, zatem będzie można oznaczyć zadanie jako wykonane poprzez przeciągnięcie go na listę *Zadania wykonane*.

7. **Otwórz plik *index.html* w edytorze tekstów i odszukaj wewnątrz niego znaczniki `<ul>` obu list: *Zadania do wykonania* oraz *Zadania wykonane*. Do obu dodaj atrybut `class="sortlist"`, tak by miały następującą postać:**

```
<ul id="todo-list" class="sortlist">
```

oraz

```
<ul id="completed-list" class="sortlist">
```

Przypisując tę samą nazwę klasy obu listom, będziesz mógł w prosty sposób zapewnić możliwość porządkowania elementów na każdej z nich.

8. **Zapisz plik *index.html*, po czym otwórz w edytorze plik *todo.js*. Na samym końcu funkcji `$(document).ready()`, lecz wciąż wewnątrz niej, wpisz następujące wywołanie:**

```
$('.sortlist').sortable();
```

W ten sposób obie listy zapewniają już możliwość porządkowania elementów. Jeśli chcesz sprawdzić, jak to działa, zapisz oba pliki, a następnie odśwież stronę *index.html* w przeglądarce. Zauważysz jednak, że choć faktycznie można zmieniać kolejność elementów na liście, jednak nie da się przeciągać elementów jednej listy na drugą. Bardzo łatwo można jednak połączyć ze sobą dwa widżety Sortable.

9. **Wewnątrz wywołania funkcji `sortable()` wstaw następujący literał obiektowy:**

```
$('.sortlist').sortable({
  connectWith : '.sortlist'
});
```

Opcja `connectWith` (opisana szczegółowo na stronie 452) umożliwia połączenie dwóch list. Ponieważ w naszej aplikacji obie listy należą do tej samej klasy, aby zatem je połączyć, wystarczy podać selektor tej klasy. W ten sposób użytkownik będzie już mógł dowolnie przeciągać elementy pomiędzy oboma listami. Jednak zanim definitywnie zakończysz pracę nad listami, dodasz do nich jeszcze kilka ostatnich, drobnych usprawnień wizualnych.

10. **Do tego samego literału obiektowego dodaj trzy kolejne właściwości (wyróżnione pogrubioną czcionką):**

```
$('.sortlist').sortable({
  connectWith : '.sortlist',
  cursor : 'pointer',
  placeholder : 'ui-state-highlight',
  cancel : '.delete,.done'
});
```

Nie zapomnij o przecinku za opcją `connectWith`. Opcja `cursor` (opisana na stronie 425) zmienia postać wskaźnika myszy podczas przeciągania elementów, a opcja `placeholder` (patrz strona 454) wyróżnia miejsce listy, gdzie użytkownik może upuścić przeciągany element. I w końcu opcja `cancel` (patrz strona 451) określa te elementy umieszczone wewnątrz elementu sortowalnego, które nie mogą posłużyć jako „uchwyty" do przeciągania. W naszym przypadku użytkownik nie będzie mógł przeciągać zadania, używając pola wyboru oznaczającego zadanie jako wykonane ani przycisku do usunięcia zadania.

11. **Zapisz pliki *index.html* oraz *todo.js*, po czym odśwież plik *index.html* w przeglądarce.**

Dodaj do listy kilka różnych zadań. Spróbuj je przeciągać pomiędzy obiema listami. Teraz powinieneś już oznaczyć zadanie jako wykonane poprzez samo przeciągnięcie go na listę *Zadania wykonane* (patrz rysunek 14.1).

Usuwanie zadań

Pozostała do zaimplementowania już tylko jedna możliwość, czyli usuwanie zadań. Jest ona całkiem ważna, gdyż może się zdarzyć, że użytkownik przez pomyłkę doda zadanie, które nie powinno się znaleźć na liście. Poza tym, jeśli lista wykonanych zadań stanie się zbyt długa, użytkownik może zdecydować się na usunięcie kilku z nich.

1. **W pliku *todo.js*, za kodem funkcji `sortable()`, lecz wciąż wewnątrz funkcji `$(document).ready()` wpisz następujący fragment kodu:**

   ```
   $('.sortlist').on('click', '.delete', function() {
   });
   ```

   Także w tym przypadku zastosowałeś delegowanie zdarzeń. W momencie wczytywania strony listy są puste i nie ma w nich żadnych przycisków, dlatego też do usuwania zadań używasz techniki delegowania. W tym przypadku wywołanie `$('.sortlist')` powoduje pobranie obu list wypunktowanych dostępnych na stronie (gdyż użytkownicy powinni mieć możliwość usuwania zadań z obu list), a wywołanie metody `on()` informuje jQuery, że należy obsługiwać zdarzenia kliknięcia skierowane do elementów klasy `delete`. W odpowiedzi na kliknięcie takiego elementu ma zostać wywołana przekazana funkcja.

   W kolejnym punkcie zajmiesz się zaimplementowaniem kodu tej funkcji.

2. **Znajdź kod między znacznikami `<script>` w sekcji nagłówkowej strony i usuń kod wyróżniony pogrubieniem:**

   ```
   $('.sortlist').on('click', '.delete', function() {
     $(this).parent('li').effect('puff', function() {
       $(this).remove();
     });
   });
   ```

   W tym kodzie dzieje się całkiem sporo, lecz powinieneś już być przyzwyczajony do takich rozwiązań. Poniżej zostały opisane wszystkie wykonywane operacje.

   - Wyrażenie `$(this).parent('li')` pobiera element kliknięty przez użytkownika (reprezentowany przez `$this`), a następnie wśród jego przodków odnajduje znacznik `<li>`. Innymi słowy, wyrażenie to pobiera element zadania, które należy usunąć.
   - Metoda `effect()` biblioteki jQuery UI odtwarza w elemencie określony efekt wizualny. W tym przypadku odtwarzamy efekt o nazwie `puff`, który powoduje że element się powiększa, stopniowo wygasa i w końcu znika.
   - W wywołaniu metody `effect()` została umieszczona funkcja zwrotna, która będzie wywołana po zakończeniu efektu. W naszym przypadku funkcja pobiera element listy, na którym operowała metoda `effect()` — czyli element listy pobrany przy użyciu wyrażenia `$(this)` — a następnie usuwa go całkowicie ze strony, wywołując metodę `remove()` jQuery (patrz strona 160). Metoda `remove()`, w odróżnieniu od `detach()` (opisanej na stronie 533), całkowicie usuwa wskazany element strony.

3. **Zapisz plik i odśwież w przeglądarce stronę *index.html*.**

   Teraz powinieneś już bez problemu dodawać, przenosić i usuwać zadania, a cała strona powinna wyglądać tak, jak na rysunku 14.1. Gdybyś miał jakieś problemy z wykonaniem tego przykładu, poniżej znajdziesz cały kod pliku *todo.js*.

```
$(document).ready(function(e) {
  $("#add-todo").button({
    icons: {
      primary: "ui-icon-circle-plus"
    }
  }).click(function() {
    $("#new-todo").dialog('open');
  });

  $("#new-todo").dialog({
    modal: true,
    autoOpen: false,
    buttons : {
      "Dodaj zadanie" : function() {
        var taskName = $('#task').val();
        if (taskName === '') {
          return false;
        }
        var taskHTML = '<li><span class="done">%</span>';
        taskHTML += '<span class="delete">x</span>';
        taskHTML += '<span class="task"></span></li>';
        var $newTask = $(taskHTML);
        $newTask.find('.task').text(taskName);
        $newTask.hide();
        $('#todo-list').prepend($newTask);
        $newTask.show('clip',250).effect('highlight',1000);
        $(this).dialog('close');
      },
      "Anuluj" : function() {
        $(this).dialog('close');
      }
    },
    close: function() {
      $('#new-todo input').val('');
    }
  });

  $("#todo-list").on('click','.done', function() {
    var $taskItem = $(this).parent('li');
    $taskItem.slideUp(250, function() {
      var $this = $(this);
      $this.detach();
      $('#completed-list').prepend($this);
      $this.slideDown();
    });
  });

  $('.sortlist').sortable({
    connectWith : '.sortlist',
    cursor : 'pointer',
    placeholder : 'ui-state-highlight',
    cancel : '.delete,.done'
  });

  $('.sortlist').on('click','.delete',function() {
    $(this).parent('li').effect('puff', function() {
      $(this).remove();
    });
  });
}); // Koniec funkcji ready.
```

Uwaga: Pełną, działającą kopię aplikacji napisanej w tym rozdziale znajdziesz w plikach *complete-index.*
↪html oraz *complete-todo.js*, umieszczonych w katalogu *R14*.

Dalsze kroki

Gratuluję — właśnie napisałeś swoją pierwszą aplikację internetową! Jednak tę aplikację można poprawić na kilka różnych sposobów. Być może już nawet masz przygotowaną listę ewentualnych usprawnień. W tym podrozdziale zamieszczona została lista potencjalnych poprawek wraz z odnośnikami do źródeł informacji, które mogą Ci się przydać podczas ich implementowania.

Edycja zadań

Aktualnie aplikacja nie zapewnia użytkownikowi możliwości poprawy ewentualnych błędów typograficznych w nazwach zadań. Jeśli ktoś wpisze „Uwiec ciasteczka", będzie musiał usunąć zadanie i utworzyć je od nowa. Problem edycji zadań można rozwiązać na dwa sposoby.

Pierwszym z nich jest dodanie do każdego z zadań przycisku *Edytuj*. Jego kliknięcie będzie powodować wyświetlenie okna dialogowego z polem tekstowym zawierającym nazwę zadania — czyli tekst umieszczony wewnątrz znacznika `<span>` klasy `task`.

Można także dodać do strony kolejne okno dialogowe, takie jak przedstawione na stronie 522. Kliknięcie przycisku *Edytuj* powodowałoby wyświetlenie tego okna dialogowego i umieszczenie nazwy zadania w polu tekstowym, a zamknięcie okna dialogowego — aktualizację nazwy zadania.

Innym rozwiązaniem może być skorzystanie z właściwości HTML o nazwie `contentEditable`. Jej użycie zapewnia możliwość edycji zawartości dowolnego elementu HTML. Przykładowo poniżej został przedstawiony kod HTML zapewniający możliwość edycji nazw zdań w naszej przykładowej aplikacji:

```
<span class="taks" contentEditable>
```

Właściwość tę można nawet zastosować dynamicznie, używając metody `prop()` jQuery:

```
$('.task').prop('contentEditable', true);
```

Jednak z zastosowaniem właściwości `contentEditable` wiąże się jeden problem. Widżet Sortable jQuery UI nie pozwala na zaznaczanie tekstu w sortowalnych elementach, a zatem nawet po zastosowaniu właściwości `contentEditable` użytkownik nie będzie mógł zaznaczyć tekstu do edycji. Problem ten można ominąć, nakazując jQuery UI ignorowanie elementów klasy `task` poprzez przekazanie odpowiedniej opcji w wywołaniu funkcji `sortable()`. Dokładnie w taki sam sposób postąpiłeś z polem wyboru do oznaczania zadania jako wykonanego oraz przyciskiem do usuwania zadania w kroku 10. na stronie 535; a tu wystarczy dodać selektor `.task` tak, jak pokazano na poniższym przykładzie:

```
cancel: '.delete,.done,.task'
```

Jeśli jednak zastosujesz to rozwiązanie, w elementach zadań nie pozostanie zbyt wiele miejsc, za które użytkownik będzie mógł je przeciągać. W takim przypadku powinieneś dodać do elementu `<li>` zadania jakiś wyraźnie widoczny element, którego użytkownik mógłby używać do przeciągania. Aby skorzystać z tego rozwiązania, wystarczy użyć opcji `handle` widżetu Sortable (patrz strona 453).

Potwierdzanie usunięcia

Obecnie kiedy użytkownik kliknie przycisk usuwający zadanie, zostaje ono usunięte raz na zawsze. Mógłbyś jednak dodać modalne okno dialogowe, zawierające prośbę o potwierdzenie usunięcia. Jeśli użytkownik kliknie przycisk *Tak*, zadanie zostanie usunięte, jeśli przycisk *Nie*, zadanie pozostanie na liście.

Zapisywanie listy

Jednak największym problemem naszej aplikacji jest to, że nie zapamiętuje ona zadań w momencie zamykania okna przeglądarki. Innymi słowy, lista ma charakter całkowicie tymczasowy i nie zostanie odtworzona po zamknięciu i ponownym uruchomieniu przeglądarki bądź też po wyświetleniu strony na innym komputerze. Istnieje kilka sposobów pozwalających na zapamiętanie stanu listy zadań.

Magazyn lokalny

Wszystkie nowoczesne przeglądarki udostępniają mechanizm określany jako *magazyn lokalny* (ang. *local storage*). Pozwala on na zapisywanie danych na komputerze użytkownika i odczytanie ich po ponownym wyświetleniu strony. Mógłbyś skorzystać z magazynu lokalnego, by zapisywać aktualny stan listy zadań po każdej zmianie ich stanu. W takim przypadku, kiedy użytkownik wróci na stronę, powinieneś sprawdzić, czy są zapisane jakieś dane w magazynie lokalnym, a jeśli są, musisz odpowiednio zaktualizować stronę.

Więcej informacji na temat magazynu lokalnego można znaleźć na witrynie Mozilla Developer Network: *https://developer.mozilla.org/en-US/docs/Web/Guide/API/DOM/Storage*. Dostępna jest nawet wtyczka jQuery ułatwiająca korzystanie z tego mechanizmu: *https://github.com/julien-maurel/jQuery-Storage-API*.

Zapis na serwerze

Kolejną możliwością jest zapisanie listy zadań na serwerze. To rozwiązanie ma tę zaletę, że takiej listy zadań można używać na dowolnym komputerze. Z drugiej strony jego wadą jest konieczność utworzenia jakiegoś systemu kontroli dostępu do listy, w przeciwnym razie każdy będzie mógł wejść na stronę i wyświetlić Twoje zadania (a nawet je usunąć).

Książka ta nie jest poświęcona zagadnieniom programowania aplikacji działających po stronie serwera. Jednak do przekazania zadań na serwer będziesz musiał użyć kodu JavaScript. Zapewne będzie to kod korzystający z technologii AJAX, opisanej w poprzednim rozdziale.

Najprostszym rozwiązaniem byłoby pobranie wszystkich elementów list i zastosowanie metody `.each()` do pobrania nazwy każdego z zadań. Następnie mógłbyś utworzyć obiekt jQuery zawierający listę wszystkich zadań, zarówno tych do wykonania, jak i już wykonanych. W końcu, abyś mógł je przesłać na serwer, musiałbyś *serializować* ten obiekt, czyli zapisać go w postaci łańcucha znaków. Poniżej został zamieszczony kod funkcji, która realizuje wszystkie te operacje:

```
function getData() {
  var todoData = {
    toDo : [],
    completed : []
  };
  $('#todo-list').each( function() {
    var task = $(this).find('.task').text();
    todoData.toDo.push(task);
  });
  $('#completed-list').each( function() {
    var task = $(this).find('.task').text();
    todoData.completed.push(task);
  });
  return $.param(todoData);
}
```

Tę funkcję mógłbyś wywoływać za każdym razem, kiedy będziesz chciał pobrać wszystkie zadania w postaci nadającej się do przesłania na serwer. Odczytanie tych zadań i zrobienie z nimi czegoś użytecznego należałoby już do programu wykonywanego na serwerze.

Inne pomysły

Jeśli masz więcej pomysłów na poprawienie zamieszczonej tu przykładowej listy zadań, warto, byś je dokładniej wypróbował. Ten projekt doskonale nadaje się do sprawdzenia nowych umiejętności stosowania języka JavaScript oraz bibliotek jQuery i jQuery UI. Bezustannie rozwijaną wersję tego projektu można znaleźć na serwisie GitHub, na stronie *https://github.com/sawmac/jquery-todo*.

CZĘŚĆ

V

Wskazówki, sztuczki i rozwiązywanie problemów

15
ROZDZIAŁ

Wykorzystywanie wszystkich możliwości jQuery

Biblioteka jQuery w ogromnym stopniu ułatwia pisanie programów w języku JavaScript oraz pozwala szybko i łatwo wzbogacać tworzone strony w wyszukane możliwości interakcji. W tej książce przedstawionych zostało kilka przykładów jej wykorzystania, takich jak walidacja formularzy lub dynamiczne podmienianie prezentowanych obrazków. Kiedy jednak sami zaczniemy jej używać, okaże się, że stosowanie nie zawsze jest proste, a pełne wykorzystanie wszystkich możliwości, jakie daje, wymaga sporej wiedzy. W tym rozdziale poznasz bardziej zaawansowane sposoby korzystania z jQuery — dowiesz się, jak używać dokumentacji, korzystać z gotowych możliwości interakcji poprzez stosowanie wtyczek i w końcu poznasz kilka przydatnych sztuczek.

Przydatne informacje i sztuczki związane z jQuery

Biblioteka jQuery ułatwia programowanie. Istnieją także pewne rozwiązania i metody, które dodatkowo ułatwiają pisanie programów wykorzystujących tę bibliotekę. W tym podrozdziale zamieszczono kilka bardziej zaawansowanych informacji dotyczących samej biblioteki, dzięki którym będziesz mógł w większym stopniu wykorzystywać jej możliwości.

$() to to samo, co jQuery()

W wielu artykułach i wpisach na blogach poświęconych jQuery można znaleźć fragmenty kodu, takie jak przedstawiony niżej:

```
jQuery('p').css('color','#FO3');
```

Choć doskonale znasz wyrażenie w postaci `$('p')`, które pobiera wszystkie znaczniki `<p>` dostępne na danej stronie, to jednak możesz się zastanawiać, czym jest funkcja `jQuery()`. Okazuje się, że jest ona tożsama z funkcją `$()`. Powyższy fragment kodu można by równie dobrze zapisać tak:

```
$('p').css('color','#F03');
```

W rzeczywistości `$()` jest jedynie nazwą zastępczą funkcji `jQuery()` i obie można stosować zamiennie. John Resig, twórca jQuery, uświadomił sobie, że programiści będą używać funkcji `jQuery()` bardzo często, zatem, zamiast ich zmuszać do ciągłego wpisywania długiego wywołania `jQuery()`, uznał, że znacznie wygodniejsza będzie nazwa `$()`.

W praktyce możemy używać obu tych wywołań — zarówno `jQuery()`, jak i `$()` — decyzja należy do nas. Ponieważ jednak wpisanie wywołania `$()` jest znacząco szybsze, zatem w większości przypadków zapewne będziesz korzystał właśnie z niego (jak czyni większość programistów).

Uwaga: W innej popularnej bibliotece JavaScript — Prototype (*http://prototypejs.org*) — także użyto funkcji `$()`. Jeśli zatem zdarzy się, że na swojej witrynie będziesz korzystał z obu tych bibliotek, lepszym rozwiązaniem będzie stosowanie funkcji `jQuery()`. Co więcej, jQuery udostępnia specjalną funkcję opracowaną z myślą o właśnie takich sytuacjach; nosi ona nazwę `.noConflict()`. Więcej informacji na jej temat można znaleźć na stronie *http://api.jquery.com/jQuery.noConflict/*.

Zapisywanie pobranych elementów w zmiennych

Za każdym razem, gdy przy użyciu funkcji `$()` pobierasz jakieś elementy strony — korzystając z takiego wywołania jak `$('#tooltip')` — wywoływana jest funkcja jQuery. Każde takie wywołanie zmusza przeglądarkę użytkownika do wykonania sporego fragmentu kodu, a to niejednokrotnie może doprowadzić do niepotrzebnego spowolnienia działania programu. Załóżmy na przykład, że chcielibyśmy pobrać jakiś element strony i zmodyfikować go, wywołując kilka funkcji jQuery. Możemy to zrobić w następujący sposób:

```
$('#tooltip').html('<p>Mrówkojad</p>');
$('#tooltip').fadeIn(250);
$('#tooltip').animate({left : "100px"},250);
```

Ten fragment kodu pobiera element o identyfikatorze `tooltip` i umieszcza w nim znacznik `<p>`. Następnie ponownie pobiera ten sam element i stopniowo wyświetla go na stronie. Ostatnia instrukcja powyższego fragmentu po raz trzeci pobiera ten sam element i wywołuje funkcję `animate()`, która przesuwa go do miejsca o współrzędnej poziomej o wartości `100`. Każde z tych wywołań — `$('#tooltip')` — powoduje wykonanie funkcji jQuery. A ponieważ każda z instrukcji operuje na tym samym elemencie strony, zatem wystarczyłoby go pobrać tylko raz.

Jednym z rozwiązań tego problemu (opisanym na stronie 156 książki) jest technika łączenia wywołań w sekwencję. Najpierw pobieramy potrzebny element, a następnie dodajemy do niego wywołania kolejnych funkcji:

```
$('#tooltip').html('<p>Mrówkojad</p>').fadeIn(250).animate({left :
"100px"},250);
```

Jednak czasami utworzenie takiej sekwencji wywołań prowadzi do powstania nieczytelnego kodu. Jednym ze sposobów rozwiązania tego problemu jest dodanie nowego wiersza za wywołaniem każdej z metod składających się na sekwencję. Ponieważ język JavaScript nie zwraca wielkiej uwagi na znaki odstępu i nowego wiersza, można to zrobić na przykład w następujący sposób:

```
$('#tooltip').html('<p>Mrówkojad</p>')
             .fadeIn(250)
             .animate({left:"100px"},250);
```

Choć wywołania funkcji `.html`, `.fadeIn` oraz `.animate` znajdują się w odrębnych wierszach, w rzeczywistości tworzą sekwencję będącą jedną instrukcją.

Innym rozwiązaniem jest zatem jednokrotne wywołanie funkcji jQuery i zapisanie uzyskanych wyników w zmiennej, co umożliwi ich wielokrotne użycie. A tak można zmodyfikować przedstawiony powyżej fragment kodu:

```
1  var tooltip = $('#tooltip')
2  tooltip.html('<p>Mrówkojad</p>');
3  tooltip.fadeIn(250);
4  tooltip.animate({left : "100px"},250);
```

W wierszu 1. wywołujemy funkcję jQuery, która pobiera element o identyfikatorze `tooltip`, a następnie zapisujemy zwrócony wynik w zmiennej `tooltip`. Po wywołaniu funkcji i zapisaniu pobranego elementu w zmiennej ponowne wywoływanie funkcji jQuery nie będzie już potrzebne. Wystarczy, że skorzystamy ze zmiennej (która zawiera obiekt jQuery) i użyjemy jej do wywoływania kolejnych funkcji.

Przy zastosowaniu takiego rozwiązania wielu programistów decyduje się na dodawanie na początku nazwy zmiennej znaku `$`, co przypomina o tym, że zawiera ona obiekt jQuery, a nie daną jakiegoś innego typu, takiego jak łańcuch znaków, liczba, tablica czy też literał obiektowy. Oto przykład:

```
var $tooltip = $('#tooltip');
```

Technika zapisywania elementów pobieranych przy użyciu jQuery w zmiennych jest bardzo często stosowana podczas obsługi zdarzeń. Jak sobie zapewne przypominasz (była o tym mowa na stronie 169), wewnątrz funkcji obsługującej zdarzenia używane jest wyrażenie `$(this)`; ono pozwala odwołać się do znacznika, do którego zdarzenie zostało skierowane. Jednak każde użycie tego wyrażenia powoduje wywołanie funkcji jQuery, a zatem jego wielokrotne stosowanie wewnątrz tej samej funkcji jest niepotrzebnym marnowaniem mocy komputera. Zamiast tego można zapisać wartość tego wyrażenia w zmiennej, a następnie używać jej, kiedy trzeba, w dalszej części kodu funkcji:

```
$('a').click(function() {
  var $this = $(this); // zapisanie odwołania do znacznika <a>
  $this.css('outline','2px solid red');
  var href = $this.attr('href');
  window.open(href);
  return false;
}); // koniec funkcji click
```

Jak najrzadsze dodawanie treści

W rozdziale 4. poznałeś funkcje jQuery umożliwiające dodawanie nowych treści do elementów stron WWW. I tak funkcja `.text()` (patrz strona 158) pozwala na zastąpienie tekstu umieszczonego wewnątrz elementu, a funkcja `.html()` (patrz strona 157) na zmianę umieszczonego wewnątrz elementu kodu HTML. Gdybyśmy na przykład chcieli umieścić komunikat o błędzie w znaczniku `<span>` o identyfikatorze `passwordError`, moglibyśmy to zrobić przy użyciu następującego wywołania:

```
$('#passwordError').text('Hasło musi mieć co najmniej 7 znaków długości.');
```

Jeszcze inne funkcje pozwalają dodawać nowe treści za elementem (funkcja `append()`, opisana na stronie 158) lub przed nim (`prepend()`, patrz strona 159).

Dodawanie i modyfikacja treści umożliwiają wyświetlanie nowych komunikatów o błędach, prezentowanie etykiet ekranowych (patrz strona 345), tworzenie wyróżnionych cytatów (patrz strona 171), jednak wszystkie te operacje stanowią duże wyzwanie dla przeglądarki. Za każdym razem gdy dodajemy do strony nowe treści, przeglądarka musi wykonać bardzo wiele pracy — w końcu każda taka operacja wymaga utworzenia nowego DOM-u (patrz strona 145), a to z kolei wiąże się z koniecznością wykonania wielu, niezauważalnych czynności. Dlatego też częste modyfikowanie zawartości strony może doprowadzić do spadku wydajności jej działania.

W tym przypadku nie liczy się wielkość dodawanych treści — znacznie większy wpływ na wydajność strony ma sama liczba wprowadzanych modyfikacji. Załóżmy na przykład, że chcemy wykorzystać na stronie etykiety: kiedy użytkownik umieści wskaźnik myszy w określonym obszarze strony, ma się na niej pojawić dodatkowy element (taki jak znacznik `<div>`) z dodatkową treścią. W tym celu musimy dodać do strony znacznik etykiety oraz jego zawartość. A tak można by to zrobić:

```
1 // dodajemy element div na końcu elementu
2 $('#elemForTooltip').append('<div id="tooltip"></div');
3 // dodajemy do niego nagłówek
4 $('#tooltip').append('<h2>Tytuł etykiety</h2>');
5 // oraz treść etykiety
6 $('#tooltip').append('<p>Treść etykiety.</p>');
```

Powyższy kod będzie działał zgodnie z oczekiwaniami. W wierszu 2. dodajemy do wskazanego elementu nowy znacznik `<div>`; w wierszu 4. do znacznika `<div>` — nagłówek, a w wierszu 6. — akapit z treścią etykiety. Jednak w trakcie tego procesu DOM strony został zmodyfikowany aż trzy razy, gdyż trzykrotnie została wywołana funkcja `append()`. Wszystkie te czynności w dużym stopniu obciążają przeglądarkę, dlatego też ograniczenie operacji modyfikujących DOM strony może w znaczącym stopniu poprawić wydajność jej działania.

W tym przykładzie możemy zmniejszyć liczbę wykonywanych operacji dodawania do strony nowej zawartości do jednej — wystarczy w zmiennej zapisać cały kod HTML etykiety, a dopiero potem dodać go do strony. Poniżej pokazano, jak to zrobić:

```
1 var tooltip = '<div id="tooltip"><h2>Tytuł etykiety</h2> ↪
  <p>Treść etykiety.</p></div';
2 $('#elemForTooltip').append(tooltip);
```

Uwaga: Symbol ➥ umieszczony na końcu pierwszego wiersza kodu oznacza, że dalszą część kodu należy wpisywać w tym samym wierszu. To *naprawdę* długi wiersz kodu, którego nie można w całości wydrukować w książce, dlatego też został tutaj zapisany w dwóch wierszach. Trzeba także pamiętać o tym, że łańcuchy znaków są jedynym przypadkiem, w którym JavaScript zwraca uwagę na odstępy i znaki nowego wiersza. Jednego łańcucha znaków nie można zapisać w dwóch kolejnych wierszach kodu:

```
var longString = "Teraz nadszedł czas, by wszyscy ludzie dobrej
woli pomogli swojemu krajowi";
```

W takim przypadku wystąpi błąd.

W tym przykładzie w wierszu 1. tworzymy zmienną zawierającą cały kod HTML, a następnie w wierszu 2. dodajemy ją do strony. W ten sposób wykonywana jest tylko jedna operacja dodawania i w zależności od używanej przez użytkownika przeglądarki ta wersja kodu może działać nawet do 20 razy szybciej niż poprzednia, wykorzystująca trzy wywołania funkcji `append()`.

Oznacza to, że jeśli chcemy wstawić nowy fragment kodu HTML w jakimś miejscu strony, należy to zrobić w ramach jednej, a nie kilku operacji (a przynajmniej ich liczbę należy jak najbardziej ograniczyć).

Optymalizacja selektorów

Elastyczność jQuery oznacza, że ten sam cel można uzyskać na kilka różnych sposobów. Przykładowo jeden element strony można pobrać na kilka sposobów, choćby przy użyciu selektora CSS lub metod służących do poruszania się po drzewie DOM (opisanych na stronie 554). Techniki opisane w tym punkcie rozdziału pozwolą przyspieszyć wykonywane operacje selekcji elementów strony i poprawić wydajność naszych programów pisanych w JavaScripcie.

- **Zawsze, gdy to tylko możliwe, warto używać selektorów z identyfikatorami elementów.** Najszybszym sposobem pobrania elementu strony jest użycie selektora identyfikatora. Już od samego powstania języka JavaScript przeglądarki udostępniały metodę pozwalającą na pobieranie elementu na podstawie jego identyfikatora i nawet dziś jest to najszybszy sposób odwołania się do elementu strony. Jeśli zatem zwracamy uwagę na wydajność działania naszej strony, możemy zrezygnować ze stosowania selektorów elementów potomnych, a zamiast tego określać identyfikatory we wszystkich elementach, do których będziemy się odwoływali w skrypcie.
- **Należy stosować selektor identyfikatora na początku selektora elementów potomnych.** Stosowanie selektorów identyfikatorów wiąże się z jednym problemem. Otóż w ten sposób można pobrać tylko jeden element. A co zrobić, gdy chcemy pobrać wszystkie znaczniki `<a>` umieszczone wewnątrz jakiegoś znacznika `<div>` lub akapitu tekstu? Jeśli strona została skonstruowana w taki sposób, że wszystkie pobierane elementy znajdują się wewnątrz elementu o pewnym identyfikatorze, warto go podać na samym początku selektora elementów potomnych. Załóżmy na przykład, że chcemy pobrać wszystkie znaczniki `<a>` umieszczone na stronie. Co więcej, tak się składa, że wszystkie one są umieszczone wewnątrz znacznika `<div>` o identyfikatorze `main`. W takim przypadku zastosowanie selektora

```
$('#main a')
```

zapewni lepszą wydajność niż zastosowanie selektora:

```
$('a')
```

- **Warto korzystać z funkcji `.find()`.** Biblioteka jQuery udostępnia funkcję pozwalającą na wyszukiwanie elementów wewnątrz zwróconych wyników selekcji. Rozwiązanie to działa w sposób przypominający selektor elementów potomnych, który odnajduje znaczniki umieszczone wewnątrz innych znaczników. Więcej informacji na temat tej funkcji można znaleźć na stronie 555, jednak najprościej rzecz ujmując, jej stosowanie polega na pobraniu konkretnych elementów w standardowy sposób i dodaniu wywołania funkcji `.find()`, wewnątrz którego przekazywany jest dodatkowy selektor. Innymi słowy, wywołanie funkcji jQuery w postaci `$('#main a')` można by zapisać tak:

  ```
  $('#main').find('a');
  ```

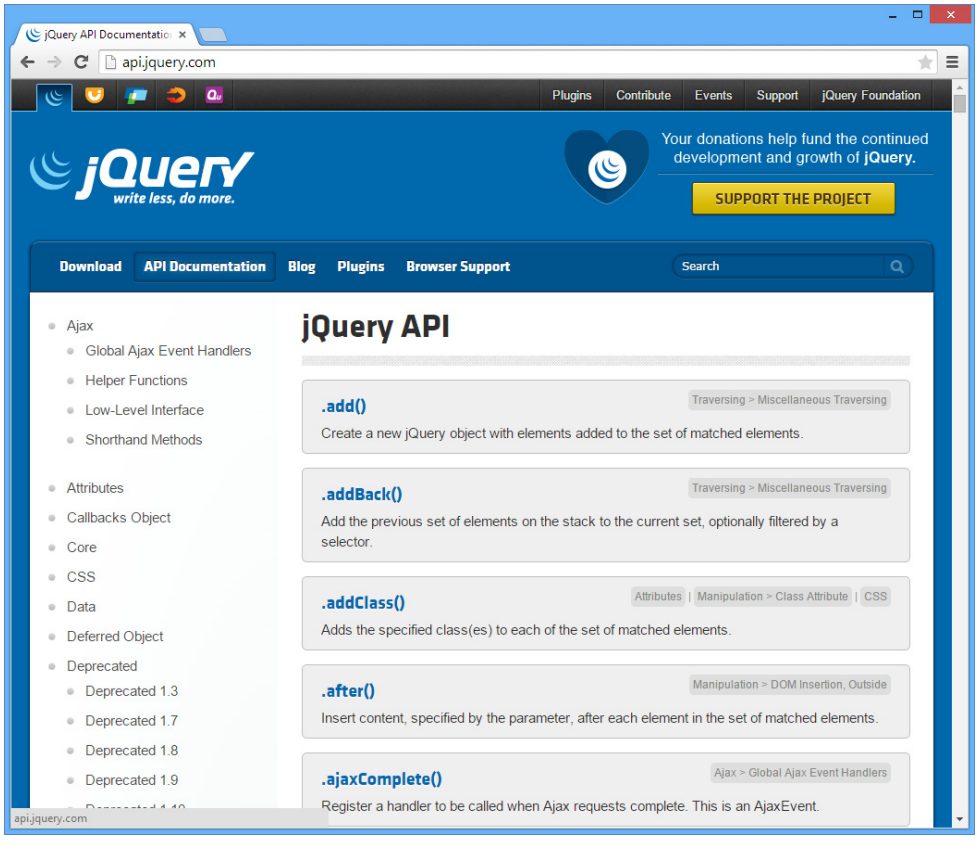
  W rzeczywistości okazuje się, że w niektórych przypadkach zastosowanie funkcji `.find()` jest dwukrotnie szybsze od wykorzystania selektora elementu podrzędnego!

Uwaga: Na stronie *http://jsperf.com/sawmac-selector-test* można znaleźć test pozwalający na sprawdzenie wydajności działania funkcji `.find()`.

- **Należy unikać zbytniej szczegółowości.** Być może jesteś przyzwyczajony do stosowania reguł szczegółowości wykorzystywanych w CSS w celu tworzenia reguł stylów ustalających postać precyzyjnie określanych elementów strony. Przykładowo reguła z selektorem `#main .sidebar .note ul.nav li a` jest bardzo szczegółowa; gdybyśmy użyli jej w wywołaniu jQuery, okazałoby się, że wykonanie zajmuje dużo czasu. O ile to tylko możliwe, należy stosować krótsze i bardziej precyzyjne selektory elementów podrzędnych, takie jak `$('.sidebar .nav a')`, bądź też skorzystać z funkcji `.find()` (opisanej w poprzednim punkcie): `$('#main').find('.sidebar').find('.nav a')`.

Korzystanie z dokumentacji jQuery

Na witrynie jQuery umieszczono bardzo szczegółową dokumentację wszystkich funkcji biblioteki, jest ona dostępna na stronie *http://api.jquery.com/* (patrz rysunek 15.1). Można tam także znaleźć przydatne odnośniki do poradników, jak zacząć korzystanie z jQuery, gdzie szukać dodatkowej pomocy i tak dalej; jednak przede wszystkim są tam dostępne szczegółowe opisy API biblioteki. API to skrót od angielskich słów *Application Programming Interface* — interfejs programowania aplikacji — co po prostu oznacza zbiór funkcji udostępnianych przez bibliotekę, takich jak funkcje do obsługi zdarzeń przedstawione w rozdziale 5. (`.click()`, `.hover()` i tak dalej), funkcje CSS opisane w rozdziale 4. (`.css()`, `.addClass()` i `.removeClass()`) oraz przede wszystkim główna funkcja jQuery — `$()` — która umożliwia pobieranie potrzebnych elementów strony.



Rysunek 15.1. *Główna strona dokumentacji jQuery zawiera odnośniki do wielu przydatnych informacji. Z lewej strony okna widoczna jest kolumna z listą wszystkich funkcji jQuery. W niektórych przypadkach są one pogrupowane według zagadnień, na przykład funkcje związane z technologią AJAX, funkcje do obsługi zdarzeń (Events) i tak dalej*

Dział dokumentacji witryny jQuery zawiera informacje o wszystkich funkcji biblioteki. Na stronie głównej wyświetlona jest ich lista. Jest to bardzo przydatne, gdy znamy nazwę funkcji, gdyż lista jest uporządkowana alfabetycznie; jednak może onieśmielać osoby, które nie znają tak dobrze jQuery. Dlatego zapewne większość będzie wolała wybierać jedną z kategorii wyświetlonych na pasku nawigacyjnym umieszczonym z lewej strony. Poniżej przedstawiono kilka najbardziej przydatnych kategorii.

- Kategoria *Selectors* (selektory; *http://api.jquery.com/category/selectors/*). Ta kategoria zawiera informacje o najbardziej użytecznych funkcjach jQuery. Warto tu zaglądać często, gdyż zawiera informacje o wielu różnych sposobach pobierania elementów. Wiele z nich poznałeś już w rozdziale 4., jeśli jednak zajrzysz do tej sekcji dokumentacji, możesz ich poznać jeszcze więcej.

  Ponieważ funkcji jQuery związanych z selektorami jest bardzo dużo, zatem zobaczysz, że w pasku bocznym zostały one podzielone na dodatkowe kategorie, takie jak *Basic* (proste selektory), *Forms* (selektory pól formularzy), *Attribute* (selektory atrybutów) i tak dalej.

- Kategoria *Attributes* (atrybuty; *http://api.jquery.com/category/attributes*). Zajrzyj na tę stronę, by znaleźć informacje o funkcjach jQuery służących do odczytywania i ustawiania wartości atrybutów elementów HTML: na przykład dodawania klasy, do jakiej znacznik ma należeć, odczytu wartości konkretnego atrybutu (przykładowo atrybutu `href` znacznika `<a>`) czy też pobierania wartości zapisanej w elemencie formularza.

Uwaga: Czasami tę samą funkcję można znaleźć w kilku różnych sekcjach dokumentacji. Przykładem może być funkcja `.val()`, używana do odczytu lub ustawienia wartości pola formularza, która została wymieniona zarówno w kategorii *Attributes* (atrybuty), jak i *Forms* (formularze).

- Kategoria *Traversing* (poruszanie się po zawartości strony; *http://api.jquery.com/category/traversing/*). Zawiera funkcje służące do manipulowania zbiorami elementów strony. Przykładowo funkcja `.find()` pozwala na poszukiwanie elementów w wynikach zwróconych przez funkcję jQuery. Możliwość ta jest bardzo przydatna, gdy na przykład chcemy pobrać element strony (taki jak `<ul>`), wykonać na nim jakieś operacje (dodać klasę, stopniowo wyświetlić i tym podobne), a następnie odszukać wewnątrz niego jakiś inny element (przykładowo znacznik `<li>` umieszczony we wcześniej pobranym znaczniku `<ul>`) i na nim także wykonać jakieś operacje. Biblioteka jQuery udostępnia wiele funkcji służących do poruszania się po elementach HTML tworzących stronę, a niektóre spośród nich zostaną przedstawione w dalszej części tego rozdziału.
- Kategoria *Manipulation* (manipulacja; *http://api.jquery.com/category/manipul-ation/*). Za każdym razem, gdy zechcemy dodać coś do strony bądź z niej coś usunąć, konieczne jest przeprowadzenie pewnych zmian w drzewie DOM. Na tej stronie znajdziesz wszystkie funkcje umożliwiające wprowadzanie zmian w zawartości strony, w tym także te, o których dowiedziałeś się w rozdziale 4. (patrz strona 157), czyli `.html()` pozwalająca dodawać do strony kod HTML, `.text()` pozwalająca dodawać do strony zwyczajny tekst i tak dalej. Jest to bardzo ważna kategoria funkcji, gdyż działanie większości programów pisanych w języku JavaScript w znaczniej mierze bazuje właśnie na dynamicznym modyfikowaniu zawartości strony.
- Kategoria *CSS* (*http://api.jquery.com/category/css/*). W tej kategorii znajdują się funkcje służące do odczytywania i ustawiania właściwości związanych z CSS, na przykład dodawanie i usuwanie klas ze znaczników, bezpośrednie podawanie wartości właściwości CSS, odczytywanie i ustawianie wysokości, szerokości oraz położenia elementów. Informacje o niektórych tych funkcjach można znaleźć na stronie 163.
- Kategoria *Events* (zdarzenia, *http://api.jquery.com/category/events/*). W rozdziale 5. dowiedziałeś się, jak można używać jQuery, by odpowiadać na czynności wykonywane przez użytkownika na stronie, takie jak przesuwanie wskaźnika myszy w obszarze jakiegoś elementu bądź kliknięcie przycisku. Na tej stronie znajdziesz informacje o wielu funkcjach jQuery powiązanych z obsługą zdarzeń. Informacje o kilku zaawansowanych funkcjach należących do tej kategorii możesz znaleźć na stronie 190.

- Kategoria *Effects* (efekty, *http://api.jquery.com/category/effects/*). W tej kategorii znajdziesz informacje o funkcjach jQuery związanych z efektami wizualnymi, takimi jak `.slideDown()`, `.fadeIn()` czy też `.animate()`, które poznałeś w rozdziale 6.
- Kategoria *Forms* (formularze, *http://api.jquery.com/category/forms/*). Do tej kategorii należą funkcje związane z… tu prosimy o werble… formularzami! Przede wszystkim znajdziemy tu wszystkie zdarzenia związane z elementami formularzy, a oprócz nich także funkcję `.val()` (pobierającą lub ustawiającą wartość pola formularza) oraz kilka innych funkcji ułatwiających przesyłanie formularzy przy użyciu technologii AJAX (więcej informacji na ten temat można znaleźć w rozdziale 13.).
- Kategoria *Ajax* (*http://api.jquery.com/category/ajax/*). Do tej kategorii należą funkcje związane z dynamicznym aktualizowaniem strony na podstawie informacji przesyłanych na serwer WWW lub pobieranych z tego serwera. Opisane zostały w rozdziale 13.
- Kategoria *Utilities* (funkcje narzędziowe, *http://api.jquery.com/category/utilities/*). Biblioteka jQuery udostępnia także kilka funkcji, które mają ułatwiać często wykonywane zadania programistyczne, takie jak odnajdywanie elementu w tablicy, wykonanie pewnych operacji na każdym elemencie tablicy lub właściwości obiektu (funkcja `$.each()` opisana na stronie 167) oraz kilka innych funkcji dla zaawansowanych. Prawdopodobnie nie będziesz musiał korzystać z żadnej z tych funkcji na tym etapie swojej programistycznej kariery (nie da się ich wykorzystać do żadnych odlotowych efektów ani nie są nam w stanie pomóc w aktualizowaniu treści strony), kiedy jednak zdobędziesz więcej wiedzy i doświadczenia, warto ponownie zajrzeć na tę stronę dokumentacji.

  Na liście w lewej kolumnie strony dostępne są także inne kategorie, zawierające rzadziej stosowane funkcje, o których jednak warto wiedzieć.

- Kategoria *Data* (dane, *http://api.jquery.com/category/data/*). Kategoria ta zawiera funkcje związane z dodawaniem danych do elementów strony. Biblioteka jQuery udostępnia funkcję `.data()` służącą do dodawania danych do elementów — można ją sobie wyobrazić jako narzędzie do dodawania do elementów par nazwa – wartość, zupełnie jakby były one miniaturową bazą danych. Zarówno ta, jak i inne funkcje należące do tej kategorii mogą się przydać podczas tworzenia aplikacji internetowych, w których trzeba przechowywać dane i ich używać. Całkiem przystępne wprowadzenie do zagadnień stosowania tych funkcji można znaleźć na stronie *http://tutorialzine.com/2010/11/jquery-data-method/*.
- Kategoria *Deffered objects* (obiekty odroczone, *http://api.jquery.com/category/deferred-object/*). Nie musisz szukać dalej, wystarczy, że zajrzysz do krótkiego opisu tej kategorii, by przekonać się, że odroczone obiekty jQuery to złożone narzędzie. (A opis ten stwierdza, że jest to obiekt pozwalający na tworzenie sekwencji wywołań, umożliwiający rejestrowanie wielu funkcji zwrotnych w specjalnych kolejkach, wywoływanie tych kolejek oraz modyfikowanie stanu — powodzenia lub niepowodzenia — wszelkich operacji zarówno synchronicznych, jak i asynchronicznych). Najprościej rzecz ujmując, pozwalają one na tworzenie

kolejek funkcji kontrolujących kolejność, w jakiej będą wywoływane. Jeśli chciałbyś się dowiedzieć czegoś więcej na ich temat, zajrzyj na tę stronę dokumentacji jQuery.

- Kategoria *Dimenstions* (wymiary, *http://api.jquery.com/category/dimensions/*). Zgromadzono w niej funkcje służące do określania wymiarów — szerokości i wysokości — elementów stron. Te same funkcje można także znaleźć w wymienionej wcześniej kategorii *CSS*.
- Kategoria *Internals* (mechanizmy wewnętrzne, *http://api.jquery.com/category/internals/*). Należy do niej tylko kilka funkcji o bardzo różnym stopniu przydatności. Przykładowo właściwość `.jquery` zwraca numer wersji biblioteki.

  ```
  // wyświetlamy okienko informacyjne z numerem wersji biblioteki
  alert($().jquery);   // na przykład 1.6.2
  ```

  Można żyć długo i szczęśliwie, i ani razu nie użyć żadnej z tych funkcji.
- Kategoria *Offset* (współrzędne, *http://api.jquery.com/category/offset/*) zawiera funkcje związane z określaniem położenia elementów na stronie, wyliczanego względem całego dokumentu lub elementu nadrzędnego. Funkcje te są używane podczas odczytywania oraz ustawiania położenia elementów.

Czytanie dokumentacji na stronie jQuery

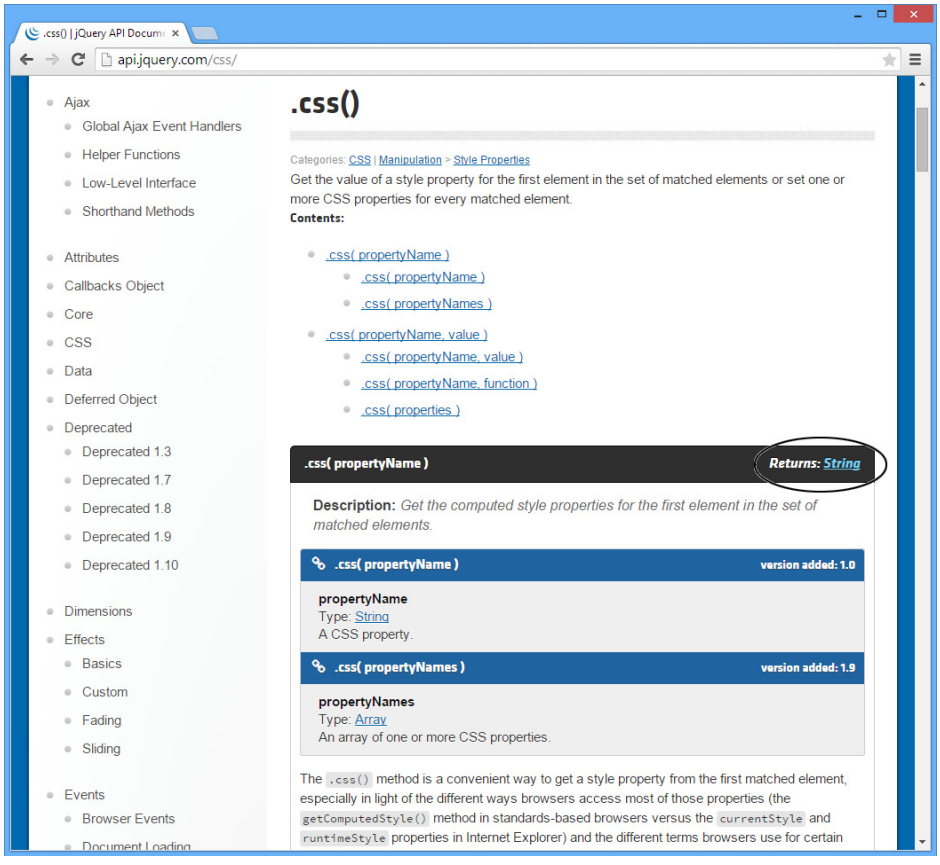
Każda funkcja jQuery została opisana na odrębnej stronie, na której znajdują się informacje o tym, co robi i jak działa. Na rysunku 15.2 przestawiony został fragment strony poświęconej funkcji `css()`. Na stronie widoczna jest nazwa funkcji (w tym przypadku jest nią `css()`) oraz lista kategorii i podkategorii, do których dana funkcja należy. Zarówno nazwę kategorii, jak i podkategorii można kliknąć, by przejść na stronę z listą wszystkich należących do niej funkcji.

W niektórych przypadkach funkcja ma dwa lub nawet trzy zastosowania i działa inaczej, w zależności od typów przekazanych do niej argumentów. W takim przypadku opis każdego z zastosowań zostanie podany osobno, poniżej nagłówka zawierającego nazwę funkcji. Przykładowo na rysunku 15.2 widać, że funkcja `css()` może przyjmować jeden lub dwa argumenty (na rysunku zostały one oznaczone cyframi 1 i 2).

Pierwszy ze sposobów używania funkcji (patrz cyfra 1) został przedstawiony jako `css(propertyName)`. W tym przypadku `propertyName` oznacza, że do funkcji należy przekazać jeden argument, który powinien być nazwą właściwości CSS. W efekcie jQuery zwróci wartość tej właściwości pobraną ze wskazanego elementu (zwróć uwagę na słowa *Returns: String* — zwraca: łańcuch znaków — zakreślone na rysunku 15.2). Informacje zamieszczone na stronie pozwalają się zorientować, że do funkcji należy przekazać jeden argument, a wynikiem wywołania jest łańcuch znaków. Załóżmy na przykład, że chcemy pobrać szerokość znacznika `<div>` o identyfikatorze `tooltip`; możemy w tym celu użyć następującego wywołania:

```
var tipWidth = $('#tooltip').css('width'); // pobieramy wartość właściwości width
```

W tym przypadku do funkcji przekazywana jest nazwa właściwości `'width'`, w efekcie zwraca ona łańcuch znaków. (Choć w tym konkretnym przypadku, ze względu na odczytywanie szerokości, zwrócony łańcuch znaków będzie zawierał liczbę — szerokość elementu w pikselach, na przykład `'300'`).



Rysunek 15.2. *Strona dokumentacji dotycząca konkretnej funkcji jQuery przedstawia listę wszystkich możliwych sposobów jej wywołania. W tym przypadku zostały one podane pod nagłówkiem Contents (zawartość); jak widać, do funkcji css() można przekazać bądź to pojedynczy argument (cyfra 1), bądź dwa argumenty (cyfra 2). W zależności od wybranego sposobu wywołania funkcja działa inaczej. Kliknij jeden z przedstawionych przykładów wywołania, a zostaniesz przeniesiony do miejsca strony, w którym został on dokładniej opisany*

Uwaga: Biblioteka jQuery udostępnia także inną wersję tej funkcji — `css( propertyNames )` — która także zwraca wartości właściwości CSS elementu. Jednak zamiast pobierać nazwę jednej właściwości i zwracać jedną wartość, funkcja ta pobiera tablicę (patrz strona 77) zawierającą nazwy właściwości i zwraca obiekt (patrz strona 165), w którym znajdują się pary nazwa – wartość. Pary te składają się z nazwy właściwości CSS z tablicy przekazanej w wywołaniu oraz wartości odpowiedniego elementu HTML strony.

Drugi sposób użycia funkcji `css()` (oznaczony cyfrą 2) został przedstawiony jako `css(propertyName, value)`. A zatem zakłada on przekazanie w wywołaniu funkcji dwóch argumentów — nazwy właściwości CSS oraz wartości. W przypadku zastosowania tego sposobu wywołania funkcja `css()` ustawia wartość podanej właściwości CSS elementu. Gdybyś chciał na przykład ustawić szerokość znacznika `<div>` o identyfikatorze `tooltip`, pierwszym argumentem wywołania musiałaby być nazwa właściwości `'widht'`, a drugim — liczba określająca zamierzoną szerokość elementu:

```
$('#tooltip').css('width',300);   // ustawiamy szerokość elementu div na 300 pikseli
```

Dokumentacja zawiera także informacje o dwóch innych sposobach stosowania funkcji `css()` w celu ustawiania wartości właściwości CSS. Oto one.

- **`.css( propertyName, function )`** — w tym przypadku istnieje możliwość dynamicznego wyznaczenia wartości, która zostanie przypisana właściwości CSS. Rozwiązanie to jest przydatne, gdy dysponujemy kolekcją elementów strony i chcemy, by w każdym z nich właściwość miała nieco inną wartość. Jako przykład można by podać sekwencję znaczników `<div>`, które chcemy rozmieścić na stronie (określając wartość właściwości `left`) tak, by były widoczne jeden obok drugiego.
- **`.css( properties )`** — ten sposób wywołania został opisany na stronie 163; oznacza on, że w wywołaniu funkcji jest przekazywany literał obiektowy, dzięki czemu, za jednym zamachem, można ustawić wartości wielu właściwości CSS.

Ważne jest, by uzmysłowić sobie, że funkcje jQuery mogą pobierać różne argumenty i — w zależności od nich — działać na różne sposoby; co więcej, jest to rozwiązanie często stosowane. Przykładowo funkcja `css()` może zarówno pobierać, jak i ustawiać wartości właściwości CSS. Zauważysz zapewne, że funkcje jQuery bardzo często działają jako oba akcesory, czyli pozwalają na pobieranie danych (akcesor „get") i ich ustawianie (akcesor „set").

Strona z dokumentacją zawsze zawiera listę wszystkich możliwych zastosowań danej funkcji i zazwyczaj przedstawia działające przykłady jej użycia. Dokumentacja jQuery jest dobrze utrzymana i, jak na dokumentację techniczną, całkiem przystępna, a także zrozumiała. Warto poświęcić trochę czasu na jej przeglądnięcie i przeczytanie informacji przynajmniej o tych funkcjach, których najczęściej używamy.

Poruszanie się po DOM

Już wiesz, w jaki sposób można pobierać elementy stron, korzystając z funkcji jQuery i selektorów CSS; i tak wywołanie `$('p')` pobiera wszystkie akapity na stronie. Po pobraniu elementów można z nimi coś zrobić, na przykład dodać do nich jakąś klasę lub ją usunąć, zmienić właściwość CSS albo ukryć element. Jednak czasami może się zdarzyć, że będziemy chcieli pobrać inne elementy strony powiązane w jakiś sposób z pobranym wcześniej. W terminologii JavaScript takie operacje są określane jako poruszanie się (albo trawersowanie) po DOM (modelu obiektów dokumentu).

Takie operacje poruszania się po DOM są często wykonywane podczas obsługi zdarzeń, gdy procedura obsługi jest skojarzona z jednym elementem, a chcemy coś zrobić z innym. Przykładowo załóżmy, że na naszej stronie znajduje się znacznik `<div>` o identyfikatorze `gallery`, wewnątrz którego umieszczona jest grupa miniaturek zdjęć. Po kliknięciu znacznika `<div>` chcemy wykonać jakieś operacje na miniaturkach: poruszyć, zmniejszyć, zwiększyć lub coś w tym stylu… Procedura obsługi zdarzeń została dołączona do znacznika `<div>` w następujący sposób:

```
$('#gallery').click(function() {

}); // koniec funkcji click
```

Wewnątrz tej procedury musimy dodać kod, który coś zrobi z miniaturkami. A zatem, choć użytkownik klika znacznik `<div>`, jednak my chcemy wykonać jakieś operacje na *miniaturkach*. W powyższym przykładzie odwołaliśmy się do znacznika `<div>`, a wtedy wewnątrz funkcji obsługującej zdarzenia wyrażenie `$(this)` będzie się odwoływać do tego znacznika (jeśli to dla Ciebie coś nowego, możesz zajrzeć na stronę 169, gdzie znajdziesz więcej informacji na temat tego wyrażenia). Wewnątrz funkcji pobranym elementem jest `<div>`, jednak my musimy znaleźć obrazki umieszczone wewnątrz tego znacznika. Na szczęście jQuery udostępnia rozwiązanie tego problemu — jest nim funkcja `.find()`. Służy ona do wygenerowania nowych wyników jQuery poprzez przeszukanie *zawartości* aktualnie pobranego elementu i odnalezienie w niej znaczników pasujących do podanego selektora. A zatem wszystkie obrazki umieszczone wewnątrz znacznika `<div>` możemy odszukać, używając następującego wywołania (wyróżnionego pogrubioną czcionką):

```
$('#gallery').click(function() {
  $(this).find('img');
}); // koniec funkcji click
```

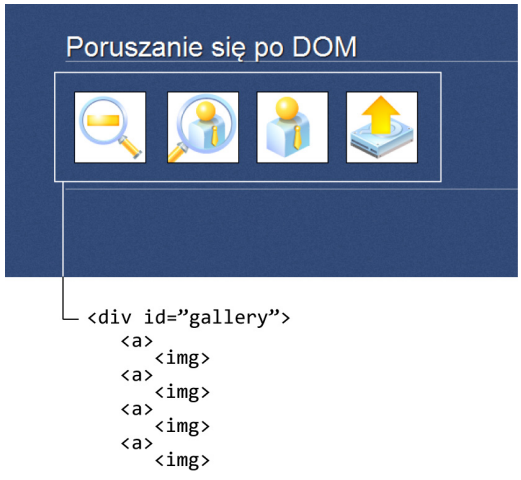
Wywołanie `$(this).find('img')` tworzy nową kolekcję pobranych elementów; wyrażenie `$(this)` odwołuje się najpierw do znacznika `<div>`, a następnie wywołanie `.find('img')` odnajduje wszystkie znaczniki `<img>` umieszczone wewnątrz pobranego wcześniej znacznika `<div>`. Oczywiście taki kod nie wykonuje żadnych operacji na pobranych znacznikach, można jednak uzupełnić go o wywołanie dowolnej z funkcji generujących efekty wizualne, które poznałeś w poprzednim rozdziale. Można użyć następującego kodu:

```
$('#gallery').click(function() {
  $(this).find('img').fadeTo(500,.3).fadeTo(250,1);
}); // koniec funkcji click
```

Zgodnie z informacjami zamieszczonymi na stronie 214, wymagane jest podanie informacji o czasie trwania efektu oraz docelowej wartości nieprzezroczystości. Powyższe wywołanie najpierw wygasza wszystkie obrazki do 30% nieprzezroczystości w czasie 500 milisekund, a następnie, w ciągu 250 milisekund zmienia ich nieprzezroczystość z powrotem do poziomu 100% (aby przekonać się, jak wygląda taki efekt, wystarczy wyświetlić w przeglądarce plik *find.html* dostępny w przykładach dołączonych do książki, w katalogu *R15*).

W rzeczywistości poruszanie się po DOM jest tak często wykonywaną operacją, że jQuery udostępnia wiele funkcji (*http://api.jquery.com/category/manipulation/*) ułatwiających pobieranie elementów, a następnie odnajdywanie innych, które są z nimi w jakiś sposób powiązane. Aby lepiej zrozumieć ich działanie, posłużymy się przykładem prostego fragmentu kodu HTML przedstawionego na rysunku 15.3. Zawiera on znacznik `<div>` o identyfikatorze `gallery`, a wewnątrz niego cztery obrazki umieszczone w odnośnikach.

Zgodnie z informacjami podanymi na stronie 151, do przedstawienia powiązań pomiędzy elementami stron WWW można wykorzystać relacje rodzinne. Przykładowo znacznik `<div>` przedstawiony na rysunku 15.3 jest rodzicem znaczników `<a>`, a z kolei one są rodzicami umieszczonych wewnątrz znaczników `<img>`. Jednocześnie znaczniki `<a>` są dziećmi znacznika `<div>` oraz rodzeństwem w stosunku do pozostałych znaczników `<a>`. I analogicznie, każdy znacznik `<img>` jest dzieckiem znacznika `<a>`, wewnątrz którego jest umieszczony; a ponieważ w znacznikach `<a>` nie ma innych znaczników, żaden ze znaczników `<img>` nie ma rodzeństwa.



Rysunek 15.3. *Często zdarza się, że pobierając jakieś elementy przy użyciu jQuery — na przykład w celu dodania procedury obsługi zdarzeń do znacznika <a> przedstawionego na tym rysunku — będziemy chcieli wykonać także jakieś operacje na innych elementach, które są z nimi w jakiś sposób powiązane (wyświetlić ramkę wokół elementu lub zmodyfikować znacznik <img> i tym podobne). Właśnie w takich przypadkach mogą pomóc funkcje służące do poruszania się po DOM*

Poniżej przedstawionych zostało kilka funkcji służących do poruszania się po DOM, dostępnych w bibliotece jQuery.

- Funkcja **`.find()`** odnajduje konkretny element wewnątrz aktualnie pobranego elementu. W tym przypadku należy zacząć do standardowego pobrania elementu, a następnie dodać wywołanie funkcji `.find()` i przekazać do niej odpowiedni selektor; oto przykład:

  ```
  $('#gallery').find('img')
  ```

  Powyższe wywołanie pobiera wszystkie znaczniki `<img>` umieszczone wewnątrz znacznika `<div>` z identyfikatorem `gallery`. Oczywiście, ten sam efekt można uzyskać, stosując selektor elementu potomnego w postaci `$('#gallery img')`. Jak już wspominaliśmy, funkcja `.find()` jest najczęściej stosowana w sytuacjach, kiedy już wcześniej został pobrany jakiś element strony, na którym wykonaliśmy jakieś operacje — takie jak dodanie procedury obsługi zdarzenia — a teraz chcemy pobrać inne, powiązane z nim elementy.

  Funkcja `.find()` używana jest do pobierania elementów potomnych (znaczników umieszczonych wewnątrz innych) aktualnie pobranego elementu. Jeśli zatem w przypadku zilustrowanym na rysunku 15.3 aktualnie pobranym elementem będzie `<div>`, funkcji `.find()` możemy użyć, by pobrać znaczniki `<a>` lub `<img>`.

Uwaga: Zajrzyj na stronę 536, by dowiedzieć się o ogromnych zaletach związanych z wydajnością działania skryptów, jakie zapewnia stosowanie funkcji `.find()`. Zazwyczaj stanowi ona szybszy sposób pobierania elementów niż stosowanie selektora elementów potomnych.

- Funkcja **`.children()`** jest nieco podobna do funkcji `.find()`. Także w jej wywołaniu można podać selektor, jednak ogranicza ona zakres pobieranych elementów jedynie do bezpośrednich potomków (dzieci) aktualnie pobranego elementu. Załóżmy, że na stronie znajduje się znacznik `<div>`, a wewnątrz niego grupa kolejnych znaczników `<div>`. Kliknięcie głównego znacznika `<div>` powinno spowodować wyświetlenie pozostałych, które są początkowo ukryte, i dodanie do nich czerwonego obramowania. Załóżmy, że zastosowaliśmy funkcję `.find()`, by zrealizować to zadanie przy użyciu następującego fragmentu kodu:

```
$('#mainDiv').click(function() {
    $(this).find('div').show().css('outline','red 2px solid');
});
```

Jednak takie rozwiązanie nie zadziałałoby prawidłowo, gdyby w znacznikach `<div>` umieszczonych wewnątrz znacznika głównego znajdowały się kolejne znaczniki `<div>`. W takim przypadku użycie powyższego kodu doprowadziłoby do zmiany wyglądu wszystkich znaczników `<div>` umieszczonych wewnątrz głównego, podczas gdy nam chodziło o wyróżnienie wyłącznie jego dzieci (potomków bezpośrednich). Problem ten rozwiązuje zastąpienie funkcji `.find()` funkcją `.children()`:

```
$('#mainDiv').click(function() {
    $(this).children('div').show().css('outline','red 2px solid');
});
```

Ta wersja kodu odnajduje wyłącznie te znaczniki `<div>`, które są dziećmi znacznika głównego i pomija wszystkie jego dalsze dzieci.

- Funkcja **`.parent()`**. W odróżnieniu od funkcji `.find()`, która poszukuje elementów wewnątrz pobranego znacznika, funkcja `.parent()` podróżuje w górę DOM i odnajduje znaczniki przodków. Taka możliwość może się przydać, gdybyśmy na przykład dodali procedurę obsługi zdarzeń do znaczników `<a>` z rysunku 15.3, lecz chcieli wykonać jakieś operacje na znaczniku `<div>` (choćby dodać do niego obramowanie lub kolor tła). W takim przypadku wystarczyłoby wywołać funkcję `.parent()`, by pobrać znacznik `<div>` i wykonać na nim zamierzone czynności; oto przykład:

```
$('#gallery a').hover(
   function() {
      var $this = $(this);
      // dodanie obramowania do odnośnika
      $this.css('outline','2px solid red');
      // dodanie koloru tła do znacznika div
      $this.parent().css('backgroundColor','rgb(110,138,195)');
   },
   function() {
      var $this = $(this);
      // usunięcie obramowania odnośnika
      $this.css('outline','');
      // usunięcie koloru tła znacznika div
      $this.parent().css('backgroundColor','');
   }
); // koniec funkcji hover
```

W tym przykładzie wskazanie odnośnika myszą powoduje wyświetlenie jego obramowania, a następnie pobranie jego rodzica (znacznika `<div>`) i zmianę jego koloru tła. Po usunięciu wskaźnika myszy z obszaru odnośnika usuwane są jego obramowanie oraz kolor tła jego rodzica (więcej informacji na temat zdarzenia `hover` można znaleźć na stronie 192). Aby przekonać się, jak w praktyce działa ten fragment kodu, wystarczy wyświetlić w przeglądarce plik *parent.html* umieszczony w przykładach do książki, w katalogu *R15*.

- Funkcja **`.closest()`** odnajduje najbliższego przodka pasującego do podanego selektora. W odróżnieniu od funkcji `.parent()`, która odnajduje bezpośredniego przodka (rodzica) elementu, funkcja `.closest()` pozwala na podanie selektora i odnajduje najbliższego przodka, który do niego pasuje. W przykładzie przedstawionym na rysunku 15.3 każdy obrazek jest umieszczony wewnątrz znacznika `<a>`; innymi słowy, ten znacznik `<a>` jest rodzicem obrazka. W jaki sposób

możemy pobrać znacznik , wewnątrz którego znajdują się te odnośniki (czyli bardziej odległego przodka w hierarchii kodu HTML)? Otóż, możemy w tym celu zastosować właśnie funkcję .closest():

```
1  $('#gallery img').click(function() {
2      var $this = $(this);
3      $this.css('outline','2px red solid');
4      $this.closest('div').css('backgroundColor','white');
5  }); // koniec funkcji click
```

Użyta wierszu 3. zmienna $this odwołuje się do znacznika <img>. Wywołanie .closest('div') oznacza natomiast, że chcemy znaleźć najbliższego przodka będącego znacznikiem . Najbliższym — bezpośrednim — przodkiem obrazka jest znacznik <a>, jednak nie jest to , dlatego jQuery go pominie i sprawdzi kolejnego przodka, i tak dalej, aż do momentu odszukania znacznika .

- Funkcja **.siblings()**. Funkcja ta może się przydać, kiedy chcemy odszukać element znajdujący się na tym samym poziomie struktury kodu HTML, co element aktualnie pobrany. Załóżmy, że cały czas posługujemy się przykładowym kodem przedstawionym na rysunku 15.3. Tym razem chcemy, by w momencie kliknięcia jednego z odnośników wszystkie pozostałe zostały nieznacznie wygaszone, a następnie ponownie rozjaśnione. Skorzystamy przy tym ze zdarzenia click, które będzie się odwoływać do klikniętego odnośnika, jednak chcemy zmodyfikować wygląd wszystkich pozostałych odnośników umieszczonych wewnątrz tego samego znacznika . Innymi słowy, zaczynamy od klikniętego odnośnika, jednak chcemy pobrać całe jego rodzeństwo. Możemy to zrobić przy użyciu następującego fragmentu kodu:

```
1  $('#gallery a').click(function() {
2      $(this).siblings().fadeTo(500,.3).fadeTo(250,1);
3  }); // koniec funkcji click
```

Wyrażenie $(this) zastosowane w powyższym przykładzie odwołuje się do klikniętego odnośnika, a zatem wywołanie funkcji .siblings() pozwoli pobrać wszystkie inne znaczniki <a> umieszczone w tym samym znaczniku .

Także funkcja .siblings() umożliwia przekazanie jednego argumentu — selektora — w ten sposób pozwala na ograniczenie liczby pobieranych elementów. Przykładowo załóżmy, że wewnątrz znacznika przedstawionego na rysunku 15.3 przed grupą odnośników znajduje się nagłówek oraz jeden akapit tekstu. Ponieważ zarówno ten nagłówek, jak i akapit umieszczone są wewnątrz znacznika razem ze wszystkimi znacznikami <a>, także stanowią rodzeństwo odnośników. Innymi słowy, po zastosowaniu przedstawionego wcześniej fragmentu kodu kliknięcie odnośnika spowodowałoby odtworzenie efektów także na tym nagłówku i akapicie. Aby zastosować efekty wyłącznie w odnośnikach, powinniśmy zmodyfikować drugi wiersz powyższego kodu w następujący sposób:

```
$(this).siblings('a').fadeTo(500,.3).fadeTo(250,1);
```

Selektor 'a' umieszczony w wywołaniu funkcji .siblings() sprawi, że zwróci ona tylko znaczniki będące rodzeństwem aktualnie pobranego elementu, które jednocześnie są znacznikami <a>. Aby przekonać się, jak w praktyce działa ten przykład, wystarczy wyświetlić w przeglądarce plik *siblings.html* umieszczony w katalogu *R15*.

KLINIKA ZAAWANSOWANEGO UŻYTKOWNIKA

Przerywanie poruszania się po DOM przy użyciu funkcji .end()

Abyś mógł zrobić jak najwięcej, pisząc jak najmniej kodu, jQuery pozwala na tworzenie sekwencji wywołań. Technika ta została opisana na stronie 156, jednak ogólnie rzecz ujmując, polega ona na pobraniu elementów, wykonaniu na nich pewnej operacji, a następnie wykonaniu kolejnych poprzez dodawanie jednej funkcji za drugą. Gdybyśmy chcieli pobrać wszystkie akapity na stronie, następnie je wygasić i ponownie wyświetlić, moglibyśmy to zrobić przy użyciu następującego kodu:

```
$('p').fadeOut(500).fadeIn(500);
```

W taki sposób można łączyć dowolnie wiele funkcji, w tym także funkcje do poruszania się po DOM opisane na poprzednich stronach. Załóżmy na przykład, że chcemy pobrać znacznik `<div>`, dodać od niego obramowanie, a następnie pobrać wszystkie znaczniki `<a>` umieszczone wewnątrz tego elementu `<div>` i zmienić ich kolor. Wszystko to możemy zrobić za pomocą następującego wywołania:

```
$('div').css('outline','2px red
➥solid').find('a').css('color','purple');
```

Oto efekt rozłożenia tej instrukcji na fragmenty:

1. `$('div')` pobiera wszystkie znaczniki `<div>`.
2. `.css('outline','2px red solid')` dodaje do tego elementu czerwone obramowanie o szerokości 2 pikseli.
3. `.find('a')` pobiera wszystkie odnośniki umieszczone wewnątrz pobranego wcześniej znacznika `<div>`.
4. `.css('color','purple')` zmienia kolor tekstu tych odnośników na fioletowy.

Gdy dodamy do takiej sekwencji wywołanie funkcji służącej do poruszania się po DOM, zmieniamy aktualnie pobrane elementy. W powyższym przykładzie początkowo pobrany był znacznik `<div>`, jednak później, w połowie sekwencji pobraliśmy wszystkie odnośniki umieszczone w tym znaczniku `<div>`. Czasami może się zdarzyć, że będziemy chcieli powrócić do początkowego stanu kolekcji pobranych elementów. Innymi słowy, najpierw będziemy chcieli pobrać pewną grupę znaczników, następnie ją zmienić, by w końcu powrócić do początkowej grupy. Przykładowo załóżmy, że użytkownik może kliknąć znacznik `<div>`, którego poziom nieprzezroczystości wynosi 50%, a w odpowiedzi chcemy zmienić jego nieprzezroczystość do 100%, zmienić kolor nagłówka umieszczonego wewnątrz tego znacznika `<div>` i dodać kolor tła do każdego akapitu (znacznika `<p>`) umieszczonego w tym znaczniku. Jedno zdarzenie — kliknięcie — musi doprowadzić do wykonania kilku operacji odnoszących się do różnych elementów strony. Jednym ze sposobów wykonania tego zadania byłoby zastosowanie następującego kodu:

```
$('div').click(function() {
  var $this = $(this);
  $(this).fadeTo(250,1);
                    // rozjaśniamy znacznik <div>
 $(this).find('h2').css('color','#F30');
  $(this).find('p')
         .('backgroundColor','#F343FF');
}); // koniec funkcji click
```

W tym przypadku możliwość tworzenia sekwencji wywołań może się okazać bardzo przydatna. Zamiast trzykrotnego stosowania wyrażenia `$(this)` możemy użyć go tylko raz i dodać do niego sekwencję odpowiednich wywołań. Jednak gdybyśmy próbowali nadać tej sekwencji poniższą postać, pojawiłyby się problemy:

```
$('div').click(function() {
   $(this).fadeTo(250,1)
          .find('h2')
          .css('color','#F30')
          .find('p')
          .('backgroundColor','#F343FF');
}); // koniec funkcji click
```

Można odnieść wrażenie, że powyższa sekwencja jest prawidłowa, ale problemy zaczynają się po wywołaniu funkcji `.find('h2')`, która zmienia aktualnie pobrany element z `<div>` na umieszczony wewnątrz niego znacznik `<h2>`. Kiedy zostaje wywołana kolejna funkcja `.find()`, czyli `.find('p')`, jQuery spróbuje odnaleźć znaczniki `<p>` wewnątrz nagłówka `<h2>`, a nie wewnątrz znacznika `<div>`. Na szczęście można wywołać funkcję `.end()`, która odtwarza ostatnie zmiany wprowadzone w kolekcji pobranych elementów i przywraca jej poprzedni stan. W naszym przypadku możemy użyć funkcji `.end()`, by przywrócić pobrany wcześniej znacznik `<div>` i dopiero potem rozpocząć poszukiwanie znaczników `<p>`:

```
$('div').click(function() {
  $(this).fadeTo(250,1)
         .find('h2')
         .css('color','#F30')
         .end()
         .find('p')
         .('backgroundColor','#F343FF');
}); // koniec funkcji click
```

Należy zwrócić uwagę na funkcję `.end()` wywoływaną bezpośrednio za funkcją `.css('color','#f30');` to właśnie ona przywraca wcześniej pobrany element `<div>`, dzięki czemu wykonywane potem wywołanie `.find('p')` będzie poszukiwać akapitów wewnątrz znacznika `<div>`.

- Funkcja **.next()** zwraca następny element, będący rodzeństwem aktualnie pobranego elementu. Miałeś już okazję zobaczyć tę funkcję w działaniu we wcześniejszej części książki, w przykładzie prezentującym najczęściej zadawane pytania, przedstawionym na stronie 204. W przykładzie tym kliknięcie pytania powodowało wyświetlenie, a następnie ukrycie, odpowiedniej odpowiedzi. Każde pytanie było reprezentowane przez znacznik `<h2>`, a odpowiedź — przez znacznik `<div>` umieszczony w kodzie strony bezpośrednio za pytaniem. Nagłówek oraz znacznik `<div>` z odpowiedzią były zatem rodzeństwem; jednak ich rodzeństwem były także wszystkie inne nagłówki i znaczniki `<div>` odpowiedzi. Dlatego też w ramach obsługi kliknięcia konieczne było pobranie znacznika umieszczonego bezpośrednio za kliknięty pytaniem (innymi słowy — następnego znacznika należącego do rodzeństwa kliknięty elementu). Funkcja `.next()`, podobnie jak `.siblings()`, pozwala na podanie opcjonalnego selektora ograniczającego zwracane wyniki. (Praktyczny przykład jej zastosowania można znaleźć w pliku *complete_faq.html* umieszczonym w katalogu *R05*).
- Funkcja **.prev()** działa tak samo jak `.next()`, z tym że pobiera nie następny, a poprzedni element.

Uwaga: Więcej funkcji jQuery pozwalających na poruszanie się po DOM można znaleźć na stronie *http://api jquery.com/category/traversing/*.

Inne funkcje do manipulacji kodem HTML

Bardzo często będziemy chcieli dynamicznie dodawać, usuwać oraz modyfikować kod HTML stron WWW. Możemy przykładowo mieć ochotę, by po kliknięciu przycisku przesyłającego na stronie został wyświetlony komunikat „Informacje zostały przesłane na serwer. Proszę czekać", albo gdy użytkownik umieści wskaźnik myszy, będziemy chcieli wyświetlić nad nim ramkę z tytułem oraz dodatkowymi informacjami na jego temat. W obu tych przypadkach pojawia się konieczność dodania do strony nowego kodu HTML. Najczęściej używane funkcje zapewniające takie możliwości zostały przedstawione na stronie 157 w rozdziale 4. Niżej zostały pokrótce przypomniane.

- Funkcja **.text()** umieszcza podany tekst wewnątrz aktualnie wybranego elementu. Oto przykład:

```
$('#error').text('Musisz podać adres email.');
```

- Funkcja **.html()** działa podobnie jak funkcja `.text()`, z tym że pozwala na dodawanie do strony dowolnego kodu HTML, a nie samego tekstu:

```
$('#tooltip').html('<h2>Esquif Avalon</h2><p>Zaprojektowany z myślą
↪o przygodzie na canoe.</p>');
```

- Funkcja **.append()** pozwala dodać przekazany do niej fragment kodu HTML na końcu elementu (na przykład na końcu elementu `div`, bezpośrednio przed zamykającym znacznikiem `</div>`). Doskonale nadaje się do dodawania nowych punktów na końcu listy.

- Funkcja **.prepend()** pozwala dodać przekazany w jej wywołaniu kod HTML na samym początku elementu (na przykład na samym początku elementu `div`, bezpośrednio za otwierającym znacznikiem `<div>`).
- Funkcja **.before()** dodaje kod HMTL przed aktualnie pobranym elementem.
- Funkcja **.after()** działa podobnie jak funkcja `.before()`, z tą różnicą, że nowy kod HTML jest dodawany za aktualnie pobranym elementem (za jego znacznikiem zamykającym).

To, której funkcji użyjemy, zależy przede wszystkim od tego, co chcemy osiągnąć. Język JavaScript jest w znacznej mierze przeznaczony do automatyzowania operacji, które projektanci stron WWW zazwyczaj wykonują ręcznie, takich jak dodawanie kodu HTML i CSS w celu utworzenia strony. Jeśli piszesz program, który ma dynamicznie dodawać do strony jakieś treści, na przykład etykietę ekranową, komunikat o błędzie, wyróżniony cytat i tym podobne, powinieneś wyobrazić sobie, jak ma wyglądać gotowy produkt i kody HTML i CSS konieczne do osiągnięcia zamierzonych celów.

Gdybyś chciał wyświetlić na stronie specjalny komunikat, w momencie gdy użytkownik wskaże myszą konkretny przycisk, spróbuj najpierw utworzyć stronę prezentującą taki komunikat bez wykorzystania kodu JavaScript — jedynie przy użyciu kodów HTML i CSS. Kiedy wstępna wersja komunikatu będzie już gotowa, przyjrzyj się, jak wygląda jej kod HTML. Czy został umieszczony przed jakimś elementem? Jeśli tak, to będziesz go mógł dodać, używając funkcji `.before()`. A może jest umieszczony wewnątrz jakiegoś znacznika? W takim przypadku będziesz mógł wykorzystać funkcje `.append()` lub `.prepend()`.

Biblioteka jQuery udostępnia także kilka funkcji służących do usuwania istniejących fragmentów strony. Oto one.

- Funkcja **.replaceWith()** całkowicie usuwa aktualnie pobrane elementy (w tym sam znacznik oraz całą jego zawartość) i zastępuje je kodem HTML podanym w wywołaniu. Aby na przykład zastąpić przycisk przesyłający formularz komunikatem „Przetwarzanie…”, można by użyć następującego wywołania:

```
$(':submit').replaceWith('<p>Przetwarzanie...</p>');
```

- Funkcja **.remove()** usuwa aktualnie pobrane elementy z DOM; co właściwie sprowadza się do usunięcia ich ze strony. Aby na przykład usunąć ze strony znacznik `<div>` o identyfikatorze `error`, można by użyć następującego wywołania:

```
$('#error').remove();
```

Choć być może wystarczą Ci funkcje opisane powyżej oraz te z rozdziału 4., jednak warto wiedzieć, że jQuery udostępnia także inne funkcje pozwalające na manipulowanie kodem HTML strony na inne sposoby.

- Funkcja **.wrap()** zapisuje aktualnie pobrane elementy wewnątrz pary znaczników HTML. Co można by zrobić, gdybyśmy chcieli opracować wymyślny efekt prezentujący tytuły zdjęć pokazywanych na stronie? Moglibyśmy zacząć od pobrania samych zdjęć, zapisania ich wewnątrz znacznika `<div>` należącego do klasy `figure` i dodania wewnątrz niego znacznika `<p>` z klasy `caption`. Następnie, korzystając z CSS, moglibyśmy w dowolny sposób sformatować oba te znaczniki. Oto sposób, w jaki można by to zrobić:

```
1  // przeglądamy listę wszystkich obrazków
2  $('img').each(function() {
3      // zapisujemy odwołanie do aktualnego obrazka
4      var $this = $(this);
5      // pobieramy wartość właściwości alt na potrzeby wyświetlenia tytułu
6      var caption = $this.attr('alt');
7      // dodanie kodu HTML
8      $this.wrap('<div class="figure"></div>').after('<p>' + caption
       ↪+ '</p>');
9  }); // koniec funkcji each
```

Powyższy kod najpierw pobiera wszystkie obrazki na stronie, a następnie przetwarza każdy z nich przy użyciu w tym celu funkcji `.each()` (opisanej na stronie 167). W wierszu 4. aktualnie pobrany obrazek jest zapisywany w zmiennej (to bardzo dobre rozwiązanie, o czym już wspominaliśmy na stronie 544). W wierszu 6. pobieramy wartość atrybutu `alt` obrazka i zapisujemy ją w zmiennej `caption`. W końcu, w wierszu 8., dodajemy ten tytuł za obrazkiem, używając do tego celu funkcji `.after()`.

Uwaga: Przykład zastosowania funkcji `.wrap()` można znaleźć w pliku *wrap.html* dostępnym w przykładach do książki, w katalogu *R15*.

W wywołaniu funkcji `.wrap()` należy przekazać kompletną parę znaczników — `$('p').wrap('<div></div>')` — bądź nawet kod HTML składający się z kilku znaczników zagnieżdżonych, takich jak te:

```
$('#example').wrap('<div id="outer"><div id="inner"></div></div>');
```

W tym przykładzie jQuery umieści aktualnie pobrane elementy wewnątrz dwóch znaczników `<div>`; powstanie kod przypominający przedstawiony poniżej:

```
<div id="outer">
  <div id="inner">
    <div id="example">To jest oryginalny kod strony.</div>
  </div>
</div>
```

- Funkcja **`.wrapInner()`** zapisuje zawartość każdego z aktualnie pobranych elementów wewnątrz podanego kodu HTML. Załóżmy na przykład, że na naszej stronie znajduje się następujący kod HTML:

```
<div id="outer">
<p>To jest zawartość elementu outer</p>
</div>
```

Jeśli teraz przeglądarka napotka i wykona następujące wywołanie: `$('#outer').wrapInner('<div id= inner ></div>');`, kod HTML strony zostanie przekształcony do następującej postaci:

```
<div id="outer">
<div id="inner">
<p>To jest zawartość elementu outer</p>
</div>
</div>
```

- Funkcja **`.unwrap()`** usuwa znaczniki nadrzędne, wewnątrz których są umieszczone aktualnie pobrane elementy. Przykładowo załóżmy, że na naszej stronie znajduje się następujący kod HTML:

```
<div>
<p>akapit</p>
<div>
```

  W takim przypadku wykonanie wywołania `$('p').unwrap()` zmieni kod strony do postaci:

```
<p>akapit</p>
```

  Jak widać, zewnętrzny znacznik `<div>` został usunięty. Zwróć uwagę, że, w odróżnieniu od innych funkcji opisywanych w tym rozdziale, funkcja `.unwrap()` nie pobiera żadnych argumentów — innymi słowy, w nawiasach umieszczonych za nazwą funkcji nie można nic zapisać, gdyż funkcja nie zadziała.

- Funkcja **`.empty()`** usuwa z aktualnie pobranych elementów całą zawartość, same elementy pozostają jednak na miejscu. Załóżmy, że na naszej stronie znajduje się znacznik `<div>` o identyfikatorze `messageBox`. Za pomocą skryptów możemy dynamicznie modyfikować treść tego elementu i wyświetlać komunikaty zależne od czynności wykonywanych przez użytkownika. Moglibyśmy dodać do niego bardzo wiele nagłówków, obrazków i akapitów tekstu, by wyświetlać użytkownikowi informacje o statusie strony. Jednak w jakiejś chwili może się okazać, że konieczne będzie usunięcie całej zawartości tego elementu (na przykład, jeśli w danej chwili nie będą miały być prezentowane żadne komunikaty), ale pozostawienie go na miejscu, by później można było w nim wyświetlić kolejne komunikaty. W celu usunięcia zawartości elementu możemy użyć następującego wywołania:

```
$('#messageBox').empty();
```

  Podobnie jak funkcja `.unwrap()`, także i `.empty()` nie pobiera żadnych argumentów.

Uwaga: Biblioteka jQuery udostępnia więcej funkcji służących do operowania na kodzie HTML strony. Pełną ich listę można znaleźć na stronie *http://api.jquery.com/category/manipulation/*.

16

ROZDZIAŁ

Zaawansowane techniki języka JavaScript

W tym rozdziale poznasz zestaw technik, które pomogą Ci stać się lepszym programistą języka JavaScript. Większość opisanych tu rozwiązań nie jest niezbędna do pisania funkcjonalnych programów, dlatego nie musisz rozumieć ich wszystkich. W kilku pierwszych podrozdziałach zamieszczone zostały porady i metody związane z posługiwaniem się łańcuchami znaków, liczbami oraz datami, a kiedy dokładnie poznasz podstawy, zamieszczone w nich informacje naprawdę pomogą Ci przetwarzać informacje podawane przez użytkownika w formularzach, operować na kodzie HTML i atrybutach znaczników oraz generować daty. Podrozdział „Łączenie różnych elementów", rozpoczynający się na stronie 606, zawiera wartościowe wskazówki dla początkujących, jednak z powodzeniem możesz napisać wiele programów bez korzystania z informacji zamieszczonych w pozostałych podrozdziałach. Jeśli jednak chcesz rozwinąć swe umiejętności, przeczytanie tego rozdziału pomoże Ci obrać właściwy kierunek.

Stosowanie łańcuchów znaków

Łańcuchy znaków są typem danych, których będziesz używał najczęściej: dane pobierane z formularzy, ścieżki do obrazków, adresy URL i kod HTML, który chcesz umieścić na stronie, to są przykłady liter, symboli i cyfr, jakie składają się na łańcuchy znaków. Podstawowe informacje o łańcuchach znaków zostały zamieszczone w rozdziale 2., jednak język JavaScript udostępnia wiele przydatnych metod ułatwiających ich stosowanie i modyfikację.

Określanie długości łańcucha

W niektórych sytuacjach konieczne może być określenie ilości znaków w łańcuchu. Załóżmy na przykład, że chcesz upewnić się, iż zawsze gdy ktoś zakłada konto użytkownika na Twojej tajnej witrynie, poda przy tym hasło o długości od 6 do 15 znaków. Każdy łańcuch znaków posiada właściwość o nazwie `length`, która zawiera właśnie tę informację. Wystarczy dodać do nazwy zmiennej kropkę, a po niej umieścić właściwość `length`, by odczytać liczbę znaków w łańcuchu — `nazwa.length`.

Przykładowo załóżmy, że na naszej stronie znajduje się formularz z polem tekstowym o identyfikatorze `password`. Aby upewnić się, że w polu tym podano łańcuch znaków o prawidłowej długości, można zastosować instrukcję warunkową (patrz strona 93) testującą wartość właściwości `length`:

```
var password = $('#password').val();
if (password.length <= 6) {
alert('Hasło jest za krótkie.');
} else if (password.length > 15) {
alert('Hasło jest za długie.');
}
```

Kod przedstawiony w tym przykładzie pobiera zawartość pola zawierającego hasło, w którym użyto możliwości biblioteki jQuery — a konkretnie wywołania o postaci: `$('#password').val()` — jednak cała jego reszta stanowi zwyczajny kod JavaScript. Doskonałym sposobem wykorzystania tego fragmentu kodu byłoby umieszczenie go w odrębnej funkcji, na przykład w sposób przedstawiony poniżej:

```
function verifyPassword() {
  var password = $('#password').val();
  if (password.length <= 6) {
    alert('Hasło jest za krótkie.');
  } else if (password.length > 15) {
    alert('Hasło jest za długie.');
  }
}
```

Taką funkcję można by później wywoływać podczas obsługi zdarzenia wysyłania formularza (patrz strona 285), aby sprawdzić, czy użytkownik podał dostatecznie długie hasło:

```
$('form').submit(verifyPassword);
```

Można by jej także użyć do obsługi zdarzeń `blur` (patrz strona 288) generowanych przez pole tekstowe do wprowadzania hasła, dzięki czemu kiedy użytkownik naciśnie klawisz *Tab* lub kliknie gdzieś poza polem, będzie można od razu sprawdzić, czy długość podanego hasła jest prawidłowa. Załóżmy na przykład, że pole hasła ma identyfikator `password`, w takim przypadku funkcji `verifyPassword` można by użyć do obsługi generowanych przez niego zdarzeń `blur` w następujący sposób:

```
$('#password').blur(verifyPassword);
```

Zmiana wielkości znaków w łańcuchu

JavaScript udostępnia dwie metody służące do zmiany wielkości wszystkich liter w łańcuchu na wielkie lub małe; za ich pomocą można zmienić łańcuch znaków `witamy` na `WITAMY` oraz `NIE` na `nie` . Możesz się zastanawiać, po co

wykonywać takie zmiany. Otóż, zmiana wielkości liter w łańcuchu na określoną wielkość ułatwia porównywanie dwóch łańcuchów. Wyobraź sobie, że piszesz program obsługujący internetowe quizy, taki jak przedstawiony w rozdziale 3. (patrz strona 124), i jedno z pytań brzmi: „Kto był pierwszym Amerykaninem, który wygrał Tour de France?". Do sprawdzenia odpowiedzi na to pytanie mógłbyś użyć następującego fragmentu kodu:

```
var correctAnswer = 'Greg LeMond';
var response = prompt('Kto był pierwszym Amerykaninem, który wygrał
➥Tour de France?', '');
if (response == correctAnswer) {
// odpowiedź prawidłowa
} else {
// odpowiedź nieprawidłowa
}
```

Oczywiście, prawidłową odpowiedzią jest Greg LeMond; co by się jednak stało, gdyby użytkownik, odpowiadając na to pytanie, wpisał *Greg Lemond*? W takim przypadku warunek testowany w instrukcji `if` przybrałby następującą postać: `'Greg LeMond' == 'Greg Lemond'`. Ponieważ język JavaScript rozróżnia wielkie i małe litery, zatem mała litera `'m'` ze słowa `'Lemond'` nie będzie równa wielkiej literze `'M'` ze słowa `'LeMond'`. Program obsługujący quiz mógłby zatem uznać tę odpowiedź za nieprawidłową. Dokładnie to samo zdarzyłoby się, gdyby użytkownik przypadkowo nacisnął klawisz *Caps Lock* i podał odpowiedź w postaci `'GREG LEMOND'`.

Aby rozwiązać ten problem, oba łańcuchy można skonwertować do liter tej samej wielkości, a dopiero potem je porównać:

```
if (response.toUpperCase() == correctAnswer.toUpperCase()) {
// odpowiedź prawidłowa
} else {
// odpowiedź nieprawidłowa
}
```

W tym przypadku wewnątrz wyrażenia warunkowego zarówno odpowiedź udzielona przez użytkownika, jak i prawidłowa odpowiedź na pytanie są przekształcane do wielkich liter, a zatem łańcuch `'Greg Lemond'` zostanie przekształcony na `'GREG LEMOND'`, a łańcuch `'Greg LeMond'` na `'GREG LEMOND'`.

Aby przekształcić łańcuch znaków do małych liter, wystarczy użyć metody `toLower➥Case()`, jak pokazano na poniższym przykładzie:

```
var answer = 'Greg LeMond';
alert(answer.toLowerCase()); // 'greg lemond'
```

Trzeba zwrócić uwagę, że żadna z tych metod nie modyfikuje oryginalnego łańcucha znaków przechowywanego w zmiennej — zwracają one nowy łańcuch, który jest zapisany odpowiednio wielkimi lub małymi literami. A zatem w powyższym przykładzie zmienna `answer` wciąż będzie zawierać łańcuch w postaci `'Greg LeMond'`, nawet po dokonaniu porównania. (Innymi słowy, metody te działają podobnie jak funkcje opisane na stronie 120, które zwracają jakieś wartości).

Przeszukiwanie łańcuchów znaków: zastosowanie indexOf()

Język JavaScript udostępnia kilka technik przeszukiwania łańcuchów znaków i odnajdywania wewnątrz nich słów, cyfr oraz określonych sekwencji znaków. Przeszukiwanie łańcuchów może się przydać w sytuacji, kiedy będziemy chcieli określić

typ przeglądarki używanej przez osobę oglądającą nasze strony. Każda przeglądarka zapisuje informacje o sobie w łańcuchu znaków. Można go z łatwością zobaczyć, wystarczy w tym celu umieścić na stronie poniższy fragment kodu, a samą stronę wyświetlić w przeglądarce:

```
<script>
alert(navigator.userAgent);
</script>
```

gdzie `navigator` jest jednym z wbudowanych obiektów przeglądarki, a `userAgent` — właściwością tego obiektu. Właściwość ta zawiera długi łańcuch znaków, w którym umieszczono wiele informacji; na przykład w przeglądarce Internet Explorer 10 działającej w systemie Windows 8 właściwość ta przyjmuje wartość `Mozilla/5.0 (compatible; MSIE 10.0; Windows NT 6.2; Trident/6.0)`. A zatem, gdybyśmy chcieli sprawdzić, czy przeglądarką użytkownika jest Internet Explorer 10, wystarczy spróbować odnaleźć we właściwości `userAgent` łańcuch znaków `MSIE 10`.

Jednym ze sposobów przeszukiwania łańcuchów znaków jest zastosowanie metody `indexOf()`. Najprościej rzecz ujmując, aby z niej skorzystać, należy umieścić za łańcuchem lub zmienną kropkę, po niej zapisać nazwę metody — `indexOf()` — a wewnątrz nawiasów podać poszukiwany łańcuch znaków. Podstawowa składnia jej wywołania wygląda następująco:

```
string.indexOf('poszukiwany łańcuch znaków')
```

Metoda `indexOf()` zwraca liczbę. Gdy poszukiwanego łańcucha nie uda się odnaleźć, zwracana jest wartość `-1`. Jeśli zatem chcemy sprawdzić, czy używaną przeglądarką jest Internet Explorer, możemy użyć następującego fragmentu kodu:

```
var browser = navigator.userAgent; // to jest łańcuch znaków
if (browser.indexOf('MSIE') != -1) {
// to jest Internet Explorer
}
```

Jeśli metoda `indexOf()` nie odnajdzie łańcucha `'MSIE'` we właściwości `userAgent`, zwróci `-1`, dlatego też warunek testuje, czy zwrócona przez nią wartość jest różna (`!=`) od `-1`.

Kiedy metoda `indexOf()` *odnajdzie* łańcuch znaków, zwraca liczbę określającą miejsce jego początku w przeszukiwanym łańcuchu. Poniższy przykład wszystko wyjaśni:

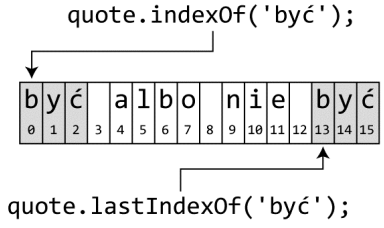
```
var quote = 'być albo nie być';
var searchPosition = quote.indexOf('być'); // zwraca 0
```

W tym przykładzie metoda `indexOf()` poszukuje położenia łańcucha `'być'` w łańcuchu `'być albo nie być'`. Dłuższy łańcuch zaczyna się od `'być'`, zatem metoda odnajdzie poszukiwane słowo na pierwszym miejscu przeszukiwanego łańcucha. Jednak, ze względu na wariactwa występujące w świecie programowania, temu pierwszemu miejscu odpowiada liczba 0, kolejnemu miejscu łańcucha (temu, w którym znajduje się litera `y`) odpowiada liczba 1, trzeciemu (literze `ć`) — liczba 2 (więcej informacji na ten temat można znaleźć na stronie 80; w podobny sposób są indeksowane także tablice).

Metoda `indexOf()` rozpoczyna poszukiwania od początku łańcucha znaków. Istnieje także możliwość rozpoczęcia poszukiwań na końcu łańcucha; służy do tego metoda `lastIndexOf()`. W przedstawionym powyżej cytacie słowo `'być'` występuje dwa razy, a zatem pierwsze `'być'` możemy odszukać, używając metody `indexOf()`, a drugie — metody `lastIndexOf()`:

```
var quote = "być albo nie być";
var firstPosition = quote.indexOf('być'); //zwraca 0
var lastPosition = quote.lastIndexOf('być'); //zwraca 13
```

Wyniki zwracane przez obie metody zostały zilustrowane na rysunku 16.1. W obu przypadkach, gdyby słowo `'być'` w ogóle nie występowało w łańcuchu, obie metody zwróciłyby wartość `-1`, a gdyby w przeszukiwanym łańcuchu poszukiwane słowo występowało tylko raz, obie metody zwróciłyby ten sam wynik — indeks początku poszukiwanego słowa w dłuższym łańcuchu.



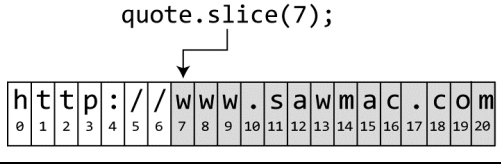
Rysunek 16.1. *Metody indexOf() oraz lastIndexOf() poszukują podanego ciągu w dłuższym łańcuchu znaków. Jeśli uda się go odnaleźć, obie zwracają jego położenie w dłuższym łańcuchu*

Pobieranie fragmentu łańcucha przy użyciu metody slice()

Do pobierania fragmentu łańcucha służy metoda `slice()`. Zwraca ona określony fragment łańcucha. Przykładowo załóżmy, że dysponujemy łańcuchem znaków `http://www.sawmac.com` i chcemy usunąć z niego początkowy ciąg `http://`. Jednym z rozwiązań jest pobranie fragmentu łańcucha rozpoczynającego się za początkowym `http://`; można to zrobić przy użyciu następującego fragmentu kodu:

```
var url = 'http://www.sawmac.com';
var domain = url.slice(7); // www.sawmac.com
```

Metoda `slice()` wymaga przekazania *liczby* określającej początkowy indeks pobieranego fragmentu łańcucha (patrz rysunek 16.2). W naszym przypadku jej wywołanie ma postać `url.slice(7)` — a indeks 7 oznacza ósmą literę łańcucha (pamiętaj, że znaki w łańcuchu są liczone od 0). Metoda ta zwraca wszystkie znaki, zaczynając od tego o podanym indeksie, a kończąc na ostatnim znaku łańcucha.



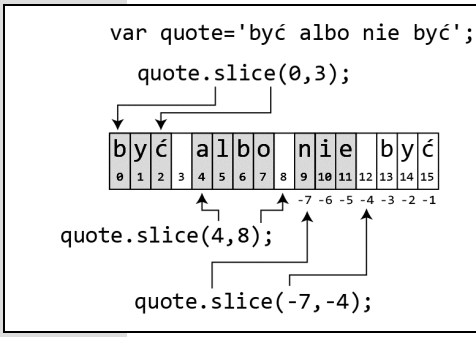
Rysunek 16.2. *Jeśli w wywołaniu metody slice() nie zostanie podany drugi argument, zwróci ona całą zawartość łańcucha, od znaku o podanym indeksie (w tym przykładzie jest to znak o indeksie 7) do kończącego łańcuch*

Można także pobrać określoną liczbę znaków z łańcucha — w tym celu wystarczy przekazać w wywołaniu metody drugi argument. Oto podstawowa składnia wywołania metody `slice()`:

```
string.slice(poczatek, koniec);
```

Argument `poczatek` jest liczbą określającą pierwszy znak pobieranego fragmentu łańcucha. Drugi argument — `koniec` — jest jednak nieco kłopotliwy. Nie określa indeksu ostatniego znaku fragmentu, lecz jego indeks powiększony o 1. Gdybyśmy

chcieli pobrać pierwsze pięć znaków łańcucha „być albo nie być", jako pierwszy argument wywołania metody `slice()` powinniśmy przekazać wartość `0`, a jako drugi — wartość `5`. Jak widać na rysunku 16.3, znak o indeksie `0` jest pierwszym znakiem łańcucha, a znak o indeksie `5` — szóstym znakiem łańcucha, jednak ten ostatni wskazany znak nie jest pobierany. Innymi słowy, znak określony drugim argumentem metody `slice()` *nigdy* nie zostanie dodany do zwracanego fragmentu łańcucha.



Rysunek 16.3. *Metoda slice() pobiera fragment łańcucha znaków. Początkowy łańcuch nie jest przy tym w żaden sposób modyfikowany. Przykładowo wywołanie quote.slice(0,3) w żaden sposób nie zmienia łańcucha zapisanego w zmiennej quote. Metoda ta zwraca wyznaczony fragment łańcucha, który można zapisać w zmiennej, wyświetlić w okienku informacyjnym albo nawet użyć jako argumentu w wywołaniu jakiejś funkcji*

Wskazówka: Jeśli chcemy pobrać z łańcucha określoną liczbę znaków, wystarczy dodać ją do wartości przekazywanej jako pierwszy argument wywołania metody `slice()`. Gdybyśmy chcieli pobrać pierwsze 10 znaków łańcucha, pierwszy argument miałby wartość `0` (pierwszy znak łańcucha), a drugi — 0 + 10, czyli `10`. Zatem wywołanie miałoby postać `slice(0,10)`.

W wywołaniu metody `slice()` można także podawać liczby ujemne, na przykład: `slice(-7,-4)`. Zastosowanie liczby mniejszej od zera sprawi, że indeks znaku będzie wyznaczony względem końca łańcucha i liczony w kierunku jego początku (co pokazano na rysunku 16.3).

Wskazówka: Gdybyśmy na przykład chcieli pobrać fragment zawierający X ostatnich znaków łańcucha, wystarczy w tym celu przekazać w wywołaniu metody slice() jeden argument: liczbę ujemną odpowiadająca ilości znaków, które należy zwrócić. Aby na przykład pobrać dokładnie 6 ostatnich znaków łańcucha, należałoby użyć wywołania o następującej postaci:

```
var end_of_string = quote.slice(-6);
```

Odnajdywanie wzorów w łańcuchach

Czasami może pojawić się konieczność przejrzenia łańcucha w poszukiwaniu nie konkretnej wartości, lecz określonego wzorca znaków. Załóżmy na przykład, że chcemy upewnić się, iż numer telefonu podany przez użytkownika podczas wypełniania internetowego zamówienia został zapisany w prawidłowym formacie. Nie chodzi przy tym o jakiś konkretny numer, taki jak 503-555-212. Interesuje nas ogólny wzorzec: trzy cyfry, łącznik, kolejne trzy cyfry, jeszcze jeden łącznik i ponownie trzy cyfry. Chcemy sprawdzić wartość podaną przez użytkownika i jeśli okaże się, że pasuje do wzorca (na przykład numer ma postać: 415-555-384, 408-555-782, 212-555-428 i tak dalej), wszystko będzie w porządku. Jeśli jednak numer nie będzie pasował do wzorca (bo użytkownik wpisał łańcuch znaków *243lllkkkmmmnn*), chcemy wyświetlić komunikat: „Hej stary, nie próbuj nas robić w konia!".

Język JavaScript pozwala na stosowanie **wyrażeń regularnych** w celu odnajdywania wzorców w łańcuchach znaków. Wyrażenie regularne to sekwencja znaków, definiujących wzorzec, który chcemy odszukać. Jak to często bywa z terminami programistycznymi, określenie „wyrażenie regularne" jest trochę myląco. Poniżej przedstawione zostało jedno z często używanych wyrażeń regularnych:

```
/^[-\w.]+@([a-zA-ZO-9][-a-zA-ZO-9]+\.)+[a-zA-Z]{2,4}$/
```

Nie ma w nim nic, co mogłoby wyglądać na „regularne", no chyba że jesteśmy superobcymi z Omicrona 9. Do tworzenia wyrażeń regularnych używane są znaki i ich sekwencje, takie jak `*`, `+`, `?` oraz `\w`, które są następnie tłumaczone przez interpreter JavaScriptu do postaci umożliwiającej dopasowywanie ich do prawdziwych znaków zapisywanych w łańcuchach — liter, cyfr i tym podobnych.

Uwaga: Profesjonaliści czasami używają także skróconej nazwy wyrażeń regularnych — *regex* (od angielskich słów *regular expression*).

Tworzenie i stosowanie podstawowych wyrażeń regularnych

W celu utworzenia wyrażenia regularnego w języku JavaScript konieczne jest zbudowanie obiektu wyrażenia, który ma postać sekwencji znaków zapisanych pomiędzy dwoma znakami ukośnika. Aby na przykład utworzyć wyrażenie regularne pasujące do słowa „witaj", należałoby użyć następującej instrukcji:

```
var myMatch = /witaj/;
```

Podobnie do pary cudzysłowów, które wyznaczają łańcuch znaków, para znaków ukośnika — `/` — tworzy wyrażenie regularne.

JavaScript udostępnia także kilka metod obiektu łańcuchów znaków, które mogą operować na wyrażeniach regularnych (zostały one opisane w tej części rozdziału, od strony 570), jednak najprostszą z nich jest metoda `search()`. Działa podobnie jak metoda `indexOf()`, lecz zamiast odnajdywać jeden łańcuch znaków wewnątrz innego, poszukuje wzorca (czyli wyrażenia regularnego). Załóżmy, że chcemy odnaleźć łańcuch znaków „być" w łańcuchu „być albo nie być". Dowiedziałeś się, jak to zrobić na stronie 567, a poniżej zobaczysz, jak można zrobić to samo, korzystając z wyrażeń regularnych:

```
var myRegEx = /być/; // wyrażenia regularne nie są zapisywane w cudzysłowach
var quote = 'być albo nie być';
var foundPosition = quote.search(myRegEx); // zwraca 0
```

Jeśli metoda `search()` odnajdzie pasujący łańcuch znaków, zwróci położenie jego pierwszego znaku; jeśli jednak niczego nie znajdzie, zwróci wartość `-1`. A zatem w powyższym przykładzie w zmiennej `foundPosition` zostanie zapisana wartość `0`, gdyż ciąg `być` znajduje się na samym początku łańcucha (`b` jest jego pierwszym znakiem).

Jak sobie zapewne przypominasz (patrz strona 567), metoda `indexOf()` działa w dokładnie taki sam sposób. Skoro obie metody działają tak samo, możesz się zastanawiać, po co w ogóle zawracać sobie głowę wyrażeniami regularnymi? Otóż, wyrażenia regularne mają tę zaletę, że pozwalają odnajdywać wzorce, czyli są w stanie realizować znacznie bardziej złożone i subtelniejsze porównania niż te, na które pozwala

metoda `index()` (gdyż ona poszukuje jedynie wystąpień konkretnego łańcucha). Przykładowo metody `indexOf()` można by użyć do odnalezienia konkretnego adresu strony — *http://www.missingmanuals.com/* — natomiast wyrażenia regularnego do odnalezienia dowolnego tekstu zapisanego w formacie przypominającym adres URL — czyli dokładnie tego, co chcielibyśmy zrobić w celu sprawdzenia, czy osoba przesyłająca komentarz na Twoim blogu podała adres swojej strony WWW.

Aby jednak opanować wyrażenia regularne, trzeba poznać tajemnicze symbole używane do ich tworzenia.

Tworzenie wyrażeń regularnych

Choć wyrażenia regularne mogą się składać z jednego lub kilku słów, zazwyczaj są w nich używane kombinacje liter i symboli specjalnych, definiujące wzorzec, którego będziemy szukać. Wyrażenia regularne udostępniają różne symbole oznaczające różne rodzaje znaków, na przykład pojedyncza kropka (`.`) reprezentuje jeden, dowolny znak, natomiast sekwencja `\w` oznacza dowolną literę lub cyfrę (jednak bez odstępów i symboli, takich jak `$` lub `%`). W tabeli 16.1 zawarto listę znaków najczęściej używanych w wyrażeniach regularnych.

Tabela 16.1. *K lka symboli najczęściej używanych w wyrażeniach regularnych*

Znak	Pasuje do
`.`	dowolnego znaku — litery, cyfry, odstępu oraz dowolnego symbolu.
`\w`	dowolnego znaku używanego w słowach, czyli należącego do zbioru zawierającego litery a – z, A – Z, cyfry 0 – 9 oraz znak podkreślenia.
`\W`	dowolnego znaku niebędącego znakiem używanym w słowach. Stanowi przeciwieństwo `\w`.
`\d`	dowolnej cyfry 0 – 9.
`\D`	dowolnego znaku z wyjątkiem cyfr. Stanowi przeciwieństwo `\d`.
`\s`	znaku odstępu, tabulacji, powrotu karetki, nowego wiesza.
`\S`	dowolnego znaku z wyjątkiem odstępu, tabulacji, powrotu karetki, nowego wiesza.
`^`	początku łańcucha znaków. Jest używany, kiedy chcemy mieć pewność, że przed poszukiwanym fragmentem pasującym do wzorca nie znajdują się żadne inne znaki.
`$`	końca łańcucha znaków. Można go używać, by upewnić się, że poszukiwany fragment jest umieszczony na samym końcu łańcucha. Przykładowo `/kom$/` pasuje do łańcucha znaków `"kom"`, ale tylko w przypadku, gdy są to trzy ostatnie znaki przeszukiwanego łańcucha. Innymi słowy, `/kom$/` pasuje do słowa „interkom", lecz nie „zakompleksiony".
`\b`	odstępu, początku łańcucha, końca łańcucha oraz dowolnego znaku, który nie jest ani literą, ani cyfrą, takiego jak +, = czy też '. Symbolu `\b` można używać do dopasowania początku lub końca słowa i to nawet w przypadku, gdy jest ono umieszczone na samym początku lub końcu łańcucha.
`[]`	dowolnego znaku podanego pomiędzy nawiasami. I tak wyrażenie `[aeiou]` będzie pasowało do dowolnej z tych liter. Aby podać zakres znaków, wystarczy użyć łącznika: wyrażenie `[a-z]` będzie pasowało do dowolnej małej litery, a wyrażenie `[0-9]` — do dowolnej cyfry (czyli odpowiada symbolowi `\d`).

Tabela 16.1. *Kilka symboli najczęściej używanych w wyrażeniach regularnych — ciąg dalszy*

Znak	Pasuje do
`[^]`	dowolnego znaku z wyjątkiem podanych w nawiasach. Przykładowo wyrażenie `[^aąeęioóuAĄEĘIOÓU]` będzie pasowało do dowolnego znaku z wyjątkiem samogłosek, a wyrażenie `[^0-9]` — do dowolnego znaku z wyjątkiem cyfr (czyli jest to odpowiednik symbolu `\D`).
`|`	znaku podanego przed kreską pionową lub za nią. Przykładowo wyrażenie `a|b` będzie pasowało do znaku `a` lub `b`, lecz nie do obu tych znaków równocześnie. (Przykład zastosowania tego symbolu został przedstawiony na stronie 581).
`\`	Służy do poprzedzania znaków specjalnych wyrażeń regularnych (takich jak `*`, `.`, `\`, `/`) i umożliwia odszukanie tego znaku w łańcuchu. Przykładowo kropka (`.`) oznacza w wyrażeniu regularnym dowolny znak, gdybyśmy jednak faktycznie chcieli odnaleźć kropkę w łańcuchu, musielibyśmy użyć wyrażenia regularnego o postaci `\. (czyli ukośnika i kropki)`.

Uwaga: Jeśli wszystkie rozważania o wyrażeniach „regularnych" powodują jedynie ból głowy, zapewne ucieszy Cię wiadomość, że w książce (na stronie 577) przedstawione zostały przykłady niektórych, najczęściej używanych wyrażeń regularnych, które możesz skopiować i wykorzystać w swoich własnych skryptach (bez zbytniego wgłębiania się w to, jak one działają).

Wyrażenia regularne są zagadnieniem, którego najlepiej można się nauczyć na przykładach; dlatego też w dalszej części tego punktu przedstawionych zostało kilka wyrażeń, które ułatwią ich poznanie i przyswojenie sobie tej wiedzy. Załóżmy, że chcemy odszukać pięć cyfr zapisanych jedna po drugiej — by na przykład sprawdzić, czy użytkownik podał w formularzu kod produktu.

1. **Dopasowanie jednej liczby.**

   Pierwszym krokiem będzie określenie, w jaki sposób możemy utworzyć wyrażenie regularne pasujące do jednej cyfry. Jeśli zajrzysz do tabeli 16.1, przekonasz się, że w wyrażeniach regularnych symbolem odpowiadającym cyfrze jest `\d`.

2. **Dopasowanie pięciu liczb podanych jedna za drugą.**

   Ponieważ symbol `\d` odpowiada pojedynczej cyfrze, najprostszym sposobem dopasowania pięciu kolejnych cyfr zapisanych jedna za drugą będzie użycie wyrażenia w postaci: `\d\d\d\d\d`. (Na stronie 574 został przedstawiony sposób zapisania tego samego wyrażenia w krótszej postaci).

3. **Dopasowanie tylko pięciu liczb podanych jedna za drugą.**

   Wyrażenie regularne jest jak rakieta precyzyjnie naprowadzana na cel — ustawia swój celownik na początek łańcucha, który ma dopasować. Dlatego też czasami otrzymujemy w efekcie dopasowanie, które jest fragmentem całego słowa lub zbiorem znaków. Wyrażenie regularne podane w poprzednim punkcie zostanie dopasowane do pierwszych pięciu cyfr, które za jego pomocą uda się odnaleźć. Przykładowo w ciągu `12345678998` zostanie ono dopasowane do fragmentu `12345`. Oczywiście, nie jest to pięciocyfrowy kod produktu, o jaki nam chodzi, dlatego też będziemy potrzebowali wyrażenia, które pozwoli odnaleźć ciąg składający się jedynie z pięciu cyfr.

   Symbol `\b` (nazywany także znakiem **granicy słowa**) odpowiada dowolnemu znakowi, który nie jest ani literą, ani cyfrą. A zatem nasze wyrażenie moglibyśmy zapisać w następującej postaci: `\b\d\d\d\d\d\b`. Moglibyśmy także użyć

symbolu ^, by dopasować wzorzec do początku łańcucha, oraz symbolu $, by dopasować go do końca łańcucha. Oba te symbole stają się bardzo przydatne, kiedy chcemy dopasować wyrażenie do całego łańcucha znaków. Gdyby na przykład w polu produktu ktoś wpisał łańcuch `kdfjalkjfajk 77777 jajajajajaj`, moglibyśmy go poprosić o jego poprawienie przed przesłaniem formularza. W końcu interesuje nas sam kod produktu, na przykład 23423 (bez żadnych dodatkowych znaków). W takim przypadku moglibyśmy użyć wyrażenia w postaci `^\d\d\d\d\d$`.

Uwaga: Ciąg składający się z pięciu cyfr przypomina nieco polski kod pocztowy, jednak w nim pierwsze dwie cyfry są oddzielone od trzech pozostałych znakiem łącznika, czyli 44-100. Wyrażenie regularne umożliwiające sprawdzanie kodów pocztowych zostało opisane na stronie 577.

4. **Zastosowanie wyrażenia w kodzie JavaScript.**

   Załóżmy, że już zapisałeś informacje wprowadzone przez użytkownika do zmiennej o nazwie `code`, a teraz chcesz sprawdzić, czy dane te są napisane w prawidłowym formacie, czy są pięcioma cyframi umieszczonymi jedna za drugą:

```
var codeTest = /^\d\d\d\d\d$/; // tworzymy testujące wyrażenie regularne
if (code.search(codeTest) == -1) {
  alert('To nie jest prawidłowy kod produktu');
} else {
  // dane są prawidłowe
}
```

Wyrażenie regularne zastosowane w tym przykładzie działa, jednak można uznać, że pięciokrotne wpisywanie symbolu `\d` to trochę dużo. Jak wyglądałoby wyrażenie pozwalające sprawdzić sto cyfr zapisanych jedna obok drugiej? Na szczęście JavaScript udostępnia kilka symboli pozwalających na sprawdzanie wielokrotnego występowania tego samego znaku. Zostały one przedstawione w tabeli 16.2. Symbole te umieszcza się bezpośrednio za sprawdzanym znakiem.

Tabela 16.2. *Znaki służące do wielokrotnego dopasowywania tego samego znaku lub wzorca*

Znak	Odpowiada
`?`	brakowi wystąpień lub dokładnie jednemu wystąpieniu poprzedzającego elementu wzorca (znaku lub grupy znaków). Oznacza to, że poprzedzający element wzorca jest opcjonalny, jeśli jednak wystąpi, może się pojawić tylko jeden raz; na przykład wzorzec `st?ernik` pasuje do słowa `"sernik"` lub `"sternik"`, lecz nie do słowa `"stternik"`.
`+`	jednemu powtórzeniu poprzedzającego elementu wzorca lub większej liczbie powtórzeń. Element ten musi wystąpić co najmniej raz.
`*`	brakowi powtórzenia poprzedzającego elementu wzorca lub dowolnej liczbie powtórzeń. Element ten jest opcjonalny i może się powtórzyć dowolną ilość razy; na przykład wyrażenie `.*` odpowiada dowolnemu łańcuchowi znaków, w tym łańcuchowi pustemu.
`{n}`	ściśle określonej liczbie powtórzeń poprzedzającego elementu wzorca; na przykład `\d{3}` odpowiada łańcuchowi składającemu się z trzech cyfr.
`{n,}`	Poprzedzający element wzorca musi się powtórzyć co najmniej *n* razy. Przykładowo wyrażenie `n{2,}` odpowiada słowu `"gehenna"` lub `"Achhhhhh!"`.
`{n,m}`	Poprzedzający element wzorca musi się powtórzyć co najmniej *n* razy, lecz nie więcej niż *m* razy. A zatem wyrażenie `\d{3,4}` będzie odpowiadało trzem lub czterem zapisanym kolejno cyfrom, lecz nie pięciu cyfrom.

Aby na przykład dopasować pięć cyfr, można użyć wyrażenia w postaci `\d{5}`, gdzie `\d` odpowiada jednej cyfrze, a wyrażenie `{5}` informuje interpreter JavaScriptu, że ma się ona powtórzyć pięć razy. Wyrażenie `\d{100}` będzie odpowiadać 100 cyfrom zapisanym jedna obok drugiej.

Przeanalizujmy kolejny przykład. Załóżmy, że chcemy znaleźć nazwę jakiegoś pliku GIF, zapisaną w łańcuchu znaków. Dodatkowo chcemy ją pobrać, aby na przykład wykorzystać w innym miejscu skryptu (możemy przy tym użyć metody `match()` opisanej na stronie 582). Innymi słowy, chodzi o odnalezienie dowolnego łańcucha znaków pasującego do podstawowego wzorca nazwy pliku z rozszerzeniem GIF, na przykład *logo.gif*, *banner.gif* bądź *ad.gif*. Oto czynności, jakie należy wykonać.

1. **Określ wspólną postać nazw plików.**

   Aby utworzyć wyrażenie regularne, musimy najpierw ustalić, jakiego wzorca znaków szukamy. Ponieważ interesują nas nazwy plików GIF, wiemy, że wszystkie będą się kończyć ciągiem znaków `.gif`. Innymi słowy, interesujący nas ciąg może się składać z dowolnej liczby znaków zakończonych ciągiem `.gif`. Jednak przed rozszerzeniem *.gif* powinien się znaleźć przynajmniej jeden znak: *a.gif* jest prawidłową nazwą pliku, natomiast *.gif* nie jest.

2. **Odszukaj rozszerzenia *.gif*.**

   Ponieważ chodzi nam o odszukanie ciągu znaków `.gif`, zatem możesz przypuszczać, że także wyrażenie regularne będzie go zawierało. Jednak, jak mogłeś się przekonać, analizując tabelę 16.1, w wyrażeniach regularnych znak kropki odpowiada dowolnemu znakowi. A zatem wyrażenie `.gif` pasowałoby — oczywiście — do ciągu `.gif`, lecz także do `tgif`. Kropka odpowiada dowolnemu znakowi, czyli oprócz faktycznego znaku kropki można ją także dopasować do litery `t` w ciągu `tgif`. Aby zbudować wyrażenie regularne z kropką, która zostanie potraktowana w sposób dosłowny, konieczne jest umieszczenie przed nią znaku odwrotnego ukośnika; dopiero wyrażenie `\.` zostanie zrozumiane jako „znajdź znak kropki". Oznacza to, że w celu odszukania ciągu `.gif` należy użyć wyrażenia regularnego w postaci: `\.gif`.

3. **Odszukaj dowolną liczbę znaków umieszczonych przed rozszerzeniem *.gif*.**

   Chodzi o dopasowanie takich łańcuchów znaków jak *a.gif*, *photo.gif*, ale nie *.gif* (bo on nie byłby prawidłową nazwą pliku). Aby znaleźć przynajmniej jeden znak, należy użyć wyrażenia `.+`, które trzeba rozumieć jako: „znajdź jeden znak (`.`) występujący raz lub więcej razy (`+`)". Jeśli jednak zastosujemy je do utworzenia wyrażenia regularnego o postaci `.+\.gif`, mogłoby się okazać, że będzie ono dopasowywane do większego fragmentu łańcucha znaków, a nie samej nazwy pliku: to wyrażenie regularne można dopasować do wszystkich znaków w łańcuchu. Jeśli na przykład będziemy dysponować łańcuchem `plik logo nazywa się logo.gif`, to wyrażenie regularne `.+\.gif` zostanie dopasowane do całego łańcucha, włącznie z nazwą pliku *logo.gif*. A przecież zależy nam wyłącznie na nazwie pliku *logo.gif*. W tym celu należy użyć sekwencji `\S`, oznaczającej wszelkie znaki z wyjątkiem znaków odstępu. Wyrażenie `\S+\.gif` należy rozumieć jako: znajdź przynajmniej jeden znak, który nie jest znakiem odstępu (dotyczy to także znaków tabulacji, spacji, powrotu karetki i nowego wiersza) i za którym występuje fragment `.gif`. W tym przypadku w naszym przykładowym łańcuchu znaków wyrażenie zostanie dopasowane wyłącznie do fragmentu *logo.gif*.

4. **Upewnij się, że za *.gif* nie będzie żadnych znaków.**

   Teraz wyrażenie regularne odnajdzie wyłącznie nazwę pliku… a może nie? Wyrażenia regularne potrafią być naprawdę trudne. Jeśli przykładowo użyjesz powyższego wyrażenia w łańcuchu znaków o postaci: "`adres e-mail alex.gifford@example`", dopasowany zostanie fragment `alex.gif`, który w ogóle nie stanowi nazwy pliku. Aby zapewnić, że takie problemy nie pojawią się, można także sprawdzić, czy za nazwą rozszerzenia (`gif`) nie występują żadne inne znaki. Można to zrobić, dodając na końcu wyrażenie odpowiadające granicy słowa (`\b`): na przykład: `\S+\.gif\b`. W ten sposób będziemy mieli pewność, że dopasowany fragment nie stanowi elementu dłuższego łańcucha, który przypadkowo zawiera sekwencję `.gif`.

5. **Zignoruj wielkość znaków.**

   Nasze wyrażenie regularne ma jeszcze jeden feler — znajduje wyłącznie pliki z rozszerzeniem *gif*. Przecież rozszerzenie *GIF* także jest prawidłowe, a mimo to, nasze wyrażenie regularne nie odnajdzie pliku o nazwie *logo.GIF*. Aby nasze wyrażenie regularne ignorowało wielkość liter, trzeba do niego dodać argument `i`:

   ```
   /\S+\.gif\b/i
   ```

   Koniecznie należy zwrócić uwagę, że argument `i` jest umieszczony poza prawym ukośnikiem wyznaczającym koniec wyrażenia regularnego.

6. **Zastosuj wyrażenia w skrypcie:**

   ```
   var testString = 'Plik logo.gif utworzony przez
   alex.gifford@witryna.com.pl'; // testowany łańcuch znaków
   var gifRegex = /\S+\.gif\b/i; // wyrażenie regularne
   var results = testString.match(gifRegex);
   var file = results[0]; // logo.gif
   ```

   Powyższy fragment kodu pobiera nazwę pliku umieszczoną w łańcuchu znaków. (Szczegółowe informacje na temat sposobu działania metody `match()` można znaleźć na stronie 582).

Grupowanie fragmentów wzorców

Wewnątrz wyrażeń regularnych można tworzyć podgrupy — w tym celu stosowane są nawiasy. Takie podgrupy są niezwykle przydatne, gdy chcemy wyszukiwać wielokrotnie powtarzające się fragmenty wzorca, korzystając przy tym ze znaków przedstawionych w tabeli 16.2.

Załóżmy, że chcemy sprawdzić, czy łańcuch zawiera fragment `Mar` lub `Marzec` — oba te fragmenty zaczynają się od liter `Mar`. Wiemy więc, co chcemy dopasować, jednak w tym przypadku nie możemy sprawdzać występowania samych liter `Mar`, gdyż można by je także dopasować do wielu innych słów, na przykład `Marcepan` lub `Margherita`. Dlatego też chcemy odnaleźć litery `Mar`, po których będzie umieszczony odstęp lub inna granica słowa (do tego służy znak `\b` opisany w tabeli 16.1) albo całe słowo `Marzec` zakończone granicą słowa. Innymi słowy, końcówka `zec` jest opcjonalna. A oto sposób na utworzenie takiego wyrażenia za pomocą nawiasów.

```
var sentence = 'Marzec jest najokrutniejszym z miesięcy.';
var aprMatch = /Mar(zec)?\b/;
if (sentence.search(aprMatch) != -1) {
  // znaleźliśmy Mar lub Marzec
} else {
  // nic nie znaleźliśmy
}
```

Wyrażenie regularne zastosowane w powyższym przykładzie — `/Mar(zec)?\b/` — sprawia, że pierwsze trzy litery (`Mar`) są obowiązkowe, natomiast podwzorzec — `(zec)` — jest opcjonalny (znak ? określa, że może on nie wystąpić lub wystąpić jeden raz). I wreszcie, umieszczony na samym końcu wzorca znak granicy słowa (`\b`) sprawia, że wyrażenie nie zostanie dopasowane do takich słów jak `Marcepan` lub `Margherita` (więcej informacji na temat stosowania podwzorców można znaleźć w ramce „Zastępowanie tekstów przy użyciu podwzorców" na stronie 586).

Aby to wyrażenie regularne było jeszcze bardziej niezawodne, można także na jego początku dodać sekwencję oznaczającą granicę słowa (`\b`):

```
var aprMatch = /\bMar(zec)?\b/;
```

Dodanie granicy słowa na początku wyrażenia zabezpiecza przed możliwością dopasowania go do takich łańcuchów znaków jak `zMarznięty` czy `ZMarł`. (Oczywiście raczej jest mało prawdopodobne, aby takie łańcuchy pojawiły się, ale lepiej się na wszelki wypadek zabezpieczyć).

Wskazówka: Obszerną bibliotekę wyrażeń regularnych można znaleźć na witrynie *www.regexlib.com*. Znajdują się w niej wyrażenia na niemal każdą okazję.

Przydatne wyrażenia regularne

Tworzenie wyrażeń regularnych może być sporym wyzwaniem. Nie tylko trzeba znać i rozumieć działanie poszczególnych symboli używanych w wyrażeniach, lecz także należy określić właściwy wzorzec. Trzeba na przykład uwzględnić, że numer telefonu stacjonarnego może się składać z siedmiu cyfr (1234567), lecz oprócz tego może być poprzedzony dwucyfrowym numerem kierunkowym (89-1234567). Aby ułatwić Ci rozpoczęcie stosowania wyrażeń regularnych, w tym punkcie rozdziału przedstawimy kilka przydatnych przykładów.

Uwaga: Jeśli nie masz ochoty samodzielnie wpisywać wszystkich przedstawianych tu wyrażeń, możesz je znaleźć w pliku *example_regex.txt* dostępnym w przykładach do książki, w katalogu *R16*. (Więcej informacji na temat pobierania przykładów znajdziesz na stronie 46).

Kod pocztowy

Kody pocztowe mają odmienną postać w różnych krajach, jednak w Polsce składają się z grupy dwóch cyfr oddzielonych łącznikiem od kolejnej grupy trzech cyfr. Oto wyrażenie regularne odpowiadające kodowi pocztowemu:

```
\d{2}-\d{3}
```

To wyrażenie regularne można podzielić na następujące fragmenty:

- `\d{2}` odpowiada grupie dwóch cyfr,
- `-\d{3}` odpowiada grupie trzech cyfr poprzedzonych łącznikiem.

Uwaga: Aby mieć pewność, że do wyrażenia zostanie dopasowany cały łańcuch znaków, należy je rozpocząć znakiem `^` i zakończyć znakiem `$`. By na przykład mieć pewność, że w polu do podania kodu pocztowego użytkownik wpisał sam kod, należy użyć wyrażenia regularnego w postaci: `/^\d{2}-\d{3}$/`. Zabezpieczy nas ono przed sytuacjami, gdy użytkownik wpisze w polu coś takiego jak łańcuch `"blabla 33-300 och ach"`.

Numer telefonu stacjonarnego

Polskie numery telefonów stacjonarnych mogą się składać z opcjonalnego, dwucyfrowego numeru kierunkowego oraz numeru lokalnego liczącego siedem cyfr; oto kilka przykładów zapisu takich numerów: (22) 555-2121, 22.555.2121 lub 22 555 2121. Wyrażenie regularne odpowiadające takim numerom ma następującą postać:

```
\(?(\d{2})\)?[ .-](\d{3})[ .-](\d{4})
```

Uwaga: Przykłady wyrażeń regularnych odpowiadających pełnym polskim numerom telefonów stacjonarnych i komórkowych można znaleźć na witrynie *http://www.regexplib.com*, a konkretnie na stronie *http://regexlib.com/Search.aspx?k=polish*.

Powyższe wyrażenie regularne wygląda na skomplikowane, jeśli jednak rozdzielimy je na fragmenty (i skorzystamy z dobrych wyjaśnień, takich jak zamieszczone poniżej), będzie można zrozumieć zasadę jego działania.

- `\(` odpowiada otwierającemu znakowi nawiasu. Ponieważ nawiasy są używane w wyrażeniach regularnych do tworzenia podwzorców grup, zatem nawias otwierający ma w nich specjalne znaczenie. By poinstruować interpreter JavaScriptu, że chodzi o faktyczny znak nawiasu otwierającego, musimy go poprzedzić odwrotnym ukośnikiem (przypomina to nieco poprzedzanie cudzysłowów i apostrofów opisane na stronie 61).
- Znak `?` oznacza, że nawias otwierający jest opcjonalny, dzięki czemu numery telefonów, w których numer kierunkowy nie jest zapisany w nawiasie, na przykład 22-555-2121, także zostaną dopasowane do wyrażenia.
- Fragment `\d{2}` odpowiada dwóm cyfrom.
- Fragment `\)?` odpowiada opcjonalnemu nawiasowi zamykającemu.
- Fragment `[ .-]` odpowiada znakowi odstępu, łącznikowi lub kropce. (Zwróć uwagę, że zazwyczaj kropkę trzeba poprzedzić znakiem odwrotnego ukośnika, by interpreter JavaScriptu potraktował ją dosłownie, a nie jako symbol specjalny odpowiadający *dowolnemu* znakowi; jednak kropka umieszczona wewnątrz nawiasów kwadratowych zawsze jest traktowana dosłownie).
- Fragment `\d{3}` odpowiada sekwencji trzech cyfr.
- Fragment `[ .-]` odpowiada znakowi odstępu, łącznikowi lub kropce.
- Fragment `\d{4}` odpowiadającym sekwencji czterech cyfr.

Adresy e-mail

Sprawdzanie poprawności adresu e-mail jest popularnym problemem, występującym podczas weryfikacji danych wpisywanych przez użytkownika w formularzu. Wiele osób stara się unikać podawania swojego adresu e-mail i wpisuje w formularzu dowolne dane, takie jak „to nie twój interes", bądź też popełnia proste błędy typograficzne (na przykład wpisuje adres w postaci *jan.kowalski@gmail.commm*). Poniższe wyrażenie regularne sprawdza, czy łańcuch znaków zawiera adres poczty elektronicznej zapisany w prawidłowej postaci:

```
[-\w.]+@([A-zO-9][-A-zO-9]+\.)+[A-z]{2,4}
```

Uwaga: To wyrażenie regularne nie sprawdza, czy podany adres należy do rzeczywistej osoby; sprawdza jedynie to, czy jest zapisany w prawidłowej postaci.

Powyższe wyrażenie można podzielić na fragmenty i zinterpretować w następujący sposób.

- Fragment `[-\w.]+` odpowiada łącznikowi, znakowi używanemu w słowach lub kropce, powtórzonym co najmniej jeden raz. A zatem będzie pasował do takich ciągów znaków jak „jan", „jan.kowalski" lub „jan-kowalski".
- Znak `@` jest znakiem `@` występującym we wszystkich adresach e-mail.
- Fragment `[A-zO-9]` odpowiada jednej literze lub cyfrze.
- Fragment `[-A-zO-9]+` odpowiada jednemu lub większej liczbie wystąpień łącznika, litery lub cyfry.
- Fragment `\.` odpowiada znakowi kropki, a zatem będzie pasować do kropki w nazwie *sawmac.com*. (*http://www.sawmac.com*).
- Znak + odpowiada jednemu lub większej liczbie powtórzeń umieszczonego wcześniej wzorca; w naszym przypadku te powtórzenia odnoszą się do grupy trzech fragmentów wzorca przedstawionych w trzech poprzednich punktach listy. Dzięki niemu wzorzec będzie pasować do nazw poddomen, takich jak *jan@mail.sawmac.com*.
- Ostatni fragment `[A-z]{2,4}` pasuje do dowolnej sekwencji składającej się z dwóch, trzech lub czterech liter, czyli odpowiada takim końcówkom jak *com* w *.com* lub *pl* w *.pl*.

Uwaga: Powyższe wyrażenie regularne nie będzie pasować do *wszystkich* adresów e-mail prawidłowych z technicznego punktu widzenia. Przykładowo adres `!#$%&'*+-/=?^_`.{|}~@example.com` jest prawidłowy, jednak powyższe wyrażenie nie uzna go za taki. Nasze wyrażenie zostało zaprojektowane z myślą o adresach, którymi użytkownicy będą się posługiwać. Jeśli jednak naprawdę zależy Ci na zachowaniu poprawności i dokładności, możesz wykorzystać poniższe wyrażenie regularne. Wpisz je całe w jednym wierszu:

```
/^[\w!#$%&\'*+\/=?^`{|}~.-]+@(?:[a-z\d][a-z\d-]*(?:\.[a-z\d][a-z\
↪d-]*)?)+\.(?:[a-z][a-z\d-]+)$/i
```

Daty

Daty można zapisywać na wiele różnych sposobów, takich jak 28/09/2009, 28-9-2009, 28 09 2009 bądź nawet 28.09.2009 (a są to formaty stosowane w Polsce; w innych krajach mogą być wykorzystywane zupełnie inne sposoby zapisu dat, na przykład w USA najczęściej spotkamy format 09/28/2009). Ponieważ osoby odwiedzające naszą witrynę mogą podawać daty w każdym z tych formatów, zatem musimy mieć jakiś sposób sprawdzenia, czy podana data została zapisana prawidłowo. (W ramce na stronie 595 dowiesz się, jak skonwertować każdy z tych zapisów na jeden, standardowy format, dzięki czemu będziemy mogli mieć pewność, że wszystkie daty wpisane w formularzu są zapisane prawidłowo).

Oto wyrażenie regularne sprawdzające poprawność zapisu daty:

```
([0123]?\d)[-\/ .]([01]?\d)[-\/ .](\d{4})
```

- Nawiasy — `()` — łączą dwa podane wewnątrz fragmenty wyrażenia w jedną grupę. Razem tworzą one numer dnia.
- Fragment `[0123]?` odpowiada cyfrom 0, 1, 2 oraz 3, które mogą zostać pominięte lub wystąpić jeden raz. Ponieważ w miesiącu nie ma dnia 40., zatem pierwszą cyfrą numeru dnia może być tylko jedna z cyfr od 0 do 3. Ten wzorzec jest opcjonalny, gdyż użytkownik może pominąć pierwszą cyfrę numeru dnia, wpisując na przykład 9 zamiast 09.
- Symbol `\d` odpowiada dowolnej cyfrze.
- Fragment `[-\/ .]` odpowiada łącznikowi, znakowi ukośnika, znakowi odstępu lub kropki. Są to dozwolone separatory oddzielające numer dnia od miesiąca, na przykład: 10/11 lub 10.11 lub 10-11.
- Nawiasy — `()` — wyznaczają kolejny podwzorzec przeznaczony do pobrania numeru miesiąca.
- Fragment `[01]?` odpowiada jednej z cyfr 0 lub 1, która może zostać pominięta lub pojawić się jeden raz. Cyfra ta określa pierwszą cyfrę miesiąca. Oczywiście, nie może być większa od 1, gdyż nie mamy miesiąca na przykład 22. Co więcej, wszystkie miesiące od stycznia do września można zapisać przy użyciu jednej cyfry — na przykład 9 zamiast 09. Właśnie z tego względu ta pierwsza cyfra jest opcjonalna.
- Symbol `\d` odpowiada dowolnej cyfrze.
- Fragment `[-\/ .]` został opisany powyżej.
- Kolejna para nawiasów pozwala pobrać numer roku.
- Fragment `\d{4}` pobiera sekwencję znaków składającą się z czterech cyfr, na przykład 1908 lub 2880.

Uwaga: Choć oczywiście można napisać mechanizm weryfikacji poprawności informacji podawanych w formularzu, korzystając z wyrażeń regularnych prezentowanych w tym rozdziale, jednak warto wiedzieć, że nie jest to jedyne dostępne rozwiązanie. Ktoś już ten problem rozwiązał, dlaczego zatem nie skorzystać z owoców czyjejś ciężkiej pracy, a samemu zająć się rozwiązywaniem innych problemów? Łatwym sposobem zagwarantowania, że osoby odwiedzające naszą stronę będą wpisywały w formularzu poprawne informacje, może być użycie wtyczki jQuery Form Validation opisanej na stronie 317.

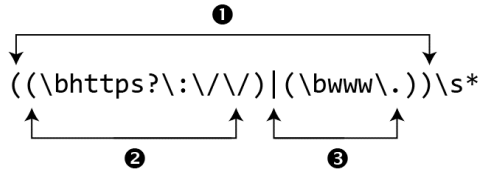
Adresy stron WWW

Sprawdzanie adresów URL jest przydatne, jeśli prosimy użytkownika o podanie adresu swojej strony i chcemy mieć pewność, że został podany, bądź też chcemy przeanalizować jakiś tekst i wyszukać w nim wszystkie podane adresy. Oto proste wyrażenie regularne reprezentujące adres URL:

```
((\bhttps?:\/\/)|(\bwww\.))\S*
```

Jest dosyć trudne, gdyż użyto w nim kilku par nawiasów, definiując przy ich użyciu kilka grup. Na rysunku 16.4 pokazano te grupy, co pomoże zrozumieć konstrukcję powyższego wyrażenia. Jedna para nawiasów (oznaczona cyfrą 1) otacza dwie pozostałe grupy (oznaczone jako 2 i 3). Znak `|` umieszczony pomiędzy dwoma pozostałymi grupami oznacza logiczne „lub". Innymi słowy, łańcuch znaków musi pasować do podwyrażenia 2 lub 3.

- `(` oznacza początek zewnętrznej grupy (na rysunku 16.4 została ona oznaczona numerem 1).
- `(` oznacza początek wewnętrznej grupy (na rysunku 16.4 została ona oznaczona numerem 2).
- `\b` odpowiada początkowi słowa.
- `http` odpowiada początkowi kompletnego adresu URL, zaczynającego się od oznaczenia protokołu — `http`.
- `s?` to opcjonalna litera `s`. Strony WWW mogą być przesyłane przy użyciu bezpiecznych połączeń, a zatem adres URL może także rozpoczynać się od oznaczenia protokołu `https`.
- Fragment `:\/\/` odpowiada sekwencji znaków `://`. Jednak ze względu na to, że znak ukośnika ma specjalne znaczenie w wyrażeniach regularnych, aby został potraktowany dosłownie, trzeba go poprzedzić odwrotnym ukośnikiem.
- `)` oznacza koniec pierwszej grupy wewnętrznej (na rysunku 16.4 została ona oznaczona numerem 2). Cała ta grupa będzie odpowiadać oznaczeniu protokołu `http` lub `https`.
- Znak `|` pozwala dopasować wyrażenie podane z jego lewej lub prawej strony (2 lub 3 na rysunku 16.4).
- `(` oznacza początek drugiej grupy wewnętrznej (na rysunku 16.4 została ona oznaczona numerem 3).
- `\b` odpowiada początkowi słowa.
- Fragment `www\.` odpowiada sekwencji znaków `www.`.



Rysunek 16.4. *Można grupować wyrażenia przy użyciu nawiasów oraz odszukać jedno lub drugie z wyrażeń, umieszczając pomiędzy nimi znak | (pionową kreskę). Przykładowo zewnętrzne wyrażenie (1) zostanie dopasowane do dowolnego tekstu pasującego do wyrażenia 2 lub 3*

- `)` oznacza koniec drugiej grupy wewnętrznej (na rysunku 16.4 została ona oznaczona numerem 3). Pozwoli ona dopasować wyrażenie do adresu URL, który nie zaczyna się od określenia protokołu `https`, lecz od sekwencji znaków `www`.
- `)` oznacza koniec grupy zewnętrznej (na rysunku 16.4 została ona oznaczona numerem 1). Do tej pory wyrażenie regularne będzie pasować do łańcucha znaków rozpoczynającego się od `http://`, `https://` lub `www`.
- Fragment `\S*` odpowiada dowolnej liczbie znaków innych niż odstępy.

To wyrażenie regularne nie jest doskonałe (na przykład można go dopasować do zupełnie bezsensownych adresów URL, takich jak *http://#$*%&*@**), jednak jest stosunkowo proste i można je prawidłowo dopasować do rzeczywistych adresów, takich jak *http://www.sawmac.com/missing/js/index.html*.

Wskazówka: Aby sprawdzić, czy łańcuch znaków zawiera wyłącznie adres URL (czy przed nim lub za nim nie wpisano żadnych niepotrzebnych znaków), na początku wyrażenia należy umieścić znak `^`, na jego końcu znak `$`, a ze środka wyrażenia usunąć znaki `\b`. W efekcie uzyskamy wyrażenie w postaci `^((https?:\/\/)|(www\.))\S*$`.

Dopasowywanie wzorców

Metoda `search()` opisana na stronie 571 jest jednym ze sposobów sprawdzenia, czy łańcuch zawiera konkretny wzorzec wyrażenia regularnego. Kolejną metodą, która na to pozwala, jest metoda `match()`. Można jej używać nie tylko w celu sprawdzenia, czy łańcuch zawiera konkretny wzorzec, lecz także po to, by pobrać fragment paskujący do tego wzorca i wykorzystać go w dalszej części skryptu. Przykładowo załóżmy, że na naszej stronie znajduje się formularz z polem służącym do podawania komentarzy. Być może chcielibyśmy sprawdzić, czy w tym komentarzu nie został podany adres URL, a gdyby został, chcemy go jakoś pobrać i przetworzyć.

Poniższy fragment kodu odnajduje i pobiera adres URL z łańcucha znaków:

```
// sprawdzany łańcuch zawierający adres URL
var text='moja witryna to www.missingmanuals.com';
// utworzenie wyrażenia regularnego
var urlRegex = /((\bhttps?:\/\/)|(\bwww\.))\S*/
// odnalezienie dopasowania wyrażenia w łańcuchu znaków
var url = text.match(urlRegex);
alert(url[0]); // www.missingmanuals.com
```

W powyższym fragmencie kodu na początku tworzymy zmienną i zapisujemy w niej łańcuch znaków zawierający adres URL *www.missingmanuals.com*. Zmienną utworzyliśmy tutaj wyłącznie w celach testowych (abyś mógł się przekonać, do czego służy metoda `match()`). Gdybyśmy faktycznie chcieli sprawdzić zawartość pola tekstowego, powinniśmy posłużyć się następującym wywołaniem:

```
var text = $('#comments').val();
```

Następnie piszemy wyrażenie regularne, które zostanie dopasowane do adresu URL (szczegóły jego konstrukcji zostały opisane na stronie 581). Następnie wywołujemy metodę `match()` na rzecz testowanego łańcucha znaków. Metoda `match()` jest metodą obiektu łańcucha znaków, dlatego też jej wywołanie rozpoczyna się od podania nazwy zmiennej zawierającej łańcuch; po tej nazwie należy zapisać kropkę oraz samą metodę `match()`. W wywołaniu tej metody przekazywane jest wyrażenie regularne.

W powyższym przykładzie w zmiennej `url` zapisywane są wyniki dopasowania. Jeśli nie uda się znaleźć fragmentu łańcucha pasującego do wyrażenia regularnego, wynikiem metody `match()` będzie specjalna wartość języka JavaScript — `null`. Jeśli jednak uda się znaleźć dopasowanie, metoda ta zwraca tablicę — jej pierwszym elementem jest dopasowany łańcuch znaków. W naszym przykładzie zmienna `url` zawiera tablicę, której pierwszym elementem jest dopasowany fragment sprawdzanego łańcucha znaków. Konkretnie rzecz biorąc, `url[0]` zawiera łańcuch znaków `www.missingmanuals.com` (więcej informacji na temat tablic można znaleźć na stronie 77).

Uwaga: W języku JavaScript `null` jest wartością traktowaną tak samo jak `false`; a zatem to, czy dopasowanie się udało, czy nie, można sprawdzić w następujący sposób:

```
var url = text.match(urlRegex);
if (! url) {
  // brak dopasowania
} else {
  // jest dopasowanie
}
```

Metoda `match()` dostarcza także pewnych dodatkowych informacji. Oprócz tablicy, której pierwszym elementem jest cały dopasowany łańcuch znaków, metoda `match()` dodaje także do tej tablicy właściwość `index`, określającą położenie początku dopasowanego fragmentu w całym łańcuchu. Oto przykład:

```
var string = 'Być albo nie być';
var regex = /albo/;
var result = string.match(regex);
alert(result.index); // wyświetla liczbę 4
```

Zmienna `result` zawiera wartości zwrócone przez metodę `match()`. W tym przypadku jest to tablica, a jej pierwszym i jedynym elementem jest dopasowany łańcuch znaków — element `result[0]` zawiera łańcuch `albo`. Metoda `match()` dodała także do tej tablicy właściwość `index`. W powyższym przykładzie będzie ona mieć wartość 4, gdyż słowo „albo" zaczyna się od czwartego znaku łańcucha (trzeba pamiętać, że w języku JavaScript elementy tablic są liczone od 0). Jest to taka sama wartość, którą zwróciłoby wywołanie metody `search()` (opisanej na stronie 571); a działanie tej metody przypomina metodę `indexOf()` (patrz strona 567), która zwraca indeks pierwszego znaku podanego fragmentu, odnalezionego w większym łańcuchu znaków. Dodatkowo jest to ta sama wartość, której należałoby użyć w metodzie `slice()` (patrz strona 569), aby pobrać fragment z większego łańcucha znaków.

Jeśli w wyrażeniu regularnym zostały umieszczone podwzorce zdefiniowane przy użyciu nawiasów `( )` (zgodnie z informacjami podanymi w ramce na stronie 586), metoda `match()` umieści w zwracanej tablicy także dodatkowe elementy. Pierwszym elementem zwracanej tablicy zawsze jest dopasowany fragment łańcucha znaków; jeśli jednak wyrażenie regularne będzie zawierać podwzorce, w każdym dodatkowym elemencie zwracanej tablicy zostanie zapisany fragment łańcucha dopasowany do kolejnego podwzorca.

Dopasowywanie każdego wystąpienia wzorca

Metoda `match()` może działać na dwa różne sposoby, zależnie od tego, jak zostało przygotowane wyrażenie regularne. W przedstawionym wcześniej przykładzie wywołanie metody zwracało tablicę zawierającą pierwszy odnaleziony fragment łańcucha pasujący do wyrażenia oraz właściwość `index` określającą położenie tego fragmentu w łańcuchu (a gdyby wyrażenie zawierało podwzorce, w tablicy mogłyby także znaleźć się dodatkowe elementy). A zatem, gdybyśmy dysponowali długim łańcuchem znaków, zawierającym wiele adresów URL, zostałby zwrócony tylko pierwszy z nich. Jednak można także skorzystać z opcji wyszukiwania globalnego, która umożliwi pobranie wszystkich dopasowań w łańcuchu.

Opcję wyszukiwania globalnego włącza się, dodając na końcu tworzonego wyrażenia literę `g` (podobnie jak było podczas tworzenia wyrażeń, w których nie jest uwzględniana wielkość liter, opisanych na stronie 576):

```
var urlRegex = /((\bhttps?:\/\/)|(\bwww\.))\S*/g
```

Warto zwrócić uwagę, że litera `g` została umieszczona poza znakiem ukośnika kończącym wyrażenie. Takie wyrażenie regularne wykonuje przeszukanie globalne; podczas użycia go w metodzie `match()` spróbuje odnaleźć każdy fragment łańcucha pasujący do podanego wyrażenia i zwróci tablicę zawierającą wszystkie. Innymi słowy, jest to doskonały sposób na odnalezienie każdego adresu URL umieszczonego na przykład we wpisie w blogu bądź też każdego wystąpienia konkretnego słowa w długim bloku tekstu.

Poniżej przedstawiony został przykład z początku tej strony zmodyfikowany tak, by korzystał z przeszukiwania globalnego:

```
// utworzenie zmiennej zawierającej tekst z adresami URL
var text='istnieje wiele wspaniałych witryn, takich jak
www.missingmanuals.com oraz http://www.oreilly.com';
// utworzenie wyrażenia regularnego z opcją przeszukiwania globalnego
var urlRegex = /((\bhttps?:\/\/)|(\bwww\.))\S*/g
// znalezienie dopasowań w sprawdzanym łańcuchu znaków
var url = text.match(urlRegex);
alert(url[0]); // www.missingmanuals.com
alert(url[1]); // http //www.oreilly.com
```

Liczbę odnalezionych dopasowań można określić przy użyciu właściwości `length` zwróconej tablicy, na przykład `url.length`. W powyższym przykładzie właściwość ta ma wartość `2`, gdyż w sprawdzanym łańcuchu znaków udało się odnaleźć dwa adresy URL. Co więcej, do każdego z tych łańcuchów można się odwołać przy użyciu odpowiedniego indeksu tablicy (zgodnie z informacjami podanymi na stronie 80); a zatem, w naszym przykładzie `url[0]` zawiera pierwszy odnaleziony adres URL, a `url[1]` — drugi.

Trzeba pamiętać, że w przypadku przeszukiwania globalnego metoda `match()` nie zwraca indeksu określającego położenie początku odszukanego fragmentu łańcucha znaków, nie zwraca także żadnych informacji o podwzorcach. W tym przypadku metoda zwraca wyłącznie tablicę zawierającą każde dopasowanie wyrażenia odnalezione w przeszukiwanym łańcuchu znaków.

Zastępowanie tekstów

Wyrażeń regularnych można także używać do zastępowania fragmentów łańcuchów znaków. Załóżmy, że mamy łańcuch znaków zawierający datę zapisaną w postaci 28/10/2014. Chcemy jednak, by na naszej stronie daty były prezentowane w formacie 28.10.2014. Taką zmianę możemy wykonać właśnie przy użyciu metody `replace()`. Oto składnia jej wywołania:

```
lancuch.replace(wyrazenieRegularne, 'zamiennik');
```

Metoda `replace()` wymaga przekazania dwóch argumentów. Pierwszym jest wyrażenie regularne, które chcemy odszukać w łańcuchu, natomiast drugim — łańcuch, którym chcemy zastąpić fragment pasujący do podanego wyrażenia regularnego. Aby zatem zmienić format zapisu dat z 28/10/2011 na 28.10.2011, moglibyśmy użyć następującego fragmentu kodu:

```
1  var date='28/10/2014'; // łańcuch znaków
2  var replaceRegex = /\//g // wyrażenie regularne
3  var date = date.replace(replaceRegex, '.'); // zastępujemy / znakiem .
4  alert(date); // 28.10.2014
```

W wierszu 1. tworzona jest zmienna zawierająca datę zapisaną jako `'28/10/2014'`. W rzeczywistym programie łańcuch ten mógłby zostać podany przez użytkownika w polu formularza. W wierszu 2. tworzone jest wyrażenie regularne. Dwa skrajne znaki ukośnika (`/`) określają odpowiednio jego początek i koniec. Umieszczony wewnątrz wyrażenia symbol `\/` odpowiada potraktowanemu dosłownie znakowi ukośnika. Litera `g` umieszczona na samym końcu wyrażenia sprawia, że będzie ono operowało globalnie, czyli zostanie zastąpiony każdy występujący w nim znak ukośnika. Gdybyśmy pominęli tę literę, zostałoby zastąpione tylko pierwsze wystąpienie ukośnika, a przez to ostatecznie data miałaby postać `'28.10/2014'`. Faktyczna zamiana jest wykonywana w wierszu 3. i to właśnie ona zmienia znaki ukośnika na kropki i zapisuje wynik w nowej zmiennej `date`. W ostatnim wierszu wyświetlamy odpowiednio sformatowaną datę — `'28.10.2014'` — w okienku informacyjnym.

Testowanie wyrażeń regularnych

Przykładowe wyrażenia regularne zostały zamieszczone w pliku *example_regex.txt*, umieszczonym w przykładach dołączonych do książki w katalogu *testy*. Dodatkowo w katalogu *testy* znajduje się także plik *regex_tester.html*. Można go wyświetlić w przeglądarce i spróbować swych sił podczas samodzielnego tworzenia wyrażeń regularnych (patrz rysunek 16.5). Wystarczy wpisać testowy łańcuch znaków w polu *Testowy łańcuch znaków*, a w polu poniżej podać wyrażenie regularne (należy przy tym pominąć znaki ukośnika podawane na początku i końcu wyrażenia w kodzie JavaScript i wpisać jedynie sam wzorzec). Oprócz tego można także wybrać metodę, której chcesz używać — *Search*, *Match* lub *Replace* — oraz dodatkowe opcje: ignorowanie wielkości liter i wyszukiwanie globalne. Aby sprawdzić wyniki, wystarczy kliknąć przycisk *Wykonaj*.

Rysunek 16.5. *Ta strona WWW, dostępna w przykładach do książki, pozwala na testowanie wyrażeń regularnych przy użyciu różnych metod JavaScriptu — na przykład search() lub match() — oraz z wykorzystaniem różnych opcji dodatkowych, takich jak ignorowanie wielkości liter oraz wyszukiwanie globalne*

PORADNIA DLA ZAAWANSOWANYCH

Zastępowanie tekstów przy użyciu podwzorców

Zastępowanie fragmentów pasujących do wzorca (takich jak ukośniki w dacie 28/10/2008) innym łańcuchem znaków (na przykład znakiem kropki) nie jest jedyną możliwością, jaką daje metoda `replace()`. Oprócz tego, za jej pomocą można także zapamiętywać *podwzorce* podane w wyrażeniu regularnym i używać ich podczas zastępowania tekstów. Podwzorzec jest dowolnym fragmentem wyrażenia regularnego zapisanym w nawiasach, na przykład podwzorcem jest `(zec)` w wyrażeniu regularnym `/Mar(zec)?\b/`.

Metoda `replace()` zastosowana w przykładzie ze strony 585 pozwalała zmienić format zapisu daty z 28/10/2008 na 28.10.2008. Co jednak moglibyśmy zrobić w przypadku, gdybyśmy chcieli zapisać w tym samym formacie 28.10 2008 daty wpisywane jeszcze inaczej, na przykład w postaci: 28 10 2008 lub 28-10-2008? Zamiast używać długiego kodu JavaScript sprawdzającego i zastępującego wszystkie wystąpienia odstępów, łączników i ukośników, można utworzyć ogólne wyrażenie regularne pasujące do każdego z tych formatów zapisu dat:

```
var date='28-10-2008';
var regex = /([0123]?\d)
➥[-\/ .]([01]?\d)[-\/ .](\d{4})/;
date = date.replace(regex, '$1.$2.$3');
```

W tym przykładzie skorzystano z wyrażenia regularnego do odnajdywania dat, opisanego na stronie 580. Należy w nim zwrócić uwagę na fragmenty umieszczone w nawiasach, takie jak `([01]?\d)`. Każdy z tych podwzorców pasuje do konkretnego fragmentu daty. Metoda `replace()` pamięta fragmenty łańcucha dopasowane do każdego z podwzorców i pozwala wykorzystać je w łańcuchu zamiennika. W powyższym przykładzie łańcuch zamiennika ma postać: `'$1.$2.$3'`. Znak dolara (`$`) wraz z zapisaną za nim liczbą reprezentuje dopasowanie do podwzorca. Przykładowo `$1` reprezentuje fragment dopasowany do pierwszego podwzorca, czyli numer dnia. A zatem, użyty tu łańcuch zamiennika należy interpretować w następujący sposób: „na początku umieść fragment dopasowany do pierwszego podwzorca, następnie zapisz kropkę, fragment dopasowany do drugiego podwzorca, kolejną kropkę i w końcu fragment dopasowany do trzeciego podwzorca”.

Stosowanie liczb

Liczby są bardzo ważnym elementem programowania. Pozwalają na realizację takich zadań jak obliczanie łącznej ceny produktu, określanie odległości między dwoma punktami bądź symulowanie rzutu kostką poprzez wygenerowanie liczby losowej z zakresu od 1 od 6. Język JavaScript udostępnia wiele sposobów posługiwania się liczbami.

Zamiana łańcucha znaków na liczbę

W tworzonej zmiennej liczbę można zapisać w następujący sposób:

```
var a = 3.25;
```

Może się jednak zdarzyć i tak, że łańcuch będzie reprezentacją jakiejś liczby. Jeśli użyjemy metody `prompt()` (opisanej na stronie 74) do pobrania danych od użytkownika, to nawet jeśli wpisze on 3.25, i tak uzyskamy `'3.25'` (czyli łańcuch znaków), a nie `3.25` (czyli liczbę). Bardzo często ta różnica typu nie przysparza większych problemów, gdyż interpreter JavaScriptu zazwyczaj konwertuje łańcuchy na liczby, jeśli zauważy, że program tak chce je traktować. Oto przykład:

```
var a = '3';
var b = '4';
alert(a*b); //12
```

W tym przykładzie, pomimo faktu, że zmienne `a` i `b` zawierają łańcuchy znaków, interpreter JavaScriptu skonwertuje je do postaci liczb w celu wykonania operacji mnożenia (`3 x 4`) i dzięki temu uzyskamy wynik `12`.

Gdybyśmy jednak zastosowali operator dodawania (+), interpreter JavaScriptu nie wykonałby tej konwersji, a my uzyskalibyśmy zupełnie inny wynik:

```
var a = '3';
var b = '4';
alert(a+b); //'34'
```

Także w tym przypadku obie zmienne — `a` i `b` — są łańcuchami znaków; jednak operator + nie służy wyłącznie do wykonywania dodawania arytmetycznego, pozwala także na łączenie (konkatenację) dwóch łańcuchów znaków (patrz strona 69). A zatem, zamiast dodania liczb `3 + 4` i uzyskania wyniku `7`, w tym przypadku uzyskamy połączenie dwóch łańcuchów, czyli `'34'`.

Gdy pojawia się konieczność konwersji łańcucha znaków na liczbę, JavaScript udostępnia kilka sposobów.

- Metoda `Number()` konwertuje na liczbę dowolny, przekazany w jej wywołaniu łańcuch znaków, na przykład:

  ```
  var a = '3';
  a = Number(a); // teraz a ma wartość 3
  ```

  A zatem problem dodawania dwóch łańcuchów zawierających liczby można rozwiązać w następujący sposób:

  ```
  var a = '3';
  var b = '4';
  var total = Number(a) + Number(b);  //7
  ```

Szybszą metodą jest jednak zastosowanie operatora +, który robi dokładnie to samo, co metoda `Number()`. Wystarczy umieścić go przed nazwą zmiennej zawierającej łańcuch znaków, a interpreter JavaScriptu skonwertuje ją do postaci liczby.

```
var a = '3';
var b = '4';
var total = +a + +b; //7
```

Oba te rozwiązania mają wadę. Gdy łańcuch znaków zawiera cokolwiek innego niż liczba — pojedynczą kropkę, znak – lub + na początku — w efekcie konwersji uzyskamy wartość `NaN` (jest to specjalna wartość języka JavaScript, która oznacza „to nie jest liczba"; z ang: *Not a Number*; więcej informacji na jej temat można znaleźć na stronie 589).

- Także funkcja `parseInt()` konwertuje łańcuch znaków na liczbę. Jednak, w odróżnieniu od `Number()`, funkcja `parseInt()` spróbuje wykonać konwersję nawet wtedy, gdy podany łańcuch znaków zawiera jakieś litery, o ile tylko zaczyna się on od cyfr. Funkcja może się przydać, gdy musimy skonwertować na liczbę łańcuch znaków w takiej postaci jak `'20 lat'`, podany na przykład jako odpowiedź na pytanie zadane na stronie:

  ```
  var age = '20 lat';
  age = parseInt(age,10);  //20
  ```

  Funkcja `parseInt()` szuka cyfry bądź znaków + lub - umieszczonych na początku łańcucha znaków, a następnie poszukuje kolejnych cyfr i tak dalej, aż do momentu odnalezienia pierwszego znaku, który nie jest cyfrą. A zatem w powyższym przykładzie wywołanie funkcji `parseInt()` zwróci liczbę `20` i zignoruje umieszczone za nią słowo `' lat'`.

  Oprócz łańcucha znaków, w wywołaniu metody `parseInt()` przekazywany jest także drugi argument, nazywany podstawą i określający system liczbowy, którego metoda ma używać. W przeważającej większości przypadków będziemy używali systemu dziesiątkowego, czyli tego, którym zazwyczaj się posługujemy. Jednak nie jest to jedyny system zapisu liczb; stosowane są także systemy liczbowe o podstawie 8, czyli *ósemkowy*, oraz o podstawie 16, czyli *szesnastkowy*.

Uwaga: Liczby ósemkowe zawsze zaczynają się od 0, może się zatem zdarzyć, że argumentem wywołania metody `parseInt()` będzie łańcuch znaków "010", a ponieważ taki łańcuch zaczyna się od zera, zatem przeglądarka uzna, że jest to liczba ósemkowa i zwróci dziwne wyniki. W tym przypadku ósemkowa liczba 10 odpowiada liczbie 8 w systemie dziesiątkowym. Niektóre przeglądarki w takich sytuacjach domyślnie uznają, że konwertowane liczby są zapisane w systemie ósemkowym, natomiast inne (jak również standard języka JavaScript) domyślnie używają systemu dziesiątkowego. Oznacza to, że różne przeglądarki mogą zwracać różne wyniki.

W efekcie najbezpieczniejszym rozwiązaniem jest jawne określanie, że metoda `parseInt()` ma używać systemu dziesiątkowego:

```
var age = '010 lat';
age = parseInt(age, 10); // 10
```

- Funkcja `parseFloat()` działa podobnie do funkcji `parseInt()`, przy czym jest używana w przypadkach, gdy liczba zapisana w łańcuchu znaków może zawierać

kropkę dziesiętną. Gdybyśmy na przykład dysponowali łańcuchem znaków `'4.5 hektarów'`, z wykorzystaniem funkcji `parseFloat()` moglibyśmy go przekształcić na liczbę z miejscami dziesiętnymi:

```
var space = '4.5 hektarów';
space = parseFloat(space); // 4.5
```

Gdybyśmy do przetworzenia tego samego łańcucha znaków użyli funkcji `parseInt()`, uzyskalibyśmy liczbę `4`, gdyż funkcja ta zwraca jedynie liczby całkowite.

Wybór jednej z tych metod jest zależny od sytuacji. Jeśli chcemy dodać do siebie dwie liczby zapisane w postaci łańcuchów znaków, możemy w tym celu wykorzystać metodę `Number()` lub operator +. Jeśli jednak chcemy pobrać liczbę zapisaną w łańcuchu znaków, który może także zawierać jakieś inne znaki, takim jak `'200px'` lub `'1.5em'`, musimy użyć funkcji `parseInt()` w przypadku pobierania wartości całkowitych lub `parseFloat()` w przypadku pobierania liczb z częścią ułamkową (takich jak `1.5`).

Sprawdzanie występowania liczb

Podczas korzystania z języka JavaScript do przetwarzania danych wpisywanych przez użytkowników czasami może się pojawić konieczność sprawdzenia, czy podane informacje są odpowiedniego typu. Jeśli na przykład prosimy o podanie roku urodzenia, chcemy mieć pewność, że została podana liczba całkowita. Podobnie w przypadku przeprowadzania operacji arytmetycznych, bo jeśli dane, na których one operują, nie będą liczbami, mogą wystąpić błędy lub nawet awaria samego skryptu.

Do sprawdzenia, czy łańcuch znaków można przekształcić na liczbę, służy metoda `isNaN()`. Metoda ta pobiera jeden argument będący łańcuchem znaków i sprawdza, czy „nie jest on liczbą". Jeśli łańcuch zawiera cokolwiek oprócz znaku plusa (lub minusa w przypadku liczb ujemnych) i umieszczonych za nim cyfr oraz opcjonalnej części dziesiętnej, funkcja uzna, że „nie jest on liczbą". A zatem łańcuch `'-23.45'` jest liczbą, a `'24 piksele'` nie jest. Funkcja zwraca wartość `true` (jeśli łańcuch nie jest liczbą) oraz `false` (jeśli jest liczbą). Funkcji tej można używać wewnątrz instrukcji warunkowej, co pokazano na poniższym przykładzie:

```
var x = '10'; // jest liczbą
if (isNaN(x)) {
  // ten fragment nie zostanie wykonany
} else {
  // ten fragment zostanie wykonany
}
```

Zaokrąglanie liczb

Język JavaScript zapewnia także możliwość zaokrąglania wartości ułamkowych do liczb całkowitych — na przykład można zaokrąglić `4.5` do `5`. To bardzo przydatne, gdy wykonujemy obliczenia, których wynik ma być liczbą całkowitą. Przykładowo załóżmy, że używamy języka JavaScript do dynamicznego wyznaczenia wysokości elementu `<div>` na stronie, w zależności od wysokości okna przeglądarki. Innymi

słowy, wysokość elementu jest uzależniona od wysokości okna przeglądarki. Wszystkie wykonywane obliczenia mogą zwrócić wynik z częścią ułamkową (na przykład 300.25), jednak, zważywszy na fakt, że nie ma czegoś takiego jak 0,25 piksela, musimy taki wynik zaokrąglić do najbliższej liczby całkowitej (na przykład 300).

Liczbę można zaokrąglić przy użyciu metody round() obiektu Math. Składnia jej wywołania wygląda następująco:

```
Math.round(liczba)
```

W wywołaniu tej metody przekazywana jest liczba (bądź też zmienna zawierająca jakąś wartość liczbową), a sama metoda zwraca wartość całkowitą. Jeśli przekazana wartość zawiera część dziesiętną mniejszą od 0,5, zostanie ona zaokrąglona w dół. Jeśli natomiast część dziesiętna liczby jest równa lub większa od 0,5, liczba zostanie zaokrąglona w górę, na przykład wartość 4.4 zostanie zaokrąglona do 4, a 4.5 do 5. Oto przykład:

```
var decimalNum = 10.25;
var roundedNum = Math.round(decimalNum); // 10
```

Uwaga: JavaScript udostępnia także dwie inne metody służące do zaokrąglania: Math.ceil() oraz Math.floor(). Używa się ich dokładnie tak samo jak metody Math.round(), jednak Math.ceil() zaokrągla przekazaną wartość w górę (na przykład wywołanie Math.ceil(4.0001) zwróci 5), natomiast metoda Math.floor() zaokrągla w dół (na przykład wywołanie Math.floor(4.9999) zwróci 4). Nazwy tych metod podchodzą od angielskich słów *ceiling* — sufit i *floor* — podłoga. Łatwo więc sobie wyobrazić, że pierwsza z nich — Math.ceil() — zaokrągla w górę, a druga — Math.floor() — w dół.

Formatowanie wartości monetarnych

Podczas obliczania ceny produktu lub prezentowania wartości koszyka z zakupami prezentowana jest zazwyczaj liczba wraz z dwoma miejscami dziesiętnymi, na przykład 9,99. Jeśli nawet wartość monetarna jest liczbą całkowitą, zazwyczaj dodaje się do niej dwa zera po przecinku dziesiętnym, na przykład 10,00. Poza tym, wartości monetarne, takie jak 9,8, także są zapisywane jako 9,80. Niestety, JavaScript nie postrzega wartości liczbowych w taki sposób — pomija niepotrzebne zera na końcu wartości dziesiętnych (wyświetli 10 zamiast 10,00 oraz 8.9 zamiast 8,90).

Na szczęście, dostępna jest metoda o nazwie toFixed(), pozwalająca konwertować liczby na łańcuchy znaków zawierające odpowiednią liczbę miejsc dziesiętnych. Aby jej użyć, wystarczy umieścić za liczbą (lub nazwą zmiennej zawierającej wartość liczbową) kropkę, a następnie dodać toFixed(2):

```
var cost = 10;
var printCost = cost.toFixed(2) . ' zł'; // 10.00 zł
printCost = printCost.replace(/\./, ',');  // 10,00 zł
```

Liczba przekazywana w wywołaniu metody toFixed() określa, ile miejsc dziesiętnych chcemy uzyskać w wyniku. W przypadku wartości monetarnych należy użyć liczby 2, dzięki czemu uzyskamy takie wartości jak 10.00 lub 9.90. W przypadku przekazania liczby 3 analogiczne wyniki miałyby postać: 10.000 oraz 9.900. Trzecia instrukcja zamienia kropkę w łańcuchu wygenerowanym przez metodę toFixed() na przecinek i uzyskujemy prawidłowo sformatowaną polską wartość monetarną.

Jeśli formatowana wartość ma początkowo więcej miejsc dziesiętnych niż podamy w wywołaniu, zostanie odpowiednio zaokrąglona. Oto przykład:

```
var cost = 10.289;
var printCost = cost.toFixed(2) + ' zł'; // 10.29 zł
printCost = printCost.replace(/\./, ','); // 10,29 zł
```

W tym przypadku wartość `10.289` zostanie zaokrąglona do `10.29`.

Uwaga: Metoda `toFixed()` operuje wyłącznie na liczbach. Jeśli spróbujemy sformatować za jej pomocą łańcuch znaków, zostanie zgłoszony błąd:

```
var cost='10'; // łańcuch znaków
var printCost=cost.toFixed(2) + ' zł'; // błąd
```

Aby ominąć ten problem, wystarczy skonwertować łańcuch znaków na liczbę przy użyciu jednego ze sposobów opisanych na stronie 70, na przykład:

```
var cost='10'; // łańcuch znaków
cost = +cost; // lub cost = Number(cost);
var printCost= cost.toFixed(2) + ' zł'; // 10.00 zł
printCost = printCost.replace(/\./, ','); // 10,00 zł
```

Tworzenie liczb losowych

Liczby losowe pozwalają wzbogacać programy o elementy losowości. Załóżmy, że dysponujemy tablicą pytań quizowych (takich jak w przykładowym programie przedstawionym na stronie 124). Zamiast wyświetlać te pytania za każdym razem w tej samej kolejności, można je wybierać i prezentować losowo. Można by także podczas wyświetlania strony losowo wybierać nazwę pliku graficznego, który zostanie na niej wyświetlony. Każde z tych zadań wymaga zastosowania liczb losowych.

Język JavaScript udostępnia metodę `Math.random()` służącą do generowania liczb losowych z zakresu od 0 do 1 (na przykład .9716907176080688 lub .10345038010895868). Choć najprawdopodobniej takie liczby nie będą szczególnie przydatne, jednak przy użyciu prostych działań matematycznych można je przekształcić na liczby całkowite z zakresu od zera do wybranej wartości. Przykładowo poniższy wzór pozwala wygenerować liczbę losową z zakresu od 0 do 9:

```
Math.floor(Math.random()*10);
```

Ten wzór można rozdzielić na dwa fragmenty. Fragment umieszczony wewnątrz wywołania metody `Math.floor()` — czyli `Math.random()*10` — generuje liczbę losową z zakresu od 0 do 10. Pozwala on wygenerować takie liczby jak 4.190788392268892, a ponieważ liczba losowa jest zakresu od 0 o 10, zatem *nigdy* nie przyjmie wartości 10. Aby uzyskać liczbę całkowitą, wystarczy przekazać tę wartość do metody `Math.floor()`, która ją zaokrągli do najbliższej liczby całkowitej, a zatem 3.4448588848 zostanie zaokrąglone do 3, a .1111939498984 do 0.

Gdybyśmy chcieli wygenerować liczbę z zakresu od 1 do innej liczby, wynik metody `Math.random()` należałoby pomnożyć przez górną granicę zakresu, a następnie zaokrąglić przy użyciu metody `Math.ceil()` (która zaokrągla w górę, do najbliższej liczby całkowitej). Gdybyśmy chcieli zasymulować rzut kostką, by uzyskać wartość z zakresu od 1 do 6, moglibyśmy to zrobić w następujący sposób:

```
var roll = Math.ceil(Math.random()*6); // 1,2,3,4,5 or 6
```

Losowe pobieranie elementu tablicy

Metody `Math.random()` można używać w celu pobierania losowo wybranego elementu tablicy. Zgodnie z informacjami podanymi na stronie 80, do elementów tablicy odwołujemy się przy użyciu indeksów liczbowych. Indeks pierwszego elementu ma wartość 0, natomiast ostatniego — wartość o jeden mniejszą od sumarycznej liczby elementów w tej tablicy. Przy zastosowaniu metody `Math.random()` pobieranie losowego elementu tablicy jest naprawdę łatwe:

```
var people = ['Romek','Tomek','Krysia','Bronek']; // tworzymy tablicę
var random = Math.floor(Math.random() * people.length);
var rndPerson = people[random];
```

Wiersz 1. powyższego fragmentu kodu tworzy tablicę zawierającą cztery imiona. Wiersz 2. wykonuje dwie podstawowe operacje. Przede wszystkim generuje liczbę losową z zakresu od 0 do liczby elementów w tablicy (czyli w naszym przypadku od 0 do 4). Następnie korzysta z metody `Math.floor()`, by zaokrąglić tę liczbę w dół, do najbliższej liczby całkowitej i zwraca w efekcie liczbę 0, 1, 2 lub 3. W końcu, w ostatnim wierszu pobieramy jedno z imion z tablicy, korzystając przy tym z wyliczonej wcześniej wartości, i zapisujemy je w zmiennej `rndPerson`.

Funkcja do generacji liczby losowej

Funkcje są doskonałym narzędziem do tworzenia użytecznych fragmentów kodu, nadających się do wielokrotnego stosowania (patrz strona 115). Jeśli często korzystasz z liczb losowych, możesz zdecydować się na napisanie funkcji, która pomoże w generowaniu losowej liczby całkowitej z określonego zakresu, na przykład od 1 do 6 lub od 100 do 1000. Przedstawiona poniżej funkcja wymaga przekazania dwóch argumentów. Pierwszy z nich określa najniższą możliwą wartość zakresu (na przykład 1), a drugi — wartość maksymalną (na przykład 6):

```
function rndNum(from, to) {
  return Math.floor((Math.random()*(to - from + 1)) + from);
}
```

Aby skorzystać z tej funkcji, dodaj ją do swojej strony WWW (zgodnie z informacjami zamieszczonymi na stronie 115) i wywołaj, tak jak w następującym przykładzie:

```
var dieRoll = rndNum(1,6); // generacja liczby losowej z zakresu od 1 do 6
```

Daty i godziny

Jeśli chcemy śledzić aktualną datę i godzinę, musimy skorzystać z obiektu `Date`. Ten specjalny obiekt języka JavaScript pozwala określać rok, miesiąc, dzień tygodnia, godzinę oraz wiele innych informacji związanych z datą i czasem. Aby z niego skorzystać, należy utworzyć zmienną i zapisać w niej nowy obiekt `Date`, zgodnie z poniższym przykładem:

```
var new = new Date();
```

Instrukcja `new Date()` tworzy nowy obiekt `Date` zawierający aktualną datę i godzinę. Po utworzeniu można pobierać z niego różne informacje, wywołując w tym celu odpowiednie metody (opisane w tabeli 16.3). Aby pobrać aktualny rok, można użyć następującego fragmentu kodu:

```
var now = new Date();
var year = now.getFullYear();
```

Tabela 16.3. *Metody służące do pobierania informacji z obiektu Date*

Metoda	Zwraca
`getFullYear()`	rok; na przykład 2014.
`getMonth()`	miesiąc wyrażony jako liczba z zakresu od 0 do 11: 0 odpowiada styczniowi, a 11 grudniowi.
`getDate()`	dzień miesiąca — jest to liczba z zakresu od 1 do 31.
`getDay()`	dzień tygodnia — jest to liczba z zakresu od 0 do 6; przy czym 0 oznacza niedzielę, a 6 sobotę.
`getHours()`	godzinę wyrażoną jako liczba z zakresu od 0 do 23.
`getMinutes()`	minuty, jako liczbę z zakresu od 0 do 59.
`getSecond()`	sekundy, jako liczbę z zakresu od 0 do 59.
`getTime()`	całkowitą liczbę milisekund, jaka upłynęła od 1 stycznia 1970 roku (więcej informacji na ten temat można znaleźć w ramce na stronie 595).

Uwaga: Instrukcja `new Date()` pobiera aktualną datę i godzinę według ustawień na komputerze użytkownika. Innymi słowy, jeśli zegar systemowy w komputerze użytkownika nie jest prawidłowo ustawiony, także pobierane informacje o dacie i godzinie nie będą dokładne.

Pobieranie miesiąca

Aby pobrać datę z wykorzystaniem obiektu `Date`, należy posłużyć się funkcją `getMonth()`, która zwraca numer miesiąca:

```
var now = new Date();
var month = now.getMonth();
```

Jednak zamiast zwracać liczbę, która byłaby sensowna dla nas — ludzi — (czyli 1 dla miesiąca stycznia), metoda ta zwraca wartość o jeden mniejszą. I tak dla stycznia metoda ta zwróci wartość 0, dla lutego — 1 i tak dalej. Jeśli zatem chcemy uzyskać liczbę odpowiadającą tradycyjnie używanej numeracji miesięcy, do wyniku zwróconego przez metodę `getMonth()` należy dodać 1:

```
var now = new Date();
var month = now.getMonth()+1; // odpowiada faktycznej numeracji miesięcy
```

JavaScript nie udostępnia żadnego mechanizmu pozwalającego na pobieranie nazw miesięcy. Na szczęście, dziwaczny sposób numeracji miesięcy używany w metodzie `getMonth()` okazuje się bardzo wygodny przy określaniu ich nazw. W celu podania nazw miesięcy należy je najpierw zapisać w tablicy, a następnie pobierać z niej przy użyciu jako indeksu liczby zwróconej przez metodę `getMonth()`:

```
var months = ['styczeń','luty','marzec','kwiecień','maj',
              'czerwiec','lipiec','sierpień','wrzesień',
              'październik','listopad','grudzień'];
var now = new Date();
var month = months[now.getMonth()];
```

Wiersz 1. powyższego kodu tworzy tablicę zawierającą nazwy wszystkich dwunastu miesięcy, zapisane w prawidłowej kolejności, czyli od stycznia do grudnia. Należy

pamiętać, że w celu pobrania wartości elementu tablicy należy podać jego indeks oraz że indeksy tablicy zaczynają się od wartości 0 (patrz strona 79). A zatem, by pobrać pierwszy element tablicy `months`, trzeba użyć wyrażenia `months[0]`. Oznacza to, że korzystając z metody `getMonth()`, możemy uzyskać liczbę, której następnie będziemy mogli użyć jako indeksu do tablicy `months` i pobrać nazwę miesiąca.

Określanie dnia tygodnia

Metoda `getDay()` zwraca numer dnia tygodnia. Podobnie jak to było w metodzie `getMonth()`, także i tutaj interpreter JavaScriptu zwraca liczbę o jeden mniejszą, niż można by się spodziewać: 0 odpowiada niedzieli (która w USA i niektórych innych krajach jest traktowana jako pierwszy dzień tygodnia), a 6 — sobocie. Ponieważ jednak nazwa dnia tygodnia jest zazwyczaj bardziej użyteczna niż jego numer, zatem możemy zapisać te nazwy w tablicy i do ich pobierania używać metody `getDay()`:

```
var days = ['niedziela','poniedziałek','wtorek','środa',
            'czwartek','piątek','sobota'];
var now = new Date();
var dayOfWeek = days[now.getDay()];
```

Pobieranie czasu

Obiekt `Date()` zawiera także informacje o czasie, a zatem można go wyświetlić na stronie bądź użyć do określenia, czy użytkownik wyświetla stronę przed południem, czy po południu. Następnie można te informacje w jakiś sposób wykorzystać — na przykład wyświetlić obraz tła ze słońcem w czasie dnia lub z księżycem w nocy.

Do pobierania liczby godzin, minut i sekund można użyć metod `getHours()`, `getMinutes()` oraz `getSeconds()`. Aby wyświetlić czas na stronie WWW, w wybranym miejscu jej kodu HTML należy dodać poniższy fragment kodu JavaScriptu:

```
var now = new Date();
var hours = now.getHours();
var minutes = now.getMinutes();
var seconds = now.getSeconds();
document.write(hours + ":" + minutes + ":" + seconds);
```

Powyższy fragment kodu generuje czas zapisany w formacie 6:35:51 oznaczający godzinę 6, 35 minut oraz 51 sekund. Jednak wygeneruje także czas zapisany jako 18:23:42 reprezentujący godzinę szóstą po południu, 23 minuty i 42 sekundy. Problem polega jednak na tym, że w dobie zegarków elektronicznych sporo osób może być przyzwyczajonych do wyrażania czasu w formie 12-godzinnej, gdzie zapis 10:34 p.m. odpowiada godzinie 22:34. Kolejnym problemem jest to, że podczas zapisu czasu godziny i minuty zawsze powinny być zapisywane przy użyciu dwóch cyfr (nawet jeśli ich wartość jest mniejsza od 10), na przykład: 6:04:09. Na szczęście powyższy przykład stosunkowo łatwo można zmodyfikować i dostosować do prezentowania czasu w zapisie 12-godzinnym.

PORADNIA DLA ZAAWANSOWANYCH

Tajemnice obiektu Date

JavaScript pozwala na dostęp do poszczególnych elementów obiektu `Date`, takich jak numer roku, dnia lub miesiąca. Jednak w rzeczywistości interpreter JavaScriptu myśli o tej dacie jak o liczbie *milisekund*, jakie upłynęły od północy 1 stycznia 1970 roku. Przykładowo poniedziałek 1 grudnia 2014 roku jest reprezentowany jako liczba 1417388400000.

Nie, to nie żart. Z punktu widzenia interpretera JavaScriptu początkiem czasu jest 1 stycznia 1970 roku według uniwersalnego czasu koordynowanego (który w większości przypadków pokrywa się z czasem GMT). Ta data (nazywana także „epoką Uniksa") została celowo wybrana w latach 70. ubiegłego wieku przez programistów tworzących system operacyjny Unix, po to by wszyscy mogli używać tego samego sposobu zapisu czasu. Od tamtej pory ten sposób reprezentacji czasu stał się popularny i jest używany w wielu językach programowania oraz na wielu platformach komputerowych.

Za każdym razem, gdy używamy jakiejś metody obiektu `Date`, takiej jak `getFullYear()`, interpreter JavaScriptu wykonuje obliczenia matematyczne (bazujące na liczbie sekund, jaka upłynęła od 1 stycznia 1970 roku), by określić, jaki rok ten obiekt reprezentuje. Aby pobrać liczbę milisekund dla danej daty, wystarczy wywołać metodę `getTime()`:

```
var sometime = new Date();
var msElapsed = sometime.getTime();
```

Przechowywanie informacji o datach i czasie w formie milisekund sprawia, że bardzo łatwo można obliczać różnice między dwiema datami. Aby obliczyć liczbę dni do nowego roku, wystarczy pobrać liczbę milisekund od 1 stycznia 1970 do 1 stycznia następnego roku i odjąć od niej liczbę milisekund od 1 stycznia 1970 roku do dnia dzisiejszego.

```
// liczba milisekund od 1.1.1970 do dnia dzisiejszego
var today = new Date();
// liczba milisekund od 1.1.1970 do następnego nowego roku
var nextYear = new Date(2015,0,1);
// obliczenie różnicy milisekund pomiędzy dniem dzisiejszym
// i następnym nowym rokiem
var timeDiff = nextYear - today;
```

W efekcie odjęcia od siebie dwóch dat uzyskujemy liczbę milisekund stanowiącą różnicę pomiędzy nimi. Aby przekształcić ją w jakąś użyteczną informację, należy ją podzielić przez liczbę milisekund w jednym dniu (by określić liczbę dni) lub liczbę milisekund w godzinie (by określić liczbę godzin) i tak dalej.

```
var second = 1000; // sekunda to 1000 millisekund
var minute = 60*second; // 60 sekund to minuta
var hour = 60*minute; // 60 minut to godzina
var day = 24*hour; // 24 godziny to jedna doba
var totalDays = timeDiff/day; // sumaryczna
                              // liczba dni
```

(W tych przykładach mogłeś zauważyć inny sposób tworzenia obiektu daty: `new Date(2015,0,1)`. Więcej informacji na jego temat można znaleźć na stronie 597.

Zmiana godzin do formatu 12-godzinnego

Aby zmienić czas z formatu 24-godzinnego na 12-godzinny, należy wykonać kilka operacji. Przede wszystkim trzeba sprawdzić, czy czas reprezentuje godziny przedpołudniowe (aby dodać do niego oznaczenie `'am'`), czy też popołudniowe (by dodać do niego oznaczenie `'pm'`). Poza tym, liczbę godzin większą od 12 należy zmienić na ich 12-godzinny odpowiednik (na przykład zmienić 14 na 2 p.m.).

Poniższy fragment kodu wykonuje taką konwersję:

```
1    var now = new Date();
2    var hour = now.getHours();
3    var am_pm;
4    if (hour < 12) {
5      am_pm = 'am';
6    } else {
7      am_pm = 'pm';
8    }
9    hour = hour % 12;
10    if (hour==0) {
11     hour = 12;
12   }
13   hour = hour + ' ' + am_pm;
```

Uwaga: Kolumna liczb widoczna z lewej strony to tylko numeracja wierszy kodu, zamieszczona po to, by łatwiej było analizować opisywany kod. Nie należy przepisywać tych liczb do własnego kodu.

Wiersze 1. i 2. pobierają aktualną datę oraz czas i zapisują aktualną godzinę w zmiennej `hour`. Wiersze od 3. do 7. sprawdzają, czy godzina wypada przed południem, czy po południu; jeśli jest mniejsza od 12 (godzinie bezpośrednio po północy odpowiada liczba 0), to znaczy, że mamy rano (a.m.), w przeciwnym razie mamy popołudnie (p.m.).

W wierszu 8. został zastosowany operator **modulo**, reprezentowany przez znak procenta (`%`). Operator ten zwraca resztę z dzielenia dwóch liczb. Przykładowo w przypadku dzielenia 5 przez 2 dwójka mieści się w liczbie 5 dwa razy (2 * 2 = 4) i pozostaje reszta o wartości 1. Innymi słowy 5 % 2 = 1. Wracając do naszego przykładu z przeliczaniem godzin, jeśli mamy godzinę 18, to 18 % 12 wynosi 6 (12 mieści się w 18 jeden raz i pozostaje reszta 6). A zatem, godzina 18 to inaczej 6 p.m., czyli dokładnie taka, jaką chcieliśmy uzyskać. Jeśli pierwsza wartość jest mniejsza od wartości umieszczonej z prawej strony operatora modulo, wynikiem działania jest pierwsza wartość, na przykład 8 % 12 daje 8. Innymi słowy, w naszym przypadku operator modulo nie zmienia godzin przed południem.

W wierszach od 9. do 11. uwzględniamy dwa możliwe wyniki działania operatora modulo. Jeśli mamy godzinę 12 (południe) lub 0 (północ), operator modulo zwróci wartość 0. W tych przypadkach ustawiamy godzinę na 12 — odpowiednio 12 p.m. oraz 12 a.m.

I w końcu, w wierszu 12. dodajemy do przeliczonej godziny odpowiednią końcówkę — `am` lub `pm` — dzięki czemu godzina zostanie wyświetlona w oczekiwanej postaci: `6 am` lub `6 pm` .

Dopełnianie pojedynczych cyfr

Zgodnie z informacjami podanymi na poprzedniej stronie, kiedy wartości minut lub sekund są mniejsze od 10, może się okazać, że prezentowane dane zostaną wyświetlone w dziwnej postaci, takiej jak 7:3:2. Aby rozwiązać ten problem i wyświetlać czas w oczekiwanej postaci, czyli jako 7:03:02, przed pojedynczymi cyframi minut i sekund należy dodawać zero. Można to zrobić bardzo łatwo przy użyciu prostej instrukcji warunkowej:

```
1  var minutes = now.getMinutes();
2  if (minutes<10) {
3    minutes = '0' + minutes;
4  }
```

W wierszu 1. pobieramy minuty z aktualnego czasu, uzyskując na przykład wartość `33` lub `3`. W wierszu 2. sprawdzamy, czy pobrana wartość jest mniejsza od `10`, co by oznaczało, że jest pojedynczą cyfrą i wymaga dodania zera na początku. Instrukcja umieszczona w wierszu 3. jest nieco bardziej złożona, gdyż normalnie nie można dodać zera przed liczbą: `0 + 2` daje `2`, a nie `02`. Jednak nic nie stoi na przeszkodzie, by w taki sposób dodawać do siebie łańcuchy znaków, a zatem `'0' + minutes` oznacza: połącz łańcuch znaków `'0'` z wartością zmiennej `minutes`. Zgodnie z informacjami podanymi na stronie 70, w przypadku dodawania łańcucha znaków do liczby interpreter JavaScriptu konwertuje tę liczbę na łańcuch znaków, w efekcie uzyskujemy łańcuch, taki jak `'08'`.

Teraz możemy połączyć wszystkie te elementy i napisać prostą funkcję, która wyświetla czas w postaci: 7:32:04 p.m. lub 4:02:34 a.m bądź nawet całkowicie pomija sekundy i zwraca czas w postaci 7:23 p.m.:

```
function getTime(secs) {
    var sep = ':'; // znak separatora
    var hours, minutes, seconds, time;
    var now = new Date();
    var am_pm;
    hours = now.getHours();
    if (hours < 12) {
       am_pm = 'am';
    } else {
       am_pm = 'pm';
    }
    hours = hours % 12;
    if (hours==0) {
        hours = 12;
    }
    time = hours;
    minutes = now.getMinutes();
    if (minutes<10) {
        minutes = '0' + minutes;
    }
    time += sep + minutes;
    if (secs) {
        seconds = now.getSeconds();
        if (seconds<10) {
            seconds = '0' + seconds;
        }
        time += sep + seconds;
    }
    return time + ' ' + am_pm;
}
```

Kod tej funkcji można znaleźć przykładach do książki, w pliku *getTime.js* umieszczonym w katalogu *R16*. Jego działanie można natomiast sprawdzić, wyświetlając w przeglądarce przykładowy plik *time.html* (umieszczony w tym samym katalogu). Aby skorzystać z tej funkcji, należy dołączyć plik *getTime.js* do strony bądź też skopiować kod i umieścić go bezpośrednio na swojej stronie lub w innym zewnętrznym pliku JavaScript. Aby wyświetlić czas, wystarczy wywołać funkcję `getTime()`. Jeśli chcemy wyświetlać także sekundy, możemy wywołać funkcję w następujący sposób: `getTime(true)`. Funkcja zwróci bieżący czas zapisany w formacie 12-godzinnym.

Tworzenie daty innej niż bieżąca

Dowiedziałeś się, jak można tworzyć obiekt `Date` (`new Date()`) reprezentujący aktualną datę i czas na komputerze użytkownika. A w jaki sposób można napisać obiekt `Date` reprezentujący 1 stycznia następnego roku bądź Wigilię najbliższych Świąt Bożego Narodzenia? JavaScript pozwala tworzyć takie daty na kilka różnych sposobów. Możemy potrzebować takiej możliwości na przykład podczas wyliczania różnicy pomiędzy dwiema datami, takiej jak „Ile dni pozostało do nowego roku?".

W przypadku stosowania metody `Date()` można podać zarówno przeszłe, jak i przyszłe datę i czas. Składnia wywołania takich metod ma następującą postać:

```
new Date(rok, miesiac, dzien, godzina, minuta, sekunda, milisekunda);
```

By na przykład uzyskać datę reprezentującą 1 stycznia 2015 roku, trzeba użyć następującego wywołania:

```
var ny2015 = new Date(2015,0,1,0,0,0);
```

Taki kod zostanie zinterpretowany w następujący sposób: „utwórz nowy obiekt `Date` reprezentujący 1 stycznia 2015 roku o godzinie 0, minut 0 i 0 sekund". W wywołaniu w takiej postaci trzeba określić przynajmniej argumenty ustalające rok oraz miesiąc, jeśli jednak informacje o czasie nie są potrzebne, to argumenty odpowiadające godzinom, minutom, sekundom i milisekundom możemy pominąć. Aby na przykład utworzyć obiekt daty reprezentujący 1 stycznia 2015 roku, możemy użyć wywołania w postaci:

```
var ny2015 = new Date(2015,0,1);
```

Uwaga: Koniecznie trzeba pamiętać, że w JavaScripcie styczniowi odpowiada wartość 0, lutemu — 1 i tak dalej; informacje na ten temat można znaleźć na stronie 593.

Tworzenie daty z zadaną liczbą dni w przód

Zgodnie z informacjami podanymi w ramce na stronie 595, interpreter JavaScriptu traktuje daty jako liczbę milisekund, która upłynęła od północy 1 stycznia 1970 roku, według uniwersalnego czasu koordynowanego (UTC). Kolejnym sposobem utworzenia obiektu `Date` jest przekazanie w wywołaniu metody `Date()` liczby milisekund:

```
new Date(milisekundy);
```

A zatem, datę 1 stycznia 2015 roku można także uzyskać w następujący sposób:

```
var ny2015 = new Date(1420066800000);
```

Oczywiście, większość z nas nie liczy jak kalkulatory, dlatego nie postrzegamy dat w taki sposób. Jednak ten sposób tworzenia dat okazuje się bardzo przydatny, gdy trzeba utworzyć nową datę przesuniętą o określony czas w stosunku do innej daty. Podczas tworzenia w kodzie JavaScript cookie (ciasteczka) konieczne jest określenie daty jego wygaśnięcia. Aby upewnić się, że cookie zostanie usunięte dokładnie za tydzień, musimy podać datę przesuniętą o 7 dni w stosunku do dnia dzisiejszego.

Uwaga: Przedstawiony powyżej przykład — `new Date(1420066800000);` — nie zadziała we wszystkich przypadkach. Działa on prawidłowo wyłącznie w środkowoeuropejskiej strefie czasowej, UTC+1, czyli strefie czasowej, w której leży Polska. Dzieje się tak dlatego, że przeglądarki uwzględniają strefy czasowe i dostosowują swój zegar na podstawie uniwersalnego czasu koordynowanego (UTC), zgodnie z informacjami podanymi w ramce na stronie 595.

Aby utworzyć taką datę, należy użyć poniższego fragmentu kodu:

```
var now = new Date(); // data dzisiejsza
var nowMS = now.getTime(); // pobranie liczby milisekund dla daty bieżącej
var week = 1000*60*60*24*7; // liczba milisekund w tygodniu
var oneWeekFromNow = new Date(nowMS + week);
```

W 1. wierszu powyższego fragmentu kodu zapisujemy w zmiennej `now` bieżącą datę i godzinę. W kolejnym wierszu za pomocą metody `getTime()` pobieramy liczbę milisekund, jakie upłynęły od 1 stycznia 1970 roku do dziś. W wierszu 3. obliczamy,

PORADNIA DLA ZAAWANSOWANYCH

Obsługa różnych stref czasowych

Komputery robią coś więcej, niż jedynie śledzenie upływu sekund, minut i dni. Muszą także koordynować swój czas z czasem na innych komputerach działających na całym świecie, w różnych strefach czasowych. W końcu godzina 8 wieczorem w Moskwie nie jest godziną 8 wieczorem w San Francisco. Ponieważ oba te miasta znajdują się niemal na przeciwnych stronach kuli ziemskiej, a czas wschodów i zachodów słońca jest niemal odwrotny, zatem godzina 8 po południu w Moskwie wypada o godzinie 8 rano w San Francisco. (Bądź też, w przypadku czasu letniego, godzina 8 wieczorem w Moskwie odpowiada godzinie 9 rano w San Francisco).

Aby ułatwić komputerom (i ludziom) z różnych stron świata wzajemną synchronizację, programiści używają konwencji określanej jako *UTC* — uniwersalny czas koordynowany. Ta strefa czasowa była niegdyś określana jako *GMT — Greenwich Mean Time* — i odpowiada południkowi o długości geograficznej 0 . Wszyscy, którzy nie mieszkają w Anglii, Portugalii lub w krajach afrykańskich leżących na tej samej długości geograficznej, co Anglia, mieszkają w krajach, których czas jest *przesunięty* względem czasu UTC.

San Francisco leży w strefie czasowej UTC -8, co oznacza, że jest tam o 8 godzin wcześniej, niż wynosi czas UTC. Kiedy w Londynie jest godzina 9:15 wieczorem, w San Francisco jest godzina 1:15 po południu. Moskwa znajduje się w strefie czasowej UTC +4, a zatem kiedy w Londynie jest godzina 9:15 wieczorem, w Moskwie jest godzina 1:15. (Choć godziny te będą nieco inne w przypadku stosowania czasu letniego).

Obiekt `Date` udostępnia metodę pozwalającą na określenie przesunięcia lokalnej strefy czasowej (czyli różnicy pomiędzy czasem lokalnym oraz czasem UTC):

```
var now = new Date();
var offset = now.getTimezoneOffset();
```

Jeśli komputer będzie działał w San Francisco, wywołanie tej metody zwróci wartość 480 lub 420. Wartość ta reprezentuje całkowitą liczbę minut, jaka różni czas lokalny komputera od czasu UTC. A zatem 480 odpowiada 8 godzinom, a 420 — 7 godzinom (ta wartość będzie zwracana w przypadku czasu letniego). Jeśli w kraju zamieszkania jest stosowany czas letni, przeglądarka także to uwzględni (czyż komputery nie są sprytne?).

Ogólnie rzecz biorąc, nie musimy się przejmować uwzględnianiem stref czasowych. Kod JavaScript jest wykonywany na komputerach użytkowników, a zatem, w przypadku wykonywania obliczeń związanych z czasem, takich jak wyświetlanie informacji, która jest godzina, przeglądarka dostosuje się do strefy czasowej użytkownika i wyświetli prawidłowy czas.

ile milisekund przypada na jeden tydzień (1000 milisekund * 60 sekund * 24 godziny * 7 dni). I w końcu, w ostatnim wierszu tworzymy nową datę, dodając liczbę milisekund w tygodniu do liczby milisekund reprezentującej datę bieżącą.

Uwaga: Obiekt Date języka JavaScript udostępnia wiele użytecznych metod służących do obliczania dat i dostosowywania ich do różnych lokalizacji. Kompletną dokumentację tej klasy można znaleźć na stronie *https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Date*.

Tworzenie bardziej wydajnego kodu JavaScript

Programowanie wymaga dużo pracy. Programiści zawsze starają się wykonywać zadania szybciej i w mniejszej liczbie wierszy kodu. Znanych jest wiele sztuczek z tego obszaru, a poniżej opisano techniki szczególnie przydatne przy korzystaniu z języka JavaScript i biblioteki jQuery.

Zapisywanie ustawień w zmiennych

Jedną z ważnych lekcji dla programistów jest nauka usuwania zbędnych szczegółów ze skryptów, co sprawia, że programy są bardziej elastyczne i łatwiejsze do aktualizowania. Załóżmy, że kolor akapitu ma się zmienić na pomarańczowy, kiedy użytkownik kliknie dany tekst. Możesz uzyskać ten efekt za pomocą funkcji `css()` biblioteki jQuery (patrz strona 163):

```
$('p').click(function() {
  $(this).css('color','#F60');
});
```

Tu kolor pomarańczowy (`#F60`) jest na stałe zapisany w instrukcji. Przyjmijmy, że chcesz użyć tej samej barwy także w innych fragmentach kodu (na przykład aby dodać kolor tła po umieszczeniu kursora nad wierszem tabeli). Wydaje się, że należy umieścić wartość `#F60` także w tych instrukcjach. Jednak lepsze rozwiązanie polega na zapisaniu koloru w zmiennej na początku skryptu i użyciu jej w dalszej części programu:

```
1  $(document).ready(function() {
2  var hColor='#F60';
3  $('p').click(function() {
4    $(this).css('color',hColor);
5  });
6  $('td').hover(
7    function() {
8      $(this).css('backgroundColor',hColor);
9    },
10   function() {
11     $(this).css('backgroundColor','transparent');
12   }
13 );
14 }); // Koniec funkcji ready.
```

Zmienna `hColor` przechowuje tu wartość szesnastkową reprezentującą kolor. Zmiennej tej skrypt używa w zdarzeniu `click` znaczników `<p>` i zdarzeniu `hover` tagów `<td>`. Jeśli później uznasz, że kolor pomarańczowy nie odpowiada Ci, możesz zmienić wartość zapisaną w zmiennej, na przykład na `var hColor='F33';`, a skrypt użyje nowej barwy.

Powyższy skrypt będzie bardziej elastyczny, jeśli zlikwidujesz powiązanie między kolorami używanymi dla znaczników `<p>` i `<td>`. Obecnie dla obu tagów używana jest ta sama barwa, jednak jeśli chcesz mieć możliwość zastosowania w przyszłości dwóch odrębnych kolorów, możesz dodać do kodu nową zmienną:

```
1  $(document).ready(function() {
2  var pColor='#F60';
3  var tdColor=pColor;
4  $('p').click(function() {
5    $(this).css('color',pColor);
6  });
7  $('td').hover(
8    function() {
9      $(this).css('backgroundColor',tdColor);
10   },
11   function() {
12     $(this).css('backgroundColor','transparent');
13   }
14 );
15 }); // Koniec funkcji ready.
```

Teraz zdarzenia `click` i `hover` korzystają z tego samego koloru `#F60`, ponieważ skrypt przypisuje w 3. wierszu wartość zmiennej `pColor` do zmiennej `tdColor`. Jeśli jednak w przyszłości zechcesz nadać komórkom tabeli inną barwę, wystarczy zmienić wiersz 3.:

```
var tdColor='#FF3';
```

W czasie tworzenia programów JavaScript staraj się przekształcać nazwy bezpośrednio używane w programie na zmienne. Dobrymi kandydatami na to są: kolory, czcionki, wymiary (szerokość i wysokość), czas (na przykład 1000 milisekund), nazwy plików (graficznych i innych), tekst komunikatów (z ramek ostrzegawczych i okien z potwierdzeniami) oraz ścieżki do plików (na przykład w odnośnikach i znacznikach `<img>`):

```
var highlightColor = '#33A';
var upArrow = 'ua.png';
var downArrow='da.png';
var imagePath='/images/';
var delay=1000;
```

Takie definicje zmiennych należy umieścić na początku skryptu lub — jeśli używasz jQuery — w funkcji `.ready()`.

Wskazówka: Szczególnie przydatne jest umieszczanie w zmiennych tekstu, który chcesz wyświetlać na stronie. Są to na przykład komunikaty o błędach („Podaj prawidłowy adres e-mail") lub informacje („Dziękujemy za podanie adresu e-mail"). Jeśli zgrupujesz takie wiadomości w formie zmiennych na początku skryptu, w przyszłości będziesz mógł je łatwo zmodyfikować (lub przetłumaczyć, jeżeli zechcesz dotrzeć do użytkowników z innych państw).

Zapisywanie ustawień w obiektach

Istnieje także nieco bardziej zaawansowany sposób przechowania ustawień: z wykorzystaniem obiektów JavaScript. Zgodnie z informacjami podanymi na stronie 165, literały obiektowe w języku JavaScript pozwalają na przechowywanie w jednym obiekcie par nazwa – wartość (lub klucz – wartość). W ostatnim przykładzie przedstawionym w poprzednim punkcie rozdziału do zachowania często używanych wartości, takich jak kolory, ścieżki do obrazów, nazwy ikon i tym podobne, wykorzystanych zostało wiele niezależnych zmiennych. Takie rozwiązanie jest pod każdym względem prawidłowe, jednak wszystkie te niezależne wartości można grupować i umieścić w jednym obiekcie, a następnie odwoływać się do nich, tak jak do właściwości, wykorzystując w tym celu zapis z kropką (patrz strona 87).

Przykładowo listę pięciu zmiennych z poprzedniego punktu rozdziału można by zapisać w formie jednego obiektu w następujący sposób:

```
var siteSettings = {
  highlightColor: '#33A',
  upArrow: 'ua.png',
  downArrow:'da.png',
  imagePath: '/images/',
  delay: 1000
}
```

Jeśli zastosowany zostanie ten zapis, wartości nie będą kojarzone z nazwami właściwości przy użyciu znaku równości, dlatego zamiast zapisu `upArrow = 'ua.png'` będzie użyty zapis `upArrow: 'ua.png'`, a poszczególne pary nazwa – wartość będą od siebie oddzielane przecinkami (przy czym za ostatnią parą przecinka nie należy umieszczać). Aby użyć takich ustawień w kodzie programu, należy odwoływać się do poszczególnych właściwości obiektu przy użyciu zapisu z kropką. Aby na przykład pobrać wartość właściwości `highlightColor`, należałoby użyć następującego wyrażenia:

```
siteSettings.highlightColor
```

Powyższe rozwiązanie ma kilka zalet. Przede wszystkim zapewnia lepszą organizację kodu niż stosowanie grupy niezależnych zmiennych. Wszystkie ustawienia są zapisywane w jednym obiekcie, umieszczonym w jednym miejscu kodu. Po drugie pozwala uniknąć problemu polegającego na nadaniu tej samej nazwy dwóm różnym zmiennym. Przykładowo nazwa „delay" jest dosyć ogólna. Jeśli na początku swojego kodu utworzysz zmienną `delay`, a później, w jakimś jego innym miejscu, utworzysz kolejną zmienną o tej samej nazwie służącą do przechowywania jakiegoś ustawienia, pierwsza zmienna `delay` zostanie usunięta. Jeśli jednak ustawienie o nazwie `delay` zostanie zapisane w obiekcie — jako `siteSettings.delay` — nie będzie kolidowało z żadną inną zmienną używaną w programie.

Uwaga: Być może pamiętasz, że na stronie 265 wspominano o wtyczkach jQuery, które używają literałów obiektowych do przekazywania informacji konfiguracyjnych. Także widżety jQuery UI (przedstawione w rozdziałach 10. i 11.) korzystają z literałów obiektowych do przekazywania informacji kontrolujących ich wygląd i działanie.

Operator trójargumentowy

Często wykonywaną operacją programistyczną jest przypisywanie wartości do zmiennej na podstawie określonego warunku. Załóżmy, że chcesz określić wartość zmiennej zawierającej tekst z informacją o tym, czy użytkownik jest zalogowany. W skrypcie znajduje się zmienna logiczna `login`. Ma ona wartość `true`, jeśli użytkownik się zalogował, i `false`, jeżeli jest niezalogowany. Oto jeden ze sposobów na utworzenie nowej zmiennej:

```
var status;
if (login) {
  status='Zalogowany';
} else {
  status='Niezalogowany';
}
```

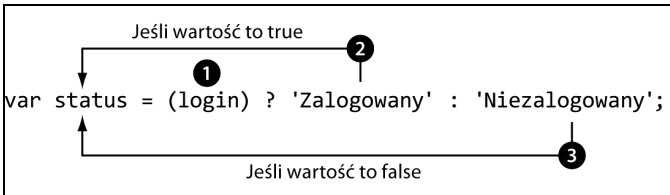
Tu prosta instrukcja warunkowa (patrz strona 93) ustawia wartość zmiennej `status` w zależności od tego, czy użytkownik jest zalogowany, czy nie. JavaScript udostępnia skrótowy zapis tej często używanej konstrukcji. Jest to tak zwany *operator trójargumentowy*. Umożliwia on tworzenie prostych instrukcji warunkowych w jednym wierszu. Składnia tego operatora wygląda następująco:

```
warunek ? A : B
```

Jeśli warunek ma wartość `true`, instrukcja zwróci `A`, a jeśli wartość warunku to `false`, zwrócone zostanie `B` (znak `?` poprzedza wynik dla wartości `true`, natomiast po symbolu `:` znajduje się rezultat zwracany dla wartości `false`). Dlatego wcześniejszy fragment można zapisać także w poniższy sposób:

```
var status = login ? 'Zalogowany' : 'Niezalogowany';
```

Sześć wierszy kodu można więc przekształcić na jeden. Na rysunku 16.6 pokazano jego działanie.



Rysunek 16.6. *Operator trójargumentowy pozwala tworzyć jednowierszowe instrukcje warunkowe. Tu punkt 1 oznacza warunek. Jeśli jest on spełniony, instrukcja zwróci kod po znaku ? (punkt 2). Jeśli warunek jest fałszywy, zwrócony zostanie kod po symbolu : (punkt 3)*

Operator trójargumentowy to tylko skrót. Nie musisz go używać, a niektórzy programiści uważają, że technika ta utrudnia zrozumienie kodu, dlatego wolą stosować bardziej czytelne instrukcje `if-else`. Ponadto operator trójargumentowy najlepiej nadaje się do określania wartości zmiennych na podstawie warunku. Techniki tej nie można użyć dla każdej instrukcji warunkowej. Na przykład nie można zastosować jej w instrukcjach wielowierszowych, w których należy uruchomić kilka wierszy kodu na podstawie wartości warunku. Jednak nawet jeśli nie będziesz korzystał z operatora trójargumentowego, warto znać jego działanie. Dzięki temu łatwiej będzie Ci zrozumieć skrypty innych programistów, które prawdopodobnie będziesz często przeglądał.

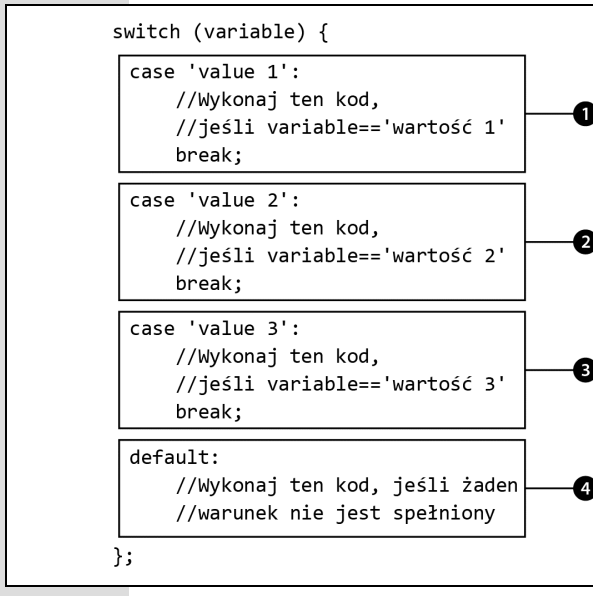
Instrukcja Switch

Jest wiele postaci instrukcji warunkowych. Operator trójargumentowy doskonale nadaje się do przypisywania wartości zmiennych na podstawie warunku, natomiast instrukcja `switch` to bardziej zwięzły sposób zapisu zbioru instrukcji `if-else` opartych na wartości jednej zmiennej.

Załóżmy, że użytkownik może podać w polu formularza ulubiony kolor, a skrypt ma wyświetlić wiadomość dostosowaną do danej barwy. Za pomocą standardowej instrukcji warunkowej można to zapisać w następujący sposób:

```
if (favoriteColor == 'niebieski') {
  message = 'Niebieski to zimny kolor.';
} else if (favoriteColor == 'czerwony') {
  message = 'Czerwony to ciepły kolor.';
} else if (favoriteColor == 'zielony') {
  message = 'Zielony to kolor liści.';
} else {
  message = 'Co to za kolor?';
}
```

Zauważ, że wielokrotnie pojawia się tu fragment `favoriteColor == 'wartość'`. Występuje on trzykrotnie w dziewięciu wierszach kodu. Jeśli chcesz wielokrotnie sprawdzić wartość danej zmiennej, bardziej eleganckie (i czytelne) będzie użycie instrukcji `switch`. Jej podstawową strukturę przedstawiono na rysunku 16.7.



Rysunek 16.7. *Instrukcja switch pozwala w zwięzły sposób wykonać różne operacje w zależności od wartości zmiennej. Nie zapomnij o poleceniu break; na końcu każdego warunku. Umożliwia ono opuszczenie instrukcji switch*

Pierwszy wiersz instrukcji `switch` rozpoczyna się od słowa kluczowego `switch`. Po nim następuje nazwa zmiennej w nawiasach i otwierający nawias klamrowy. Kod ten oznacza: „Sprawdźmy, czy wartość danej zmiennej pasuje do jednej z podanych dalej wartości". Każda operacja porównywania wartości to **warunek** (instrukcja `switch` może zawierać ich wiele). Na rysunku 16.7 znajdują się trzy warunki o numerach od 1 do 3. Podstawowa struktura warunku wygląda następująco:

```
case wartość1:
  // Wykonaj operacje.
  break;
```

Słowo kluczowe `case` oznacza początek warunku. Następnie znajduje się wartość i dwukropek. Ten wiersz to skrót dłuższego zapisu `if (zmienna=='wartość1')`. Wartością może być liczba, łańcuch znaków lub wartość logiczna (albo zmienna zawierająca dane dowolnego z tych typów). Jeśli chcesz sprawdzić, czy zmienna ma wartość `37`, możesz użyć poniższego testu:

```
case 37:
  // Wykonaj operacje.
  break;
```

Aby sprawdzić, czy zmienna zawiera wartość `true`, należy użyć następującego kodu:

```
case true:
  // Wykonaj operacje.
  break;
```

Po pierwszym wierszu należy dodać instrukcje wykonywane, jeśli warunek jest spełniony. Na końcu trzeba umieścić instrukcję `break;`. To ważne — instrukcja `break;` powoduje zakończenie wykonywania instrukcji `switch`. Jeśli ją pominiesz,

interpreter przejdzie do sprawdzania następnych warunków! Innymi słowy, kiedy interpreter JavaScript dopasuje warunek `catch`, przestanie już sprawdzać kolejne warunki i wykona cały kod umieszczony w dalszej części instrukcji `switch`.

Oto jak za pomocą instrukcji `switch` można przekształcić kod instrukcji `if-else` ze strony 603:

```
switch (favoriteColor) {
  case 'niebieski':
    message = 'Niebieski to zimny kolor.';
    break;
  case 'czerwony':
    message = 'Czerwony to ciepły kolor.';
    break;
  case 'zielony':
    message = 'Zielony to kolor liści.';
    break;
  default:
    message = 'Co to za kolor?';
}
```

Ten kod działa tak samo jak wcześniejsza instrukcja `if-else`, jednak jest bardziej zrozumiały i nie wymaga stosowania powtarzających się warunków (takich jak `if (favouriteColor === blue)`).

Możesz też umieścić kilka warunków `case` jeden pod drugim (i celowo pominąć słowo kluczowe `default`), aby uruchomić ten sam kod dla kilku różnych wartości, na przykład:

```
switch (favoriteColor) {
  case 'granatowy':
  case 'niebieski':
  case 'indygo':
    message = 'Niebieski to zimny kolor.';
    break;

  case 'czerwony':
    message = 'Czerwony to ciepły kolor.';
    break;
  case 'zielony':
    message = 'Zielony to kolor liści.';
    break;
  default:
    message = 'Co to za kolor?';
}
```

Odpowiada to zapisowi `if (favoriteColor == 'granatowy' || favoriteColor == 'niebieski' || favoriteColor == 'indygo')` w instrukcji `if-else`.

Łączenie tablic i dzielenie łańcuchów znaków

Tablice w języku JavaScript można porównać do list zakupów. Są to kolejne wartości zapisane wewnątrz jednej zmiennej. Przykładowo tablicę zawierającą nazwy wszystkich dni tygodnia można utworzyć w następujący sposób:

```
var days = ['poniedziałek','wtorek','środa','czwartek','piątek',
↪'sobota','niedziela'];
```

Każdy element takiej listy jest niezależną wartością, do której można się odwoływać, używając charakterystycznego dla tablic zapisu, opisanego na stronie 79.

Aby odwołać się do pierwszego elementu na liście, musiałbyś użyć wyrażenia `days[0]`. Od czasu do czasu może się jednak zdarzyć, że będziesz chciał pobrać wszystkie elementy listy w formie jednego łańcucha znaków. Takie rozwiązanie może się przydać na przykład do wyświetlenia całej zawartości tablicy. Możesz to zrobić, korzystając z metody `join()`. Ta metoda języka JavaScript pobiera wszystkie elementy przechowywane w tablicy i zamienia je w jeden łańcuch znaków. W tym łańcuchu wszystkie elementy tablicy są oddzielone przecinkami bądź też innym separatorem określonym w wywołaniu metody, na przykład tak:

```
var weekdays = days.join(); // poniedziałek,wtorek,środa...
```

Jeśli w wywołaniu metody `join()` nie przekażesz żadnego argumentu, poszczególne elementy tablicy będą od siebie oddzielone przecinkami. Istnieje jednak możliwość określenia innego separatora:

```
var weekdays = days.join(':'); // poniedziałek wtorek środa...
```

JavaScript nie dodaje żadnych odstępów pomiędzy elementami tablicy oraz separatorami, a zatem, jeśli chcesz, by elementy były oddzielone na przykład znakiem przecinka i odstępem, konieczne będzie przekazanie łańcucha `', '` w wywołaniu metody `join()`:

```
var weekdays = days.join(', '); // poniedziałek, wtorek, środa...
```

Z drugiej strony, jeśli tylko łańcuch znaków zawiera jakieś separatory określające początek i koniec elementów, możesz go przekształcić na tablicę, używając metody `split()`. Przykładowo załóżmy, że dysponujesz następującym łańcuchem znaków:

```
var weekdays =
        'poniedziałek,wtorek,środa,czwarek,piątek,sobota,niedziela';
```

Taki łańcuch możesz podzielić na części i przekształcić w tablicę, używając następującego wywołania:

```
var dayList = weekdays.split(','); // teraz dayList zawiera tablicę
                                   // składającą się z 7 elementów.
```

Łączenie różnych elementów

Wiesz już, że język JavaScript umożliwia wykonywanie wielu zadań. Niektóre z nich to: walidacja formularzy, dodawanie efektu podmiany rysunków, tworzenie galerii zdjęć lub wzbogacanie interfejsu użytkownika (na przykład o zestawy kart lub akordeony). Możesz się jednak zastanawiać, jak połączyć wszystkie te elementy w jednej witrynie. W końcu kiedy zaczniesz korzystać z języka JavaScript, prawdopodobnie zechcesz usprawnić każdą stronę witryny. Oto kilka wskazówek ułatwiających korzystanie w witrynie z wielu skryptów.

Używanie zewnętrznych plików JavaScript

Na stronie 49 dowiedziałeś się, że zewnętrzne pliki JavaScript umożliwiają wydajne używanie tego samego kodu JavaScript na różnych stronach witryny. Technika ta ułatwia aktualizowanie kodu JavaScript. Jeśli będziesz musiał to zrobić, wystarczy

zmodyfikować (lub naprawić) jeden plik. Ponadto pobrany plik JavaScript jest zapisywany w pamięci podręcznej przeglądarki, dlatego nie trzeba go wczytywać po raz drugi. Dzięki temu strony reagują szybciej i są krócej wczytywane.

Przy korzystaniu z bibliotek języka JavaScript (takich jak jQuery) zewnętrzne pliki JavaScript są niezbędne. Strony byłyby bardzo duże i trudne w konserwacji, gdyby na każdej z nich znalazł się kod JavaScript biblioteki jQuery. Także wtyczki są udostępniane jako zewnętrzne pliki, dlatego trzeba je dołączyć do strony, aby można było ich używać. Dołączanie zewnętrznych plików JavaScript jest bardzo proste:

```
<script src="js/jquery.min.js"></script>
```

Umieszczenie własnego kodu JavaScript w zewnętrznym pliku pomaga powtórnie wykorzystać dany program i przyspiesza działanie witryny, jednak tylko wtedy, gdy dany skrypt jest potrzebny na wielu stronach. Przypomnij sobie skrypt do walidacji formularza, który utworzyłeś na stronie 311. Nie ma sensu umieszczać go w zewnętrznym pliku, ponieważ wszystkie reguły walidacji i komunikaty o błędach są specyficzne dla elementów formularza z danej strony, dlatego nie będą działać dla formularza o innych polach. W podobnych sytuacjach najlepiej jest umieścić kod do obsługi walidacji bezpośrednio na stronie.

Jednak wtyczki walidacyjnej, którą poznałeś na stronie 301, można użyć w dowolnym formularzu, dlatego warto umieścić ją w odrębnym pliku. To samo dotyczy każdego kodu używanego w wielu miejscach. Na stronie 290 dowiedziałeś się, jak za pomocą kodu JavaScript aktywować pierwsze pole formularza. Tę technikę możesz zastosować w każdym formularzu. Podobnie rzecz się ma z metodą opisaną w ramce na stronie 292. Rozwiązanie to zapobiega wielokrotnemu przesłaniu formularza, kiedy użytkownik kilkakrotnie kliknie przycisk *Wyślij*, co może być przydatne na wielu stronach. Te dwa skrypty warto umieścić w pojedynczym zewnętrznym pliku (na przykład *forms.js*) z następującym kodem JavaScript:

```
1  $(document).ready(function() {
2    // Aktywowanie pierwszego pola tekstowego formularza.
3    $(":text")[0].focus();
4
5    // Wyłączanie przycisku przesyłania.
6    $('form').submit(function() {
7      var subButton = $(this).find(':submit');
8      subButton.attr('disabled',true);
9      subButton.val('...przesyłanie w toku...');
10   });
11 }); // Koniec funkcji ready.
```

Uwaga: Zgodnie z informacjami podanymi na stronie 190, dowolny kod JavaScript umieszczony w sekcji `<head>` strony, który wymaga biblioteki jQuery, musi zostać umieszczony wewnątrz wywołania funkcji `$(document).ready()`. Biblioteka jQuery zapewnia możliwość stosowania dowolnie wielu wywołań tej funkcji. Możesz na przykład używać kilku zewnętrznych plików JavaScript, które wykonują różne operacje na stronie i w każdym z nich możesz umieścić odrębne wywołanie funkcji `$(document).ready()`. Dodatkowo możesz umieścić wywołanie tej funkcji wewnątrz elementu `<script>` na samej stronie. Biblioteka jQuery nie będzie miała żadnych problemów z takim kodem. Jednak stosowania funkcji `$(document).ready()` można uniknąć, umieszczając znaczniki `<script>` na samym końcu strony, bezpośrednio przed zamykającym znacznikiem `</body>`.

Używanie tego samego skryptu na wielu stronach wymaga odpowiedniego planu. Na przykład wiersz 3. umieszcza kursor w pierwszym polu tekstowym na stronie. Przeważnie ma to sens — warto aktywować pierwsze pole, aby użytkownik mógł zacząć wypełniać formularz. Jednak jeśli na stronie znajduje się kilka formularzy, rozwiązanie może działać w niepożądany sposób.

Jeśli w górnej części strony umieścisz pole wyszukiwania, a poniżej odrębny formularz do składania zamówień, kod z wiersza 3. aktywuje kontrolkę do wyszukiwania, a nie pierwsze pole tekstowe formularza zamówień. W takich przypadkach trzeba zastanowić się nad takimi problemami i sprawić, aby skrypt umieszczał kursor w odpowiednim miejscu. Oto dwa możliwe rozwiązania:

- Dodaj nazwę klasy do pola, które chcesz aktywować przy wczytywaniu strony. Załóżmy, że przypiszesz polu tekstowemu klasę `focus`:

  ```
  <input type="text" class="focus" name="firstName">
  ```

  Następnie można użyć kodu JavaScript do aktywowania tego pola:

  ```
  $('.focus').focus();
  ```

  Aby użyć tej instrukcji, wystarczy dodać klasę `focus` do odpowiednich pól tekstowych na wszystkich stronach z formularzem i dołączyć do każdej z nich zewnętrzny plik JavaScript z tym kodem.

- Ten sam efekt możesz uzyskać przez dodanie nazwy klasy do samego znacznika `<form>` i użycie poniższej instrukcji:

  ```
  $('.focus :text')[0].focus();
  ```

  Ten kod automatycznie aktywuje pierwsze pole tekstowe formularza klasy `focus`. Zaletą tego podejścia jest to, że zawsze powoduje umieszczenie kursora w pierwszym polu tekstowym, dlatego jeśli zmienisz układ formularza (na przykład dodasz na jego początku kilka nowych pól tekstowych), skrypt aktywuje odpowiednią kontrolkę, a nie inny obiekt klasy `focus` w dalszej części strony.

Często zbiór skryptów jest potrzebny na wszystkich (lub prawie wszystkich) stronach witryny. Możesz na przykład przygotować efekt podmienianych obrazków (patrz strona 239), a także użyć języka JavaScript do wyświetlania odnośników do stron zewnętrznych w nowym oknie (patrz strona 260). Wtedy warto utworzyć zewnętrzny plik JavaScript ze wszystkimi skryptami działającymi w całej witrynie (możesz nazwać go na przykład *site_scripts.js* lub po prostu *site.js*).

Uwaga: Biblioteka jQuery ma mechanizm zapobiegający zgłaszaniu niektórych usterek w kodzie JavaScript. Język ten wygeneruje błąd, jeśli spróbujesz wykonać operację na nieistniejącym elemencie, na przykład zechcesz pobrać pole tekstowe na stronie, która nie zawiera takiej kontrolki. Na szczęście jQuery ignoruje takie problemy.

Tworzenie kodu JavaScript o krótkim czasie wczytywania

Kiedy zaczniesz umieszczać skrypty w zewnętrznych plikach JavaScript, odwiedzający odczują przyspieszenie wczytywania stron witryny. Dzięki pamięci podręcznej przeglądarki po pobraniu zewnętrznych plików JavaScript dla jednej strony witryny nie trzeba ponownie wczytywać ich dla następnych stron. Jednak istnieje też inny sposób na przyspieszenie wczytywania witryny. Polega on na kompresowaniu zewnętrznych plików JavaScript.

Uwaga: Pliki przesyłane bezpiecznie za pomocą *SSL* (ang. *Secure Socket Layer*) *zazwyczaj* nie są umieszczane w pamięci podręcznej. Dlatego jeśli użytkownik otwiera stronę za pomocą protokołu *https://* (na przykład przez wpisanie adresu *https://www.ore lly.com/*), to pobrane pliki — w tym zewnętrzne pliki JavaScript — najprawdopodobniej trzeba będzie wczytywać za każdym razem, kiedy będą potrzebne. (Można zmienić ustawienia na serwerze WWW, by umożliwić przechowywanie w pamięci podręcznej plików przesyłanych bezpiecznym połączeniem).

Aby skrypt był bardziej zrozumiały, programiści zwykle używają odstępów, nowych wierszy i komentarzy z opisem działania kodu. Są to elementy ważne dla programisty, jednak niekoniecznie dla przeglądarki, która potrafi przetworzyć kod JavaScript bez nowych wierszy, tabulacji, dodatkowych odstępów i komentarzy. Przy użyciu programu do kompresji można zminimalizować wielkość plików. W tej książce polecana jest *zminimalizowana* wersja biblioteki jQuery, a jej wielkość jest o około połowę mniejsza od nieskompresowanego pliku.

Istnieje kilka programów, które pozwalają skrócić kod JavaScript. Są to między innymi utworzony przez Douglasa Crockforda JSMin (*http://crockford.com/javascript/jsmin.html*) i Packer opracowany przez Deana Edwarda (*http://dean.edwards.name/packer*). Jednak preferowanym rozwiązaniem będzie skorzystanie z narzędzia używanego w witrynie Yahoo! (i jQuery), ponieważ pozwala ono znacznie zmniejszyć rozmiar pliku bez modyfikowania kodu (niektóre programy kompresujące przekształcają skrypty, co może czasem prowadzić do awarii).

Program kompresujący z witryny Yahoo!, *YUI Compressor*, jest dostępny pod adresem *http://yui.github.io/yuicompressor/*. Na szczęście dostępna jest internetowa wersja tego narzędzia, która pozwala na korzystanie z niego bez konieczności instalowania na własnym komputerze.

1. **Uruchom przeglądarkę i wyświetl w niej stronę *http://refresh-sf.com***

   To jest witryna internetowej wersji kompresora YUI.

2. **Kliknij odnośnik *File(s).***

   Ewentualnie możesz także skopiować kod JavaScript z edytora tekstów i wkleić go do dużego pola tekstowego na stronie głównej witryny; w takim przypadku przejdź bezpośrednio do kroku 4.

3. **Kliknij przycisk *Choose File* i odszukaj zewnętrzny plik JavaScript na swoim komputerze.**

Plik musi zawierać wyłącznie kod JavaScript. Nie można przykładowo wybrać pliku strony WWW, który oprócz kodu JavaScript zawiera także kod HTML.

4. **Zaznacz pole wyboru *Redirect to gzipped output* umieszczone tuż powyżej przycisku *Compress*.**

   Zaznaczenie tej opcji pozwala pobrać zminimalizowany kod, zapisany w formie nowego pliku w formacie ZIP. To będzie Twój nowy, skompresowany, zewnętrzny plik JavaScript, który powinieneś zapisać na swojej witrynie.

5. **Kliknij przycisk *Compress*.**

   Strona przetworzy przesłany kod, po czym pozwoli pobrać skompresowany plik i zapisać go na swoim komputerze. Można mu zmienić nazwę (gdyż zawsze będzie zapisywany pod nazwą *min.js*), a następnie używać na własnej witrynie. Po dokonaniu minimalizacji strona generuje także estetyczny raport przedstawiający wielkość oryginalnego pliku, wielkość nowego, spakowanego pliku oraz wartość procentową określającą, o ile udało się zmniejszyć wielkość pliku.

Ostrzeżenie: Koniecznie należy pamiętać o tym, by po skompresowaniu nie pozbywać się oryginalnego pliku JavaScript, gdyż jego zmniejszona wersja jest całkowicie nieczytelna i nie będzie można jej edytować, gdyby w przyszłości pojawiła się konieczność wprowadzenia kodzie zmian.

17
ROZDZIAŁ

Diagnozowanie i rozwiązywanie problemów

Wszyscy popełniają pomyłki, a usterki w kodzie JavaScript mogą sprawić, że skrypty nie będą działać prawidłowo (a nawet całkowicie przestaną funkcjonować). Początkujący programiści popełniają zwykle wiele błędów. Ustalanie przyczyn nieoczekiwanego działania skryptów bywa frustrujące, jest to jednak nieodłączny element programowania. Na szczęście wraz z nabywaniem doświadczenia nauczysz się określać, dlaczego pojawiły się błędy, i naprawiać je.

W tym rozdziale opisano najczęstsze pomyłki programistów, a także — co ważniejsze — sposoby diagnozowania problemów w skryptach (w języku technicznym proces ten nazywa się *debugowaniem*). Ponadto w przykładzie zobaczysz krok po kroku, jak zdiagnozować program z usterkami.

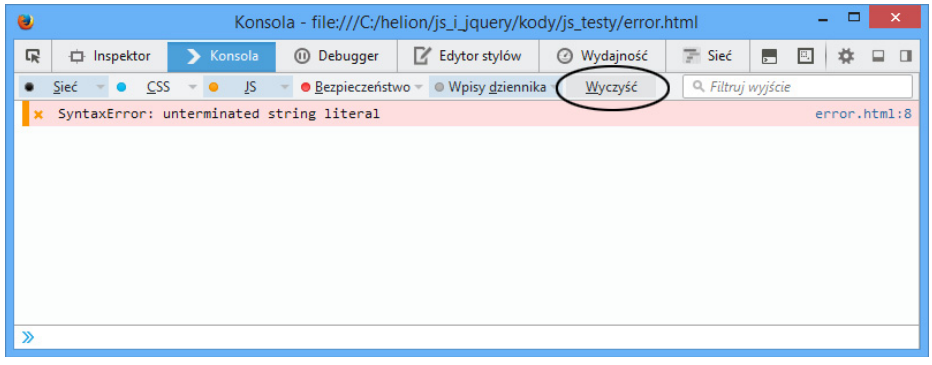
Najczęstsze błędy w kodzie JavaScript

W programach pojawiają się usterki wielu różnego rodzaju — od prostych literówek po bardziej ukryte błędy, które ujawniają się tylko w określonych warunkach. Niektóre pomyłki przydarzają się początkującym (a także doświadczonym) programistom języka JavaScript szczególnie często. Zapoznaj się z listą błędów opisanych w tym podrozdziale i pamiętaj o nich w trakcie tworzenia kodu. Znajomość tych standardowych usterek powinna pomóc Ci w wykrywaniu i rozwiązywaniu problemów we własnych programach.

Brak symboli końcowych

Jak pewnie zauważyłeś, kod JavaScript jest pełen nawiasów zwykłych i klamrowych, średników, cudzysłowów i innych znaków specjalnych. Ze względu na drobiazgową naturę komputerów pominięcie jednego takiego znaku może wstrzymać działanie programu. Jednym z najczęściej popełnianych błędów jest pominięcie zamykającego znaku specjalnego. Przykładowo instrukcja `alert('witaj';` jest nieprawidłowa, ponieważ brakuje w niej końcowego nawiasu. Poprawny zapis to `alert('witaj');`.

Pominięcie zamykającego nawiasu to błąd składniowy (patrz ramka na stronie 54). Tego rodzaju usterki „gramatyczne" uniemożliwiają uruchomienie skryptu. Kiedy spróbujesz przetestować taki kod, przeglądarka poinformuje Cię, że popełniłeś błąd składniowy. Mylące jest to, że poszczególne przeglądarki opisują usterki w różny sposób. W konsoli błędów Firefoksa (patrz strona 56) pojawia się komunikat o błędzie „SyntaxError: Missing) after argument list" [czyli: błąd składni, brak) po liście argumentów]. Internet Explorer (patrz strona 55) wyświetla informację w stylu „Oczekiwano znaku ')'". Konsola błędów przeglądarki Chrome pokazuje nieco mylący komunikat „SyntaxError: Unexpected token ;", a konsola Safari (patrz strona 57) udostępnia przydatny komunikat typu „SyntaxError: Expected token ')'". Firefox zwykle wyświetla najbardziej zrozumiałe komunikaty o błędach, dlatego warto rozpocząć analizę problemów właśnie od tej przeglądarki (patrz rysunek 17.1).



Rysunek 17.1. *Konsola błędów w Firefoksie wyświetla wszystkie błędy języka JavaScript wykryte przez przeglądarkę. Aby wyświetlić tę konsolę, wybierz z menu głównego opcję Narzędzia/Dla twórców witryn/Konsola WWW w systemie Windows (Ctrl+Shift+K) na komputerach z systemem Windows lub Narzędzia/Dla twórców witryn/Konsola WWW (⌘+Option+K) na komputerach Mac. Ponieważ znajdują się tu usterki wykryte na wszystkich stronach, warto często czyścić listę za pomocą przycisku Wyczyść (w kółku)*

Błąd składniowy w instrukcji `alert('witaj' ;` jest dobrze widoczny. Jednak jeśli w kodzie znajdują się zagnieżdżone nawiasy, łatwo pominąć zamykający znak tego typu, natomiast trudno szybko dostrzec taką pomyłkę, na przykład:

```
if ((x>0) && (y<10) {
  // Różne operacje.
}
```

W tym fragmencie brakuje ostatniego zamykającego nawiasu w instrukcji warunkowej (po wyrażeniu `(y<10)`). Pierwszy wiersz powinien wyglądać następująco: `if ((x>0) && (y<10)) {`. Firefox udostępnia najbardziej zrozumiały opis tego problemu:

„Missing) after condition" (czyli brak) po warunku). W tabeli 17.1 znajdziesz listę komunikatów o błędach składniowych, wyświetlanych przez konsolę błędów przeglądarki Firefox.

Tabela 17.1. *Komunikaty, które najczęściej można zobaczyć w konsoli błędów przeglądarki Firefox, oraz ich znaczenie*

Komunikat o błędzie	Wyjaśnienie

`Unterminated string literal`	Brak otwierającego lub zamykającego cudzysłowu:
	`var name = Janek';`
	Ten błąd pojawia się też przy braku dopasowania ograniczników:
	`var name = 'Janek";`

`Missing ) after argument list`	Brak zamykającego nawiasu w wywołaniu funkcji lub metody: `alert('witaj' ;`
`Missing ) after condition`	Brak zamykającego nawiasu w instrukcji warunkowej: `if (x==0`
`Missing ( before condition`	Brak otwierającego nawiasu w instrukcji warunkowej: `if x==0)`

`Missing } in compound statement`	Brak zamykającego nawiasu klamrowego w instrukcji warunkowej:
	`if (score == 0) {`
	`  alert('Koniec gry');`
	`  // Brak znaku } w tym wierszu.`

`Missing } after property list`	Brak zamykającego nawiasu klamrowego w obiekcie JavaScript:
	`var x = {`
	`  fName: 'Robert',`
	`  lName: 'Kowalski'`
	`  // Brak znaku } w tym wierszu.`

`Syntax error`	Ogólny problem, który uniemożliwił interpreterowi odczytanie skryptu.
`Missing ; before statement`	Informuje o uruchomieniu dwóch instrukcji w jednym wierszu bez rozdzielenia ich średnikiem. Może wynikać także z błędnego zagnieżdżenia apostrofów lub cudzysłowów: `var message='Mike'u, tu wystąpił błąd.';`
`Missing variable name`	Wynika z próby użycia zarezerwowanego słowa języka JavaScript (patrz strona 65) jako nazwy zmiennej: `var if="Błąd składniowy.";`

Błąd składniowy wystąpi także wtedy, jeśli zapomnisz podać drugi cudzysłów lub apostrof. I tak instrukcja `alert('witaj);` jest nieprawidłowa, ponieważ brakuje ostatniego apostrofu (poprawny zapis to `alert('witaj');`). Firefox wyświetli wtedy komunikat „Unterminated string literal", natomiast Internet Explorer — tekst „Brak zakończenia stałej znakowej". Przeglądarki Chrome oraz Safari, kiedy napotkają taki błąd, wyświetlą komunikat: „SyntaxError: Unexpected token ILLEGAL".

Także nawiasy klamrowe występują w parach. Są one potrzebne w instrukcjach warunkowych (patrz strona 93), pętlach (patrz strona 93), przy tworzeniu funkcji (patrz strona 115) i w obiektach JSON (patrz strona 500):

```
if (score==0) {
  alert('Koniec gry');
```

W tym fragmencie brakuje zamykającego znaku }, dlatego w skrypcie pojawi się błąd składniowy.

Jednym ze sposobów na przezwyciężenie problemu brakujących znaków przestankowych jest wpisywanie obu symboli przed wprowadzeniem dalszego kodu. Załóżmy, że chcesz dodać następujący fragment:

```
if ((name=='robert') && (score==0)) {
  alert('Przegrałeś, ale przynajmniej masz piękne imię');
}
```

Uwaga: Wiele dobrych edytorów tekstu udostępnia mechanizm kolorowania składni, który automatycznie zaznacza pary odpowiadających sobie nawiasów, nawiasów kwadratowych oraz klamrowych. Niektóre z nich potrafią nawet zaznaczać brakujące znaki przestankowe, dzięki czemu znacznie łatwiej i szybciej można poprawiać błędy.

Najpierw wpisz zewnętrzne elementy, aby utworzyć szkielet instrukcji warunkowej:

```
if ( ) {

}
```

Na tym etapie prawie nie ma kodu, dlatego łatwo zauważyć, czy wpisano wszystkie znaki specjalne. Następnie dodawaj krok po kroku dalszy kod do czasu utworzenia całego programu. W ten sam sposób warto tworzyć złożone literały obiektowe języka JavaScript, używane na przykład do ustawiania opcji wtyczki Validation (patrz strona 299) lub tworzenia obiektów JSON (patrz strona 500). Zacznij od podstawowej struktury:

```
var options = {

};
```

Następnie rozwiń ją:

```
var options = {
  rules : {

  },
  messages : {

  }
};
```

Teraz można dokończyć obiekt:

```
var options = {
  rules : {
    name : 'required',
    email: 'email'
  },
  messages : {
    name : 'Podaj nazwę użytkownika',
    email: 'Podaj adres e-mail'
  }
};
```

To podejście pozwala sprawdzić kod na różnych etapach i znacznie ułatwia wykrywanie błędów związanych ze znakami specjalnymi. Konsola błędów przeglądarki Firefox (przedstawiona na stronie 56) udostępnia najbardziej zrozumiałe opisy błędów. Jeśli skrypt nie działa, warto wyświetlić stronę w tej przeglądarce i zajrzeć do konsoli błędów. Kilka najczęściej spotykanych błędów zostało przedstawionych w tabeli 17.1.

WIEDZA W PIGUŁCE

Rodzaje błędów

Są trzy podstawowe kategorie błędów występujące w programach w języku JavaScript. Niektóre usterki są natychmiast widoczne, natomiast inne można wykryć dopiero po uruchomieniu skryptu.

- **Błędy składniowe**. Usterki tego typu to pomyłki gramatyczne, które sprawiają, że kod jest niezrozumiały dla interpretera. Błędy tego rodzaju to efekt między innymi braku zamykających nawiasów zwykłych lub klamrowych albo cudzysłowów. Przeglądarka natychmiast wykrywa takie usterki, dlatego nie uruchamia skryptu. Komunikaty o błędach składniowych pojawiają się w konsoli błędów przeglądarki.
- **Błędy czasu wykonania**. Także kiedy przeglądarka z powodzeniem wczyta skrypt, a interpreter go przetworzy, nadal mogą pojawić się problemy. Nawet jeśli składnia programu jest poprawna, mogą wystąpić błędy *czasu wykonania*. Załóżmy, że na początku skryptu utworzyłeś zmienną `message`. W dalszej części programu kod dodaje do rysunku funkcję obsługi zdarzenia `click`, aby po kliknięciu obrazka pojawiało się okno dialogowe. W tej funkcji może znajdować się instrukcja `alert(MESSAGE);`. Jej składnia jest prawidłowa, jednak użyto tu nazwy `MESSAGE` zamiast `message` (duże litery zamiast małych). Ten kod nie jest nieprawidłowy, jednak odwołuje się do zmiennej MESSAGE zamiast message. Zgodnie z tym, czego dowiedziałeś się na stronie 64, język JavaScript uwzględnia wielkość znaków, dlatego nazwy `MESSAGE` i `message` oznaczają dwie różne zmienne. Kiedy użytkownik kliknie rysunek, interpreter spróbuje znaleźć nieistniejącą zmienną `MESSAGE` i zgłosi błąd czasu wykonania.

  Inny często spotykany błąd czasu wykonania występuje przy próbie dostępu do elementu strony, który albo nie istnieje, albo nie został jeszcze wczytany do pamięci przeglądarki. Opis tego problemu znajdziesz w omówieniu funkcji `$(document).ready()` biblioteki jQuery (patrz strona 190).
- **Błędy logiczne**. Czasem skrypt działa, ale w nieoczekiwany sposób. W kodzie może znajdować się instrukcja `if-else` (patrz strona 93), która wykonuje zadanie A, jeśli warunek jest spełniony, i operację B, jeśli warunek jest fałszywy. Niestety, okazuje się, że program nigdy nie uruchamia kodu B, nawet jeśli warunek jest w oczywisty sposób fałszywy. Błędy tego rodzaju to wynik niepoprawnego użycia operatora równości (patrz strona 67). Dla interpretera języka JavaScript kod jest technicznie poprawny, jednak programista popełnił w logice programu błąd, który uniemożliwia prawidłowe działanie skryptu.

  Inny przykład błędu logicznego to pętla nieskończona. Jest to fragment kodu działający w *nieskończoność*, co zwykle powoduje „zawieszenie" programu, a nawet awarię przeglądarki. Oto przykładowa pętla nieskończona:

```
for (var i=1; i>0; i += 1) {
  // Ten kod będzie działał w nieskończoność.
}
```

Pętla ta będzie działać, dopóki warunek `i>0` będzie spełniony. Ponieważ zmienna `i` ma początkowo wartość `1` (`var i=1`), a każde uruchomienie pętli powoduje zwiększenie jej o 1 (`i += 1`), wartość zmiennej będzie zawsze większa od 0. Oznacza to, że pętla nigdy nie przerwie działania (aby przypomnieć sobie informacje o pętlach `for`, zajrzyj na stronę 109).

Błędy logiczne są zwykle najtrudniejsze do wykrycia. Jednak korzystając z technik diagnostycznych, opisanych na stronie 621, będziesz w stanie wykryć i rozwiązać wiele często występujących problemów.

Cudzysłowy i apostrofy

Początkujący programiści często mają problemy z cudzysłowami i apostrofami. Symbole te służą do tworzenia łańcuchów liter i innych znaków (i są nazywane *literałami łańcuchowymi*). Takich ciągów można używać jako komunikatów na stronach lub zmiennych w programach. JavaScript, podobnie jak inne języki programowania, umożliwia tworzenie literałów łańcuchowych za pomocą *cudzysłowów* i *apostrofów*. Poniższa instrukcja:

```
var name="Janek";
```

oznacza to samo co następna:

```
var name='Janek';
```

W poprzednim punkcie dowiedziałeś się, że trzeba użyć cudzysłowu otwierającego i zamykającego. Jeśli o tym zapomnisz, Firefox wyświetli komunikat „Unterminated string literal" (także inne przeglądarki nie uruchomią błędnego skryptu). Ponadto, co opisano na stronie 62, należy używać pasujących do siebie ograniczników, na przykład dwóch apostrofów lub dwóch cudzysłowów. Dlatego instrukcja `var name='Janek` spowoduje błąd.

Inny często spotykany problem związany jest z używaniem cudzysłowów i apostrofów w łańcuchach znaków. Bardzo łatwo można popełnić następujący błąd:

```
var message='Mike'u, tu kryje się błąd.';
```

Zwróć uwagę na apostrof w słowie „Mike'u". Interpreter potraktuje ten znak jak zamykający apostrof, dlatego wykryje instrukcję `var message='Mike'`, a pozostałą część wiersza uzna za błędną. W konsoli błędów Firefoksa pojawi się komunikat „Missing ; before statement", ponieważ przeglądarka potraktuje drugi apostrof jak koniec prostej instrukcji języka JavaScript, a dalszy kod — jak następne polecenie.

Możesz uniknąć takich problemów na dwa sposoby. Pierwszy z nich polega na łączeniu apostrofów i cudzysłowów. Możesz otoczyć cudzysłowami łańcuch znaków z apostrofami lub na odwrót. Na przykład wcześniejszy błąd można naprawić w następujący sposób:

```
var message="Mike'u, problem został rozwiązany.";
```

Jeśli łańcuch znaków zawiera cudzysłowy, można użyć poniższej techniki:

```
var message='Jacek powiedział: "Rozwiązałem problem".';
```

Inne podejście polega na użyciu w łańcuchu znaków *sekwencji ucieczki* z apostrofem lub cudzysłowem. Technikę tę szczegółowo opisano na stronie 62, a tu znajdziesz krótkie przypomnienie. Aby utworzyć sekwencję ucieczki, poprzedź dany znak specjalny ukośnikiem:

```
var message='Mike\'u, ten zapis jest poprawny.';
```

Interpreter traktuje sekwencję `\'` jak znak apostrofu, a nie jak symbol służący do otwierania i zamykania łańcuchów znaków.

Używanie słów zarezerwowanych

Na stronie 65 wymieniono długą listę słów zarezerwowanych dla języka JavaScript. Są to słowa używane w składni języka, na przykład `if`, `do`, `for` i `while`, a także właściwości obiektu przeglądarki, między innymi `alert`, `location`, `window` i `document`.

Poniższy kod wywoła błąd składniowy:

```
var if = "To nie zadziała.";
```

Ponieważ słowo `if` służy do tworzenia instrukcji warunkowych, na przykład `if (x==0)`, nie można nazwać w ten sposób zmiennej. Jednak niektóre przeglądarki nie wygenerują błędów, jeśli w nazwie zmiennej użyjesz słowa z obiektowego modelu przeglądarek. Przykładowo słowo `document` określa dokument HTML. Przeanalizujmy następujący fragment kodu:

```
var document='Dzieje się coś dziwnego.';
alert(document);
```

Podczas próby wykonania takiego kodu przeglądarki nie wygenerują błędu, a jedynie okienko komunikatu z tekstem „[object HTMLDocument]", który nie odnosi się bezpośrednio do obiektu dokumentu HTML. Innymi słowy, przeglądarki te nie pozwolą na nadpisanie obiektu dokumentu łańcuchem znaków.

Pojedynczy znak równości w instrukcjach warunkowych

Instrukcje warunkowe (patrz strona 93) umożliwiają programom reagowanie w różny sposób w zależności od wartości zmiennej, stanu elementu na stronie lub innych warunków występujących w skrypcie. Instrukcja warunkowa może wyświetlać rysunek, *jeśli* jest ukryty, a *w przeciwnym razie* — ukrywać go. Warunki mogą być tylko prawdziwe (`true`) lub fałszywe (`false`). Niestety, łatwo utworzyć instrukcję, w której warunek jest zawsze spełniony:

```
if (score=100) {
  alert('Wygrałeś!');
}
```

Ten kod ma sprawdzać wartość zmiennej `score`. Jeśli wynosi ona `100`, powinno pojawić się okno dialogowe z wiadomością „Wygrałeś!". Jednak ten fragment wyświetli takie okienko *zawsze*, niezależnie od wartości zmiennej `score` przed uruchomieniem instrukcji warunkowej. Dzieje się tak, ponieważ pojedynczy znak równości to operator *przypisania*, a więc instrukcja `score=100` przypisze wartość `100` do zmiennej `score`. Interpreter potraktuje operację przypisania jak wartość `true` i nie tylko wyświetli komunikat w oknie dialogowym, ale też zmieni wartość zmiennej `score` na `100`.

Aby uniknąć tego błędu, należy zawsze używać *dwóch* znaków równości przy sprawdzaniu, czy dwie wartości są takie same:

```
if (score==100) {
  alert('Wygrałeś!');
}
```

Wielkość znaków

Pamiętaj, że język JavaScript uwzględnia wielkość znaków. Interpreter sprawdza nie tylko litery użyte w nazwach zmiennych, funkcji, metod i słów kluczowych, ale też ich wielkość. Dlatego dla interpretera instrukcje `alert('hej')` i `ALERT('hej')` nie są tym samym. Pierwsze polecenie, `alert('hej')`, wywołuje wbudowane polecenie `alert()` przeglądarki, natomiast druga instrukcja, `ALERT('hej')`, spowoduje wywołanie funkcji `ALERT()`, zdefiniowanej przez użytkownika.

Przy korzystaniu z rozwlekłych metod pobierania elementów modelu DOM, `get ↪ElementsByTagName()` i `getElementById()`, mogą wystąpić problemy, ponieważ nazwy tych metod są zapisywane przy użyciu zarówno małych, jak i dużych liter (co jest kolejnym powodem przemawiającym za korzystaniem wyłącznie z biblioteki jQuery). Także przy stosowaniu dużych i małych liter w nazwach zmiennych oraz funkcji mogą czasem pojawić się kłopoty.

Jeśli zobaczysz komunikat o błędzie „*x* is not defined" (gdzie *x* to nazwa zmiennej, funkcji lub metody), problem może wynikać z nieodpowiedniej wielkości znaków.

Nieprawidłowe ścieżki do zewnętrznych plików JavaScript

Inny często pojawiający się błąd to niepoprawne ścieżki do zewnętrznych plików JavaScript. Na stronie 49 opisano, jak należy dołączać takie pliki do stron — trzeba wskazać odpowiedni plik we właściwości `src` znacznika `<script>`. Dlatego w sekcji `<head>` strony HTML należy umieścić tag `<script>`:

```
<script src="site_js.js"></script>
```

Właściwość `src` działa jak atrybut `href` odnośników i wskazuje ścieżkę do pliku JavaScript. Jak wspomniano w ramce „Rodzaje adresów URL", są trzy sposoby wskazywania plików: ścieżki bezwzględne (*http://www.site.com/site_js.js*), podane względem katalogu głównego (*/site_js.js*) i podane względem dokumentu (*site_js.js*).

Ścieżki określane względem dokumentu opisują, jak przeglądarka ma przejść od bieżącego dokumentu (strony WWW) do konkretnego pliku. Odnośniki tego rodzaju są używane często, ponieważ umożliwiają przetestowanie strony i kodu JavaScript bezpośrednio na komputerze. Jeśli użyjesz odsyłaczy podanych względem katalogu głównego, na potrzeby testów będziesz musiał zainstalować serwer sieciowy na komputerze (lub przenieść pliki na serwer). Podczas stosowania adresów określanych względem katalogu głównego w celu testowania stron na własnym komputerze konieczne będzie zainstalowanie i uruchomienie na nim serwera WWW (lub przeniesienie stron na taki serwer w celu ich przetestowania).

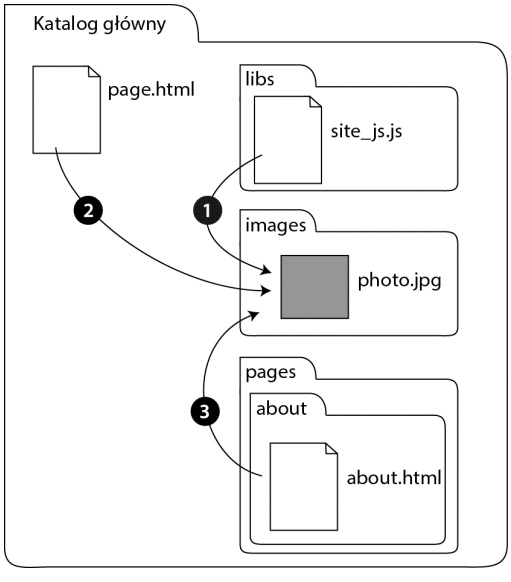
Więcej informacji o ścieżkach znajdziesz na stronie 45. Jednak, ogólnie rzecz biorąc, jeśli używasz zewnętrznych plików JavaScript i odkryjesz, że skrypt nie działa, dokładnie sprawdź, czy wpisałeś poprawne ścieżki.

Wskazówka: Jeśli korzystasz z biblioteki jQuery i w konsoli błędów Firefoksa zobaczysz komunikat „$ is not defined", prawdopodobnie nie dołączyłeś prawidłowo pliku *jquery.js* (patrz strona 135).

Nieprawidłowe ścieżki w zewnętrznych plikach JavaScript

Inny problem pojawia się przy używaniu ścieżek podawanych względem dokumentu w zewnętrznych plikach JavaScript. Skrypt może wyświetlać na stronie rysunki (na przykład pokaz slajdów lub obrazek wybrany losowo w danym dniu). Jeśli rysunki są wskazywane względem dokumentu, mogą wystąpić problemy, jeżeli ścieżki znajdują się w zewnętrznym pliku JavaScript. Dlaczego? Kiedy przeglądarka dołącza zewnętrzny plik JavaScript do strony, punktem wyjścia w ścieżkach podanych względem dokumentu jest dana strona. Dlatego każdą taką ścieżkę należy zapisać względem *strony*, a nie pliku JavaScript.

Oto prosty przykład ilustrujący ten problem. Na rysunku 17.2 widoczna jest struktura bardzo prostej witryny. Składa się ona z dwóch stron (*page.html* i *about.html*), czterech katalogów (*libs*, *images*, *pages* i *about*), zewnętrznego pliku JavaScript (*site_js.js* w katalogu *libs*) i rysunku (*photo.jpg* w katalogu *images*). Załóżmy, że w pliku *site_js.js* znajduje się ścieżka do pliku *photo.jpg*, potrzebna na przykład do wstępnego pobrania obrazka (patrz strona 242) lub dynamicznego wyświetlenia go na stronie.



Rysunek 17.2. *Ścieżki podawane względem dokumentu zależą od lokalizacji plików wyjściowego i docelowego. Na przykład ścieżka tego typu z pliku site_js.js do rysunku photo.jpg (numer 1) to ../images/photo.jpg. Ścieżka do tego samego obrazka ze strony page.html (numer 2) to images/photo.jpg, a z pliku about.html — ../../images/photo.jpg*

W pliku *site_js.js* ścieżka do rysunku *photo.jpg* podana względem dokumentu to `../images/photo.jpg` (numer 1 na rysunku 14.2). Ta ścieżka nakazuje przeglądarce wyjście z katalogu *libs* (`../`), wejście do folderu *images* (`images/`) i pobranie pliku *photo.jpg*. Jednak na stronie *page.html* ścieżka do obrazka (numer 2 na rysunku 14.2) to tylko `images/photo.jpg`. Oznacza to, że ścieżka do tego samego zdjęcia jest w obu plikach inna.

Jeśli chcesz użyć skryptu *site_js.js* na stronie *page.html*, musisz podać ścieżkę numer 2, aby określić lokalizację pliku *photo.jpg* (ścieżkę trzeba podać względem dokumentu *page.html*). Oznacza to też, że pliku *site_js.js* nie można użyć na stronie znajdującej się w dowolnym innym katalogu witryny, ponieważ ścieżka względna musi być wtedy inna (numer 3 na rysunku 17.2).

Istnieje kilka sposobów na rozwiązanie tego problemu. Przede wszystkim możliwe jest, że nigdy nie natrafisz na taką sytuację, ponieważ nie będziesz umieszczał w plikach JavaScript ścieżek do innych dokumentów. Jeśli jednak chcesz stosować tę technikę, powinieneś używać ścieżek podawanych względem katalogu głównego (patrz strona 45), które są takie same dla wszystkich stron witryny. Inna możliwość to określenie ścieżki do pliku na poszczególnych stronach WWW. Możesz na przykład dołączyć do każdej z nich zewnętrzny plik JavaScript i zdefiniować zmienną przechowującą ścieżkę do odpowiedniego pliku podaną względem dokumentu (danej strony).

I w końcu możesz także zastosować podejście użyte w pokazie slajdów ze strony 249. Ścieżki są tam zapisane w odnośnikach na poszczególnych stronach, a kod JavaScript pobiera odpowiednie ścieżki z kodu HTML. Jeśli dana ścieżka działa na stronie, będzie poprawna także w skrypcie.

Znikające zmienne i funkcje

Czasem możesz napotkać błąd typu „*x* is not defined", gdzie *x* to nazwa zmiennej lub wywoływanej funkcji. Usterka ta może wynikać z błędnego wpisania nazwy zmiennej lub funkcji albo użycia liter nieodpowiedniej wielkości. Jednak jeśli zajrzysz do kodu i stwierdzisz, że dana jednostka jest prawidłowo zdefiniowana w skrypcie, mógł wystąpić problem z zasięgiem.

Zasięg zmiennych i funkcji opisano szczegółowo na stronie 121. Warto pamiętać, że jeśli zmienna została zdefiniowana wewnątrz funkcji, będzie dostępna tylko w niej (i w innych funkcjach zagnieżdżonych w funkcji głównej). Oto prosty przykład:

```
1 function sayName(name) {
2   var message = 'Twoje imię to ' + name;
3 }
4 sayName();
5 alert(message); // Błąd — nazwa message jest niezdefiniowana.
```

Zmienna `message` jest zdefiniowana w funkcji `sayName()`, dlatego istnieje tylko w niej. Poza funkcją zmienna jest niedostępna, dlatego przy próbie jej użycia w wierszu 5. wystąpi błąd.

Ten problem może pojawić się także przy korzystaniu z jQuery. Na stronie 190 dowiedziałeś się, jak ważna przy stosowaniu tej biblioteki jest funkcja `$(document).ready()`. Wszystkie instrukcje z tej funkcji są uruchamiane dopiero po wczytaniu kodu HTML strony. Jeśli zdefiniujesz w metodzie `$(document).ready()` zmienne i funkcje, a następnie spróbujesz użyć ich poza nią, pojawi się problem:

```
$(document).ready(function() {
  var msg = 'witaj';
});
alert(msg); // Błąd — msg jest niezdefiniowana.
```

Dlatego kiedy używasz jQuery, pamiętaj o umieszczeniu całego kodu w funkcji `$(document).ready()`:

```
$(document).ready(function() {
  var msg = 'witaj';
  alert(msg); // Zmienna msg jest dostępna.
});
```

WIEDZA W PIGUŁCE

Jak zmniejszyć liczbę błędów?

Najlepszy sposób na radzenie sobie z błędami w programach to szybkie ich wykrywanie. Jeśli zaczniesz testy z wykorzystaniem przeglądarki dopiero po napisaniu 300-wierszowego skryptu, znalezienie przyczyny problemu może być naprawdę trudne. Oto dwie najważniejsze techniki zapobiegania usterkom.

- Tworzenie skryptów w krótkich fragmentach. Jak już prawdopodobnie zauważyłeś, programy JavaScript bywają mało czytelne z uwagi na znaki `}`, `)`, `'`, instrukcje `if`, `else`, funkcje i tak dalej. Nie próbuj pisać całego skryptu naraz (chyba że jesteś naprawdę dobrym programistą, program jest krótki lub czujesz, że masz szczęście). Jest tak wiele źródeł potencjalnych błędów w kodzie, że warto rozwijać skrypty stopniowo.

  Załóżmy, że obok pola tekstowego chcesz wyświetlić liczbę wpisanych w nim znaków. Rozwiązanie to jest stosowane w witrynach, w których liczba znaków w polu jest ograniczona na przykład do 300. Za pomocą języka JavaScript można łatwo wykonać opisane zadanie, jednak składa się ono z kilku kroków: reagowania na wystąpienie zdarzenie `keydown` (kiedy użytkownik wpisze literę w polu), odczytywania wartości z pola, zliczania wprowadzonych znaków i wyświetlania ich liczby na stronie.

  Możesz spróbować napisać cały skrypt za jednym razem, jednak warto najpierw utworzyć kod dla etapu 1. (reagowanie na zdarzenie `keydown`), a następnie przetestować go w przeglądarce (może Ci w tym pomóc wykorzystanie polecenia `alert()` lub funkcji `console.log()`, opisanej na następnej stronie, wykonywanych w odpowiedzi na zdarzenie `keydown`). Jeśli pierwszy fragment działa, można przejść do etapu 2., przetestować kod i tak dalej.

  Wraz z nabywaniem doświadczenia nie będziesz musiał testować tak krótkich fragmentów. Warto wtedy napisać od razu kilka części skryptu, a następnie przetestować jego większą porcję.

- Częste testowanie. Należy często testować skrypty w przeglądarce. Warto to robić przynajmniej po ukończeniu każdego fragmentu programu, co opisano w poprzednim punkcie. Ponadto należy sprawdzić program w różnych przeglądarkach: Internet Explorerze 8 i nowszych, oraz w najnowszych wersjach przeglądarek Firefoks, Chrome i Safari oraz w innych aplikacjach, których mogą używać osoby odwiedzające daną witrynę.

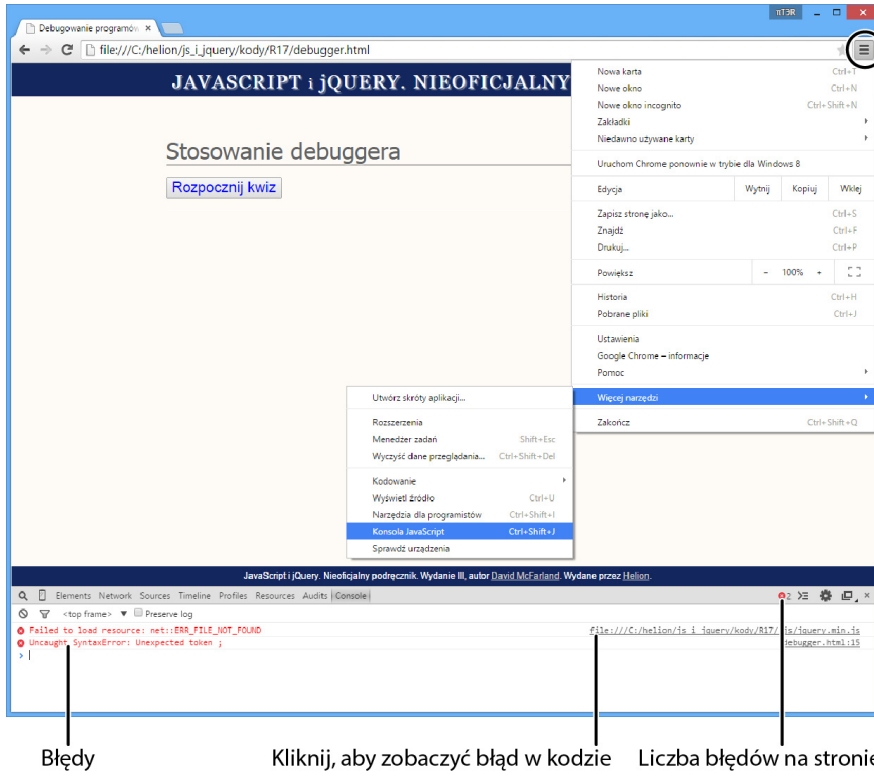
Testowanie aplikacji przy użyciu konsoli

Jeśli jeszcze nie używałeś konsoli JavaScript przeglądarki, nie miałeś okazji poznać jednego z najlepszych narzędzi dla twórców stron WWW. Wszystkie nowoczesne przeglądarki dysponują takimi konsolami, które mogą pomóc w poprawianiu kodów HTML, CSS oraz JavaScript.

Otwieranie konsoli

Aby skorzystać z konsoli, najpierw należy wyświetlić stronę w przeglądarce. Następnie trzeba wykonać jedną z poniższych czynności.

- **Google Chrome:** Kliknij przycisk ustawień przeglądarki (zakreślony na rysunku 17.3), a następnie wybierz opcję *Więcej narzędzi/Konsola JavaScript*. Możesz także nacisnąć kombinację klawiszy *Ctrl+Shift+J* (w systemie Windows) lub *⌘+Options+J* (w systemie Mac OS).



Rysunek 17.3. *Konsola JavaScript wyświetla wszystkie błędy, które wystąpiły na bieżącej stronie. W przedstawionej obok przeglądarce Chrome można ją wyświetlić, klikając przycisk widoczny z prawej strony paska adresu (zakreślony) i wybierając opcję Więcej narzędzi/Konsola JavaScript. Aby zobaczyć wiersz, w którym wystąpił błąd, należy kliknąć fragment kodu bezpośrednio pod komunikatem o nim. Przeglądarka wyświetli kartę Sources i podświetli wiersz, w którym wystąp ł błąd*

- **Internet Explorer:** Naciśnij klawisz *F12*, aby wyświetlić okno narzędzi deweloperskich, a następnie przejdź na kartę *Konsola*.
- **Firefox:** Aby w systemie Windows wyświetlić konsolę, kliknij przycisk *Otwórz menu* dostępny z prawej strony paska narzędzi przeglądarki, następnie ikonę *Narzędzia* i wybierz opcję *Konsola WWW*. W systemie Mac OS wybierz opcję *Narzędzia/Dla twórców witryn/Konsola WWW*. Możesz także skorzystać z kombinacji klawiszy: *Ctrl+Shift+I* (w systemie Windows) lub *⌘+Option+K* (w systemie Mac OS).
- **Safari:** Konsolę błędów możesz wyświetlić z menu *Programowanie*, wybierając opcje *Programowanie/Pokaż konsolę błędów* (lub naciskając kombinację klawiszy *Ctrl+Alt+C* w systemie Windows lub *⌘+Option+C* w systemie Mac OS). Jednak to menu nie jest domyślnie dostępne po zainstalowaniu przeglądarki, dlatego też trzeba je włączyć w oknie dialogowym właściwości. W tym celu kliknij przycisk wyświetlający menu ustawień przeglądarki i wybierz opcję *Preferencje*. W wyświetlonym oknie dialogowym przejdź na kartę *Zaawansowane*, zaznacz pole wyboru *Pokazuj menu Programowanie w pasku menu* i zamknij okno dialogowe.

Po ponownym uruchomieniu przeglądarki opcja *Programowanie* zostanie wyświetlona w menu głównym pomiędzy opcjami *Zakładki* oraz *Okno* (w systemie Mac OS), natomiast w systemie Windows będzie dostępna w menu wyświetlanym po kliknięciu przycisku ustawień strony. Z tego menu wybierz opcję *Pokaż konsolę błędów*.

Przeglądanie błędów przy użyciu konsoli

W oknie konsoli wyświetlane są wszystkie błędy, jakie wystąpią w kodzie JavaScript. Jako pierwsze przeglądarka wyświetla odszukane błędy syntaktyczne. Zgodnie z informacjami podanymi na stronie 51, błędy syntaktyczne w programach komputerowych przypominają nieco błędy gramatyczne. Kiedy interpreter języka JavaScript w przeglądarce odnajdzie taki błąd, zaprzestaje dalszego analizowania i wykonywania skryptu. Interpreter poinformuje o błędzie, lecz jeśli na stronie będzie ich więcej, nie dowiemy się o nich aż do momentu, gdy poprawimy pierwszy z nich.

Po poprawieniu wszystkich błędów syntaktycznych mogą się pojawić błędy *czasu wykonywania* (ang. *runtime errors*; patrz strona 606), czyli błędy, o których przeglądarka informuje w trakcie wykonywania skryptu. Taki błąd może na przykład spowodować próba odczytania wartości zmiennej, która nigdy nie została utworzona. Konsola może stanowić naszą pierwszą linię obrony podczas prób lokalizowania i poprawiania błędów w kodzie JavaScript (patrz rysunek 17.3).

Śledzenie działania skryptu za pomocą funkcji console.log()

Kiedy skrypt zacznie działać, funkcjonuje jak czarna skrzynka. Programista nie wie, co dzieje się w programie, i widzi tylko końcowe efekty, na przykład komunikat na stronie, okno wyskakujące i tak dalej. Nie zawsze można sprawdzić, czy pętla działa prawidłowo lub jaką wartość ma zmienna w danym momencie.

Programiści języka JavaScript od dawna używają metody `alert()` do wyświetlania okienek z aktualną wartością zmiennych (patrz strona 46). Aby ustalić, jakie dane zapisano w zmiennej `elementName` w pętli, możesz umieścić w niej polecenie `alert(elementName);`. Jest to jeden ze sposobów na zajrzenie do czarnej skrzynki skryptu. Jednak okna dialogowe przeszkadzają w pracy. Aby je ukryć, musisz je kliknąć, a jeśli program uruchomi pętlę 20 razy, będziesz musiał zamknąć naprawdę wiele okienek.

Konsola JavaScript udostępnia lepszy sposób obserwacji programu. Nie tylko wyświetla błędy (patrz poprzedni punkt), ale też służy do wyświetlania komunikatów z programu. Funkcja `console.log()` działa podobnie jak `document.write()` (patrz strona 48), ale zamiast wyświetlać informacje na stronie, zapisuje je w konsoli.

Wskazówka: Wszystkie nowoczesne przeglądarki obsługują metodę `console.log()`. A zatem można jej używać w przeglądarkach Chrome, Safari, Internet Explorer oraz Opera.

Aby wyświetlić w konsoli aktualną wartość zmiennej `elementName`, możesz użyć następującej instrukcji:

```
console.log(elementName);
```

Ta metoda — w odróżnieniu od polecenia `alert()` — nie zakłóca działania programu, a jedynie dodaje komunikat do konsoli.

Aby rejestrowane komunikaty były bardziej zrozumiałe, można dołączyć do nich łańcuch znaków z dodatkowym tekstem. Jeśli na przykład chcesz sprawdzić, jaką wartość ma zmienna `name` w danym miejscu programu, możesz użyć funkcji `console.log()` w następujący sposób:

```
console.log(name);
```

Możesz też poprzedzić wartość zmiennej informacją:

```
console.log('Nazwa użytkownika: ', name);
```

To wywołanie spowoduje wyświetlenie w konsoli pojedynczego wiersza tekstu, zawierającego łańcuch `Nazwa użytkownika:` oraz zawartość zmiennej `name`.

Ale co zrobić w sytuacji, gdy wartość zmiennej chcemy umieścić gdzieś *wewnątrz* łańcucha? Załóżmy, że rejestrujemy wynik zdobyty przez użytkownika i chcemy wyświetlić w konsoli zrozumiały komunikat, taki jak: „Janek zdobył 50 punktów". Innymi słowy, zależy nam na tym, by na początku łańcucha znaków umieścić imię gracza, a wynik — gdzieś pośrodku łańcucha.

W tym celu w wywołaniu funkcji `console.log()` należy zapisać: łańcuch znaków zawierający sekwencje `%s` (po jednej dla każdej zmiennej, której wartość chcemy wyświetlić), przecinek, nazwę zmiennej, przecinek i w końcu nazwę drugiej zmiennej. Oto przykład takiego wywołania:

```
console.log('%s zdobył %s punktów', name, score);
```

Sekwencja `%s` oznacza: „Wstaw zamiast mnie wartość zmiennej". Innymi słowy, skrypt zastąpi pierwsze wystąpienie symbolu `%s` wartością zmiennej `name`, a drugie — wartością zmiennej `score`.

Funkcja `log()` służy jedynie do zbierania informacji o działaniu skryptu w trakcie jego *rozwijania*. Kiedy program jest gotowy, należy usunąć z programu wszystkie wywołania `console.log()`.

Przykład — korzystanie z konsoli

W tym przykładzie dowiesz się, jak użyć funkcji `console.log()` do sprawdzenia, co dzieje się w programie. Analizowany skrypt będzie wyświetlał liczbę znaków wpisanych w polu tekstowym formularza.

Uwaga: Informacje o pobieraniu przykładowych plików znajdziesz na stronie 46.

W przedstawionym tu przykładzie będziemy używać przeglądarki Google Chrome. Możesz go także wykonać w swojej ulubionej przeglądarce, lecz jej działanie może się nieco różnić od opisanego poniżej.

1. **Otwórz w edytorze tekstu plik *console.html*.**

   Skrypt wymaga do działania biblioteki jQuery, dlatego jej zewnętrzny plik JavaScript jest już dołączony do strony. Dodano także otwierający i zamykający znacznik `<script>`. Teraz należy wprowadzić funkcję `$(document).ready()` biblioteki jQuery.

2. **Między znacznikami `<script>` w górnej części strony wpisz kod wyróżniony pogrubieniem:**

```
<script>
$(document).ready(function() {

});  // koniec funkcji ready
</script>
```

   Funkcję `$(document).ready()` poznałeś na stronie 190. Sprawia ona, że przeglądarka wczytuje cały kod strony przed uruchomieniem programu JavaScript. Najpierw użyj funkcji `console.log()`do wyświetlenia komunikatu o uruchomieniu funkcji `.ready()`.

3. **Dodaj do skryptu kod wyróżniony pogrubieniem:**

```
<script>
$(document).ready(function() {
  console.log('GOTOWE');
}); // koniec funkcji ready
</script>
```

   Funkcja `console.log()` jest uruchamiana w miejscu, w którym ją wywołasz. Oznacza to, że kiedy przeglądarka wczyta kod HTML strony (na ten moment czeka funkcja `ready()`), zapisze tekst „GOTOWE" w konsoli. Używanie funkcji `ready()` to standardowa i prosta operacja, dlatego zwykle nie trzeba wywoływać na tym etapie funkcji `console.log()`, jednak tu pozwala to zademonstrować działanie funkcji `log()`. Do rozwijanej strony dodasz później wiele wywołań tej funkcji, aby dobrze ją poznać.

4. **Zapisz plik i otwórz go w przeglądarce Chrome. Jeśli okno konsoli nie jest widoczne, wyświetl je, naciskając kombinację *Ctrl+Shift+J* (w systemie Windows) lub *⌘+Option+J* (w systemie Mac OS).**

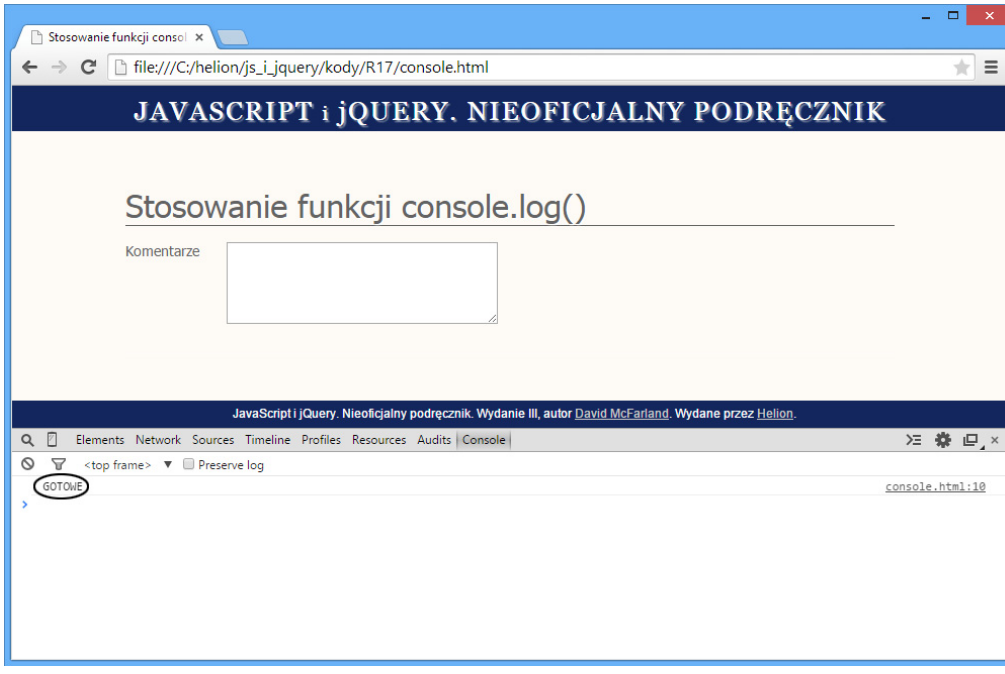
   W konsoli powinno pojawić się słowo *GOTOWE* (wyróżnione kółkiem na rysunku 17.4). Rozwijany skrypt ma wyświetlać liczbę znaków w polu formularza. Wartość ta ma się zmieniać po każdym wprowadzonym znaku. Aby uzyskać ten efekt, należy dodać do pola tekstowego zdarzenie `keyup` (patrz strona 182). Na każdym etapie rozwijania skryptu dodasz funkcję `console.log()`, aby kontrolować, co dzieje się w programie.

5. **Po wierszu dodanym w kroku 3. wpisz poniższy kod:**

```
$('#comments').keyup(function() {
  console.log('Zdarzenie: keyup');
});  // koniec funkcji keyup
```

   Koniecznie zapisz ten kod wewnątrz funkcji `$(document).ready()`.

   Na stronie znajduje się znacznik `<textarea>` o identyfikatorze `comments`. Element ten można pobrać za pomocą selektora jQuery, `$('#comments')`. W tym fragmencie dodano też funkcję obsługi zdarzenia `keyup` (informacje o dołączaniu



Rysunek 17.4. *Możesz używać konsoli JavaScript do wyświetlania w przeglądarce tajemnych komunikatów. Jednak możliwości te znacznie lepiej nadają się do użycia podczas testowania programów. W tym przypadku prosty komunikat — GOTOWE — informuje, że programowi udało się pomyślnie wywołać funkcję $(document).ready() jQuery*

zdarzeń znajdziesz na stronie 182). Wywołana tu funkcja `console.log()` wyświetla w konsoli JavaScript komunikat o stanie, informujący o każdym zgłoszeniu zdarzenia `keyup`. Jest to wygodny sposób na sprawdzenie, czy funkcja obsługi zdarzenia jest uruchamiana, czy może coś blokuje zgłoszenie danego zdarzenia.

Zapisz stronę, odśwież ją w przeglądarce i wpisz kilka znaków w polu tekstowym. Upewnij się, że jest widoczna konsola JavaScript — powinieneś w niej zobaczyć kilka wierszy z tekstem „Zdarzenie: keyup". Obok komunikatu powinna zostać wyświetlona także liczba (w niektórych przeglądarkach będzie ona umieszczona po lewej, a w innych po prawej stronie komunikatu), oznaczająca, ile razy dany komunikat został wyświetlony w konsoli.

Skoro zdarzenie `keyup` działa, można pobrać zawartość pola tekstowego i przypisać ją do zmiennej. Aby sprawdzić, czy skrypt zapisuje odpowiednie dane, należy wyświetlić zawartość zmiennej w konsoli.

6. **Dodaj wiersze 3. i 4. pod kodem wpisanym w kroku 5.:**

```
1 $('#comments').keyup(function() {
2   console.log('Zdarzenie: keyup');
3   var text = $(this).val();
4   console.log('Treść komentarza: ', text);
5 });   // koniec funkcji keyup
```

Wiersz 3. pobiera zawartość pola tekstowego i zapisuje ją w zmiennej `text` (informacje o sprawdzaniu wartości pól tekstowych znajdziesz na stronie 283). Wiersz 4. wyświetla komunikat w konsoli. Tu wiadomość składa się z łańcucha

znaków `'Treść komentarza:'` i aktualnej zawartości pola tekstowego. Jeśli program nie działa prawidłowo, standardową techniką diagnostyczną jest wyświetlenie wartości zmiennych. Pozwala to upewnić się, że zmienne zawierają oczekiwane informacje.

7. **Zapisz plik, odśwież go w przeglądarce i wpisz dowolny tekst w polu komentarza.**

   Po wpisaniu każdej litery w konsoli powinna pojawić się zawartość pola komentarza. Korzystanie z konsoli nie powinno już sprawiać Ci dużych problemów, dlatego dodaj jeszcze jeden komunikat i dokończ skrypt.

8. **Zmodyfikuj funkcję obsługi zdarzenia `keyup` przez dodanie dwóch nowych wierszy (5. i 6. w poniższym kodzie):**

```
1 $('#comments').keyup(function() {
2   console.log('Zdarzenie: keyup');
3   var text = $(this).val();
4   console.log('Treść komentarza: ', text);
5   var chars = text.length;
6   console.log('Liczba znaków: ', chars);
7 });   // koniec funkcji keyup
```

   Wiersz 5. sprawdza liczbę znaków zapisanych w zmiennej `text` (właściwość `length` omówiono na stronie 565) i przypisuje tę wartość do zmiennej `chars`. Aby się upewnić, że skrypt poprawnie pobiera liczbę znaków, należy za pomocą funkcji `log()` wyświetlić komunikat w konsoli (wiersz 6.).

   Pozostała do wykonania jeszcze jedna operacja — ukończenie skryptu, aby wyświetlał liczbę znaków użytkownikowi.

9. **Dodaj ostatni wiersz na końcu funkcji obsługi zdarzenia `keyup` (wiersz 10.). Gotowy skrypt powinien wyglądać następująco:**

```
1  <script>
2  $(document).ready(function() {
3    console.log('READY');
4    $('#comments').keyup(function() {
5      console.log('Zdarzenie: keyup');
6      var text = $(this).val();
7      console.log('Treść komentarza: ', text);
8      var chars = text.length;
9      console.log('Liczba znaków: ', chars);
10     $('#count').text("Liczba znaków: " + chars);
11   });   // koniec funkcji keyup
12 });   // koniec funkcji ready
13 </script>
```

10. **Zapisz plik i wyświetl go w przeglądarce.**

    Teraz konsola powinna wyglądać taki, jak ta z rysunku 17.5. Gotową wersję rozwiązania zawiera plik *complete_console.html* w katalogu *R17* w archiwum z przykładami.

Uwaga: Po utworzeniu działającego programu należy usunąć ze skryptu wszystkie wywołania `console.log()`. Funkcja `log()` wywoła błędy w niektórych starszych przeglądarkach.

Rysunek 17.5. *Konsola JavaScript to doskonałe narzędzie do wyświetlania diagnostycznych informacji w czasie działania programu. Możesz też pogrupować zbiory komunikatów (na przykład wszystkie wiadomości zapisane w pętli). W tym celu dodaj funkcję console.group() przed pierwszym wywołaniem console.log() w grupie, a po wyświetleniu ostatniego komunikatu ze zbioru wywołaj funkcję console.groupEnd()*

Diagnozowanie zaawansowane

Konsola JavaScript to doskonałe narzędzie do wyświetlania komunikatów z informacjami o funkcjonowaniu programu. Jednak czasem skrypt działa tak szybko, że trudno zauważyć, jakie operacje zachodzą na poszczególnych etapach. Trzeba wtedy spowolnić program. Na szczęście przeglądarki udostępniają użyteczne debuggery skryptów JavaScript, które pozwalają na prześledzenie działania programu wiersz po wierszu i zobaczenie, co dzieje się na każdym etapie skryptu.

Uwaga: W tym przykładzie zastosowana została przeglądarka Google Chrome, jednak debuggery JavaScriptu o bardzo podobnych możliwościach dostępne są także w innych przeglądarkach, takich jak Firefox, Opera, Safari oraz Internet Explorer.

Diagnozowanie to proces naprawiania nieprawidłowo działających programów. Aby dobrze zrozumieć funkcjonowanie skryptu (lub występujące w nim problemy), czasem trzeba krok po kroku prześledzić jego działanie.

Aby użyć debuggera, należy umieścić w określonych wierszach kodu *punkty wstrzymania* (nazywane też punktami przerwania). Są to miejsca, w których interpreter wstrzymuje działanie i czeka na polecenia. Należy wtedy użyć kontrolek debuggera, które pozwalają uruchomić program wiersz po wierszu. W ten sposób możesz dokładnie prześledzić, jak działają poszczególne instrukcje. Proces korzystania z debuggera przebiega następująco:

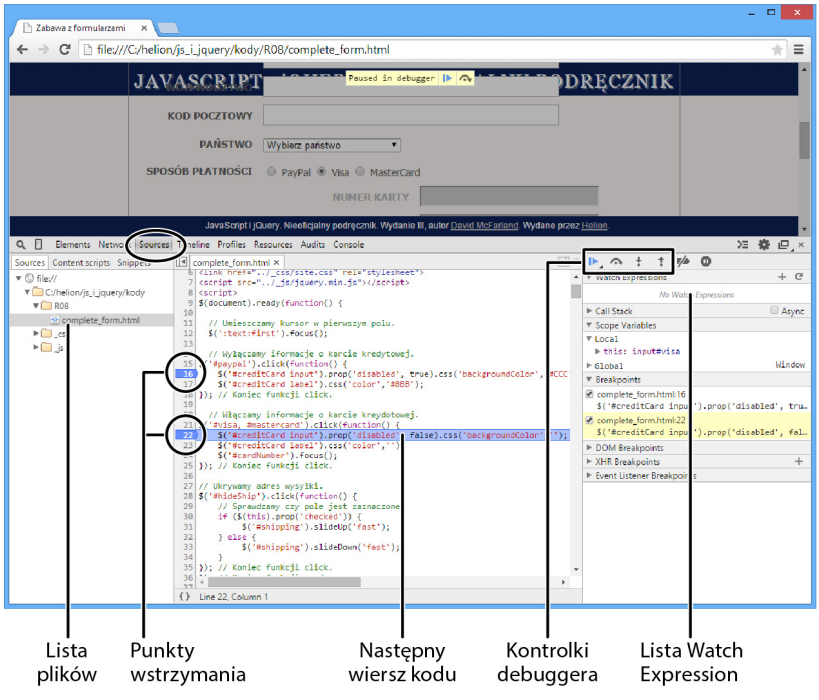
1. **Otwórz stronę w przeglądarce.**

   Pamiętaj, że w opisywanym tu przykładzie zastosowana została przeglądarka Google Chrome oraz jej wbudowany debugger. W debuggerach dostępnych w innych przeglądarkach konkretne wykonywane czynności oraz wygląd paneli mogą być nieco inne.

2. **Otwórz konsolę JavaScript.**

   Aby otworzyć narzędzia dla programistów używanej przeglądarki, postępuj zgodnie z instrukcjami opisanymi na stronie 51. Kiedy korzystasz z przeglądarki Chrome, konsolę JavaScript najprościej otworzysz, używając kombinacji klawiszy *Ctrl+Shift+J* (w systemie Windows) lub *⌘+Options+J* (w systemie Mac OS).

3. **Kliknij kartę *Sources*, a następnie z listy plików wybierz plik zawierający kod JavaScript, który chcesz przetestować (patrz rysunek 17.6).**



Rysunek 17.6. *Debugger umożliwia dodawanie punktów wstrzymania (wierszy, w których skrypt wstrzymuje działanie i czeka na polecenia), kontrolowanie wykonywania kodu i podglądanie zmiennych na liście Czujka. W czasie wykonywania programu aktualny wiersz (oczekujący na uruchomienie) jest podświetlony*

W przeglądarce Chrome kod źródłowy pliku, który chcemy testować, jest wyświetlany na karcie *Sources*. W przypadku skryptu umieszczonego bezpośrednio na stronie WWW, na karcie jest wyświetlany cały kod strony (włącznie z kodem HTML). Jeśli jednak wybierzemy zewnętrzny plik JavaScript, to na karcie zostanie wyświetlony wyłącznie kod tego pliku.

4. **Wybierz z menu kodu źródłowego (patrz rysunek 17.6) plik z diagnozowanym skryptem.**

   Skrypty często znajdują się w różnych miejscach strony i w zewnętrznych plikach JavaScript. Jeśli na stronie działają skrypty z kilku plików, trzeba wybrać dokument zawierający diagnozowany kod.

5. **Dodaj punkty wstrzymania.**

   Aby dodać punkt wstrzymania, kliknij lewy margines obok numeru wiersza. Pojawi się znacznik reprezentujący taki punkt.

Uwaga: Dodawanie punktów wstrzymania do wierszy, które zawierają tylko komentarze, jest bezcelowe. Debugger nie zatrzyma skryptu w takim miejscu. Punkty wstrzymania należy dodawać tylko do wierszy z wykonywalnym kodem JavaScript.

---

6. **Odśwież stronę.**

   Aby dodać punkty wstrzymania w wybranych miejscach skryptu, trzeba najpierw wyświetlić stronę w przeglądarce, zatem może się zdarzyć, że diagnozowany kod JavaScript już zostanie uruchomiony (jeszcze przed dodaniem punktów wstrzymania). Należy wtedy odświeżyć stronę, aby ponownie włączyć skrypt.

   Jeśli dodałeś punkt wstrzymania do funkcji reagującej na zdarzenie (na przykład chcesz zdiagnozować kod uruchamiany po kliknięciu przycisku lub umieszczeniu kursora nad odnośnikiem), musisz je wywołać — kliknąć przycisk lub najechać kursorem na odsyłacz — aby dojść do punktu wstrzymania i rozpocząć proces diagnozowania.

   Kiedy skrypt dojdzie do punktu wstrzymania, przerwie działanie. Program zostanie zatrzymany w czasie i będzie oczekiwał na wykonanie wiersza po pierwszym punkcie wstrzymania.

7. **Użyj kontrolek debuggera do przejścia przez program krok po kroku.**

   Większość debuggerów udostępnia cztery kontrolki (patrz rysunek 17.6), które określają, jak program ma działać po zatrzymaniu się w punkcie wstrzymania. Informacje o tych kontrolkach znajdziesz w następnym podpunkcie rozdziału.

8. **Obserwuj stan programu na liście *Watch Expressions* (patrz rysunek 17.7).**

   Celem analizowania działania programu krok po kroku jest śledzenie, co dzieje się w każdym wierszu skryptu. Lista *Watch Expressions* udostępnia podstawowe informacje o stanie programu i pozwala wskazać dodatkowe zmienne, które chcesz obserwować. Możesz w ten sposób śledzić na przykład wartość zmiennej `score`. Na stronie 632 dowiesz się, jak korzystać z listy *Watch Expressions*.

9. **Napraw skrypt w edytorze tekstu.**

W czasie przechodzenia przez skrypt powinieneś wykryć problem i na przykład dowiedzieć się, dlaczego wartość danej zmiennej nigdy się nie zmienia, a warunek zawsze jest prawdziwy. Po uzyskaniu potrzebnych informacji można przejść do edytora tekstu i zmodyfikować skrypt (na stronie 633 znajduje się przykład ilustrujący naprawianie skryptu).

10. **Przetestuj stronę w przeglądarce. Jeśli to konieczne, powtórz powyższe kroki, aby usunąć wykryte problemy.**

Kontrolowanie działania skryptu za pomocą debuggera

Po dodaniu do skryptu punktów wstrzymania i odświeżeniu strony można uruchomić kod wiersz po wierszu. Jeśli dodałeś punkt wstrzymania do fragmentu wykonywanego przy wczytywaniu strony, skrypt zatrzyma się w tym punkcie. Jeżeli punkt wstrzymania znajduje się w wierszu uruchamianym po wystąpieniu zdarzenia (na przykład po kliknięciu odnośnika), aby dojść do tego punktu, trzeba wywołać dane zdarzenie.

Kiedy debugger przerwie program w punkcie wstrzymania, nie uruchomi danego wiersza, ale zatrzyma się tuż *przed* nim. Można wtedy kliknąć jeden z czterech przycisków debuggera, aby określić jego dalsze działanie (patrz rysunek 17.7).

- ***Resume script execution*** (wznów wykonywanie skryptu). Przycisk *Kontynuuj* ponownie uruchamia skrypt. Program nie zatrzyma się do momentu napotkania przez interpreter następnego punktu wstrzymania lub do czasu zakończenia działania. Jeśli skrypt ponownie dojdzie do punktu wstrzymania, zatrzyma się w oczekiwaniu na polecenia użytkownika.

  Użyj przycisku *Kontynuuj*, jeśli chcesz uruchomić program lub przejść do następnego punktu wstrzymania.

- ***Step over next funcion call*** (przeskocz wywołanie następnej funkcji). Ta przydatna opcja wykonuje aktualny wiersz kodu, a następnie zatrzymuje działanie skryptu. Jej nazwa wynika z tego, że jeśli bieżący wiersz zawiera wywołanie funkcji, debugger nie wyświetli jej kodu ani się w nim nie zatrzyma, ale wykona go i zatrzyma się w następnym wierszu. Warto korzystać z tej opcji, jeśli wiadomo, że dana funkcja działa bezbłędnie. Jeżeli przykładowo skrypt wywołuje funkcję biblioteki jQuery, warto ją przeskoczyć. Jeśli tego nie zrobisz, będziesz musiał przez długi czas śledzić wiersz po wierszu skomplikowany kod tej biblioteki. Z tej opcji będziesz zazwyczaj korzystał, chyba że w aktualnym wierszu kodu znajduje się wywołanie funkcji, którą sam utworzyłeś — jeśli chcesz sprawdzić, co się wewnątrz niej dzieje, powinieneś skorzystać z opisanej poniżej opcji *Step into next function call*.

- ***Step into next function call*** (wejdź do wywołania następnej funkcji). Ta opcja powoduje wkroczenie debuggera w kod funkcji. Oznacza to, że jeśli bieżący wiersz zawiera wywołanie funkcji, debugger wkroczy w nią i zatrzyma się na jej pierwszym wierszu. Ta opcja jest przydatna, jeżeli nie jesteś pewien, czy problem występuje w głównym skrypcie, czy w funkcji.

  Jeśli jesteś pewien, że dana funkcja działa poprawnie (na przykład korzystałeś z niej już dziesiątki razy), nie warto stosować tej opcji. Dobrze jest też używać *Step over next function call* opcji zamiast *Step into next function call* przy

diagnozowaniu wierszy kodu z selektorami i poleceniami biblioteki jQuery. Na przykład `$('#button')` to wyrażenie umożliwiające bibliotece jQuery pobranie elementu strony. Jest to też funkcja tej biblioteki, dlatego jeśli klikniesz przycisk *Step into next function call*, wkroczysz w złożony świat jQuery. Jeśli tak się stanie, zauważysz to, ponieważ karta ze skryptem zmieni się i wyświetli cały kod JavaScript z pliku biblioteki jQuery.

Jeśli w czasie korzystania z debuggera zagubisz się w funkcji lub w kodzie biblioteki języka JavaScript, takiej jak jQuery, możesz wydostać się z danego fragmentu za pomocą kontrolki *Step out of current function*.

- ***Step out of current function*** (wyjdź z bieżącej funkcji). Ten przycisk powoduje wyjście debuggera z wywołania funkcji. Zwykle jest używany po kliknięciu przycisku *Step into next fuction call*. Wybranie tej opcji powoduje wykonanie kodu funkcji bez zatrzymywania się w każdym jej wierszu. Kiedy klikniesz ten przycisk, debugger wróci do wiersza wywołania funkcji i zatrzyma skrypt.

Obserwowanie skryptu

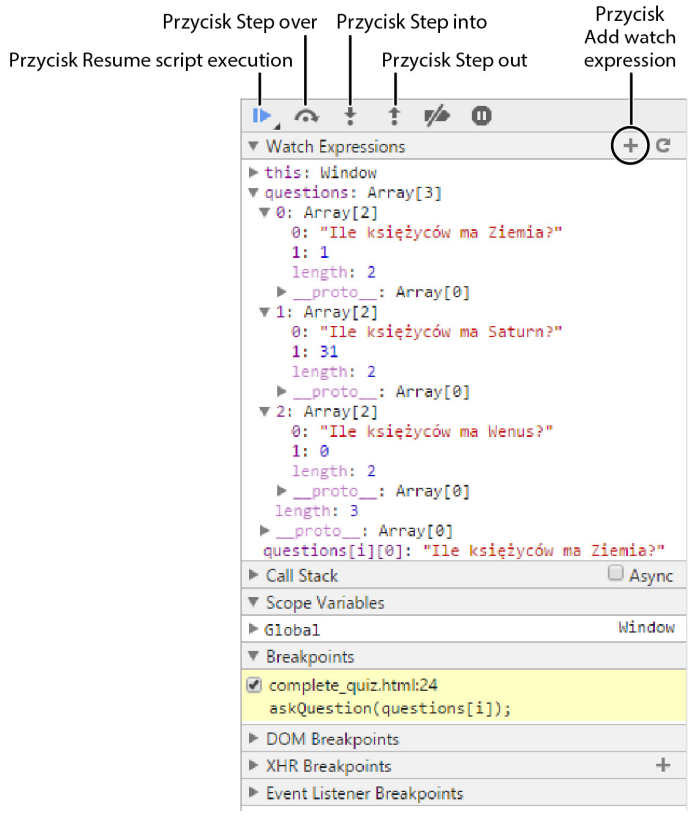
Choć przyciski debuggera pozwalają kontrolować wykonywanie skryptu, celem korzystania z tego narzędzia jest śledzenie tego, co dzieje się w programie. Pomocna jest w tym lista *Watch Expressions* (patrz rysunek 17.7). Wyświetla ona zmienne i funkcje dostępne w kontekście wykonywanego wiersza kodu. Oznacza to, że jeśli umieścisz punkt wstrzymania w funkcji, zobaczysz listę wszystkich zdefiniowanych w niej zmiennych. Jeżeli dodasz taki punkt w głównym ciele skryptu, pojawią się funkcje zdefiniowane w tym obszarze. Ponadto na liście *Watch Expressions* widoczne są wszystkie utworzone funkcje.

Przy użyciu żółtego paska z napisem „Nowe wyrażenie czujki" możesz dodawać własne zmienne i wyrażenia. Wystarczy kliknąć ten pasek, a pojawi się pole tekstowe. Wpisz nazwę przeznaczonej do obserwacji zmiennej lub instrukcję języka JavaScript, którą chcesz uruchomić. Ponieważ debugger nie śledzi wartości zmiennej licznika w pętlach `for` (patrz strona 112), możesz ją dodać, a następnie obserwować jej zmiany przy każdym uruchomieniu danej pętli.

Listę *Watch Expressions* możesz traktować jak okno z ciągle wywoływanym poleceniem `console.log()`. Lista ta wyświetla wartość zmiennej lub wyrażenia w momencie wykonywania danego wiersza kodu.

Lista *Watch Expressions* zapewnia wartościowy wgląd w program i udostępnia efekt „zatrzymania filmu", który pozwala dokładnie określić miejsce wystąpienia błędu w skrypcie. Na przykład jeśli wiesz, że dana zmienna przechowuje liczbę, możesz przejść przez program krok po kroku, aby zobaczyć, jaką wartość zapisuje w zmiennej w momencie jej utworzenia i jak modyfikuje przechowywane w niej instrukcje. Jeżeli po kliknięciu przycisków *Step over next funcion call* lub *Step out of current function* zauważysz, że zmienna przyjmuje nieoczekiwaną wartość, prawdopodobnie znalazłeś wiersz, w którym pojawia się błąd.

Uwaga: Pamiętaj, że kiedy program jest zatrzymany, debugger podświetla następny wiersz kodu, który zostanie wykonany. A zatem, jeśli wyróżniony wiersz ustawia wartość zmiennej, nie zobaczysz jej wartości na liście *Watch Expressions* aż do momentu naciśnięcia przycisku *Step over next function call*.



Rysunek 17.7. *Lista Watch Expressions wyświetla wartość różnych zmiennych w czasie działania programu. Możesz dodać do listy własne wyrażenia, które pojawią się na paskach o szarym tle w górnej części okna*

Przykład diagnozowania

W tym przykładzie użyjesz konsoli JavaScript przeglądarki Chrome do zdiagnozowania pliku zawierającego błędy różnego rodzaju (składniowe, czasu wykonania i logiczne). Strona to prosty quiz, który wyświetla trzy pytania i uzyskany wynik. Aby zobaczyć, jak powinna ona działać, otwórz w dowolnej przeglądarce plik *complete_debugger.html* z katalogu *R17*.

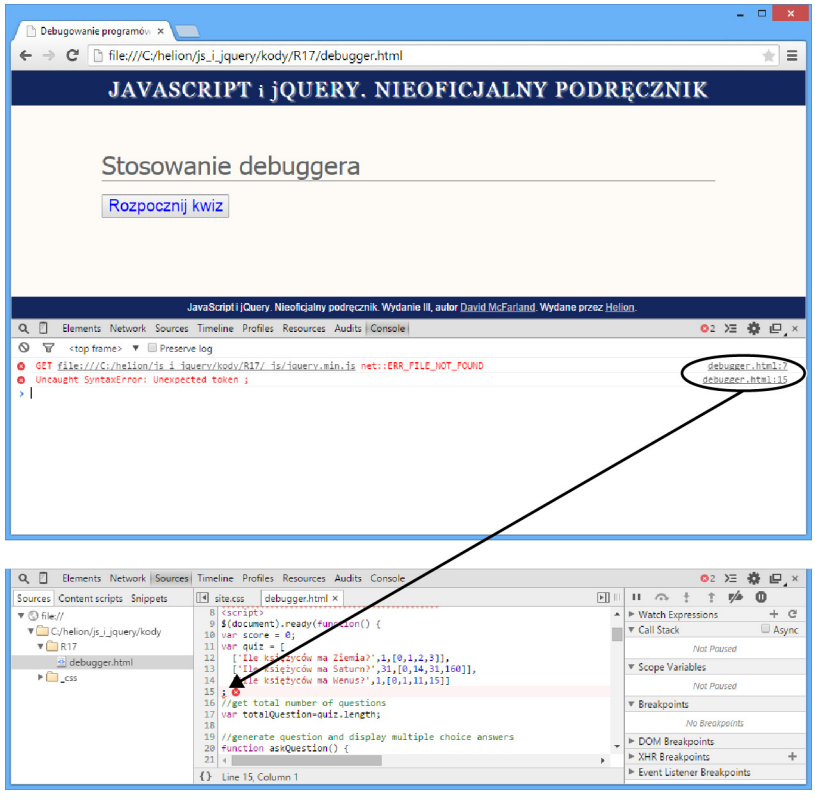
Uwaga: Informacje o pobieraniu przykładowych plików znajdziesz na stronie 46.

Podobnie jak wszystkie inne przykłady prezentowane w tym rozdziale, także i ten zostanie wykonany przy użyciu przeglądarki Chrome. Możesz spróbować wykonać go w innej przeglądarce, jednak w takim przypadku zarówno konsola JavaScript, jak i debugger mogą wyglądać inaczej niż w zamieszczonym tu opisie.

1. **Uruchom przeglądarkę Chrome i otwórz plik *debugger.html* z katalogu *R17*.**

   Otwórz konsolę JavaScript — w przeglądarce Chrome zrobisz to najprościej, używając kombinacji klawiszy *Ctrl+Shift+J* (w systemie Windows) lub *⌘+Options+J* (w systemie Mac OS).

Konsola wyświetli informacje o dwóch błędach. Najpierw zajmiesz się drugim z nich. Został opisany jako: „Uncaught SyntaxError: Unexpected token ; ”[1]. Znaczenie tego komunikatu nie jest tak od razu oczywiste, jednak informuje on o tym, że przeglądarka nie oczekiwała średnika w miejscu, w którym go znalazła. Zważywszy, że średnik oznacza koniec instrukcji, można dojść do wniosku, że to z tą instrukcją jest coś nie w porządku. Zwróć także uwagę, że podany został numer wiersza kodu, w którym przeglądarka znalazła błąd (został on zakreślony na rysunku 17.8).



Rysunek 17.8. *Konsola przeglądarki Chrome to pierwszy przystanek przy wykrywaniu błędów składniowych i czasu wykonania, które uniemożliwiają wykonanie skryptu*

2. **Kliknij numer wiersza (na rysunku 17.8 został on zakreślony).**

   Spowoduje to wyświetlenie panelu *Sources* i podświetlenie wiersza, w którym wystąpił problem. Pamiętaj, że bardzo dużo błędów to zwyczajne, proste pomyłki typograficzne. Do tych, które zdarzają się najczęściej należą pominięcie zamykającego nawiasu, nawiasu klamrowego (`}`) lub kwadratowego (`]`), które można stosunkowo łatwo zlokalizować i poprawić. W tym przypadku, w wierszu 11 rozpoczyna się definicja tablicy — `var quiz = [` — lecz w kodzie zabrakło końca definicji, czyli zamykającego nawiasu `]`.

-

[1] Nieprzechwycony wyjątek SyntaxError: Nieoczekiwany leksem ; — *przyp. tłum.*

3. **Uruchom edytor tekstu i otwórz plik *debugger.html*. Znajdź wiersz 15. (znajduje się w nim tylko znak ;). Dodaj zamykający nawias kwadratowy przed symbolem ;, aby wiersz wyglądał następująco:**

```
];
```

Nowy nawias kwadratowy zamyka zagnieżdżoną tablicę, która obejmuje wszystkie pytania i odpowiedzi quizu.

4. **Zapisz plik. Wróć do przeglądarki Chrome i odśwież stronę.**

W kodzie wciąż znajduje się pierwszy z początkowych błędów. A do tego pojawił się następny! Tym razem konsola informuje, że „$ is not defined", i wskazuje wiersz 9. z funkcją `$(document).ready()` biblioteki jQuery. Kiedy przeglądarka informuje, że coś jest „not defined", oznacza to, iż kod wskazuje na nieistniejący element, na przykład zmienną lub funkcję, której jeszcze nie utworzyłeś. Może to też być skutek popełnienia literówki. Tu jednak kod wygląda poprawnie. Przyczyna znajduje się we wcześniejszej części kodu:

```
<script src="_js/jquery.min.js"></script>
```

Standardowym problemem przy korzystaniu z zewnętrznych plików jest błędnie podana ścieżka do skryptu. Tu plik *jquery.min.js* znajduje się w katalogu o nazwie *_js poza* katalogiem danej strony, natomiast ścieżka informuje, że katalog *_js* jest w tym samym folderze. Ponieważ Firefox nie potrafi znaleźć pliku *jquery.min.js* (to w nim zdefiniowano specjalną funkcję `$()` biblioteki jQuery), informuje o błędzie.

5. **Zmień znacznik `<script>`, aby wyglądał jak ten poniżej:**

```
<script src="../_js/jquery.min.js"></script>
```

Fragment `../` informuje, że katalog *js* znajduje się poza bieżącym folderem, dlatego ścieżka prowadzi teraz do pliku jQuery. Jakie błędy mogą się jeszcze kryć w programie?

6. **Zapisz plik, wróć do przeglądarki Chrome i odśwież stronę.**

Brak błędów; a to dlatego, że oba były spowodowane przez ten sam problem. Pierwszy z nich informował, że nie można znaleźć pliku biblioteki jQuery; a ponieważ przeglądarka nie mogła go znaleźć, zatem nie wiedziała, co oznacza `$`. Wygląda na to, że strona została naprawiona, ale czy na pewno?

7. **Kliknij przycisk *Rozpocznij quiz*.**

Następny błąd! Tym razem konsola informuje, że „askQuestions is not defined", i wskazuje na wiersz 69. w końcowej części skryptu. Ponieważ ten problem pojawia się tylko po uruchomieniu programu, jest to błąd czasu wykonania (patrz ramka na stronie 615). Problem pojawia się w końcowej części skryptu, w poniższej instrukcji warunkowej:

```
if (quiz.length>0) {
  askQuestions();
} else {
  giveResults();
}
```

Prawdopodobnie zauważyłeś już, że kiedy dany element jest niezdefiniowany, często wynika to z popełnienia prostej literówki. Tu `askQuestions()` to wywołanie funkcji, dlatego zajrzyj do kodu i spróbuj ją znaleźć.

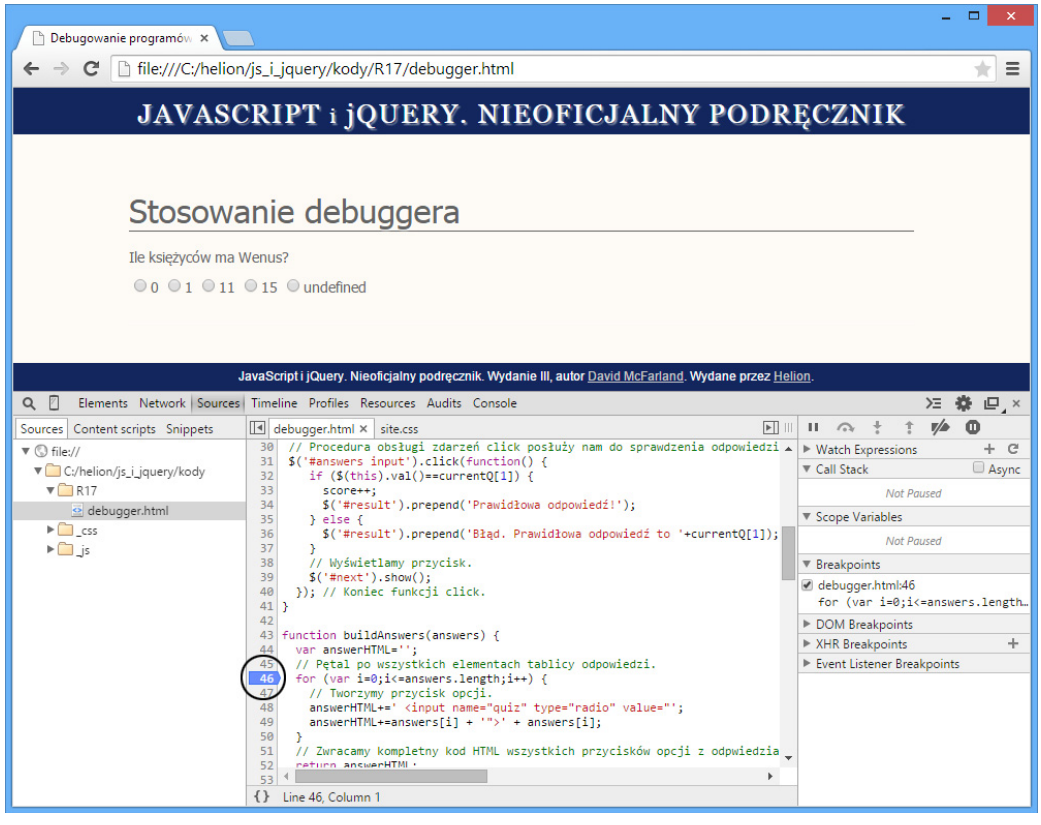
Czy znalazłeś funkcję? Choć w kodzie nie ma metody `askQuestions()`, powinieneś zauważyć funkcję `askQuestion()` (bez litery „s").

8. **Wróć do edytora tekstu i usuń ostatnią literę „s" z wywołania `askQuestions()` w wierszu 69. (w końcowej części skryptu). Zapisz plik, odśwież go w przeglądarce i ponownie kliknij przycisk *Rozpocznij quiz*.**

   Tym razem pojawi się pytanie z pięcioma odpowiedziami w formacie wielokrotnego wyboru. Niestety, przy ostatniej możliwości znajduje się etykieta *undefined*. Wygląda to na usterkę, jednak konsola jest pusta, dlatego technicznie nie ma błędu w kodzie JavaScript. Problem musiał wystąpić w **logice** programu. Aby odkryć przyczynę błędu, trzeba użyć debuggera.

9. **W przeglądarce Chrome kliknij kartę *Sources* i z listy wybierz plik *debugger.*
   ↪*html* (patrz rysunek 17.9).**

   Karta *Sources* zapewnia dostęp do kodu JavaScript strony. Jeśli strona zawiera kod w tym języku, a ponadto dołączono do niej zewnętrzne pliki JavaScript, w menu kodu źródłowego możesz określić, który plik chcesz zdiagnozować.



Rysunek 17.9. *W debuggerze przeglądarki można zdiagnozować dowolny skrypt używany przez daną stronę. Menu kodu źródłowego pozwala wybrać kod JavaScript zagnieżdżony w tej stronie lub zapisany w dołączonych zewnętrznych plikach JavaScript*

Ponieważ problemem jest przycisk opcji z napisem „undefined", warto rozpocząć szukanie przyczyny kłopotów w kodzie tworzącym takie przyciski. Jeśli jesteś autorem danego skryptu, prawdopodobnie wiesz, gdzie szukać potrzebnego kodu. Jednak jeżeli otrzymałeś program z usterkami, musisz go przejrzeć, aby znaleźć właściwy fragment.

Tu przyciski opcji tworzy funkcja o nazwie `buildAnswers()`, która przygotowuje zbiór odpowiedzi do wyboru reprezentowanych przez wspomniane elementy. Ta funkcja przyjmuje tablicę z tekstem każdego przycisku, a zwraca łańcuch znaków z kodem HTML utworzonych przycisków. Warto rozpocząć diagnozowanie od tej właśnie funkcji.

10. **Przewiń zawartość środkowego panelu karty *Sources* (prezentującego kod HTML i JavaScript wybranej strony) w dół, tak by widoczny był wiersz 47. Ustaw punkt wstrzymania w wierszu 46. (w kółku na rysunku 17.9).**

    Numer wiersza zostanie podświetlony, co będzie oznaczać, że znajduje się w nim punkt wstrzymania, czy też miejsce, w którym interpreter JavaScriptu przerwie wykonywanie kodu. Oznacza to, że kiedy ponownie uruchomisz program, interpreter dojdzie do tego wiersza i zatrzyma skrypt, a Ty będziesz mógł przejść przez kod wiersz po wierszu, aby zobaczyć, jak działa.

    Debugger umożliwia też podgląd wartości zmiennych w czasie działania program, co przypomina korzystanie z funkcji `console.log()` (patrz strona 623). Musisz tylko poinformować debugger, jakie zmienne chcesz obserwować.

11. **W panelu z prawej strony kliknij przycisk „+" (umieszczony bezpośrednio na prawo od nagłówka listy *Watch Expressions*), wpisz literę `i`, a następnie wciśnij klawisz *Enter*.**

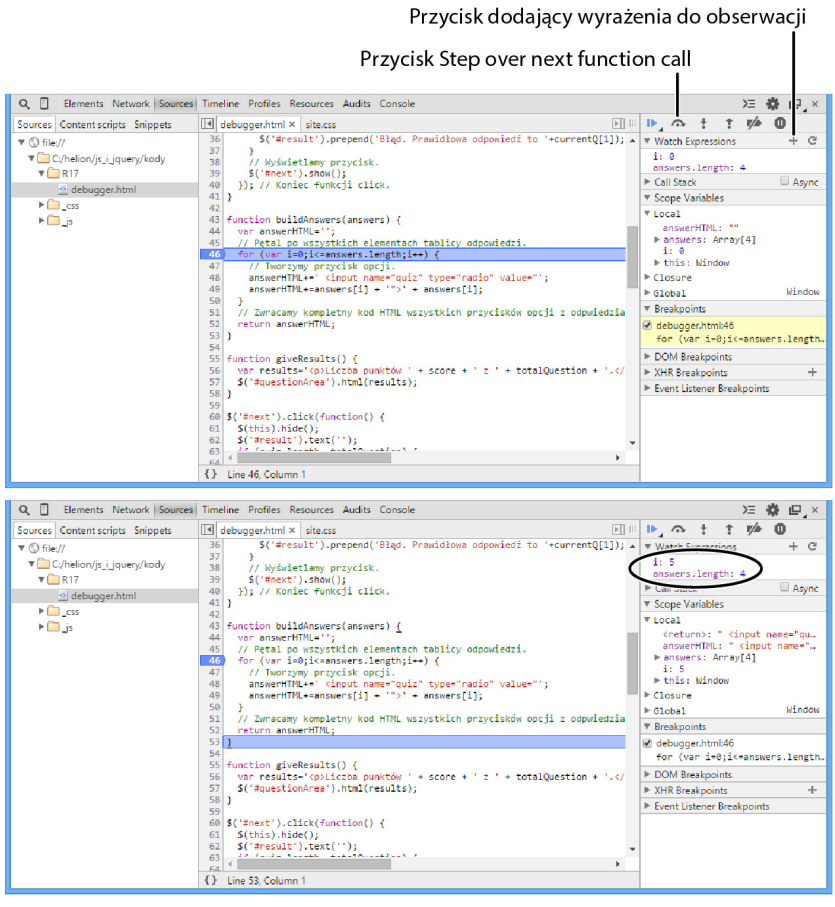
    Dodałeś w ten sposób zmienną `i` do listy *Watch Expressions*. Jest to zmienna licznika pętli `for`, określająca liczbę uruchomień tej pętli (więcej informacji o pętlach znajdziesz na stronie 109). W czasie działania skryptu będziesz mógł śledzić zmiany tej wartości. Następnie dodaj nową obserwowaną zmienną.

12. **Ponownie kliknij przycisk „+" obok nagłówka *Watch Expressions*, wpisz wyrażenie `answers.length` i wciśnij klawisz *Enter*.**

    Nie przejmuj się wartością, którą debugger wyświetli na tym etapie (prawdopodobnie będzie to tekst „answers is not defined"). Nie możesz śledzić wartości zmiennych, dopóki debugger nie wejdzie do odpowiedniej funkcji. Teraz można już przyjrzeć się działaniu skryptu.

13. **Kliknij przycisk *Załaduj tę stronę ponownie* lub wciśnij kombinację klawiszy *Ctrl+R* (*⌘+R*). Kiedy przeglądarka odświeży stronę, kliknij przycisk *Uruchom quiz*.**

    Skrypt rozpocznie działanie, a na stronie pojawi się pierwsze pytanie. Jednak tuż przed utworzeniem przycisków opcji (w wierszu 46.) debugger wstrzyma program (patrz górna część rysunku 17.10). Zauważ, że na liście *Watch Expressions* zmienna `i` ma wartość „not defined". Dzieje się tak, ponieważ punkt wstrzymania blokuje program tuż przed uruchomieniem danego wiersza. Oznacza to, że pętla nie została jeszcze uruchomiona, dlatego zmienna `i` na razie nie istnieje.



Rysunek 17.10. *Lista Watch Expressions może zawierać nie tylko zmienne i wyrażenia, które do niej dodamy. Przykładowo sekcja Scope Variables prezentuje wszystkie zmienne dostępne w „zasięgu" bieżącej funkcji. W tym przypadku (górny rysunek) widać, że w funkcji buildAnswer() dostępne są zmienne answerHTML oraz answers*

Z kolei wartość wyrażenia `answers.length` to `4`. Tablica `answers` zawiera odpowiedzi przekazane do funkcji. Właściwość `length` tej tablicy określa liczbę jej elementów. Tu dostępne są cztery odpowiedzi, dlatego po zakończeniu działania funkcji powinny pojawić się cztery przyciski opcji.

14. **Kliknij przycisk *Step over next function call* (patrz rysunek 17.10).**

    Ten przycisk powoduje przejście do następnego wiersza programu. Zauważ, że zmienna `i` ma teraz wartość `0`. Kliknij kilkakrotnie, aby przejść przez pętlę.

15. **Klikaj przycisk *Step over next function call*, dopóki zmienna `i` na liście *Watch Expressions* nie przyjmie wartości `5` (patrz dolna część rysunku 15.10).**

    Choć tablica `answers` zawiera tylko cztery elementy, pętla `for` jest uruchamiana pięciokrotnie (to wartość zmiennej `i`). Dlatego pętla kończy działanie w nieoczekiwanym momencie. Pamiętaj, że w pętli `for` środkowa instrukcja to warunek, który musi być spełniony, aby skrypt uruchomił kod w pętli (patrz strona 112). Tu ten warunek to `i<=answers.length;`. Oznacza to, że pętla zaczyna działanie, kiedy zmienna `i` ma wartość `0`. Skrypt ponownie uruchamia pętlę, dopóki zmienna `i` ma wartość mniejszą od liczby elementów tablicy `answers` lub równą jej. Dlatego przed wyjściem z pętli zmienna `i` przyjmie wartości `0`, `1`, `2`, `3` oraz `4`,

co oznacza pięciokrotne uruchomienie pętli. Ponieważ jednak tablica odpowiedzi zawiera tylko cztery elementy, podczas piątej iteracji pętli zabraknie już odpowiedzi do wyświetlenia: komunikat „undefined" pojawia się dlatego, że w tablicy odpowiedzi nie ma piątego elementu.

16. **Wróć do edytora tekstu i zmodyfikuj pętlę `for` w wierszu 46., aby wyglądała następująco:**

```
for (i=0;i<answers.length;i++) {
```

Teraz pętla zostanie uruchomiona tyle razy, ile elementów znajduje się w tablicy `answers`, dlatego utworzy jeden przycisk opcji dla każdej odpowiedzi.

17. **Zapisz plik i wyświetl go w przeglądarce.**

Możesz kliknąć podświetlony numer wiersza na karcie *Sources*, aby usunąć punkt wstrzymania i bez przeszkód obserwować działanie ukończonej strony.

Gotową wersję przykładu zawiera plik *complete_debugger.html*. Jak widać, wyszukiwanie błędów w programie może być czasochłonne. Na szczęście debuggery znacznie ułatwiają zbadanie „wnętrza" skryptu i ustalenie przyczyn problemów.

CZĘŚĆ

VI

Dodatki

- Dodatek A. Materiały związane z językiem JavaScript
- Skorowidz

A

DODATEK

Materiały związane z językiem JavaScript

Ta książka zawiera informacje i omówienie praktycznych technik, które pozwolą Ci rozpocząć karierę programisty języka JavaScript. Jednak żadna książka nie obejmuje odpowiedzi na wszystkie pytania z tej dziedziny. Programowanie w języku JavaScript to bardzo obszerne zagadnienie, a w tym dodatku dowiesz się, gdzie możesz kontynuować dociekania i naukę.

Źródła informacji

Czasem aby przeczytać książkę, trzeba posłużyć się słownikiem. W trakcie tworzenia programów w języku JavaScript warto mieć kompletne źródło informacji na temat różnych słów kluczowych, pojęć, metod i innych elementów składni tego języka. Takie materiały są dostępne zarówno w internecie, jak i w formie książek.

Witryny

- Witryna **ECMAScript** (*http://www.ecmascript.org/*) zawiera dokumentację i informacje na temat języka ECSAScript (to oficjalna nazwa JavaScriptu). Na niej należy szukać informacji o aktualnym stanie tego języka (oraz o jego przyszłości).
- **Encyklopedia języka JavaScript w witrynie Mozilla Developer Center** (*https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference*) to kompletne źródło informacji na temat języka JavaScript. Dokumentacja ta jest niezwykle szczegółowa, choć czasami trudna do zrozumienia, gdyż kierowana dla odbiorców z odpowiednim wykształceniem informatycznym.

- **WebPlatform** (*http://www.webplatform.org/*) zawiera informacje dotyczące języka JavaScript, DOM i CSS, jak również dane o tym, które możliwości są obsługiwane przez poszczególne wersje przeglądarek. To prawdziwa encyklopedia dla twórców aplikacji internetowych.
- **Dokumentacja języka JavaScript na witrynie MSDN** (*http://msdn.microsoft.com/en-us/library/d1et7k7c(v=VS.94).aspx*) Microsoftu to doskonałe źródło wiedzy, jeśli tworzysz witryny przeznaczone dla przeglądarki Internet Explorer. Choć udostępnia ona informacje na temat języka JavaScript w wersjach obsługiwanych przez inne przeglądarki, jednak zawiera wiele informacji o implementacji języka używanej w Internet Explorerze.

Książki

- *JavaScript: The Definitive Guide*[1] autorstwa Davida Flanagana (wydawnictwo O'Reilly) to najbardziej wyczerpująca drukowana encyklopedia języka JavaScript. Jest to treściwa i obszerna pozycja i zawiera wszystkie szczegóły potrzebne do dobrego zrozumienia języka JavaScript.

Podstawy języka JavaScript

JavaScript nie jest prosty w nauce, dlatego zawsze warto korzystać z wielu źródeł informacji, aby opanować wszelkie niuanse tworzenia aplikacji sieciowych. Wymienione w tym podrozdziale materiały pomagają poznać podstawy tego języka (co czasem bywa trudne).

Witryny

- **Samouczek języka JavaScript w witrynie W3 Schools** (*www.w3schools.com/js*) to rozbudowany (choć nie zawsze zawierający szczegółowe wyjaśnienia) poradnik, który opisuje większość aspektów programowania w języku JavaScript.
- **Wprowadzenie do języka JavaScript** na witrynie howtocreate.co.uk (*http://www.howtocreate.co.uk/tutorials/javascript/introduction*) stanowi ogólnie dostępny, szczegółowy opis języka. Oczywiście, ponieważ korzystasz z biblioteki jQuery, nie będziesz potrzebował wielu spośród zamieszczonych tam informacji, gdyż dotyczą one tradycyjnych sposobów pobierania elementów DOM i manipulowania nimi.

Książki

- *Head First JavaScript*[2] autorstwa Michaela Morrisona (wydawnictwo O'Reilly) to ciekawe i bogato ilustrowane wprowadzenie do języka JavaScript. Znajdziesz tu wiele informacji o języku JavaScript, podanych w zabawnym i dowcipnym stylu.

[1] Wydanie polskie: *JavaScript. Podręcznik programisty*, RM, 2002 — *przyp. tłum.*

[2] Wydanie polskie: *Head First JavaScript. Edycja polska*, Helion, Gliwice 2009 — *przyp. tłum.*

jQuery

Niniejsza książka jest poświęcona głównie bibliotece jQuery, jednak warto dużo lepiej poznać to rozbudowane, przyspieszające pracę i ciekawe narzędzie.

Witryny

- **Blog jQuery** (*http://blog.jquery.com/*) pozwala być na bieżąco ze wszystkimi zmianami wprowadzanymi w jQuery.
- **Dokumentacja jQuery** (*http://docs.jquery.com*) to miejsce stanowiące główne źródło poszukiwań odpowiedzi na wszelkie pytania związane z tą biblioteką. Wszystkie możliwości, funkcje i tajniki biblioteki jQuery zostały tu dokładnie opisane. Przykłady demonstrują działanie wszystkich funkcji biblioteki, dzięki czemu można się dowiedzieć, jakie są zalecane sposoby ich stosowania i jak powinny działać.
- **jQuery Fundamentals** (*http://jqfundamentals.com/*) to witryna prezentująca jQuery w unikalnym, praktycznym stylu. Nie tylko wyjaśnia podstawowe pojęcia związane z tą biblioteką, lecz także na każdej stronie udostępnia „piaskownicę" JavaScript, dzięki której czytelnik może eksperymentować z prezentowanym kodem i na bieżąco oglądać wyniki jego działania.

Książki

- *JQuery in Action* autorstwa Beara Bibeaulta i Yehudy Katza (wydawnictwo Manning) to szczegółowe omówienie biblioteki jQuery z wieloma przykładami. Książka ta wymaga pewnej wiedzy z zakresu języka JavaScript i programowania.
- *jQuery Cookbook* to książka wydana przez wydawnictwo O'Reilly, zawierająca wiele „przepisów" na rozwiązywanie najczęściej spotykanych zadań i problemów, przed jakimi stają programiści. Została napisana przez sporą grupę osób, spośród których wiele to najbardziej błyskotliwe umysły w dziedzinie stosowania jQuery.

Zaawansowany język JavaScript

O tak, język JavaScript jest *bardziej* skomplikowany, niż możesz sądzić po lekturze tej książki. Kiedy dobrze opanujesz podstawy, możesz zechcieć wzbogacić wiedzę na temat tego złożonego języka.

Artykuły i prezentacje

- **JS-Must-Watch** (*https://github.com/bolshchikov/js-must-watch/*) to repozytorium na serwisie GitHub zawierające listę najlepszych prezentacji i klipów wideo poświęconych językowi JavaScript.

Witryny

- **Eloquent JavaScript** (*http://eloquentjavascript.net*) to witryna z samouczkiem języka JavaScript. Jest dobrze uporządkowana, a lekcje są przedstawione w pomysłowy sposób. Choć teoretycznie witryna ta jest przeznaczona dla początkujących, autor pisze tak, jakby odbiorcami byli zawodowi informatycy, dlatego nie jest to najlepsze źródło informacji, jeśli dopiero uczysz się języka JavaScript lub programowania. Jest także dostępna w formie drukowanej książki.
- **Sekcja poświęcona językowi JavaScript w witrynie Douglasa Crockforda** (*http://javascript.crockford.com/*) zawiera wiele (skomplikowanych) materiałów na temat tego języka. Witryna jest bardzo bogata w informacje, przy czym zrozumienie niektórych z nich wymaga posiadania specjalistycznej wiedzy.
- **Dział poświęcony językowi JavaScript na witrynie Mozilla Developers Network** (*https://developer.mozilla.org/en-US/docs/Web/JavaScript*) zawiera bardzo dużo informacji na temat JavaScriptu, w tym jego encyklopedię, o której wspomniano na początku tego dodatku, a także przewodnik (*https://developer.mozilla.org/en-US/docs/Web/JavaScript/Guide*) opisujący różne istniejące wersje języka oraz szczegółowe przykłady przeznaczone dla programistów o różnym poziomie wiedzy — od początkujących do zaawansowanych.

Książki

- *The Principles of Object-Oriented JavaScript*[3] napisana przez Nicholę Zakasa to krótka książka (ma mniej niż 100 stron) poświęcona zaawansowanym sposobom organizowania kodu. To pozycja dla profesjonalistów, którą warto przeczytać.
- *JavaScript Patterns*[4] (wydawnictwo O'Reilly). Jeśli naprawdę chcesz rozwinąć swoje umiejętności programowania w języku JavaScript, w tej książce znajdziesz programistyczne „wzorce" pokazujące, jak rozwiązywać najczęściej spotykane problemy, takie jak najlepsze sposoby korzystania z literałów obiektowych, formatu JSON oraz tablic. To pozycja dla zaawansowanych programistów.
- W tej nieco już starej, lecz wciąż dobrej książce *JavaScript: The Good Parts*[5] Douglas Crockford (wydawnictwo O'Reilly) opisuje najbardziej przydatne elementy języka JavaScript i metody unikania błędów. Douglas wie, o czym

[3] Wydanie polskie: *JavaScript. Zasady programowania obiektowego*, Helion, Gliwice 2014 — *przyp. tłum.*

[4] Wydanie polskie: *JavaScript. Wzorce*, Helion, Gliwice 2012 — *przyp. tłum.*

[5] Wydanie polskie: *JavaScript — mocne strony*, Helion, Gliwice 2009 — *przyp. tłum.*

pisze, ponieważ jest starszym architektem do spraw języka JavaScript w firmie Yahoo!, jak również twórcą formatu JSON. Książka jest krótka i treściwa, jednak zawiera wiele wartościowej wiedzy na temat właściwego korzystania z JavaScriptu.

CSS

Jeśli postanowiłeś zmierzyć się z tą książką, prawdopodobnie znasz już język CSS. JavaScript pozwala wykorzystać możliwości stylów CSS nie tylko do kontrolowania wyglądu elementów, ale też na przykład do przenoszenia ich po ekranie za pomocą animacji. Jeśli potrzebujesz przypomnieć sobie informacje o języku CSS, w tym podrozdziale znajdziesz listę kilku przydatnych źródeł.

Witryny

- **Kompletny przewodnik po języku CSS w witrynie WestCiv** (*http://www.westciv.com/style_master/academy/css_tutorial/*) opisuje niemal każdy aspekt kaskadowych arkuszy stylów. Nie poznasz tu zestawu różnych technik, ale znajdziesz solidne omówienie podstaw języka CSS oraz tworzenia stylów i ich arkuszy.
- **Dokumentacja CSS na witrynie Mozilla Developer Network** (*https://developer.mozilla.org/en-US/docs/Web/CSS/Reference*) zawiera alfabetyczną listę wszystkich właściwości CSS wraz z wyczerpującymi informacjami na ich temat.
- **Poradnik Selectutorial** (*http://css.maxdesign.com.au/selectutorial/*) to doskonałe miejsce na naukę składni selektorów języka CSS. Ponieważ jQuery oparto na pomyśle stosowania selektorów do manipulowania kodem HTML strony, warto bardzo dobrze opanować to zagadnienie.

Książki

- *CSS: The Missing Manual*[6] wydanie 3. autorstwa Davida Sawyera McFarlanda (wydawnictwo O'Reilly) to wyczerpująca książka o kaskadowych arkuszach stylów napisana w formie samouczka. Obejmuje szczegółowe omówienie języka CSS, a także praktyczne przykłady i wskazówki pomocne przy rozwiązywaniu problemów, które pozwolą się upewnić, że kod CSS będzie działał w różnych przeglądarkach.
- *CSS: The Definitive Guide*[7] autorstwa Erica Meyera (wydawnictwo O'Reilly). Tytuł mówi sam za siebie — w tej książce opisano język CSS na takim poziomie szczegółowości, że praktycznie niemożliwe jest przeczytanie jej całej za jednym razem.

[6] Wydanie polskie: *CSS3. Nieoficjalny podręcznik. Wydanie III*, Helion, Gliwice 2013 — *przyp. tłum.*

[7] Wydanie polskie: *CSS. Kaskadowe arkusze stylów. Przewodnik encyklopedyczny. Wydanie III*, Helion, Gliwice 2008 — *przyp. tłum.*

Skorowidz

E

F

G

H

I

J

K

L

Ł

M

N

O

P

R

S

Ś

T

U

W

z, ź