

Buduj zaawansowane i interaktywne strony WWW!

JavaScript i jQuery

nieoficjalny podręcznik



David Sawyer McFarland

O'REILLY®

Helion 

Tytuł oryginału: JavaScript & jQuery: The Missing Manual

Thumaczenie: Piotr Rajca

ISBN: 978-83-246-5703-2

© 2012 Helion S.A.

Authorized Polish translation of the English edition of JavaScript & jQuery: The Missing Manual, 2nd Edition ISBN 9781449399023 © 2012 David Sawyer McFarland.

This translation is published and sold by permission of O'Reilly Media, Inc., which owns or controls all rights to publish and sell the same.

All rights reserved. No part of this book may be reproduced or transmitted in any form or by any means, electronic or mechanical, including photocopying, recording or by any information storage retrieval system, without permission from the Publisher.

Wszelkie prawa zastrzeżone. Nieautoryzowane rozpowszechnianie całości lub fragmentu niniejszej publikacji w jakiejkolwiek postaci jest zabronione. Wykonywanie kopii metodą kserograficzną, fotograficzną, a także kopiowanie książki na nośniku filmowym, magnetycznym lub innym powoduje naruszenie praw autorskich niniejszej publikacji.

Wszystkie znaki występujące w tekście są zastrzeżonymi znakami firmowymi bądź towarowymi ich właścicielami.

Wydawnictwo HELION dołożyło wszelkich starań, by zawarte w tej książce informacje były kompletne i rzetelne. Nie bierze jednak żadnej odpowiedzialności ani za ich wykorzystanie, ani za związane z tym ewentualne naruszenie praw patentowych lub autorskich. Wydawnictwo HELION nie ponosi również żadnej odpowiedzialności za ewentualne szkody wynikłe z wykorzystania informacji zawartych w książce.

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/jsjqnp.zip>

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

Printed in Poland.

- [Poleć książkę na Facebook.com](#)
- [Kup w wersji papierowej](#)
- [Oceń książkę](#)

- [Księgarnia internetowa](#)
- [Lubię to! » Nasza społeczność](#)

Spis treści

Nieoficjalna czołówka	11
Wprowadzenie	15
Część I. Wprowadzenie do języka JavaScript	33
Rozdział 1. Pierwszy program w języku JavaScript	35
Wprowadzenie do programowania	36
Czym jest program komputerowy?	38
Jak dodać kod JavaScript do strony?	38
Zewnętrzne pliki JavaScript	40
Pierwszy program w języku JavaScript	42
Dodawanie tekstu do stron	45
Dołączanie zewnętrznych plików JavaScript	46
Wykrywanie błędów	48
Konsola JavaScript w przeglądarce Firefox	48
Wyświetlanie okna dialogowego błędów w Internet Explorerze 9	50
Konsola błędów w przeglądarce Chrome	51
Konsola błędów w przeglądarce Safari	51
Rozdział 2. Gramatyka języka JavaScript	55
Instrukcje	55
Wbudowane funkcje	56
Typy danych	56
Liczby	57
Łańcuchy znaków	57
Wartości logiczne	58
Zmienne	59
Tworzenie zmiennych	59
Używanie zmiennych	62

Używanie typów danych i zmiennych	63
Podstawowe operacje matematyczne	64
Kolejność wykonywania operacji	65
Łączeniełańcuchów znaków	65
Łączenie liczb iłańcuchów znaków	66
Zmienianie wartości zmiennych	67
Przykład — używanie zmiennych do tworzenia komunikatów	69
Przykład — pobieranie informacji	70
Tablice	72
Tworzenie tablic	74
Używanie elementów tablicy	75
Dodawanie elementów do tablicy	76
Usuwanie elementów z tablicy	79
Przykład — zapisywanie danych na stronie za pomocą tablic	79
Krótkalekcja o obiektach	82
Komentarze	85
Kiedy używać komentarzy?	86
Komentarze w tej książce	87
Rozdział 3. Dodawanie struktur logicznych i sterujących	89
Programy reagujące inteligentnie	89
Podstawy instrukcji warunkowych	91
Uwzględnianie planu awaryjnego	94
Sprawdzanie kilku warunków	94
Bardziej skomplikowane warunki	97
Zagnieżdżanie instrukcji warunkowych	99
Wskazówki na temat pisania instrukcji warunkowych	100
Przykład — używanie instrukcji warunkowych	101
Obsługa powtarzających się zadań za pomocą pętli	104
Pętle while	104
Pętle i tablice	106
Pętle for	107
Pętle do-while	109
Funkcje — wielokrotne korzystanie z przydatnego kodu	110
Krótki przykład	112
Przekazywanie danych do funkcji	113
Pobieranie informacji z funkcji	114
Unikanie konfliktów między nazwami zmiennych	116
Przykład — prosty quiz	118
Cześć II. Wprowadzenie do biblioteki jQuery	125
Rozdział 4. Wprowadzenie do jQuery	127
Kilka słów o bibliotekach JavaScript	127
Jak zdobyć jQuery?	129
Dodawanie jQuery do strony	132
Podstawowe informacje o modyfikowaniu stron WWW	134

Zrozumieć DOM	138
Pobieranie elementów stron na sposób jQuery	140
Proste selektory	141
Selektory zaawansowane	143
Filtry jQuery	146
Zrozumienie kolekcji jQuery	148
Dodawanie treści do stron	149
Zastępowanie i usuwanie wybranych elementów	152
Ustawianie i odczyt atrybutów znaczników	154
Klasy	154
Odczyt i modyfikacja właściwości CSS	155
Jednoczesna zmiana wielu właściwości CSS	157
Odczyt, ustawienia i usuwanie atrybutów HTML	159
Wykonanie akcji na każdym elemencie kolekcji	160
Funkcje anonimowe	160
this oraz \$(this)	162
Automatycznie tworzone, wyróżniane cytaty	163
Opis rozwiązania	164
Kod rozwiązania	165
Rozdział 5. Akcja i reakcja — ożywianie stron za pomocą zdarzeń	169
Czym są zdarzenia?	169
Zdarzenia związane z myszą	171
Zdarzenia związane z dokumentem i oknem	172
Zdarzenia związane z formularzami	173
Zdarzenia związane z klawiaturą	174
Obsługa zdarzeń przy użyciu jQuery	174
Przykład — wyróżnianie wierszy tabeli	177
Zdarzenia specyficzne dla biblioteki jQuery	181
Oczekiwanie na wczytanie kodu HTML	182
Zdarzenia biblioteki jQuery	183
Obiekt reprezentujący zdarzenie	185
Blokowanie standardowych reakcji na zdarzenia	186
Usuwanie zdarzeń	187
Zaawansowane zarządzanie zdarzeniami	188
Inne sposoby stosowania funkcji bind()	189
Przykład — jednostronicowa lista FAQ	191
Omówienie zadania	191
Tworzenie kodu	192
Rozdział 6. Animacje i efekty	197
Efekty biblioteki jQuery	197
Podstawowe wyświetlanie i ukrywanie	198
Wygaszanie oraz rozjaśnianie elementów	200
Przesuwanie elementów	202
Przykład: wysuwany formularz logowania	202
Tworzenie kodu	203



Animacje	205
Tempo animacji	207
Wykonywanie operacji po zakończeniu efektu	209
Przykład: animowany pasek ze zdjęciami	211
Tworzenie kodu	213
Cześć III. Dodawanie mechanizmów do stron WWW	217
Rozdział 7. Efekty związane z rysunkami	219
Zamiana rysunków	219
Zmienianie atrybutu src rysunków	220
Wstępne wczytywanie rysunków	221
Efekt rollover z użyciem rysunków	222
Przykład — dodawanie efektu rollover z użyciem rysunków	223
Omówienie zadania	224
Tworzenie kodu	225
Przykład — galeria fotografii z efektami wizualnymi	228
Omówienie zadania	228
Tworzenie kodu	230
Wzbogacona galeria z wtyczką FancyBox biblioteki jQuery	234
Podstawy	235
Tworzenie galerii zdjęć	237
Personalizacja efektu FancyBox	238
Przykład — galeria fotografii oparta na wtyczce FancyBox	244
Rozdział 8. Usprawnianie nawigacji	249
Podstawowe informacje o odnośnikach	249
Pobieranie odnośników w kodzie JavaScript	249
Określanie lokalizacji docelowej	250
Blokowanie domyślnego działania odnośników	251
Otwieranie zewnętrznych odnośników w nowym oknie	252
Tworzenie nowych okien	255
Właściwości okien	255
Otwieranie stron w okienku na pierwotnej stronie	259
Przykład — otwieranie strony na stronie	262
Animowane menu nawigacyjne	263
Kod HTML	264
Kod CSS	266
Kod JavaScript	268
Przykład	268
Rozdział 9. Wzbogacanie formularzy	271
Wprowadzenie do formularzy	271
Pobieranie elementów formularzy	273
Pobieranie i ustawianie wartości elementów formularzy	275
Sprawdzanie stanu przycisków opcji i pól wyboru	276
Zdarzenia związane z formularzami	277

Inteligentne formularze	281
Aktywowanie pierwszego pola formularza	282
Wyłączanie i włączanie pól	283
Ukrywanie i wyświetlanie opcji formularza	284
Przykład — proste wzbogacanie formularza	285
Aktywowanie pola	286
Wyłączanie pól formularza	286
Ukrywanie pól formularza	289
Walidacja formularzy	291
Wtyczka Validation	293
Podstawowa walidacja	294
Zaawansowana walidacja	297
Określanie stylu komunikatów o błędach	302
Przykład zastosowania walidacji	303
Prosta walidacja	303
Walidacja zaawansowana	305
Walidacja pól wyboru i przycisków opcji	308
Formatowanie komunikatów o błędach	311
Rozdział 10. Rozbudowa interfejsu stron WWW	313
Organizowanie informacji przy użyciu kart	314
Kod HTML	315
Kod CSS	316
Kod JavaScript	319
Przykład — panel kart	320
Dodawanie sliderów	325
Stosowanie slidera AnythingSlider	326
Przykład — AnythingSlider	327
Modyfikowanie wyglądu slidera	329
Modyfikacja działania slidera	332
Określanie wielkości i położenia elementów strony	333
Określanie wysokości i szerokości elementów	334
Określanie położenia elementu na stronie	337
Uwzględnianie przewinięcia strony	339
Dodawanie etykietek ekranowych	340
Kod HTML	340
Kod CSS	342
Kod JavaScript	343
Przykład — etykiety ekranowe	344
Część IV. AJAX — komunikacja z serwerem sieciowym	355
Rozdział 11. Wprowadzenie do AJAX-a	357
Czym jest AJAX?	357
AJAX — podstawy	360
Elementy układanki	360
Komunikacja z serwerem sieciowym	362



AJAX w bibliotece jQuery	365
Używanie funkcji load()	365
Przykład — korzystanie z funkcji load()	368
Funkcje get() i post()	372
Formatowanie danych przesyłanych na serwer	373
Przetwarzanie danych zwróconych z serwera	376
Obsługa błędów	380
Przykład — korzystanie z funkcji get()	380
Format JSON	386
Dostęp do danych z obiektów JSON	388
Złożone obiekty JSON	389
Rozdział 12. Flickr oraz Google Maps	393
Prezentacja JSONP	393
Dodawanie do witryny kanału Flickr	395
Tworzenie adresu URL	395
Stosowanie funkcji \$.getJSON()	398
Prezentacja danych kanału Flickr w formacie JSON	398
Przykład — dodawanie zdjęć z Flickr na własnej stronie	400
Wyświetlanie na własnej stronie map Google Maps	404
Określanie lokalizacji na mapie	407
Inne opcje wtyczki GoMap	409
Dodawanie znaczników	411
Dodawanie okienek informacyjnych do znaczników	415
Przykład zastosowania wtyczki GoMap	415
Cześć V. Rozwiązywanie problemów, wskazówki i sztuczki	419
Rozdział 13. Wykorzystywanie wszystkich możliwości jQuery	421
Przydatne informacje i sztuczki związane z jQuery	421
\$() to to samo, co jQuery()	421
Zapisywanie pobranych elementów w zmiennych	422
Jak najrzadsze dodawanie treści	423
Optymalizacja selektorów	425
Korzystanie z dokumentacji jQuery	426
Czytanie dokumentacji na stronie jQuery	430
Poruszanie się po DOM	432
Inne funkcje do manipulacji kodem HTML	438
Zaawansowana obsługa zdarzeń	441
Rozdział 14. Zaawansowane techniki języka JavaScript	445
Stosowanie łańcuchów znaków	445
Określanie długości łańcucha	446
Zmiana wielkości znaków w łańcuchu	446
Przeszukiwanie łańcuchów znaków: zastosowanie indexOf()	447
Pobieranie fragmentu łańcucha przy użyciu metody slice()	449

Odnajdywanie wzorów w łańcuchach	450
Tworzenie i stosowanie podstawowych wyrażeń regularnych	451
Tworzenie wyrażeń regularnych	451
Grupowanie fragmentów wzorców	456
Przydatne wyrażenia regularne	456
Dopasowywanie wzorców	461
Zastępowanie tekstów	463
Testowanie wyrażeń regularnych	464
Stosowanie liczb	464
Zamiana łańcucha znaków na liczbę	465
Sprawdzanie występowania liczb	467
Zaokrąglanie liczb	468
Formatowanie wartości monetarnych	468
Tworzenie liczb losowych	469
Daty i godziny	471
Pobieranie miesiąca	471
Określanie dnia tygodnia	472
Pobieranie czasu	472
Tworzenie daty innej niż bieżąca	476
Łączenie różnych elementów	477
Używanie zewnętrznych plików JavaScript	477
Tworzenie bardziej wydajnego kodu JavaScript	479
Zapisywane ustawień w zmiennych	479
Operator trójargumentowy	481
Instrukcja Switch	482
Tworzenie kodu JavaScript o krótkim czasie wczytywania	484
Rozdział 15. Diagnozowanie i rozwiązywanie problemów	487
Najczęstsze błędy w kodzie JavaScript	487
Brak symboli końcowych	488
Cudzysłowy i apostrofy	491
Używanie słów zarezerwowanych	492
Pojedynczy znak równości w instrukcjach warunkowych	493
Wielkość znaków	493
Nieprawidłowe ścieżki do zewnętrznych plików JavaScript	494
Nieprawidłowe ścieżki w zewnętrznych plikach JavaScript	494
Znikające zmienne i funkcje	496
Diagnozowanie przy użyciu dodatku Firebug	496
Instalowanie i włączanie dodatku Firebug	497
Przeglądanie błędów za pomocą dodatku Firebug	498
Śledzenie działania skryptu za pomocą funkcji console.log()	499
Przykład — korzystanie z konsoli dodatku Firebug	500
Diagnozowanie zaawansowane	503
Przykład diagnozowania	508
Dodatek A. Materiały związane z językiem JavaScript	515
Źródła informacji	515
Witryny	515
Książki	516



Podstawy języka JavaScript	516
Artykuły i prezentacje	516
Witryny	516
Książki	517
jQuery	517
Artykuły i prezentacje	517
Witryny	517
Książki	518
AJAX	518
Witryny	518
Książki	519
Zaawansowany język JavaScript	519
Artykuły i prezentacje	519
Witryny	519
Książki	520
CSS	520
Witryny	521
Książki	521
Skorowidz	525

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” od początku jej powstania, czyli wystarczająco długo, aby pamiętać aplikację HyperCard.

Holly Bauer (redaktorka) mieszka w Ye Olde Cambridge, w stanie Massachusetts. Za dnia jest redaktorką prowadzącą, a wieczorami i w weekendy — zapaloną kucharką, zdolną majsterkowiczką i entuzjastką projektowania nowoczesnego. Jej adres e-mail to: *holly@oreilly.com*.

Carla Spoon (recenzentka techniczna) jest niezależną pisarką i adjustatorką. Zapalona biegaczka, która pracuje i zaspokaja zamiłowanie do nowoczesnych gadżetów w swoim domowym biurze położonym w cieniu Mount Rainier. Jej adres e-mail to: carla_spoon@comcast.net.

Angela Howard (indeksatorka) tworzy indeksy do książek już od ponad 10 lat, zajmuje się głównie książkami technicznymi oraz, od czasu do czasu, także publikacjami z innych dziedzin, takich jak podróże, medycyna alternatywna czy też gekony tygrysie. Mieszka z mężem w Kalifornii, ma córkę i dwa koty.

Podziękowania

Gorąco dziękuję wszystkim, którzy pomagali mi w czasie prac nad tą książką i ostrzegli mnie od popełnienia kłopotliwych błędów, w tym Shelley Powers oraz Sevowi Suehringowi. 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 mądro, 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
- Buying a Home: The Missing Manual*, Nancy Conner
- CSS. Nieoficjalny podręcznik*, David Sawyer McFarland
- Fotografia cyfrowa według Davida Pogue'a*, David Pogue
- Dreamweaver CS5.5: The Missing Manual*, David Sawyer McFarland
- Droid X2: The Missing Manual*, Preston Gralla
- Droid 2: The Missing Manual*, Preston Gralla
- Excel 2007 PL. Nieoficjalny podręcznik*, Matthew MacDonald

- Facebook: The Missing Manual, Third Edition, E.A. Vander Veer*
FileMaker Pro 11: The Missing Manual, Susan Prosser i Stuart Gripman
Flash CS5.5: The Missing Manual, Chris Brover
Fotografia cyfrowa. Nieoficjalny podręcznik, Chris Grover i Barbara Brundage
Galaxy Tab: The Missing Manual, Preston Gralla
Google Apps: The Missing Manual, Nancy Conner
Google SketchUp: The Missing Manual, Chris Grover
iMovie '11 & iDVD: The Missing Manual, David Pogue i Aaron Miller
iPad 2: The Missing Manual, J.D. Biersdorfer
iPhone: The Missing Manual, Fourth Edition, David Pogue
*iPhone App Development: The Missing Manual, Fourth Edition,
Craig Hockenberry*
iPhoto '11: The Missing Manual, David Pogue i Lesa Snider
iPod: The Missing Manual, Ninth Edition, J.D. Biersdorfer i David Pogue
Komputery PC. Nieoficjalny podręcznik, Andy Rathbone
Living Green: The Missing Manual, Nancy Conner
Mac OS X Snow Leopard: The Missing Manual, David Pogue
Mac OS X Lion: The Missing Manual, David Pogue
Microsoft Project 2010: The Missing Manual, Bonnie Biafore
Motorola Xoom: The Missing Manual, Preston Gralla
Mózg. Nieoficjalny podręcznik, Matthew MacDonald
Netbooks: The Missing Manual, J.D. Biersdorfer
*Office 2010 PL. Nieoficjalny podręcznik, Nancy Connor, Chris Grover
i Matthew MacDonald*
Office 2011 for Macintosh: The Missing Manual, Chris Brover
Palm Pre: The Missing Manual, Ed Baig
Personal Investing: The Missing Manual, Bonnie Biafore
Photoshop CS5.5: The Missing Manual, Lesa Snider
Photoshop Elements 10: The Missing Manual, Barbara Brundage
PowerPoint 2007 PL. Nieoficjalny podręcznik, E.A. Vander Veer
Premiere Elements 8: The Missing Manual, Chris Grover
QuickBase: The Missing Manual, Nancy Conner
QuickBooks 2011: The Missing Manual, Bonnie Biafore
QuickBooks 2012: The Missing Manual, Bonnie Biafore
Switching to the Mac: The Missing Manual, Snow Leopard Edition, David Pogue
Switching to the Mac: The Missing Manual, Lion Edition, David Pogue
Internet. Nieoficjalny podręcznik, David Pogue i J.D. Biersdorfer

Tworzenie stron WWW. Nieoficjalny podręcznik. Wydanie II,
Matthew MacDonald

Wikipedia: The Missing Manual, John Broughton

Windows 7: The Missing Manual, David Pogue

Windows Vista PL. Nieoficjalny podręcznik, David Pogue

Word 2007 PL. Nieoficjalny podręcznik, Chris Grover

Your Body: The Missing Manual, Matthew MacDonald

Your Money: The Missing Manual, J.D. Roth

Wprowadzenie

Jeszcze nie tak dawno 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. Na przykład 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 przesyłania 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 (jak to zrobić, dowiesz się na stronie 327). Można też zrobić coś bardziej wyrafinowanego, na przykład pokazać na stronie znacznie więcej danych bez jednokrotnego natłoku informacyjnego, umieszczając je w niewielkich panelach, które użytkownik może wyświetlić po kliknięciu myszą (patrz strona 313). Można także utworzyć coś zabawnego i atrakcyjnego, takiego jak etykiety ekranowe pokazujące uzupełniające informacje na temat elementów prezentowanych na stronie (patrz strona 340).

Kolejną zaletą języka JavaScript jest błyskawiczność działania. Ta cecha umożliwia natychmiastowe reagowanie na działania użytkowników: kliknięcie odnośnika, wypełnienie formularza lub przeniesienie 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 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

JavaScript opracowano w firmie Netscape w 1995 roku, co oznacza, że język ten ma niemal tyle samo lat co sieć WWW. Choć JavaScript jest dziś uważany za w pełni wartościowe 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 kursem 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 po-mył 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 funkcjonowały prawidłowo w obu aplikacjach.

Uwaga: Po udostępnieniu przez Netscape języka JavaScript Microsoft wprowadził jScript — własną wersję języka JavaScript obsługiwanej 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 9, 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, dla tego 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 przeglądarki).

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. ECMAScript jest „oficjalną” specyfikacją języka, opracowaną i utrzymywana przez międzynarodową organizację standaryzacyjną o nazwie Ecma International (<http://www.ecmascript.org/>).

JavaScript jest wszędzie

JavaScript działa nie tylko na stronach WWW. Język ten okazał się tak użyteczny, że zastosowano go do tworzenia widżetów Yahoo i Apple Dashboard, programów dla telefonów iPhone oraz dodawania skryptów do wielu programów firmy Adobe, między innymi do Acrobatu, Photoshopa, Illustratora i Dreamweavera. Ta ostatnia aplikacja zawsze umożliwiała zaawansowanym programistom używającym języka JavaScript dodawanie nowych poleceń.

Ponadto język programowania używany we Flashu — ActionScript — oparto na języku JavaScript, dlatego opanowanie podstaw JavaScriptu to dobry wstęp do pracy nad projektami flashowymi.

Czym jest jQuery?

JavaScript ma jeden mały, krępujący sekret: pisanie w nim języków jest dosyć trudne. Choć pisanie w JavaScriptie 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 stworzona w celu ułatwienia pisania programów w tym języku. Biblioteka jQuery jest złożonym programem napisanym w JavaScriptie, który zarówno ułatwia rozwiązywanie 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 do trwał 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 FancyBox (która poznasz na stronie 234) pozwala przekształcić prostą stronę z grupą miniaturek w interaktywny pokaz slajdów — a wszystko 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 języków HTML i CSS znajdziesz w książce *Head First HTML with CSS and XHTML. Edycja polska* Elisabeth Freeman i Erica Freeman. Szczegółowe omówienie kaskadowych arkuszy stylów przedstawia książka *CSS. 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 HTML. 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 błędą 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.

Obecnie powszechnie używa się pięciu typów dokumentów HTML: HTML 4.01 Transitional, HTML 4.01 Strict, XHTML 1.0 Transitional, XHTML 1.0 Strict oraz HTML5 (nowa postać na scenie WWW). Są one do siebie bardzo podobne. Drobne różnice związane są ze sposobem zapisu znaczników oraz listą dozwolonych znaczników i atrybutów. Większość edytorów stron WWW dodaje odpowiednie deklaracje przy tworzeniu nowej strony. Jeśli chcesz się dowiedzieć, jak wyglądają deklaracje dokumentów różnych typów, ich szablony znajdziesz na stronie www.webstandards.org/learn/reference/templates.

Nie jest istotne, której wersji HTML-a używasz. Wszystkie współczesne przeglądarki obsługują każdą deklarację i prawidłowo wyświetlają strony pięciu podanych typów. Wybór deklaracji nie jest tak istotny jak sprawdzenie, czy strona przechodzi *validację*, co opisałem w ramce na stronie 21.

Uwaga: Język XHTML uznawano początkowo za przełom w obszarze tworzenia stron WWW. Jednak choć wciąż część osób uważa, że należy używać tylko tego języka, trendy się zmieniły. World Wide Web Consortium (W3C) wstrzymało prace nad standardem XHTML, koncentrując się na rozwoju języka HTML5.Więcej informacji na temat tego języka możesz zdobyć, sięgając po jedną z książek *HTML5: The Missing Manual* napisaną przez Matthew MacDonalda lub *HTML5: Up and Running* napisaną przez Marka Pilgrima (obie zostały wydane przez wydawnictwo O'Reilly).

Działanie znaczników HTML

W przykładowym kodzie z poprzedniej strony, podobnie jak w kodzie HTML każdej strony WWW, większość poleceń występuje w parach zawierających bloki tekstu i inne instrukcje. **Znaczniki** (inaczej **tagi**) wewnętrz 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 (<). Na przykład znacznik <p> oznacza początek akapitu, a znacznik </p> — jego koniec.

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 wewnętrz 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 , który służy do wyróżniania tekstu. Umieszczenie słów między tym znacznikiem a tagiem powoduje pogrubienie czcionki. Fragment kodu HTML Uwaga! 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 Kliknij tutaj!.

WIEDZA W PIŁUŁCE

Walidacja stron WWW

Na stronie 19 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.

Validator 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. Validator wskazuje między innymi błędnie napisane nazwy tagów i niezamknięte znaczniki. Organizacja W3C (ang. *World Wide Web Consortium*), odpowiedzialna za zarządzanie licznymi technologiami sieciowymi, udostępnia bezpłatny validator pod adresem <http://validator.w3.org>. Wystarczy skopiować kod HTML i wkleić go w formularzu, przesłać stronę lub wskazać validatorowi istniejącą witrynę. Narzędzie sprawdzi wtedy kod HTML i poinformuje, czy strona jest prawidłowa. Jeśli wykryje błędy, opisze je i wskaza wiersz ich wystąpienia w pliku HTML.

Jeśli używasz przeglądarki Firefox, możesz pobrać ze strony <http://users.skynet.be/mguerry/mozilla> wtyczkę HTML Validator. Umożliwia ona walidację strony bezpośrednio w przeglądarce. Wystarczy otworzyć stronę (także bezpośrednio z komputera), a validator wskaże błędy w kodzie HTML. Na stronie <http://zappatic.net/projects/safarivalidator> dostępne jest podobne narzędzie, Safari Tidy, przeznaczone dla przeglądarki Safari.

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. Pominiecie znacznika zamykającego, dwukrotne użycie tego samego identyfikatora lub błędne zagnieźdzenie tagów 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 *atribut*, 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, 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 tagi

zapewniają dodatkowe wskazówki strukturalne. Na przykład element `` określa listę wypunktowaną, która pozwala na przykład 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, na przykład 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 właściwość innego (na przykład animować kolor tła, by zmienił się z żółtego na czerwony). Można na przykład 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: formowanego elementu strony (*selekторa*) i instrukcji formatujących (*bloku deklaracji*). Selektem 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. Na przykład aby zmienić kolor i rozmiar czcionki wszystkich akapitów strony, należy użyć następującego kodu:

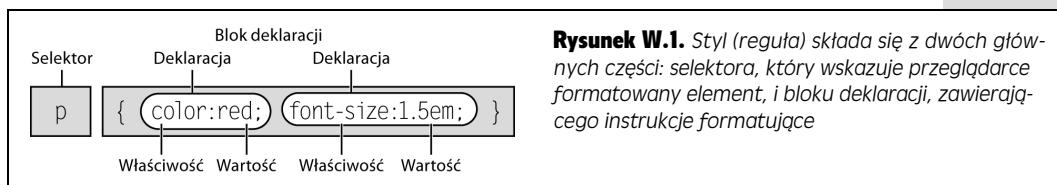
```
p { color: red; font-size: 1.5em; }
```

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. Są to:

- **Selektor**, który wskazuje przeglądarkę 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 tagach 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 140, 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 zamkającym tego typu (`}`).
- **Deklaracja**, czyli instrukcje formatujące, które znajdują się między nawiasami klamrowymi. Każda deklaracja ma dwie części — **właściwość** i **wartość** — a kończy się średnikiem.
- **Właściwości**, czyli opcje formatowania, których w CSS dostępnych jest duża liczba. 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 CSS. *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 jaskrawożasnozielony. 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 zamkający 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 kod 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:

- **Notepad++** (Windows, <http://notepad-plus-plus.org/>) to przyjaciel programisty. Koloruje składnię kodu 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, 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), skróty do dodawania znaczników HTML i tak dalej.
- **CoffeeCup Free HTML Editor** (Windows, www.coffeecup.com/free-editor) to bezpłatna wersja komercyjnego edytora CoffeeCup HTML (49 dolarów).
- **TextWrangler** (Mac, 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; 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/indigo_r; dostępna jest także wtyczka ułatwiająca dodawanie mechanizmu automatycznego uzupełniania dla jQuery (<http://marketplace.eclipse.org/category/free-tagging/jquery>).
- **Aptana Studio** (Windows, Linux, Mac; www.aptana.org) jest potężnym, bezpłatnym środowiskiem przeznaczonym do tworzenia kodów HTML, CSS, JavaScript, PHP oraz Ruby on Rails.

Oprogramowanie komercyjne

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

- **EditPlus** (Windows, www.editplus.com, 35 dolarów) to niedrogi edytor tekstu z obsługą kolorowania składni, protokołu FTP, automatycznego uzupełniania tekstu i innymi funkcjami przyspieszającymi pracę.
- **CoffeeCup** (Windows, www.coffeecup.com, 49 dolarów) to połączenie edytora tekstowego i graficznego. Programista może wpisać kod HTML lub utworzyć stronę za pomocą interfejsu graficznego.
- **textMate** (Mac, <http://macromates.com>, 44,85 euro) to nowy faworyt użytkowników Maca. Ten edytor tekstu zawiera wiele funkcji pozwalających zaoszczędzić czas programistom, którzy używają języka JavaScript. Jedną z nich jest automatyczne dodawanie drugiego symbolu w parach znaków przestankowych (program wstawia na przykład zamkający nawias po wpisaniu nawisu otwierającego).
- **BBEdit** (Mac, 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, www.macromedia.com/software/dreamweaver, 399 dolarów) 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 CS5.5: The Missing Manual*.
- **Expression Web Designer** (Windows, http://www.microsoft.com/expression/products/StudioWebPro_Overview.aspx, 149 dolarów) to nowe narzędzie Microsoftu do tworzenia stron WWW. Udostępnia ono wiele narzędzi dla profesjonalistów, w tym świetne funkcje do zarządzania stylami CSS.

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](http://developer.mozilla.org/en/JavaScript/Reference) (<http://developer.mozilla.org/en/JavaScript/Reference>), [Ecmascript.org](http://www.ecmascript.org/docs.php) (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. 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://docs.jquery.com/Main_Page. Jednak została 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 przedstawieniem się na sposób myślenia specyf-

ficzny 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 500 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, *znaczącą* — 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 „cegielki” 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, „Dodawanie mechanizmów do stron WWW”, zawiera wiele praktycznych przykładów wykorzystania języka JavaScript. Nauczysz się tworzyć wyskakujące paski nawigacyjne i interaktywne galerie fotografii. Zobaczysz, jak zwiększyć użyteczność formularzy przez dodanie mechanizmu ich walidacji (co zapobiega przesyłaniu niekompletnych danych), udostępnienie kontrolki kalendarza (co ułatwia wybieranie dat) i zmianę dostępnych opcji na podstawie wyborów dokonanych przez użytkownika. Na zakończenie przygotujesz ciekawy interfejs użytkownika — zawierający panele z kartami, slidery i wyskakujące okna dialogowe, które atrakcyjnie wyglądają i bezbłędnie działają.
- Część IV, „AJAX — komunikacja z serwerem sieciowym”, jest poświęcona technologii, która sprawiła, że JavaScript to jeden z najatrakcyjniejszych języków związanych ze stronami WWW. W tej części dowiesz się, jak używać języka JavaScript do komunikacji z serwerem sieciowym, co umożliwia pobieranie informacji i aktualizowanie stron bez konieczności wczytywania nowej strony na podstawie danych udostępnionych przez serwer.

Wskazówka: Na stronie poświęconej oryginalnej wersji książki znajdziesz instrukcje wyjaśniające krok po kroku instalację serwera na własnym komputerze, co umożliwia wypróbowanie tej ciekowej technologii (jej opis zawiera Część III). Więcej informacji o tej stronie znajdziesz w punkcie „Przykłady do pobrania” w dalszej części tego Wstępu.

- Część V, „Rozwiązywanie problemów, wskazówki i sztuczki”, to zakończenie zagadnień podstawowych i przejście do bardziej zaawansowanych. W tej części książki dowiesz się, jak efektywnie korzystać z biblioteki jQuery oraz jak posłużywać się jej bardziej zaawansowanymi funkcjami. Dowiesz się tu także, co można zarobić, gdy program nie będzie działał zgodnie z oczekiwaniemi 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 wcisnięcie oraz zwolnenie 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** wcisniętym 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 wcisniętym odpowiednim klawiszem, 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 dloni od klawiatury i używanie myszy do wyboru polecenia z menu (na przykład w celu włączenia pogrubienia) bywa rozpraszające. Dlatego wielu doświadczonych programistów woli uruchamiać instrukcję 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 *System preferences* (Mac OS X).

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

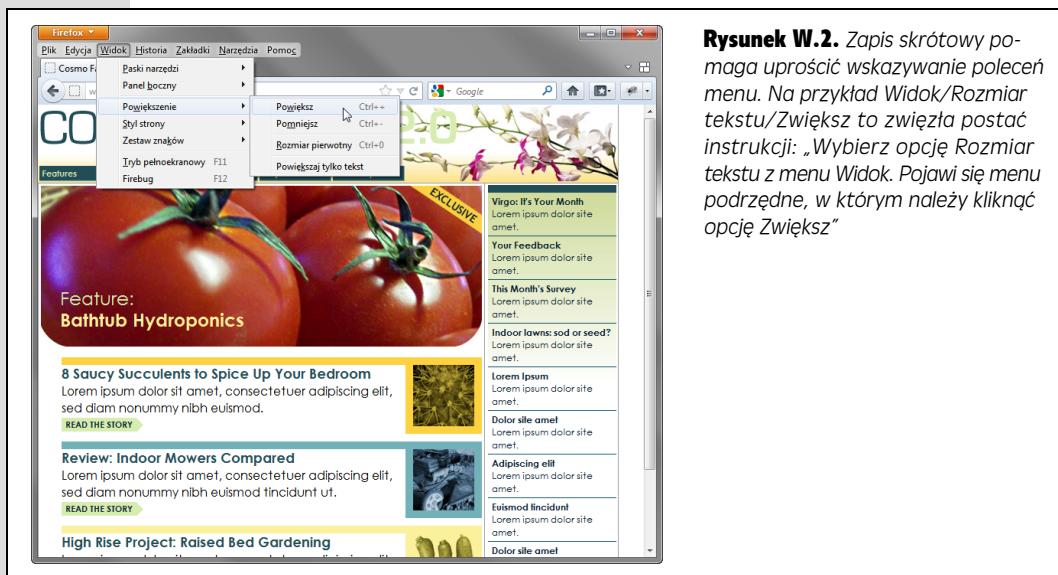
O/tych/ukośnikach

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.

Zasoby internetowe

Książka ma sprawić, byś mógł szybko i profesjonalnie zacząć tworzenie stron WWW. Dlatego też naturalne jest, że jakaś jej część jest związana z 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 im informacje o tym, co Ci się w tej książce podoba (lub co doprowadza Cię do szalu).



Rysunek W.2. Zapis skrótowy po-maga uprościć wskazywanie polecen menu. Na przykład Widok/Rozmiar tekstu/Zwiększa to związała postać instrukcji: „Wybierz opcję Rozmiar tekstu z menu Widok. Pojawi się menu podrzędne, w którym należy kliknąć opcję Zwiększa”

Przykłady do pobrania

Ta 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 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/przyklad/jsjqnp.zip>). Prawdopodobnie nie nauczysz się zbyt wiele, jeśli tylko przeczytasz tekst, leżąc wygodnie w hamaku. Jednak jeśli przeszledzisz 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/jsjqnp>.

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.





I

CZĘŚĆ

Wprowadzenie do języka JavaScript

- Rozdział 1. „Pierwszy program w języku JavaScript”
- Rozdział 2. „Gramatyka języka JavaScript”
- Rozdział 3. „Dodawanie struktur logicznych i sterujących”

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 i efektywnych 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 dla tego, 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 Marka Pilgrima *HTML5: Up and Running*, wydanej przez wydawnictwo O'Reilly.



Rysunek 1.1. JavaScript umożliwia reagowanie stron WWW na działania odwiedzających. W witrynie Amazon.com umieszczenie kurSORA nad odnośnikiem „Wish Lists” powoduje otwarcie zakładki, która pojawia się nad treścią strony i udostępnia dodatkowe opcje

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 są programistami.

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 oczekwać.

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. Nauka składni JavaScriptu przypomina poznawanie słówek i gramatyki obcego języka naturalnego. 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. Na początku będziesz prawdopodobnie popełniał wiele pomyłek tego rodzaju, co jest normalne przy nauce programowania.

WIEDZA W PIGUŁCE

Skrypty po stronie klienta i serwera

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 zwiększenia możliwości stron można użyć także wtyczek. Aplikacje 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 powodują 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.

Prawdę mówiąc, JavaScript może być także językiem programowania używanym po stronie serwera. 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.

Na początku programowanie w języku JavaScript może Ci się wydawać 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 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 JavaScriptie. 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 niezbędne do osiągnięcia celu. 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 obsługującej języki HTML i CSS to *silnik zarządzania układem (renderujący)*. Większość przeglądarek ma też *interpretator języka JavaScript*. Ten mechanizm przetwarza kod JavaScript i wykonuje operacje opisane w programie. Ponieważ przeglądarki obsługują domyślnie kod HTML, trzeba informować je o wystąpieniu kodu JavaScript za pomocą znacznika `<script>`.

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, komplikacja 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 komplikacji tworzenie i testowanie programów w językach skryptowych trwa dużo krócej.

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>
```

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 21). 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 otwierającym a zamykającym znacznikiem `<script>` wpisz kod JavaScript:

```
<!doctype html>
<html>
<head>
<meta charset="UTF-8">
<title>Strona WWW</title>
<script type="text/javascript">
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 umieszczenie 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 program, który dodaje animowane menu rozwijane. Ten sam pasek nawigacyjny powinien znaleźć się na każdej stronie witryny, jednak rozwiązanie polegające na kopiowaniu i wklejaniu kodu JavaScript we wszystkich plikach ma wiele wad.

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 wzrobić 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 używasz 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>
<title>Strona WWW</title>
<script src="navigation.js"></script>
</head>
```

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

WIEDZA W PIGUŁCE

Rodzaje adresów URL

Przy dodłączaniu 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 (innej strony, grafiki lub pliku JavaScript).

Ścieżka bezwzględna przypomina adres pocztowy — zawiera wszystkie informacje potrzebne do znalezienia pliku przeglądarkce uruchomionej w dowolnym komputerze. Taka ścieżka zawiera człon `http://` oraz nazwę domeny, katalogu i pliku, na przykład `http://www.cosmofarmer.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.sawmac.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. Założmy, 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 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 witryny na własnym komputerze bez wykorzystania serwera. Możesz przygotować zewnętrzny plik JavaScript, dodać 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.

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>`:

```
<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 234 dowiesz się, jak to zrobić). 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 46.

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 zobacysz, że niektóre skrypty korzystają z kodu umieszczonego w plikach zewnętrznych. Zdarza się to często przy korzystaniu z bibliotek JavaScript (zawierają one kod JavaScript upraszczający wykonywanie złożonych zadań programistycznych). W przykładzie rozpoczynającym się na stronie 46 zobacysz, jak używa 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 24 znajdziesz listę zalecanych aplikacji), przeglądarki i plików dostępnych na stronie książki w witrynie *helion.pl* (dokładne instrukcje znajdziesz w następnej 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 *Wyodrębni 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 zamkającym 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.

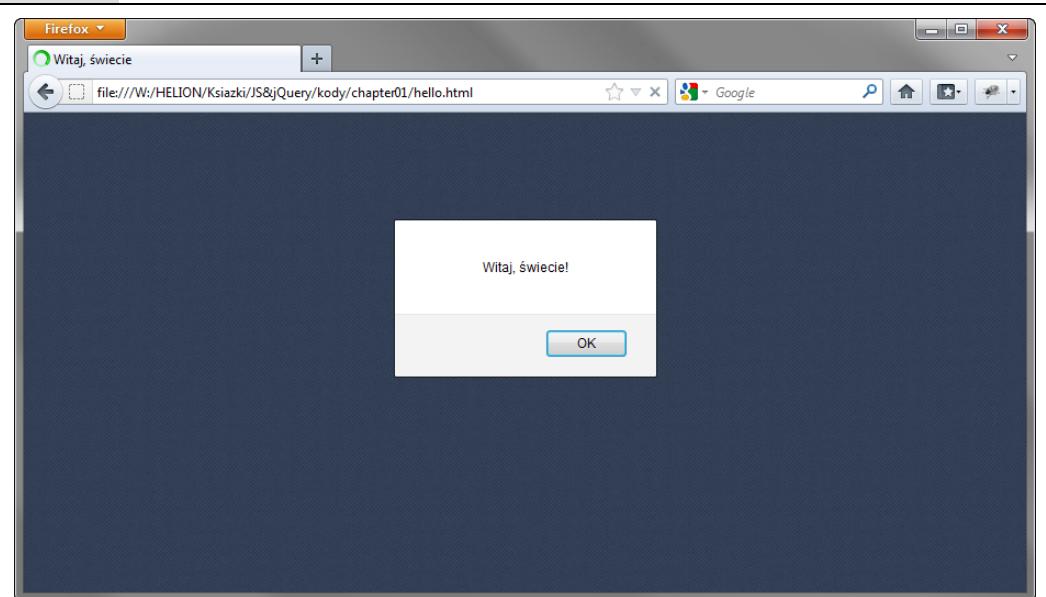
4. Wciśnij ponownie klawisz *Enter* i dodaj znacznik *</script>*. Kod powinien wyglądać następująco:

```
<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 jak ten w tekście.

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

Pojawi się okno dialogowe języka JavaScript (patrz rysunek 1.2). Zauważ, że w momencie pojawienia się okna strona jest pusta. Jeśli nie widzisz okienka przedstawionego na rysunku 1.2, możliwe, że w kodzie pojawił się błąd. Sprawdź dokładnie kod i zapoznaj się z następną wskazówką.



Rysunek 1.2. Okno dialogowe języka JavaScript pozwala szybko przykuć uwagę internauty. Wyświetlające je polecenie jest jednym z najprostszych w nauce i użytkowaniu

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

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

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.

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: Internet Explorer nie lubi 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 internetu. Przy przeglądaniu przykładów dołączonych do tej książki lepiej korzystać z innej przeglądarki, takiej jak Firefox, Chrome lub Safari, unikając dzięki temu konieczności klikania przycisku podczas otwierania każdej strony.

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.

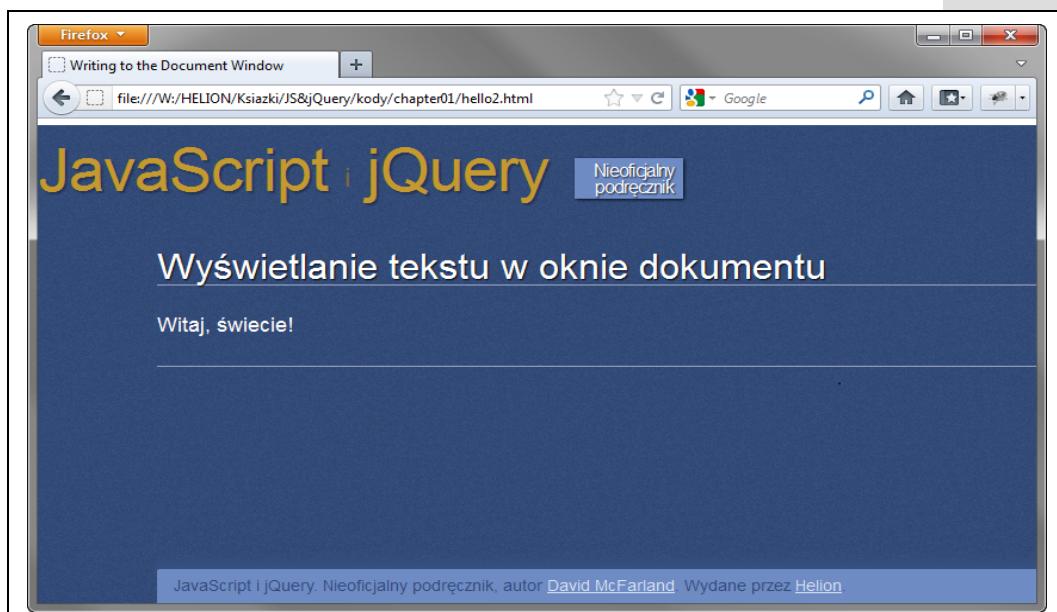
2. Bezpośrednio pod kodem `<h1>Wyświetlanie tekstu w oknie dokumentu</h1>` wpisz następujący fragment:

```
<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 1.3).



Rysunek 1.3. 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

Uwaga: Wśród pobranych plików znajdziesz też gotową wersję każdego przykładu. Jeśli wpisany kod JavaScript nie działa, porównaj własne rozwiążanie 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. Ważne, aby najpierw dobrze opanować podstawy. W dalszych 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.

Dołączanie zewnętrznych plików JavaScript

Na stronie 40 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 127. Szczególnie dobrze poznasz używaną w tej książce bibliotekę jQuery.

Jednak na razie poszerzysz swą wiedzę przez dołączenie do strony zewnętrznego pliku JavaScript i napisanie krótkiego programu, który wykonuje zdumiewające operacje:

1. Otwórz w edytorze plik *fadeIn.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 zamkającym znacznikiem *</head>* w górnej części strony i wpisz następujący kod:

```
<script src="../../js/jquery-1.6.3.min.js"></script>
```

Ten kod dołącza do dokumentu plik o nazwie *jquery-1.6.3.min.js*, znajdujący się w katalogu *js*. Kiedy przeglądarka wczyta stronę, pobierze także plik *jquery-1.6.3.min.js* i uruchomi zapisany w nim kod.

Następnie trzeba dodać do strony własny kod JavaScript.

Uwaga: Popularną praktyką jest dodawanie do nazw plików JavaScript numeru wersji. W tym przykładzie plik ma nazwę *jquery-1.6.3.min.js*. Numer 1.6.3 określa wersję biblioteki jQuery. Z kolei słowo min oznacza, że plik został *zminimalizowany*, czyli specjalnie zmniejszony, 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 tagami.

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 182, a na razie zapamiętaj, że fragment ten wykorzystuje kod z pliku *jquery-1.6.3.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:

```
$('body').hide().fadeIn(3000);
```

Wyniki wykonania tej instrukcji są magiczne. Sprawia ona, że zawartość strony najpierw znika, a następnie, powoli, ponownie staje się całkowicie widoczna (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-1.6.3.min.js"></script>
<script>
$(function() {
$('body').hide().fadeIn(3000);
});
</script>
</head>
```

8. Zapisz plik HTML i otwórz go w przeglądarce.

Powinieneś zobaczyć, jak zawartość strony powoli się pojawia. 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 *Zezwól na zablokowaną zawartość*, wyświetlony u dołu okna programu (przeczytaj także notatkę zamieszczoną na stronie 45).

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 trzech następnych rozdziałach 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 profesjonalisi 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.

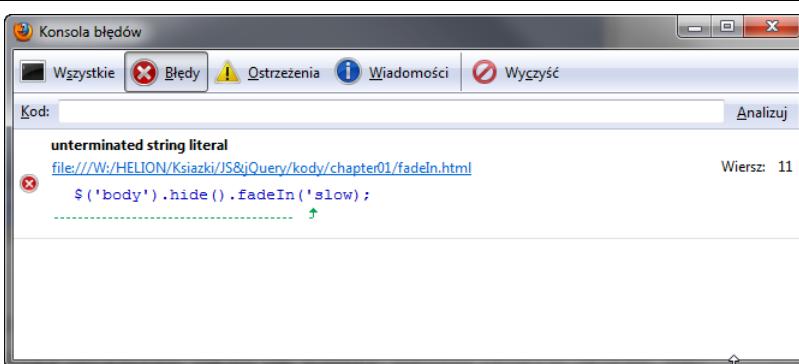
Jak więc można wykryć, gdzie wystąpił błąd? Jest wiele sposobów wyszukiwania usterek w programach JavaScript. W rozdziale 13. poznasz zaawansowane techniki *diagnostyczne*, jednak najprostszą metodą jest użycie przeglądarki. Większość przeglądarek śledzi błędy w kodzie JavaScript i zapisuje je w odrębnym 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 Firefox

Konsola przeglądarki Firefox to najlepsze miejsce do rozpoczęcia wykrywania błędów w kodzie. Nie tylko udostępnia stosunkowo jasny opis problemu (przy programowaniu żaden opis nie jest *w pełni zrozumiały*), ale też wskazuje numer wiersza, w którym wystąpiła usterka.

Konsola widoczna na rysunku 1.4 informuje o odnalezieniu niezamkniętego literatuła łańcuchowego (ang.: *unterminated string literal*), który pojawił się, ponieważ zabrakło drugiego apostrofu tworzącego parę z umieszczonym przed słowem „slow”.



Rysunek 1.4. Konsola JavaScript w Firefoksie wskazuje błędy w programach. Konsola zachowuje informacje o usterkach z wcześniejszych stron, dlatego lista może szybko stać się bardzo długa. Aby usunąć wszystkie dane o błędach, kliknij przycisk Wyczść

Konsola wyświetliła także nazwę pliku JavaScript, w jakim wystąpił błąd (w tym przypadku jest to plik *fadeIn.html*), oraz numer wiersza, w którym został odnaleziony (jest to wiersz 11.). Co więcej, miejsce wystąpienia błędu zostało nawet pokazane — w tym przypadku wyróżnione kolorem i wskazane strzałką.

Ostrzeżenie: Choć konsola wskazuje za pomocą strzałki lokalizację, w której Firefox napotkał usterkę, nie zawsze jest to miejsce popełnienia błędu. Czasem trzeba poprawić wcześniejszy lub dalszy kod.

Aby wyświetlić konsolę JavaScript, wybierz opcję *Dla twórców witryn/Konsola błędów* (ang. *Web Developer/Error Console* w systemie Windows oraz *Tools/Error Console* na komputerach Mac). Konsola to niezależne okno, które można swobodnie przemieszczać. Zawiera nie tylko błędy w kodzie JavaScript, ale również usterki w stylach CSS. Pamiętaj, aby wybrać przycisk *Błędy* w górnej części konsoli. Jeśli tego nie zrobisz, pojawią się również ostrzeżenia i komunikaty niezwiązane z problemami w kodzie JavaScript.

Wskazówka: Ponieważ konsola błędów wyświetla numer wiersza, w którym wystąpił problem, warto używać edytora tekstu z funkcją numerowania wierszy. Dzięki temu można szybko przejść z konsoli do edytora tekstu i znaleźć wiersz kodu, który trzeba poprawić.

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 błędów, które mogą często się powtarzać przy powielaniu kodu z tej książki:

- **Brakujący nawias) po liście argumentów.** Programiści czasem zapominają dodać nawias zamknięcy na końcu instrukcji. Na przykład w kodzie `alert =>('Witaj');` brakuje nawiasu po słowie Witaj.

- **Niezamknięty literal znakowy.** Łącuch znaków to zbiór znaków umieszczony w cudzysłowach lub apostrofach (więcej informacji na temat znajdziesz na stronie 57), na przykład słowo Witaj w kodzie alert('Witaj');. Łatwo zapomnieć o dodaniu otwierającego lub zamykającego cudzysłowu.
- **Niezdefiniowany element XXX.** Jeśli błędnie wpiszesz polecenie języka JavaScript, tak jak w kodzie aler('Witaj');, pojawi się błąd z informacją o tym, że (błędnie wpisana) instrukcja jest niezdefiniowana, na przykład „aler is not defined”.
- **Błąd składni.** Czasem Firefox 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 usterki tego typu często wymaga doświadczenia i zrozumienia kodu JavaScript.

Lista ta 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).

Uwaga: Doskonałym rozszerzeniem konsoli błędów w przeglądarce Firefox jest dodatek Firebug (<http://getfirebug.com/>). To wzór, na podstawie którego tworzone są narzędzia dla programistów dostępne w innych przeglądarkach, takich jak IE9, Chrome oraz Safari (opisane w dalszej części rozdziału). Dodatek Firebug został dokładniej opisany w rozdziale 15.

Wyświetlanie okna dialogowego błędów w Internet Explorerze 9

Przeglądarka Internet Explorer 9 udostępnia zbiór wyszukanych narzędzi programistycznych, służących nie tylko do odnajdywania błędów w kodzie JavaScript, lecz także do analizowania kodu CSS, HTML oraz przesyłu danych siecią. Po otwarciu okna narzędzi programistycznych jest widoczne w dolnej części okna przeglądarki (patrz rysunek 1.5). Okno narzędzi programistycznych można wyświetlić, naciśkając klawisz F12; ponowne naciśnięcie tego samego klawisza spowoduje jego zamknięcie. Błędy odnalezione w kodzie JavaScript są wyświetlane na zakładce *Konsola* (na rysunku 1.5 została zazkreślona). W odróżnieniu od konsoli w przeglądarce Firefox, na której wyświetlane są wszystkie błędy na odwiedzonych stronach, aby wyświetlić błąd w konsoli JavaScript w przeglądarce IE9, konieczne będzie odświeżenie strony.

Konsola przeglądarki IE9 wyświetla komunikaty błędów, podobne do prezentowanych w przedstawionej wcześniej przeglądarce Firefox. Przykładowo *Brak zakończenia stałej znakowej* (ang.: *Unterminated string constant*) jest odpowiednikiem komunikatu *Unterminated string literal*, jaki możemy zobaczyć w przeglądarce Firefox; informuje on o brakującym cudzysłówie lub apostrofie. Oprócz tego Internet Explorer podaje informacje o numerze wiersza w pliku HTML, w jakim wystąpił błąd.



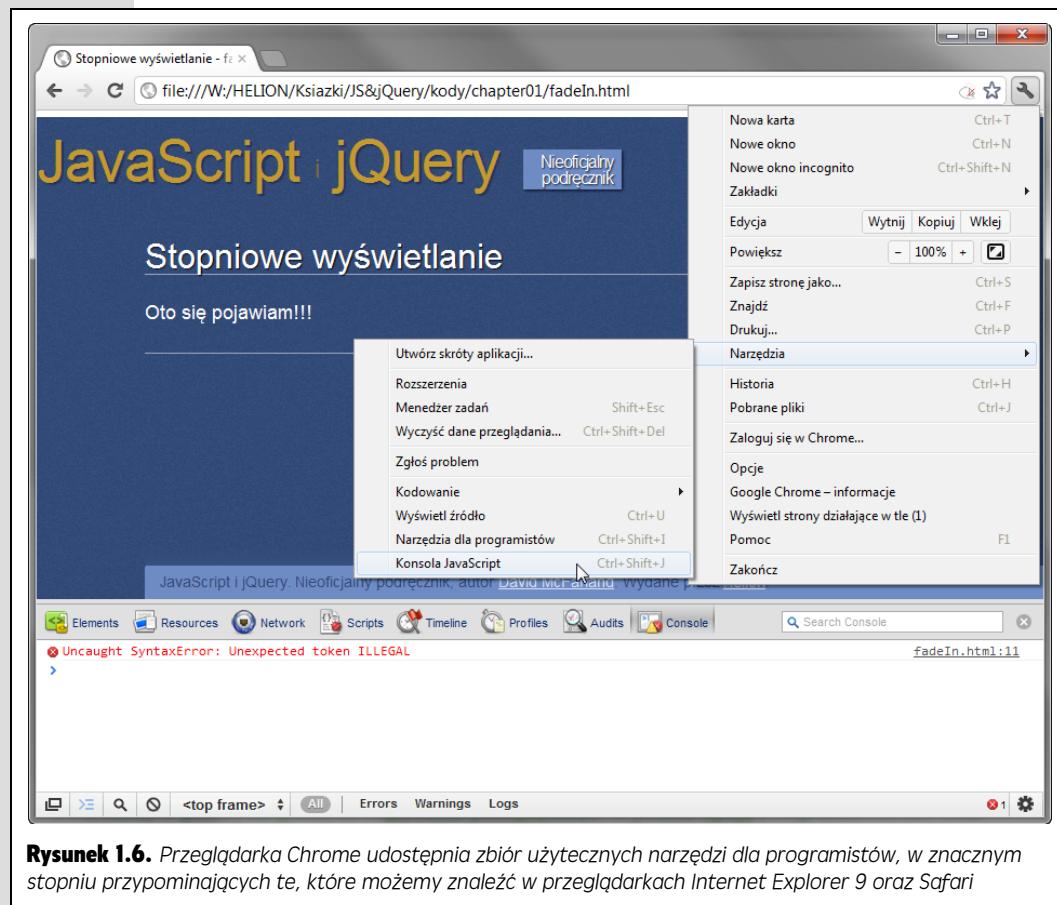
Rysunek 1.5. Okno narzędzi deweloperskich w przeglądarce Internet Explorer zapewnia dostęp do błędów w kodzie JavaScript, a także do bardzo wielu innych informacji

Konsola błędów w przeglądarce Chrome

Także przeglądarka Chrome firmy Google umożliwia przeglądanie błędów na udostępnianej konsoli JavaScript. Aby wyświetlić konsolę, należy kliknąć ikonę narzędzi (zakreślona na rysunku 1.6), wybrać opcję *Narzędzia*, a następnie *Konsola JavaScript*. Po wyświetleniu konsoli można kliknąć przycisk *Errors*, aby wyświetlić na niej tylko błędy w kodzie JavaScript, które nas teraz najbardziej interesują. Niestety, komunikaty błędów generowane przez Chrome są nieco bardziej tajemnicze niż w innych przeglądarkach. I tak pominięcie zamkniętego znaku cudzysłowu lub apostrofu spowoduje wyświetlenie komunikatu „Uncaught SyntaxError: Unexpected token ILLEGAL”. Nie jest to ani czytelne, ani łatwe do zrozumienia. Można odnieść wrażenie, że po popełnieniu jeszcze jednego podobnego błędu przeglądarka krzyknie: „Wypuścić mechatroniczne harty”.

Konsola błędów w przeglądarce Safari

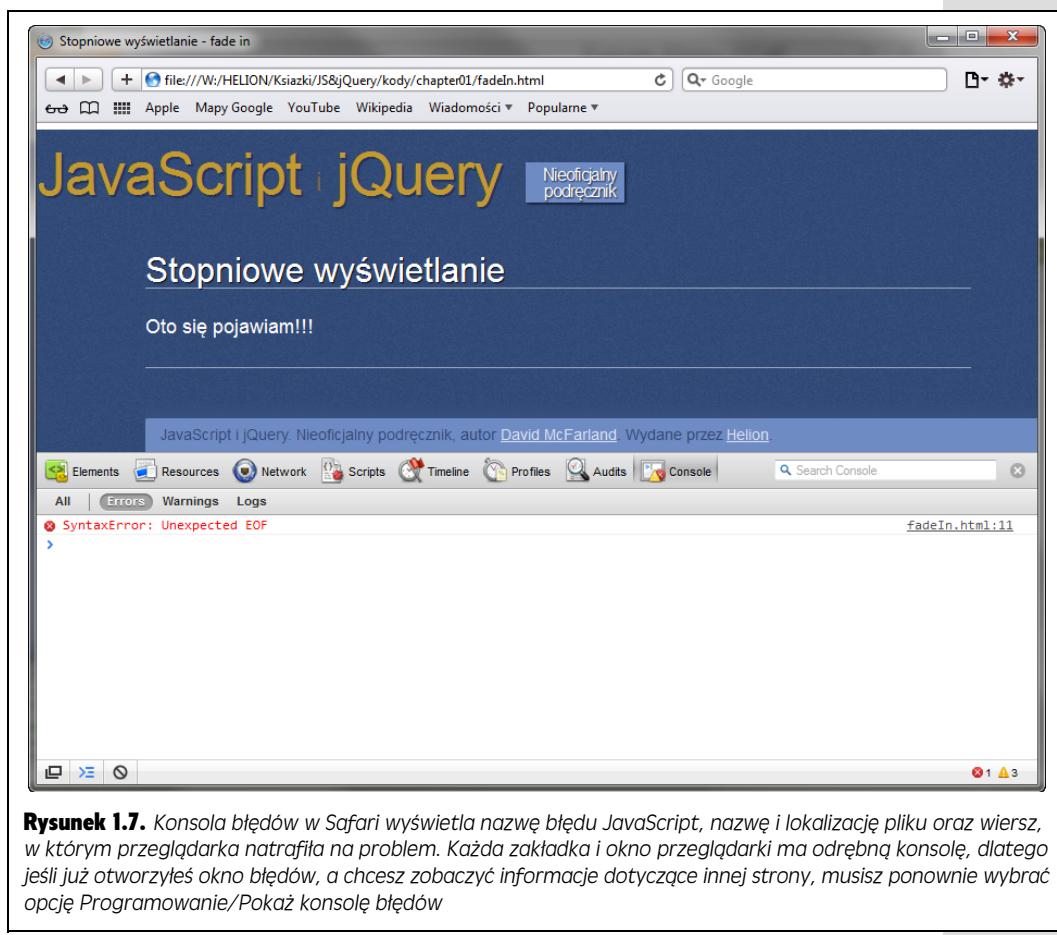
Konsolę błędów w przeglądarce Safari można otworzyć przy użyciu opcji *Programowanie/Pokaż konsolę błędów* (na komputerach Mac można użyć skrótu *Option+⌘+C*, a na komputerach z systemem Windows skrótu *Ctrl+Alt+C*). Jednak po zainstalowaniu Safari menu *Programowanie* jest zwykle wyłączone, dla tego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



Rysunek 1.6. Przeglądarka Chrome udostępnia zbiór użytecznych narzędzi dla programistów, w znacznym stopniu przypominających te, które możemy znaleźć w przeglądarkach Internet Explorer 9 oraz Safari

Aby wyświetlić menu *Programowanie* na komputerach Mac, wybierz opcję *Safari/Preferencje*. W systemie Windows kliknij ikonę koła zębatego wyświetlzoną w prawym, górnym rogu okna przeglądarki i wybierz opcję *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 na komputerze Mac, na pasku menu między opcjami *Zakładki* a *Okno* pojawi się menu *Programowanie*. Na komputerach z systemem Windows będzie ono także dostępne jako opcja w menu pokazywanym po kliknięciu ikony stron, wyświetlonej w prawym, górnym rogu okna przeglądarki. Aby uruchomić konsolę, wybierz opcję *Programowanie/Pokaż konsolę błędów* (patrz rysunek 1.7).



Rysunek 1.7. 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

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 móc 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 побieżnie 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.

Uwaga: 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 484.

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.

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 45, 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 aplikacji firmy Adobe, takich jak Acrobat i Dreamweaver, oraz w używanym we Flashu języku ActionScript, którego podstawą jest JavaScript).

Inne polecenia są uniwersalne i działają wszędzie tam, gdzie można używać 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 467.

JavaScript udostępnia wiele różnych poleceń, które poznasz na kartach tej książki. Szybkim sposobem na znalezienie polecień 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 na stronie 110.

Typy danych

Każdego dnia stykasz się z informacjami różnego typu. Nazwisko, cena posiłku, adres gabinetu doktora i data urodzin — to wszystko to istotne dane. Na ich podstawie podejmujesz decyzje związane z tym, co robić i jak żyć. Programy komputerowe funkcjonują podobnie — także przy wykonywaniu zadań polegają na informacjach. Na przykład aby 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 cyfra 5 odpowiada liczbie pięć. Można też przedstawić liczby dziesiętne: 5.25 lub 10.333333. JavaScript pozwala również stosować liczby ujemne, na przykład -130.

Ponieważ liczby są często używane w obliczeniach, w programach pojawia się wiele operacji matematycznych. Szczegółowy opis operatorów znajdziesz na stronie 63, 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 464.

Łańcuchy znaków

Do wyświetlania imion, zdań i serii znaków służą łańcuchy znaków. *Łańcuch znaków* to po prostu ciąg 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 cudzysłowa, 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 445 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 e-mail jest albo prawidłowy (`true`), albo nieprawidłowy (`false`). Strona może następnie odpowiednio zareagować na uzyskaną odpowiedź. Na przykład jeśli adres jest prawidłowy (`true`), formularz zostanie przesłany. Jeżeli adres nie jest prawidłowy (`false`), strona wyświetli komunikat o błędzie i zablokuje wysyłanie formularza.

CZĘSTO ZADAWANE PYTANIA

Umieszczanie cudzysłów 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 cudzysłowa, 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.

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

Inne rozwiązanie polega na nakazaniu interpreterowi dosłownego traktowania cudzysłów 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. Wcześniejszyszy fragment można zapisać również w następujący sposób: „Jan powiedział: \"Witaj\"”. Czasem użycie sekwencji ucieczki to jedyna możliwość, na przykład w kodzie „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 dodają ukośniki, aby zaznaczyć, że cudzysłowy są częścią łańcucha.

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 95.

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 *zmienną* pomocne przy takich zadaniach.

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 kulą 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. Na rysunku 2.1 widoczna jest inna gra, w której wykorzystano zmienne.

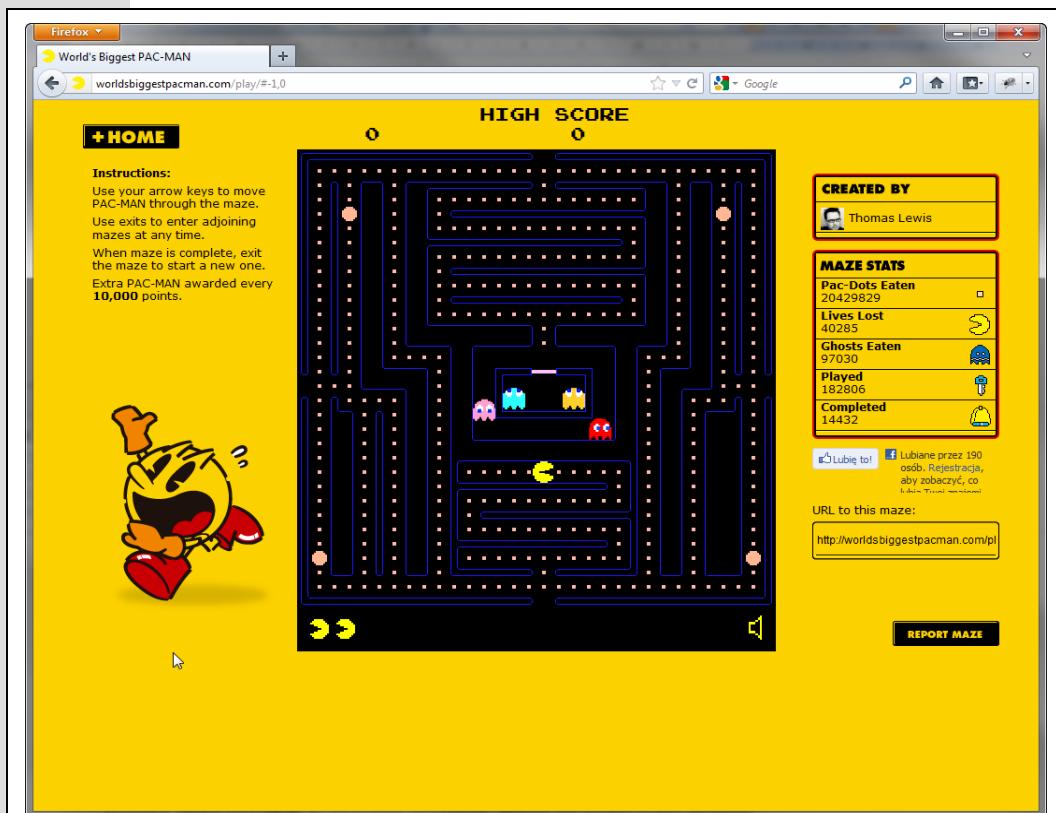
Możesz traktować zmienne jak specyficzny koszyk. Możesz umieszczać przedmioty w koszyku, zatrzymać go, a także zmienić jego zawartość. Jednak choć koszyk przechowuje różne elementy, wciąż pozostaje tym samym pojemnikiem.

Tworzenie zmiennych

Tworzenie zmiennych to dwuetapowy proces, który obejmuje *deklarowanie zmiennej* i jej *nazwanie*. W języku JavaScript można utworzyć zmienną o nazwie `score` 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. Na witrynie World's Biggest Pac-Man (<http://worldbiggestpacman.com>) język JavaScript został użyty do utworzenia gry Pac-Man udostępniającej tysiące różnych labiryntów. Gra przechowuje informacje o aktualnym wyniku, o najlepszym wyniku gracza, całkowitej ilości kropek „zjedzonych” przez wszystkich graczy oraz wiele innych danych statystycznych (wyświetlanych z prawej strony). Wszystkie są przykładami zmiennych, gdyż zmieniają wartość wraz z przebiegiem gry

Zmienne można nazwać w dowolny sposób, jednak trzeba przestrzegać przy tym 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 zmienność `SCORE` nie jest zmiennością `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 przedstawia tabela 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 zapisywaćłańcuchy 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 zisaną w zmiennej można pobrać za pomocą nazwy tej zmiennej. Na przykład aby wyświetlić w oknie dialogowym wartość umieszczoną w zmiennej `score`, należy użyć następującego kodu:

```
alert(score);
```

Warto zauważyc, że nazw zmiennych nie należy podawać w cudzysłowach. Cudzysłówne są potrzebne przy łańcuchach znaków, dlatego kod `alert('score')` wyświetli słowo „`score`”, a nie wartość zisaną 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.

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 rzeczą biorąc, język JavaScript traktuje znaki odstępu, nowego wiersza i tabulacji w sposób dosyć pozbawiony. 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 dodać 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 wstawać 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 158) oraz tablicom (patrz strona 72).

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

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

Wstawienie w takim miejscu znaku nowego wiersza przez wcisnięcie 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.

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

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 koniec 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. Tabela 2.2 przedstawia 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 \times (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 iłańcuchy 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 464), 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 oczekiwaniemi, 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:

$4 + (5 * 10);$

Łączanie łańcuchów znaków

Łączanie łańcuchów znaków w jeden ciąg to standardowa operacja programistyczna. Na przykład jeśli strona ma 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”.

Łączanie 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` jestłączona 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 łączaniu 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 (tu były to trzy łańcuchy).

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

Łączanie 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.”. 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) naka-
zuje 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 numSocks = 4;
var totalItems = Number(numOfShoes) + numSocks;
```

Polecenie Number() przekształca łańcuch na liczbę (jeśli jest to możliwe). Jeżeli łań-
cuch 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 za-
wierającego 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 cza-
sie 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 po-
pularnej 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 odejmowa-
nie, 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
<code>+=</code>	Dodaje wartość podaną po prawej stronie znaku równości do zmiennej podanej po stronie lewej.	<code>score += 10;</code>	<code>score = score + 10;</code>
<code>-=</code>	Odejmuje wartość podaną po prawej stronie znaku równości od zmiennej podanej po stronie lewej.	<code>score -= 10;</code>	<code>score = score - 10;</code>
<code>*=</code>	Mnoży wartość podaną po prawej stronie znaku równości przez zmienną podaną po stronie lewej.	<code>score *= 10;</code>	<code>score = score * 10;</code>
<code>/=</code>	Dzieli wartość podaną po prawej stronie znaku równości przez zmienną podaną po stronie lewej.	<code>score /= 10;</code>	<code>score = score / 10;</code>
<code>++</code>	Jeśli znajduje się bezpośrednio przed nazwą zmiennej, dodaje do niej 1.	<code>score++;</code>	<code>score = score + 1;</code>
<code>--</code>	Jeśli znajduje się bezpośrednio przed nazwą zmiennej, odejmuje od niej 1.	<code>score--;</code>	<code>score = score - 1;</code>

Te same reguły obowiązują przy dołączaniu do zmiennej łańcucha znaków. Na przykład jeśli zmienna zawiera łańcuch, można dodać do niej kilka kolejnych 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 43.

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 zamknięty 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 21).

3. Miedzy znacznikami `<script>` wpisz następujący kod:

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

Właśnie utworzyłeś dwie pierwsze zmienne — `firstName` i `lastName` — oraz zapisałś w nich łańcuchy znaków. Później dodasz do siebie 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 znajdziesz się wyświetlany na stronie tekst.

Uwaga: Istnieją bardziej wydajne metody dodawania kodu HTML do stron niż używanie polecenia `document.write()`. Poznasz je na stronie 149.

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 kilkułańcuchów znaków w jeden długilańcuch, 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 jedenłańcuch 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 swojej 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 48, dotyczące diagnozowania skryptów w przeglądarkach Firefox, Safari, Chrome oraz IE9.

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

```
var lastName = 'Wieczór';
```

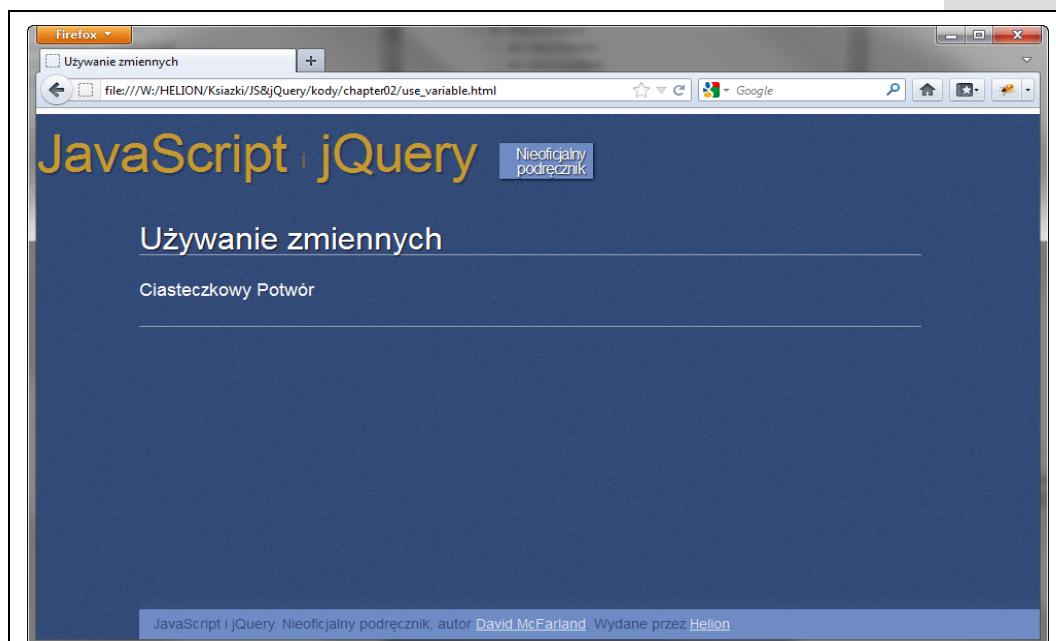
Zapisz stronę i wyświetl ją w przeglądarce. Gotowe. Teraz komunikat to „Ciasteczkowy Wieczór”. Działająca wersja tego skryptu znajduje się w pliku `complete_use_variable.html`.

Przykład — pobieranie informacji

Poprzedni skrypt ilustrował tworzenie zmiennych, jednak nie dowiedziałeś się, jak za pomocą zmiennych wyświetlać użytkownikom wyjątkowe, dopasowane wiadomości. W tym przykładzie zobaczyłeś, 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.

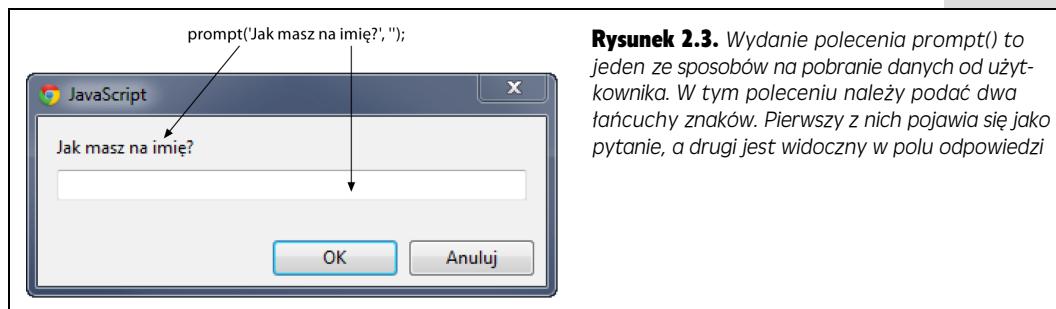


Rysunek 2.2. Choć pewnie nie sięgnąłeś po tę książkę, aby nauczyć się wyświetlać tekst „Ciasteczkowy potwór”, przykładowy skrypt ilustruje ważne zagadnienie — tworzenie i używanie zmiennych w języku JavaScript

2. Między pierwszymi znacznikami <script> (w sekcji nagłówkowej) wpisz powyższy 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

Uwaga: Przeglądarka IE7 nie pozwala korzystać z funkcji `prompt()` bez wcześniejszego zezwolenia na to w ustawieniach programu. Na szczęście przeglądarka ta bardzo szybko wychodzi z użycia.

Drugi łańcuch pojawia się w polu, w którym użytkownik wprowadza dane. Tu użyto *pustego łańcucha znaków*, czyli dwóch apostrofów, dlatego pole tekstowe jest puste. 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 69, ten kod łączy kilka łańcuchów znaków — otwierający znacznik akapitu, tekst, wartość zmiennej i zamkujący 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). Odswież stronę i wprowadź nowe dane, a tekst się zmieni. Właśnie tak powinny działać zmienne.

Tablice

Proste zmienne, takie jak te 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.



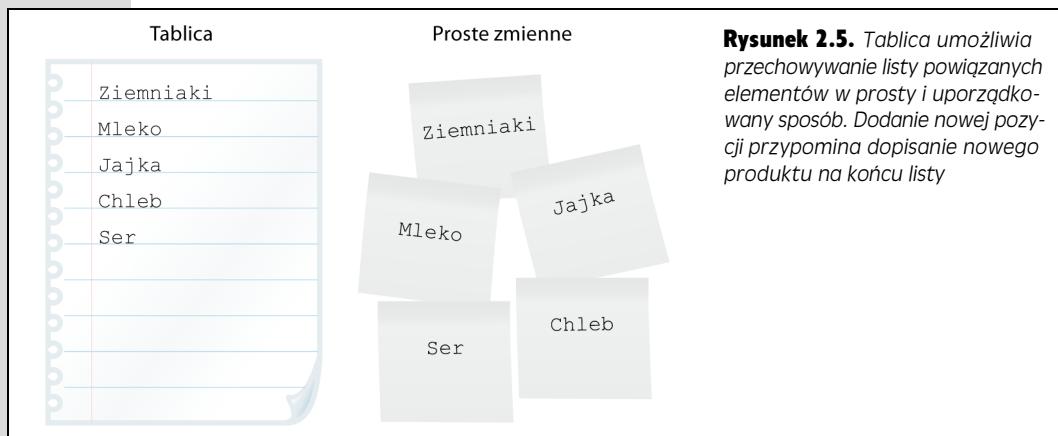
Wyobraź sobie, że masz utworzyć w języku JavaScript system obsługi koszyka zakupów, który zapisuje produkty zamawiane przez użytkownika. Gdyby program miał przechowywać każdy towar za pomocą prostych zmiennych, kod wyglądałby następująco:

```
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, jednak 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.



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 60. Przy tworzeniu tablicy należy umieścić listę elementów między otwierającym a zamkającym nawiasem kwadratowym — `[]`. Na przykład aby przygotować tablicę ze skrótnymi 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 59. 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 76.

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 (*literaty tablicowe*) preferują profesjoniści, 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, 223, 'www.oreilly.com', false];
```

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

Wcześniejsze tablice tworzyły 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 63, 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',), wcisnij klawisz *Enter*, a następnie wcisnij kilka razy klawisz spacji, aby odpowiednio umieścić 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 ponieważ tablica może zawierać wiele wartości, 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.', 'Sr.', '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.

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

```
days[0] = 'Poniedziałek';
```

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]

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, 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 104).

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 75 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. 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 poznaleś 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.

Jednak jeśli w tablicy trzeba uwzględnić kolejność elementów, używana metoda jest istotna. Założymy, 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()`.

PORADNIA DLA ZAAWANSOWANYCH

Tworzenie kolejek

Metody używane do dodawania (`push()`) i usuwania (`pop()` i `shift()`) elementów są często używane 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 kolejki FIFO (ang. *First In, First Out*, czyli pierwszy wchodzi — pierwszy wychodzi). Można zasymulować taki system za pomocą poleceń `push()` i `shift()`. Na przykład aby 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 i zarządzaniu różnymi kolejkami, na przykład listami odtwarzania, zadaniami do wykonania lub pokazami slajdów.

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 początku, a polecenie `shift()` — z końca. 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
<code>pop()</code>	<code>var p = [0,1,2,3]</code>	<code>p.pop()</code>	<code>[0,1,2]</code>	Usuwa z tablicy ostatni element.
<code>shift()</code>	<code>var p = [0,1,2,3]</code>	<code>p.shift()</code>	<code>[1,2,3]</code>	Usuwa z tablicy pierwszy element.

Polecenia `pop()` i `shift()`, podobnie jak `push()` i `unshift()`, po zakończeniu działania zwracają dane —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.pop();
```

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

Uwaga: Wśród przykładowych plików do tego rozdziału znajduje się strona, która umożliwia przedstawianie różnych poleceń związanych z tablicami. Jej nazwa to `array_methods.html`, a plik ten znajduje się w katalogu `Przykłady/R02`. 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.

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 43.

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 kod 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 74 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 ``. Znajduje się tu tylko zwykły kod HTML. Następny wiersz dodaje wartość pierwszego elementu tablicy `authors` oraz zamykające znaczniki `` 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 48 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. 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>');
```

W punkcie „Dodawanie elementów do tablicy” 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].

Następnie dodasz nowy element na początek tablicy.

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

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

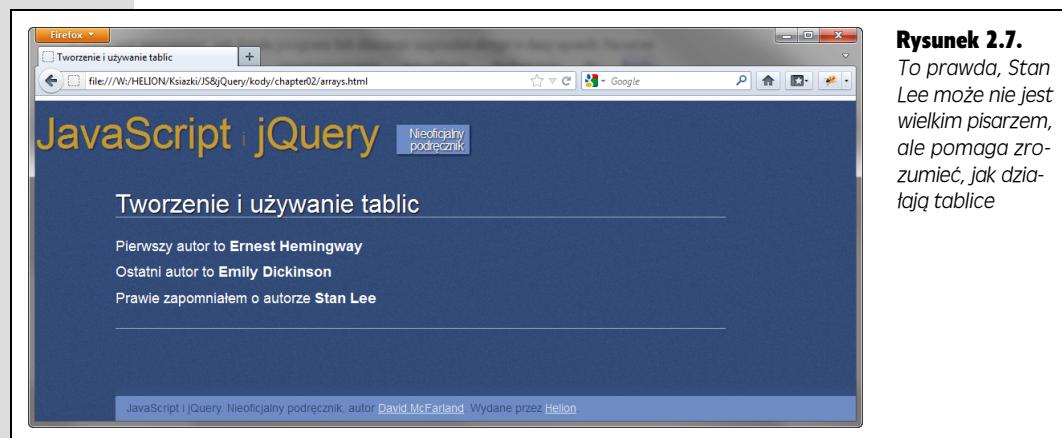
Na stronie 77 napisano, że metoda `unshift()` dodaje elementy na początek tablicy. Po wykonaniu tej instrukcji tablica `authors` będzie rozpoczynała się od wartości `['Stan Lee', 'Ernest Hemingway', 'Charlotte Bronte', 'Dante Alighieri', 'Emily Dickinson']`.

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 48).



Krótka lekcja o obiektach

Dotychczas dowiedziałeś się, że na stronie można coś zapisywać przy użyciu funkcji `document.write()`. Wiesz także, że w celu sprawdzania 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 przyrzyć 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 zapisałyśmy 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. Założymy 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 471). 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 pominać; 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 przyczyczaisz.

Wskaźówka: 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"
```

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

Czasem w trakcie 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 prosi o wprowadzenie zmian i dodanie nowych funkcji do świetnego skryptu, niegdyś znajomy kod JavaScript będzie mniej zrozumiałym. 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 jest zapomnieć, jak działa program lub dlaczego napisałeś skrypt w dany sposób. Na szczęście większość języków programowania umożliwia dodawanie do kodu notatek na użytku własny i innych programistów analizujących skrypt. JavaScript pozwala dodawać do kodu *komentarze*. Jeśli używałeś ich w kodzie 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 wykona wszystkie operacje z tego wiersza do czasu napotkania symboli *//*, kiedy to przejdzie na początek następnego 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. Założymy, że chcesz na początku kodu opisać działanie programu. Możesz to zrobić w następujący sposób:

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

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

/ To komentarz jednowierszowy. */*

Zwykle jeśli chcesz dodać krótki, jednowierszowy komentarz, powinieneś użyć sekwencji `//`. Do tworzenia komentarzy wielowierszowych warto stosować kombinacje `/* i */`.

Kiedy używać komentarzy?

Komentarz to nieocenione narzędzie w długich lub złożonych programach, których chcesz używać (lub 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 móc szybko sobie przypomnieć, jak działa skrypt, warto dodać komentarze. Pomogą one zrozumieć ogólną logikę programu i jego szczególnie skomplikowane fragmenty.

Uwaga: Dodawanie wielu komentarzy zwiększa rozmiar skryptu i wydłuża jego wczytywanie. Ogólnie rzeczą 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 484.

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.

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

```
/*
 * jQuery JavaScript Library v1.6.3
 * http://jquery.com/
 *
 * Copyright 2011, John Resig
 * Dual licensed under the MIT or GPL Version 2 licenses.
 * http://jquery.org/license
 *
 * Includes Sizzle.js
 * http://sizzlejs.com/
 * Copyright 2011, The Dojo Foundation
 * Released under the MIT, BSD, and GPL Licenses.
 *
 * Date: Wed Aug 31 10:35:15 2011 -0400
 */
```

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. Na przykład jeśli 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 móc ustalić, do czego służyły 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 powiniensem się ubrać?”, „Co mam zjeść na obiad?”, „Gdzie spędzić piątkowy wieczór?”. Wiele decyzji zależy od warunków. Założmy, że na piątkowy wieczór planujesz wyjście do kina. Prawdopodobnie zadasz sobie kilka pytań: „Czy grają jakiś dobry film?”, „Czy film jest wyświetlany o odpowiedniej godzinie?”, „Czy mam wystarczająco dużo pieniędzy, aby kupić bilet i dużą porcję popcornu?».

Założmy, że o odpowiedniej godzinie *jest* wyświetlany film. Następne pytanie to: „Czy mam wystarczającą ilość pieniędzy?». Jeśli tak, możesz wyruszać do kina. Jeżeli nie, zostaniesz w domu. Jeśli 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 najważniejszych elementów programistycznych. Umożliwiają one programom reagowanie

na różne sytuacje i działanie w inteligentny sposób. Będziesz używał ich bardzo często, a w tym miejscu możesz zapoznać się z kilkoma przykładami ich użyteczności:

- **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.

Rysunek 3.1 przedstawia 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 swoje 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 wykonyują 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.
}
```

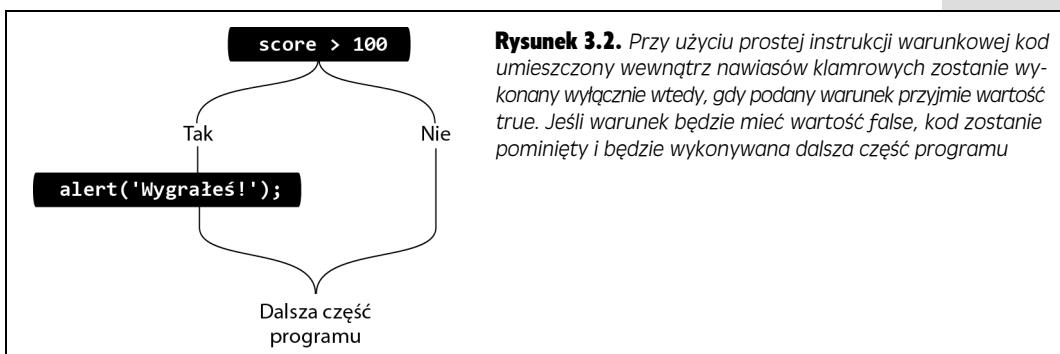
Ta instrukcja ma trzy części. Słowo kluczowe `if` informuje, że dalszy kod to instrukcja warunkowa. Nawiąsy zawierają pytanie typu „tak-nie”, nazywane *warunkiem* (więcej na ten temat dowiesz się już za chwilę). Nawiąsy klamrowe (`{ }`) wyznaczają początek i koniec kodu JavaScript, wykonywanego, jeśli 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 85.

Warunek to często porównanie dwóch wartości. Założmy, 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.



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
<code>==</code>	Równa się. Sprawdza, czy dwie wartości są takie same. Służy do porównywania liczb i łańcuchów znaków.
<code>!=</code>	Nie równa się. Sprawdza, czy dwie wartości są różne . Służy do porównywania liczb i łańcuchów znaków.
<code>====</code>	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 <code>'2' === 2</code> będzie prawdziwe, to wyrażenie <code>'2' === 2</code> już nie będzie, gdyż pierwsza wartość została zapisana w apostrofach (czyli jest łańcuchem znaków), natomiast druga jest liczbą. Korzystając z tego operatora, należy zachować dużą ostrożność, gdyż wartości wpisywane w formularzach — nawet liczby — zawsze są łańcuchami znaków; a zatem wykorzystanie tego operatora do porównywania liczb z wartością odczytaną z jakiegoś pola formularza na stronie WWW zawsze zwróci nieprawdę — wartość <code>false</code> .
<code>!==</code>	Nieidentyczny. Podobnie jak opisany powyżej operator dokładnie równy, także i ten porównuje zarówno wartości, jak i typy, na przykład wyrażenie <code>'2' != 2</code> zwraca wartość <code>false</code> , natomiast <code>'2' !== 2</code> zwraca <code>true</code> , bo choć porównywane wartości są takie same, to jednak ich typy są różne.
<code>></code>	Większy niż. Sprawdza, czy liczba po lewej stronie znaku jest większa od liczby po prawej stronie. Wyrażenie <code>2 > 1</code> jest prawdziwe, ponieważ liczba 2 jest większa od 1, jednak warunek <code>2 > 3</code> jest fałszywy, ponieważ liczba 2 nie jest większa od 3.
<code><</code>	Mniejszy niż. Sprawdza, czy liczba po lewej stronie znaku jest mniejsza od liczby po prawej stronie. Wyrażenie <code>2 < 3</code> jest prawdziwe, ponieważ liczba 2 jest mniejsza od 3, jednak warunek <code>2 < 1</code> jest fałszywy, ponieważ liczba 2 nie jest mniejsza od 1.
<code>>=</code>	Większy lub równy. Sprawdza, czy liczba po lewej stronie znaku jest większa od liczby po prawej stronie lub jej równa. Wyrażenie <code>2 >= 2</code> jest prawdziwe, ponieważ liczba 2 jest równa 2, jednak warunek <code>2 >= 3</code> jest fałszywy, ponieważ liczba 2 nie jest większa od 3 ani jej równa.
<code><=</code>	Mniejszy lub równy. Sprawdza, czy liczba po lewej stronie znaku jest mniejsza od liczby po prawej stronie lub jej równa. Wyrażenie <code>2 <= 2</code> jest prawdziwe, ponieważ liczba 2 jest równa 2, jednak warunek <code>2 <= 1</code> 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. Założymy, 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 155.

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.");  
}
```

Wykrywki odpowiadają 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 zamkającym nawiasem klamrowym można umieścić dowolną liczbę wierszy kodu JavaScript. W przykładowym quizie można też przechowywać liczbę prawidłowych odpowiedzi udzielonych przez gracza. Jeśli 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 68 2. wiersz kodu umieszczony w instrukcji warunkowej — numCorrect += 1 — dodaje 1 do aktualnej wartości zmiennej numCorrect.

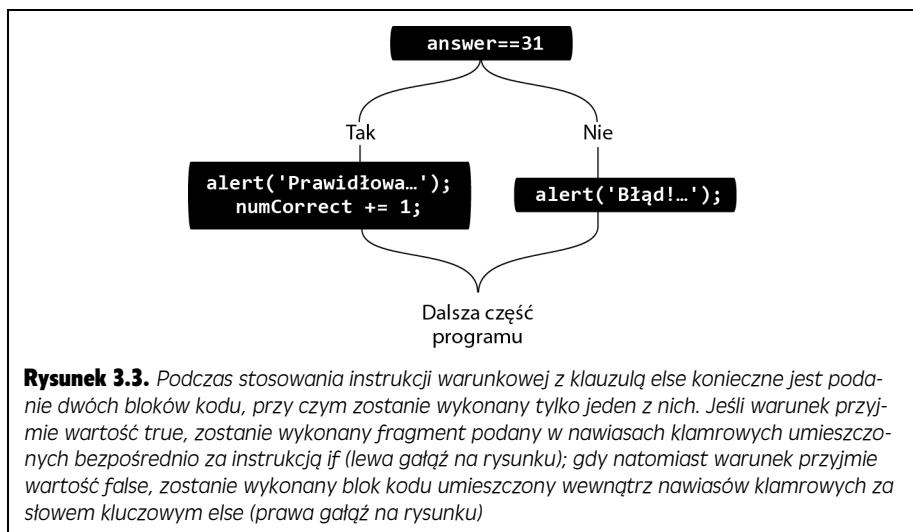
Miedzy nawiasy można wstawić więcej dodatkowych wierszy kodu, który program ma uruchomić, jeśli warunek będzie spełniony.

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*. Założymy, że program do obsługi quizu ma powiadomić gracza, czy udzielił dobrzej, 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.



Aby utworzyć klauzulę `else`, wystarczy wpisać słowo „`else`” po zamkającym 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 58 poznaleś 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. Na przykład możesz 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!

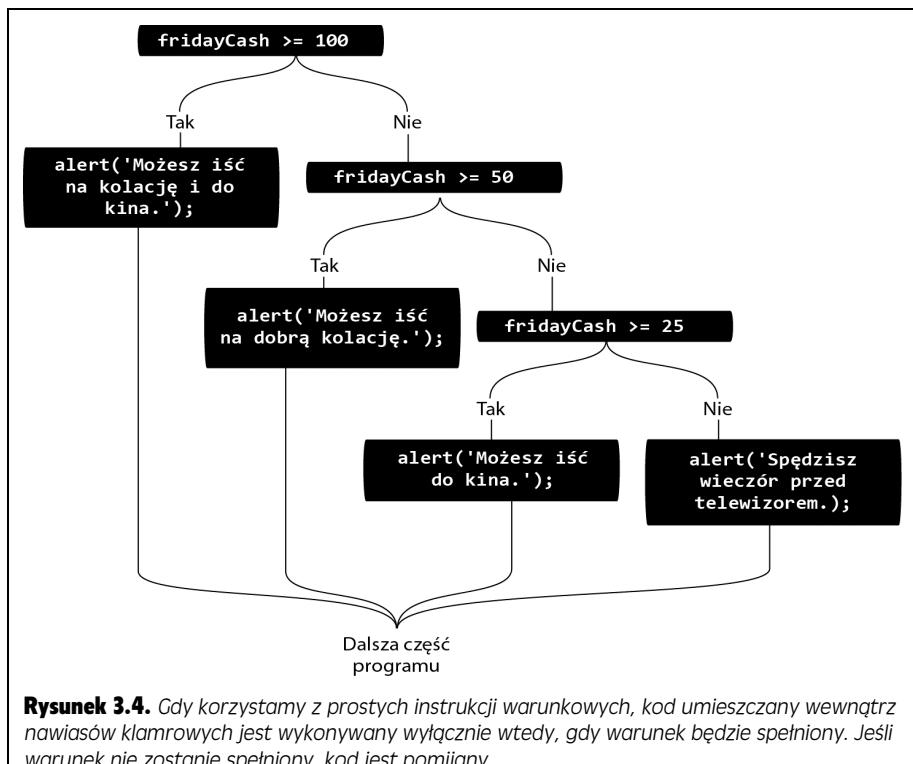
JavaScript umożliwia sprawdzenie serii warunków za pomocą instrukcji else if. Struktura ta działa w następujący sposób: instrukcja if jest powiązana z możliwością numer 1, a dalsze instrukcje else if pozwalają zadać dodatkowe pytania, które prowadzą do dodatkowych operacji. Na końcu jako możliwość rezerwową 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 poznaleś na stronie 70, 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 wewnętrznie nawiasów klamrowych jest wykonywany wyłącznie wtedy, gdy warunek będzie spełniony. Jeśli warunek nie zostanie spełniony, kod jest pomijany

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`.

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 482.

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

Ten kod testuje dwa warunki. Fragment `a < 10` sprawdza, czy wartość zmiennej `a` jest mniejsza od 10. Drugi warunek, `a > 1`, to pytanie: „Czy wartość `a` jest większa od 1?”. 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. Założmy, 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 klawiszem *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, wcisnij 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. Założmy, 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 jest 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. Założymy, że zmienna `valid` przyjmuje wartość logiczną `true` lub `false` (patrz ramka na stronie 95). 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, jednak jeśli będziesz dłujo 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 wybórów dokonuje skrypt, tym więcej jest możliwych skutków i wyższa „inteligencja” programu. Po przetworzeniu jednej instrukcji warunkowej często okazuje się, że trzeba podjąć następne decyzje.

Założymy, że chcesz rozbudować program wyświetlający propozycje rozrywki na piątkowy wieczór o obsłudze 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('Mozesz iść na kolację i do kina.');
    } else if (fridayCash >= 50) {
        alert('Mozesz iść na dobrą kolację.');
    } else if (fridayCash >= 25) {
        alert('Mozesz 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 wewnętrz 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 programu ma uruchomić 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. 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 43.

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>
```

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 100 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 48 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*.

Uwaga: A dlaczego dwa zestawy znaczników skryptu? 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 pojawiением 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 dwa 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 1, 5 lub 100). Więcej o tej funkcji dowiesz się na stronie 466, 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 liczbe). 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(luckyNumber) 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 kolejnym 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. Założymy, 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 powiniene 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.

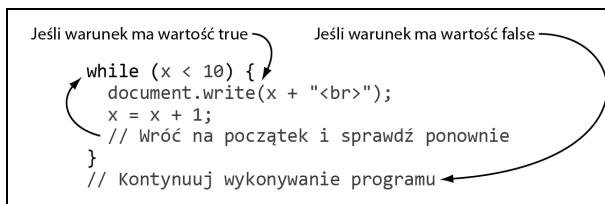
Pętle while

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 zamkającym nawiasem klamrowym, jeśli warunek jest spełniony.

Różnica polega na tym, że kiedy interpreter dojdzie do zamkającego 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 wyświetlana 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
' (
 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 68).

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 68, 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 móc 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. Założymy, 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 pisalem na stronie 72, 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 kas (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. Założmy, 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 81 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 i przecinek oraz zwiększa wartość zmiennej o 1 (instrukcja `counter++` działa tak samo jak `counter += 1`; zobacz wskazówkę na stronie 105). Po wykonaniu kodu w pętli skrypt ponownie sprawdza warunek. Pętla działa, dopóki warunek jest spełniony. Proces ten przedstawia rysunek 3.6.

```

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

```

Wartość zmiennej <code>counter</code> przed sprawdzeniem warunku	Warunek	Wejście w pętlę?	<code>days [counter]</code>	Wartość zmiennej <code>counter</code> po instrukcji <code>counter++</code>
0	$0 < 7$	Tak	<code>days [0]</code>	1
1	$1 < 7$	Tak	<code>days [1]</code>	2
2	$2 < 7$	Tak	<code>days [2]</code>	3
3	$3 < 7$	Tak	<code>days [3]</code>	4
4	$4 < 7$	Tak	<code>days [4]</code>	5
5	$5 < 7$	Tak	<code>days [5]</code>	6
6	$6 < 7$	Tak	<code>days [6]</code>	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”

Pętle for

Język JavaScript udostępnia także *pętlę for*, która jest bardziej zwięzła (i trochę bardziej skomplikowana). Pętle `for` służą zwykle do powtarzania danej operacji określona 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 106:

```

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 jej 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 + '
')`) to kod JavaScript uruchamiany w ramach pętli.

W tabeli 3.2 znajdziesz opis trzech elementów podawanych w nawiasach. 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
<code>for</code>	Rozpoczyna pętlę <code>for</code> .	
<code>var num = 1;</code>	Przypisuje zmiennej <code>num</code> wartość 1.	Jeden raz przy uruchamianiu pętli.
<code>num <= 100;</code>	Czy <code>num</code> ma wartość mniejszą od lub równą 100? Jeśli tak, należy powtórzyć pętlę. Jeśli nie, należy ją pominić i kontynuować wykonywanie skryptu.	Na początku instrukcji i przed każdym powtórzeniem pętli.
<code>num++;</code>	Dodaje 1 do zmiennej <code>num</code> . Działa tak samo jak <code>num = num + 1</code> oraz <code>num += 1</code> .	Po każdym powtórzeniu pętli.

Ponieważ pętle `for` umożliwiają łatwe powtarzanie serii operacji określonej 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 = days.length; j > 0; 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. Założmy, że chcesz wyświetlić elementy tablicy od końca. Można to zrobić w następujący sposób:

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

W tym fragmencie wartość zmiennej `j` to początkowo liczba elementów tablicy (4). Przy każdym powtórzeniu pętli należy sprawdzić, czy wartość tej zmiennej jest większa od 0. Jeśli tak, skrypt uruchamia kod między nawiasami klamrowymi, odejmując 1 od `j` (`j--`) i ponownie sprawdza warunek. Jedyna skomplikowana instrukcja to pobieranie elementów tablicy (`example[j-1]`). Ponieważ tablice są indeksowane od zera, indeks ostatniej wartości jest o 1 mniejszy od łącznej liczby elementów tablicy (patrz strona 75). Wartość zmiennej `j` to początkowo liczba wszystkich wartości tablicy, dlatego aby uzyskać dostęp do ostatniego elementu, trzeba odjąć od tej zmiennej 1.

Pętle do-while

Istnieje też inny, mniej popularny rodzaj pętli — `do-while`. Pętle tego typu działają prawie identycznie jak pętle `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 100). Skrypt ten prosi użytkownika o wpisanie liczby. Kod ma wbudowany system zabezpieczający, dlatego jeśli internauta nie poda odpowiednich danych, program powtarza prośbę. Jednak jeżeli 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 102:

1. Otwórz w edytorze tekstu plik `conditional.html` utworzony na stronie 102.

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

2. Znajdź kod między znacznikami `<script>` w sekcji nagłówkowej strony i usuń kod wyrożniony 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 102 (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”). Założmy, 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. Założmy, że chcesz utworzyć stronę z galerią fotografii, która ma zawierać 50 miniatur. Kiedy użytkownik kliknie jeden z małych obrazków, program ma przyciemić tło strony oraz wyświetlić tytuł i większą wersję wybranego zdjęcia (na stronie 234 dowiesz się, jak uzyskać ten efekt). 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 60). 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, zamkujący 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 471.

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 jednak można uruchomić funkcję? Programiści używają słowa *wywoływanie* na określenie procesu uruchomienia funkcji, kiedy ta ma wykonać swoje zadania. Wywoływanie funkcji polega na podaniu jej nazwy wraz z parą nawiasów. Na przykład aby 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 zauważyc. 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 `` kod wyróżniony pogrubieniem:

```
<p>Dziś jest <strong>  
<script>printToday();</script>  
</strong></p>
```

Zapisz stronę i wyświetl ją w przeglądarce. Na stronie pojawi się aktualna data. Jeśli chcesz wyświetlić ją także na dole strony, możesz powtórnie wywołać tę samą funkcję.

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 110, 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 pepperoni, 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. Na przykład jeśli 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 (wskaźówki na ten temat znajdziesz na stronie 60). Założmy, 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);
}
```

Teraz 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*.

Działanie nawet prostych funkcji może być nieco skomplikowane dla początkujących programistów. Poniżej znajdziesz szczegółowy opis wszystkich operacji, które ilustruje rysunek 3.7:

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!';`.

```
❶ function print(message) {  
    document.write(message); ❷  
}  
  
❸ print('Witaj, świecie!');
```

Rysunek 3.7. 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

4. Funkcja jest uruchamiana i wyświetla na stronie wartość zapisaną w zmiennej `message`.

Funkcja może przyjmować wiele parametrów. 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`, `argument1` trafia do zmiennej `parametr2` i tak dalej. Założymy, 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'. Te wartości 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 70) 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;
}
```

Założymy, 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 464. Tu ważny jest czwarty wiersz funkcji — instrukcja `return`, która zwraca wartość zapisaną w zmiennej `formattedTotal`.

Aby użyć zwrotionej 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.

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 3.8).

```
var message = 'Poza funkcją';
function warning(message) {
    alert(message);
}
warning('W funkcji');
alert(message);
```

Rysunek 3.8. 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 wewnętrz 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 zmenną `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 zmenną `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 wewnętrz niej przy użyciu słowa kluczowego `var`. Ilustruje to rysunek 3.9. Górný 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'`; Słowo `var` pozwala utworzyć zmienną lokalną.

Porównaj to z kodem z dolnej części rysunku 3.9. 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.

Zmienna lokalna w funkcji

```
var message = 'Poza funkcją';
function warning() {
    var message = 'W funkcji';
    alert(message); // 'Poza funkcją'
}
warning();
alert(message); // 'W funkcji'
```

Zmienna globalna w funkcji

```
var message = 'Poza funkcją';
function warning() {
    message = 'W funkcji';
    alert(message); // 'W funkcji'
}
warning();
alert(message); // 'W funkcji'
```

Rysunek 3.9. Przy przypisywaniu wartości do zmiennych w funkcji należy pamiętać o pewnym drobnym, ale istotnym szczególe. Jeśli chcesz, aby zmienność 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)

Zagadnienie zasięgu zmiennych jest skomplikowane, dlatego wcześniejszy opis może być nie w pełni zrozumiałym. Na razie zapamiętaj, że jeśli zmienne w skrypcie mają nieoczekiwanyą 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 określaniu zadań programu. Tworzony skrypt ma wykonywać kilka operacji. Są to:

- **Zadawanie pytań.** Program do obsługi quizu musi mieć funkcję zadawania pytań. Do tej pory poznaleś 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.** Najpierw trzeba ustalić, czy odpowiedź jest 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 polecenia `alert()` lub `document.write()`.

Opisany program można napisać na wiele sposobów. Niektórzy poczatkujący 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 ksiązyców ma Ziemia?','');
if (answer1 == 1) {
    alert('Prawidłowa odpowiedź!');
} else {
    alert('Błąd. Prawidłowa odpowiedź to 1.');
}
var answer2=prompt('Ile ksiązyców 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 kolejnym 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 74.

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 zagnieżdzonej*, 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 zagnieżdzone 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 zagnieżdzoną 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 106 napisałem, że doskonałym narzędziem do poruszania się po tablicach są pętle.

7. Kliknij kolumnę za sekwencją 1; (za tablicą `answers`) i wciśnij klawisz *Enter*, aby utworzyć nowy, pusty wiersz. Dodaj w nim następujący kod:

```
for (var i=0; i<questions.length; i++) {
```

Ten wiersz to pierwsza część pętli `for` (patrz strona 107). 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 jak 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. Ta 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 zamkającego nawiasu klamrowego.

Nowa funkcja przyjmuje jeden argument i zapisuje go w zmiennej `question`. Nie jest to tablica `questions[]` utworzona w kroku 6. W zmiennej `question` skrypt będzie zapisywać elementy z tablicy `questions[]`. Jak wiesz z kroku 8., każdy element tej tablicy to lista zawierająca dwie wartości: 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 3.10). Jeśli skrypt nie działa, wypróbuj techniki rozwiązywania problemów wymienione na stronie 48. Możesz też porównać skrypt z gotową, działającą wersją programu z pliku *compete_quiz.html*.



Rysunek 3.10. Efekt działania prostego programu do obsługi quizów. Kiedy nauczysz się manipulować stronami WWW (patrz strona 149), reagować na zdarzenia zachodzące na stronach (patrz strona 169) oraz obsługiwać formularze umieszczane na stronach WWW (patrz strona 271), zobaczyś, jak zmodyfikować ten program, by pytania pojawiały się bezpośrednio na stronie, a wynik dynamicznie się zmieniał po udzieleniu każdej odpowiedzi. Wkrótce dowieś się, jak zastąpić nieporęczne polecenie `prompt()`

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 kolejnym rozdziale poznasz bibliotekę jQuery, dowieś się, czym ona jest i jak jej używać, a co najważniejsze, jak się przy tym doskonale bawić i realizować wiele zadań programistycznych.

II

CZĘŚĆ

Wprowadzenie do biblioteki jQuery

- Rozdział 4. „Wprowadzenie do jQuery”
- Rozdział 5. „Akcja i reakcja — ożywianie stron za pomocą zdarzeń”
- Rozdział 6. „Animacje i efekty”

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 drapaleś się w zadumie po głowie lub sięgałeś po pastylkę aspiryny (na przykład pętle `for` opisane na stronie 107). 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 stu 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. Często zdarza się, 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 używamy popularnej biblioteki jQuery (<http://www.jquery.com>). Dostępne są także inne biblioteki JavaScript (patrz ramka na następnej stronie), jednak jQuery ma nad nimi przewagę i to z kilku powodów. Oto one.

- **Jest stosunkowo niewielka.** Skompresowana wersja biblioteki zajmuje jedynie około 90 kB. (Jeśli serwer dodatkowo korzysta z kompresji „gzip”, rzeczywistą wielkość przesyłanego pliku biblioteki można ograniczyć do około 30 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 Digg, Dell, Onion oraz NBC. Jest nawet używana w niektórych rozwiązańach i aplikacjach firmy Google oraz wbudowana na stałe w najpopularniejszą platformę blogerską — WordPress. Popularność jQuery jest najlepszym świadectwem jej jakości.
- **Jest bezpłatna.** A tego nie można przebić!
- **Ma ogólną 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.

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://developer.yahoo.com/yui/>) 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 postużyć do rozwiązania niemal każdego problemu.
- ◆ **Mootools** (<http://mootools.net/>) jest kolejną biblioteką służącą głównie do tworzenia płynnych animacji i innych efektów wizualnych. Dysponuje doskonałą dokumentacją i wspaniałą witryną.
- ◆ **Prototype** (<http://www.prototypejs.org/>) to jedna z pierwszych bibliotek JavaScript, jakie się pojawiły. Bardzo często jest używana wraz z dodatkową biblioteką *scriptaculous* (<http://script.aculo.us/>), pozwalającą tworzyć animacje i inne efekty wizualne.

- **Wtyczki, wtyczki i jeszcze raz wtyczki.** Biblioteka jQuery pozwala programistom tworzyć wtyczki — dodatkowe programy napisane w JavaScriptie i współpracujące z biblioteką w celu wykonywania określonych zadań, tworzenia efektów wizualnych i zapewniania nowych możliwości, z których w niezwykle prosty 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 46) 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 40), także i plik jQuery należy dołączyć do swojej strony. Jednak, ze względu na ogólną popularność jQuery, można to zrobić na kilka sposobów: można pobrać wersję biblioteki udostępnianą na serwerach firm Google, Microsoft lub na serwerze [jQuery.com](http://jquery.com) bądź też można skopiować plik biblioteki na własny komputer i umieścić go na witrynie.

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 takich sytuacjach konieczne będzie pobranie biblioteki z witryny [jQuery.com](http://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 wyłączeno 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 1.6.3 biblioteki pobieranej z serwera CDN firmy Microsoft, w sekcji nagłówka naszej strony (tuż przed zamkającym znacznikiem `</head>`) musielibyśmy umieścić znacznik o następującej postaci:

```
<script src="http://ajax.aspnetcdn.com/ajax/jquery/jquery-1.6.3.min.js">
</script>
```

Gdybyśmy chcieli skorzystać z serwera CDN jQuery, musielibyśmy użyć następującego kodu:

```
<script src="http://code.jquery.com/jquery-1.6.3.min.js"></script>
```

Poniższy kod pozwala użyć serwera CDN firmy Google:

```
<script src="http://ajax.googleapis.com/ajax/libs/jquery/1.6.3/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, z której z nich skorzystać, wybierz właśnie tę.

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; ponieważ jednak biblioteka ta jest regularnie aktualizowana, jej najnowszą wersję zawsze można znaleźć na stronie http://docs.jquery.com/Downloading_jQuery, w sekcji *Current Release* (aktualna wersja).

Aby pobrać najnowszą wersję biblioteki jQuery, wykonaj kolejno następujące kroki.

- Wejdź na stronę http://docs.jquery.com/Downloading_jQuery.**

Strona ta zawiera informacje na temat kodu, listę CDN, o których wspomniano już wcześniej, oraz listę starszych wersji biblioteki. Najnowszej wersji szukaj pod nagłówkiem *Current Release*.

- Kliknij odnośnik *Current Release* w górnej części strony bądź przewiń jej zawartość tak, by pojawił się nagłówek *Current Release*.**

Na witrynie umożliwiającej pobieranie biblioteki jQuery jej pliki są udostępniane w dwóch wersjach — zminimalizowanej (ang. *minified*) oraz nieskompresowanej (ang. *uncompressed*). Plik wersji nieskompresowanej jest bardzo duży (ma ponad 200 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 85), które ułatwiają zrozumienie przeznaczenia poszczególnych fragmentów kodu. (Aby zrozumieć te komentarze, trzeba naprawdę dobrze znać język JavaScript).

Na swojej witrynie należy jednak używać wersji zminimalizowanej. Pliki zminimalizowane zajmują znacznie mniej miejsca niż zwyczajne pliki z kodem JavaScript — nie zawierają żadnych komentarzy oraz niepotrzebnych znaków (takich jak znaki odstępu, nowego wiersza, tabulacji i tak dalej). Pliki te znacznie trudniej przeglądać i analizować, lecz przeglądarki mogą je 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-1.6.3.min.js* oznacza, że plik zawiera zminimalizowaną wersję biblioteki jQuery.

- Kliknij odnośnik *Minified* prawym przyciskiem myszy (lub środkowym w przypadku komputerów Mac) i z wyświetlnego 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*.

- 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*.

CZĘSTO ZADAWANE PYTANIA

Wersje biblioteki jQuery

Jak widzę, w tej książce używana jest biblioteka jQuery w wersji 1.6.3, natomiast najnowszą wersję dostępną na witrynie jQuery jest 1.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 przeglądarki, 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.

Ilość różnic pomiędzy dwiema wersjami biblioteki można zazwyczaj określić na podstawie porównania numerów ich wersji. Pierwsza cyfra numeru reprezentuje tak zwaną

bardzo ważną wersję. Aktualnie jest to wersja 1.; mogliśmy więc znaleźć bibliotekę w wersji 1., 1.2, 1.3, 1.4 i tak dalej. Wersja 2. (która pewne nie pojawi się zbyt szybko) na pewno będzie udostępniać jakieś bardzo ważne, nowe możliwości. Po tej pierwszej cyfrze zapisywana jest kropka, a po niej kolejny numer, na przykład 6 — jak w jQuery 1.6. 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 1.6.3, zazwyczaj odnosi się do kolejnej grupy poprawek wprowadzanych do biblioteki w wersji 1.6. A zatem, jeśli używamy biblioteki w wersji 1.6.3, a pojawiła się wersja 1.6.8, warto z niej skorzystać, gdyż zazwyczaj będzie zawierała poprawki do błędów odnalezionych w kodzie biblioteki 1.6.3.

Aby zobaczyć, jakie zmiany wprowadzono w konkretnej wersji biblioteki, wystarczy wyświetlić sekcję *Current Release* na stronie zawierającej kod biblioteki do pobrania: http://docs.jquery.com/Downloading_jQuery#Current_Release. Można będzie tam zobaczyć odnośnik o nazwie *Release notes* (uwagi do wydania). Można kliknąć ten odnośnik, aby przejść na stronę prezentującą informacje o zmianach wprowadzonych w danej wersji biblioteki. 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 pominać 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).

Dodawanie jQuery do strony

Jeśli używamy jednej z wersji jQuery udostępnianej przez sieci dystrybucji (patrz strona 130), możemy ją dodać do naszej strony, korzystając z jednego z fragmentów kodu zamieszczonych na stronie 130. 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/1.6.3/jquery.min.js"></script>
```

Wskazówka: Podczas korzystania z sieci dystrybucji treści (CDN) firmy Google można pominąć wszystkie fragmenty numeru wersji. Jeśli zamiast numeru 1.6.1 podamy o odnosić numer 1.6 (`<script src="http://ajax.googleapis.com/ajax/libs/jquery/1.6/jquery.min.js"></script>`), Google udostępnii najnowszą wersję biblioteki z rodziny 1.6 — na przykład wersję 1.6.3. Jeśli później pojawi się aktualizacja o numerze 1.6.9, serwer Google będzie udostępniał właśnie ją. Technika ta jest całkiem sprytna, gdyż (zgodnie z informacjami podanymi na stronie 132) zmiany reprezentowane przez ostatni numer zazwyczaj sprowadzają się do poprawek w kodzie, które mogą usprawnić działanie naszej witryny.

Jeśli pobrałeś plik biblioteki i umieściłeś go na własnym komputerze, będziesz go musiał dodać do strony, na której chcesz używać biblioteki. Plik jQuery jest zwykłym zewnętrznym plikiem JavaScript, a zatem dodaje się go do strony dokładnie tak samo jak inne pliki tego typu, w sposób opisany na stronie 40. Założymy 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-1.6.3.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 `<scripts>` zawierających nieco bardziej wymagający kod JavaScript:

```
<script src="js/jquery-1.6.3.min.js"></script>
<script>
$(document).ready(function() {
    // to jest miejsce na nasz kod
});
</script>
```

Wewnątrz drugiej pary znaczników `<scripts>` 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 odniesienia 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 powinieneś stosować funkcję `ready()` i umieszczać swój kod dokładnie pomiędzy `$(document).ready(`
`↳(function(){` oraz zamkającą 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 `<scripts>` odwołującymi się do jQuery nie należy umieszczać innych znaczników `<scripts>`.
- Własny kod JavaScript należy umieszczać *za wszystkimi arkuszami stylów* (zówno 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 `<scripts>` wewnątrz sekcji nagłówka strony, poniżej jakiegoś innego zawartości, lecz — oczywiście — przed zamkającym znacznikiem `</head>`.
- Do własnego kodu JavaScript warto dodawać komentarze — na przykład komentarz `// koniec funkcji ready`, umieszczony po zamkająccej 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 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 zamkajającego nawiasu klamrowego, okrągłego oraz średnika. Umieszczając za nimi komentarze, można sobie znacznie ułatwić określenie, do jakiego fragmentu kodu odnosi się dana sekwencja.

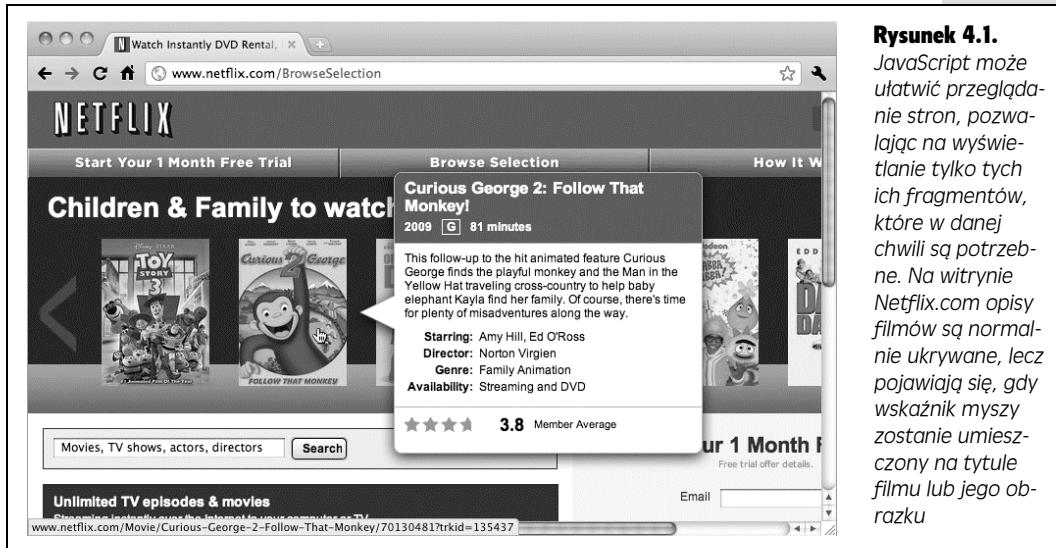
Wskazówka: jQuery udostępnia skrócony zapis wywołania funkcji `$(document).ready(function(){}):`

```
$ (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 ją 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 4.1). W obu tych przypadkach program napisany w języku JavaScript zmienia kod HTML pobrany początkowo przez przeglądarkę.



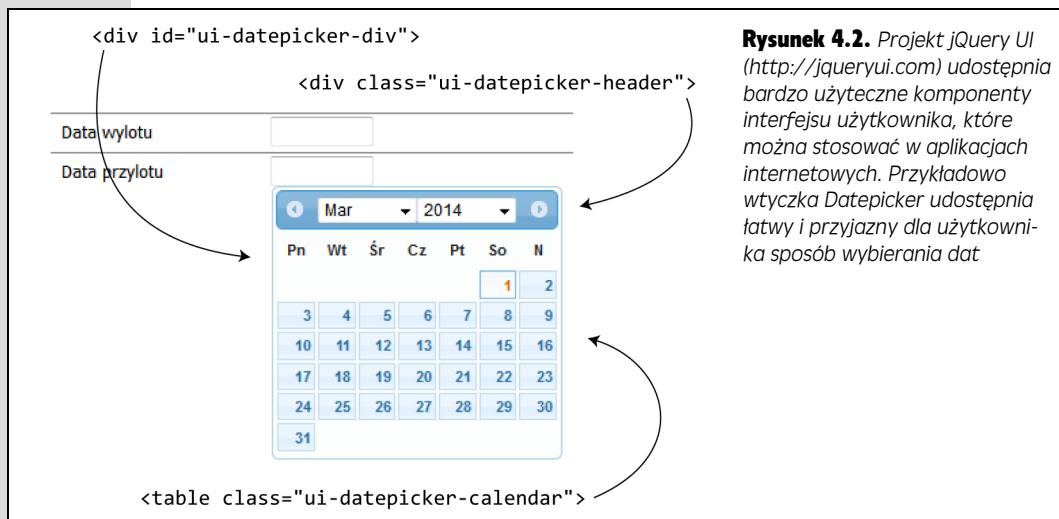
Rysunek 4.1.

JavaScript może ułatwić przeglądanie stron, pozwalając na wyświetlanie tylko tych ich fragmentów, które w danej chwili są potrzebne. Na witrynie Netflix.com opisy filmów są normalnie ukrywane, lecz pojawiają się, gdy wskaźnik myszy zostanie umieszczony na tytule filmu lub jego obrazku

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 JavaScript 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 języku 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 4.2). 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 znanego kodu 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



Rysunek 4.2. 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

kolory, typografię oraz formatowanie kalendarza. Innymi słowy, korzystając z języka 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.

A zatem 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 Netflix.com, przedstawionej na rysunku 4.1, kod JavaScript wyświetla okienko z informacjami po umieszczeniu wskaźnika myszy na tytule filmu, jednak najfajniejszą część całego tego rozwiązania (czyli utworzenie tego okienka) sprowadza się do napisania odpowiedniego kodu HTML i stylów CSS..., a to przecież już umiemy 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 rozwiążaniem jest wyświetlanie komunikatu o błędzie, takiego jak: „Proszę podać prawidłowy adres e-mail”. W takim przypadku dodajemy nową zawartość strony, umiejscowioną względem konkretnego pola formularza.
- **Usunięcie elementu.** Na witrynie *Netflix.com*, przedstawionej na rysunku 4.1, okienko informacyjne znika po usunięciu wskaźnika myszy z tytułu filmu. 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 rozwiążaniem 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 wybór 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 określaniu 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).

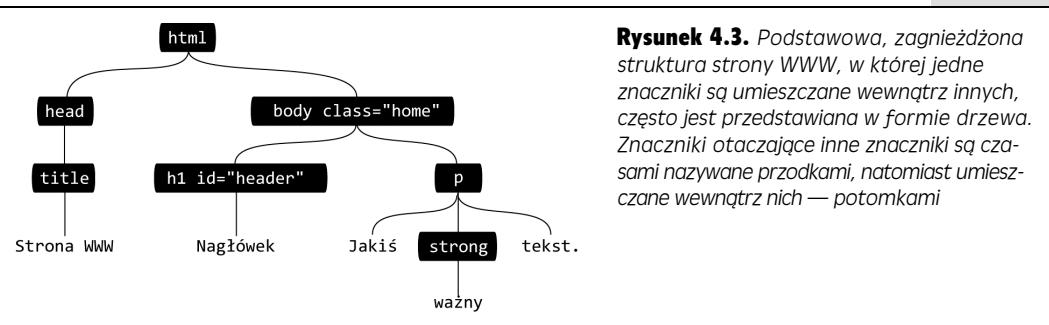
DOM dostarcza informacje niezbędne językowi JavaScript do operowania na elementach strony WWW. Oferuje także narzędzia pozwalające na poruszanie się po zawartości strony, modyfikowanie istniejących elementów HTML oraz dodawanie nowych. Sam DOM nie jest (w zasadzie) związany z językiem JavaScript — stanowi odrębny standard opracowany przez W3C (ang. *World Wide Web Consortium*), który większość producentów przeglądarek zaakceptowała i wykorzystuje w swoich programach. DOM pozwala skryptom pisany w języku JavaScript komunikować się 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 4.3). „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 4.3 wewnątrz znacznika `<h1>`) oraz **atrybuty** przypisywane poszczególnym znacznikom (takie jak atrybut `class` w znaczniku `<body>` oraz `id` w znaczniku `<h1>` na przykładzie z rysunku 4.3). W rzeczywistości DOM traktuje te wszystkie znaczniki (nazywane także elementami), atrybuty i teksty jako odrębne jednostki, nazywane **węzłami** (ang. *node*).



Rysunek 4.3. Podstawowa, zagnieżdzona 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 umieszczone wewnątrz nich — potomkami

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');
```

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 reguły CSS.

WARSZTATY UŁATWIENI

Problem z DOM

Niestety, wszystkie główne przeglądarki już tradycyjnie interpretowały DOM w różny sposób. Wszystkie wcześniejszej wersje Internet Explorera, z wyjątkiem Internet Explorera 9, obsługują zdarzenia inaczej niż inne przeglądarki; dokładnie ten sam kod HTML może spowodować wygenerowanie większej liczby węzłów tekstowych w Firefoxie i Safari niż w Internet Explorzerze, a dodatkowo Internet Explorer nie zawsze pobiera atrybuty znaczników HTML w taki sam sposób jak Firefox, Safari bądź Opera. Co więcej, różne przeglądarki w odmienny sposób traktują białe znaki (takie jak znaki odstępu oraz tabulacji) umieszczone w kodzie HTML — czasami są one traktowane jak dodatkowe węzły tekstowe (na przykład w przeglądarkach Safari lub Firefox), a czasami — zupełnie ignorowane (w Internet Explorzerze). A to jedynie kilka spośród wielu różnic w sposobie traktowania standardu DOM przez najpopularniejsze przeglądarki WWW.

Przezwyciężenie problemów związanych z zapewnieniem prawidłowego działania kodu JavaScript w wielu różnych przeglądarkach jest tak obszernym tematem, że tylko

temu jednemu zagadnienu można by poświęcić bardzo dużą (i bardzo nudną) książkę. Wiele książek dotyczących JavaScriptu poświęca wiele miejsca i uwagi na pokazanie kodu, który zmusza przeglądarki do odpowiedniego działania. Jednak życie jest zbyt krótkie — lepiej poświęcać czas na tworzenie interaktywnych interfejsów użytkownika i dodawanie wizualnych wodotrysków do swoich witryn niż na zwracanie sobie głowy zapewnieniem identycznego sposobu działania skryptów w Internet Explorzerze, Firefoxie, Safari oraz Operze.

Co więcej, tradycyjne metody DOM służące do pobierania elementów stron nie są zbyt intuicyjne dla projektantów, zwłaszcza że dysponują oni innym, doskonalszym narzędziem służącym do tego samego celu — selektorami CSS. Na szczęście, jQuery rozwiązuje te wszystkie problemy i udostępnia łatwe, działające we wszystkich przeglądarkach metody pozwalające zarówno na pobieranie, jak i na wykonywanie różnych operacji na elementach stron WWW.

Pobieranie elementów stron na sposób jQuery

Zgodnie z tym, czego się dowiedziałeś na poprzedniej stronie, przeglądarki WWW udostępniają dwie podstawowe metody służące do pobierania elementów stron — `document.getElementById()` oraz `document.getElementsByTagName()`. Niestety, nie zapewniają one dostatecznej kontroli, która pozwoliłaby na pobieranie elementów na podstawie bardziej wyszukanych kryteriów. Gdybyśmy na przykład chcieli pobrać wszystkie elementy `<a>`, którym przypisano klasę `navButton`, najpierw musielibyśmy pobrać wszystkie znaczniki odnośników, a następnie wybrać z nich te, które mają odpowiednią klasę.

Tak się składa, że 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 wielu 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. Założymy przykładowo, że chceliśmy zmienić kod HTML umieszczony wewnętrz 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 149. Jednak na początek musisz dowiedzieć się czegoś więcej na temat pobierania tych elementów przy użyciu biblioteki jQuery.

Uwaga: Najnowsze wersje przeglądarek udostępniają już metody pozwalające na pobieranie elementów stron na podstawie nazwy klasy oraz na podstawie selektorów CSS. Ponieważ jednak dostępność i obsługa tych metod nie jest jednakowa we wszystkich aktualnie używanych przeglądarkach, stosowanie biblioteki jQuery zapewnia, że pobieranie elementów będzie działało zawsze, nawet w archaicznym Internet Explorerze 6.

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 4.4).

Uwaga: Więcej informacji na temat przykładów z książki można znaleźć na stronie 43.

Selektory identyfikatorów

Każdy element strony, który ma określony identyfikator (wartość atrybutu *id*), można pobrać przy użyciu selektora identyfikatora. Założmy, że na stronie WWW znajduje się kod HTML w następującej postaci:

```
<p id="message">Komunikat specjalny</p>
```

Aby pobrać taki element przy użyciu starego sposobu korzystającego tylko z metod dostępnych w DOM, należałoby użyć następującego kodu:

```
var messagePara = document.getElementById("message");
```

Analogiczne wywołanie z zastosowaniem biblioteki jQuery ma następującą postać:

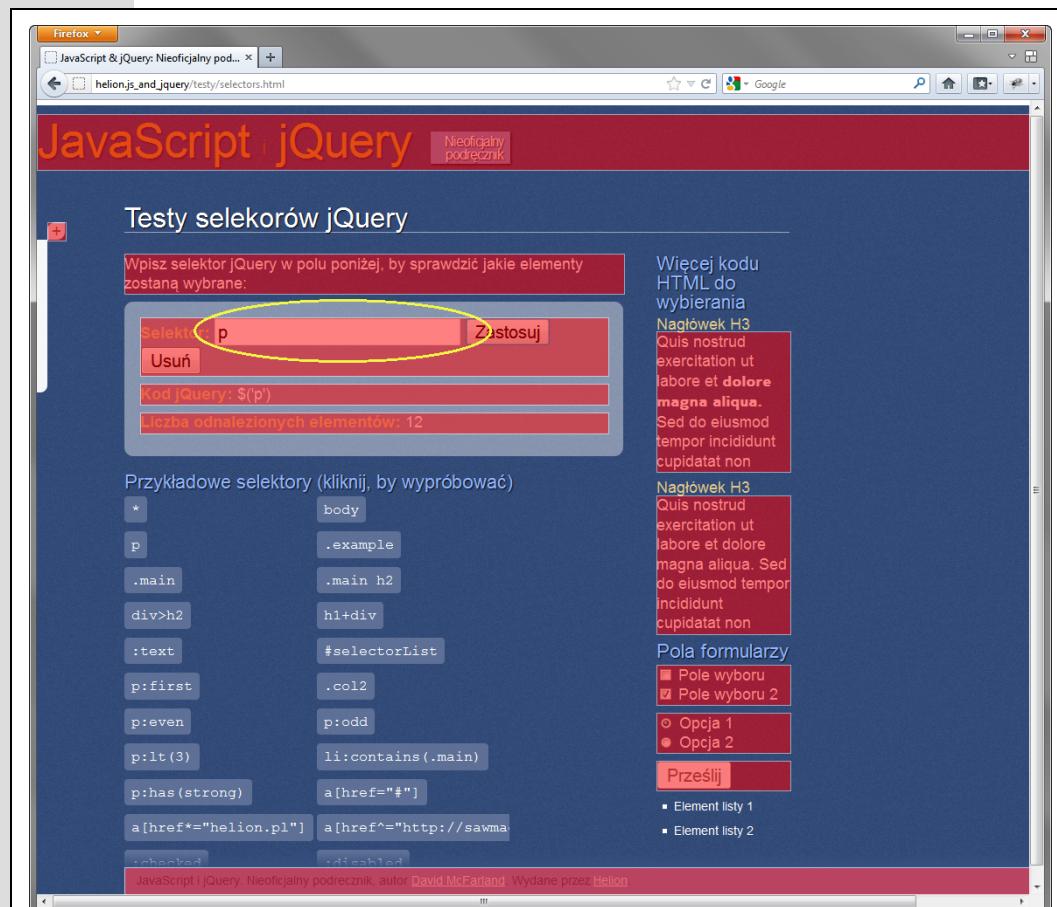
```
var messagePara = $('#message');
```

W odróżnieniu od sposobu wykorzystującego metodę DOM, w przypadku selektora jQuery nie wystarczy podać 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 — `getElementsByName()`. 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.getElementsByName('a');
```



Rysunek 4.4. 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 `p`, w związku z czym zostały wyróżnione wszystkie znaczniki `<p>` (a jest ich w sumie 12)

Po zastosowaniu jQuery analogiczne wywołanie ma następującą postać:

```
var linksList = $('a');
```

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. Założmy, 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 za-

programowania 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 rozwiązyania 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 (swoją drogą, jak to zrobić, dowiesz się na stronie 263). 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 198., 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 je opanowywać: po przeczytaniu kilku kolejnych rozdziałów książki i lepszym poznaniu sposobów działania biblioteki jQuery oraz sposobów 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 wewnętrz innego. Przykładowo założymy, że utworzyliśmy wypunktowaną listę odnośników i znacznikowi `` 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 wewnętrz listy, możemy to zrobić przy użyciu selektora w następującej postaci:

```
$('#navBar a')
```

WIEDZA W PIŁUŁCE

Zrozumieć CSS

Kaskadowe arkusze stylów są obszernym zagadnieniem pojawiającym się zawsze w ramach dyskusji o języku JavaScript. Aby korzyść z czytania tej książki możliwe największa, konieczna jest przynajmniej pobiczna 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 ułatwia 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 korzystanie z CSS, możesz skorzystać z wielu dostępnych zasobów.

Podstawowe informacje na temat CSS można znaleźć na stronie *HTML Dog CSS Tutorials* (<http://www.htmldog.com/guides/>).

Dostępne są na niej samouczki na poziomie podstawowym, średnim oraz zaawansowanym.

Można także kupić egzemplarz książki *CSS. Nieoficjalny podręcznik*, 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 zatrzymać się w celu odświeżenia informacji na temat różnych, dostępnych selektorów:

- ◆ <http://css.maxdesign.com.au/selectutorial/>,
- ◆ http://www.456bereastreet.com/archive/200601/css_3_selectors_explained/.

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 4.3 znaczniki `<h1>` oraz `<p>` są dziećmi znacznika `<body>`, natomiast znacznik `` nim nie jest (gdyż jest umieszczony wewnętrznie znacznika `<p>`). Selektor dziecka tworzy się, zapisując najpierw element rodzica, następnie znak mniejszości (`>`) 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. Założymy na przykład, że dysponujemy ukrytym panelem, wyświetlonym 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 selektoram (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 ``, w których użyto atrybutu `alt`, a nawet te spośród nich, w których w atrybutie `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 wyświetlane w nowym oknie przeglądarki.

Selektor atrybutu jest umieszczany za nazwą elementu, którego atrybuty chcemy sprawdzać. Aby na przykład znaleźć wszystkie znaczniki ``, w których został podany atrybut `alt`, można użyć następującego wyrażenia:

```
$('.img[alt]')
```

Istnieje kilka różnych rodzajów selektorów atrybutów:

- `[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 — `` — 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 273).

- `[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:

```
 $('input[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:"'])
```

- `[atribut$="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 odzyskać 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"]')
```

- `[atribut*= "wartość"]` pozwala pobrać wszystkie elementy, których określony atrybut zawiera podany ciąg znaków. Dzięki temu można odzyskać 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"]')
```

Powyzszy 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. Pozwalały one na pobieranie takich elementów jak pola tekstowe, pola haseł czy też przyciski opcji. Więcej informacji na ich temat można znaleźć na stronie 273.

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 odzyskać wszystkie parzyste wiersze tabeli, należałoby użyć następującego selektora jQuery:

```
$('tr:even')
```

Powyzsze wywołanie pobiera każdy parzysty znacznik `<tr>`. Aby dodatkowo zwiększyć 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 72). Każdy element pobrany przez jQuery ma swój indeks,

a trzeba pamiętać, że numeracja indeksów tablic w języku JavaScript zaczyna się od zera (patrz strona 75). 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. Założymy 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 chcemyignorować. 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. Założymy, że interesują nas wszystkie znaczniki , jednak wyłącznie wtedy, gdy wewnętrz nich umieszczony jest znacznik <a>. W tym celu możemy użyć następującego wywołania:

```
$('li:has(a)');
```

Takie rozwiązań różni się znaczco od selektora elementów potomnych, gdyż pozwala pobrać nie elementy <a>, lecz elementy zawierające wewnętrz 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 198), elementy o wysokości lub szerokości wynoszącej zero oraz ukryte pola formularzy. (Filtr ten nie zwraca elementów, których właściwość visibility CSS ma wartość invisible). Przykładowo założymy, 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 198).

- Filtr `:visible` jest przeciwnieństwem filtra `:hidden`. Zwraca on wszystkie widoczne elementy strony.

Zrozumienie kolekcji jQuery

Wybierając elementy strony przy użyciu obiektu jQuery — na przykład: `$('nav > Bar a')` — nie otrzymujemy w efekcie tradycyjnej listy węzłów DOM, takich jak zwracane przez metody `getElementById()` lub `getElementsByName()`. Zamiast tego zwracana jest specjalna, charakterystyczna dla biblioteki jQuery kolekcja elementów. Do elementów tych nie możesz wykorzystać tradycyjnych metod DOM, a zatem, jeśli już poznajeś 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 niezbędny do jego tworzenia. Jednym z jej podstawowych celów jest zapewnienie możliwości wykonywania wielu złożonych operacji przy użyciu możliwie krótkiego kodu. By tak się działało, jQuery używa dwóch, niezwykłych zasad.

Automatyczne pętle

Podczas korzystania ze standardowych metod DOM dysponujemy zazwyczaj grupą elementów strony, a następnie musimy utworzyć pętlę (patrz strona 104), 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 wykoną 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. Rozwiążanie 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. Założymy, 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 świadomie wykonanie liczby operacji. Założymy, 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 łańcuchach jest dosyć niezwykłą i charakterystyczną cechą biblioteki jQuery, innymi słowy, w takiej sekwencji można używać wyłącznie wbudowanych funkcji biblioteki (czyli nie można skorzystać ani z funkcji pisanych samodzielnie, ani z wbudowanych funkcji języka JavaScript).

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, założymy, że dysponujemy stroną zawierającą następujący fragment kodu HTML:

```
<div id="container">
  <div id="errors">
    <h2>Błędy:</h2>
  </div>
</div>
```

- 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>');
```

Powыższe 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.

- Funkcja `.text()` działa podobnie jak funkcja `.html()`, jednak nie akceptuje znaczników HTML. Jest przydatna, gdy chcemy zmienić tekst umieszczony wewnątrz znacznika. Przykładowo we fragmencie kodu umieszczonego na początku tego podrozdziału znajduje się znacznik `<h2>` zawierający słowo "Błędy:". Założymy, 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 `<p>` 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. Założmy, że wybraliśmy wcześniej znacznik `<div>`, lecz zamiast zmieniać jego zawartość, chcemy na jej końcu, tuż przed zamkającym znacznikiem `</div>`, coś dodać. Funkcja `.append()` jest doskonałym sposobem dodawania nowych punktów na końcach list uporządkowanych (``) lub wypunktowanych (``). W ramach przykładu założmy, że na stronie zawierającej fragment kodu HTML przedstawiony na początku tego podrozdziału wykonaliśmy poniższe wywołanie:

```
$( '#errors' ).append( '<p>W formularzu znalezione 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 znalezione 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 założmy, że na tej samej stronie, co wcześniej, chcemy wykonać poniższą instrukcję:

```
$( '#errors' ).prepend( '<p>W formularzu znalezione cztery błędy.</p>' );
```

W efekcie kod strony przyjmie następującą postać:

```
<div id="container">
  <div id="errors">
    <p>W formularzu znalezione 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 zamkającym, 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. Założmy, że kod HTML pola formularza ma następującą postać:

```
<input type="text" name="userName" id="userName">
```

A teraz założmy, 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 zaprzatać sobie głowy, jak można sprawdzić, czy zawartość pola formularza jest prawidłowa — te informacje można znaleźć na stronie 293), 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.

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()`. Założymy, 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 `` należące do klasy `error`. Wystarczy skorzystać z następującego wywołania:

```
$('.span.error').remove();
```

Dodatkowo mamy także możliwość całkowitego zastąpienia wybranych elementów zupełnie nową zawartością. Założymy, że dysponujemy stroną ze zdjęciami produktów oferowanych przez naszą firmę. Kiedy użytkownik kliknie zdjęcie produktu, zostaje on dodany do koszyka. Założymy też, że chcemy, by po kliknięciu obrazka znacznik `` został zastąpiony jakimś tekstem (takim jak: "Dodano do koszyka."). W rozdziale 10. dowiesz się, jak sprawić, by konkretne elementy reagowały na zdarzenia (takie jak kliknięcie obrazka), jak na razie jednak przyjmijmy, że na stronie istnieje znacznik `` o identyfikatorze `product101`, który chcemy zamienić na tekst. Oto sposób, w jaki możemy to zrobić za pomocą biblioteki jQuery:

```
$('#product101').replace('<p>Dodano do koszyka.</p>');
```

Powyższe wywołanie usuwa ze strony znacznik 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 163.

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 natręca 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 zamierzanej postaci. Gdybyśmy na przykład mogli oglądać kod HTML strony po programowym dodaniu do niego dziesięciu komunikatów o błędach odnalezionych w formularzu lub po utworzeniu rozbudowanego okna dialogowego z formularzem, znacznie ułatwiliby to sprawdzenie, czy generowany kod HTML ma taką postać, jaką planowaliśmy.

Na szczęście, większość nowoczesnych przeglądarek udostępnia 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 wyświetlonej strony. Różne karty pozwalają na dostęp do kodów JavaScript, HTML, CSS oraz innych, przydatnych zasobów. Konkretnie nazwy poszczególnych kart oraz metody wyświetlania tych narzędzi różnią się w poszczególnych przeglądarkach.

- ◆ W przeglądarce Firefox trzeba zainstalować dodatek Firebug (informacje na ten temat można znaleźć na stronie 496). Aby użyć narzędzia, otwórz stronę z kodem JavaScript, którego efekty działania chcesz przeanalizować, a następnie wyświetl dodatek Firebug (Narzędzia/Firebug/Pokaż Firebug, ewentualnie Widok/Firebug lub naciśnij klawisz F12). Po kliknięciu karty HTML w panelu Firebuga zostanie wyświetlone kompletne drzewo DOM (w tym

także kod HTML wygenerowany przez JavaScript). Alternatywnym rozwiązaniem jest skorzystanie z dodatku *Web Developer Toolbar* (<https://addons.mozilla.org/en-US/firefox/addon/web-developer/>), który pozwala wyświetlić zarówno zwyczajny, jak i wygenerowany kod HTML.

- ◆ W przeglądarce Internet Explorer 9 trzeba naciąć klawisz F12, co spowoduje wyświetlenie panelu *Narzędzi Deweloperskich*. Na nim wystarczy kliknąć kartę *HTML*, by wyświetlić kod HTML strony. W przeglądarce IE 9 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 pokazany wyświetlony kod HTML, zawierający także wszelkie zmiany wprowadzone przez kod JavaScript.
- ◆ W przeglądarce Chrome należy kliknąć przycisk *Ustawienia Google Chrom*, a następnie wybrać opcję *Narzędzia/Narzędzia dla programistów*. W wyświetlonym u dołu okna panelu trzeba kliknąć przycisk *Elements*.
- ◆ 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 wyświetlonym u dołu okna panelu kliknąć przycisk *Elements*.
- ◆ W przeglądarce Opera należy wybrać z menu głównego opcję *Narzędzia/Zaawansowane/Opera Dragonfly*. (Dragonfly to wbudowany w tę przeglądarkę zestaw narzędzi dla programistów). W panelu wyświetlonym u dołu okna przeglądarki trzeba kliknąć kartę *Dokumenty*.

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.

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 dokonaniu selekcji elementów, a przekazywany do niej ciąg 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 zmieni 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.

Ta 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 założymy, że przycisk, który użytkownik kliką, 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. Założymy 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 zwarcane 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 nie przyjmuje skróty właściwości CSS, takich jak `font`, `margin`, `padding` bądź `border`. Zamiast nich należy używać precyzyjnych właściwości, takich jak `font-face`, `margin-top`, `padding-bottom` bądź `border-bottom-width` i za ich pomocą określić właściwości, które łatwiej można podać z wykorzystaniem właściwości skrótowej. 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 ustawać 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%, należało by 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:

```
$('.highlight').css('border', '1px solid black');
```

Często bardzo użyteczna może być możliwość zmieniania właściwości CSS na podstawie ich bieżącej wartości. Założymy, że chcemy umieścić na stronie przycisk *Powiększ czcionkę*, który użytkownik mógłby kliknąć, by 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 jestłańcuchem znaków i ma postać '`16px`'. Ponieważ chcemy podwoić tę wartość — pomnożyć ją przez dwa — zatem musimy zamienićłańcuch znaków na liczbę, pozbywając się umieszczonejona końcu liter '`px`'. Właśnie tę operację wykonuje drugi wiersz kodu, w którym używamy metody `parseInt()`języka JavaScript, opisanej bardziej szczegółowo na stronie 466. Metoda ta usuwa wszystkie znaki zapisane po liczbie. A zatem, po wykonaniu tego drugiego wiersza kodu w zmiennej `baseFont` będzie zapisana wartość liczbową, 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:

```
$('#highlightedDiv').css('background-color', '#FF0000');
$('#highlightedDiv').css('border', '2px solid #FE0037');
```

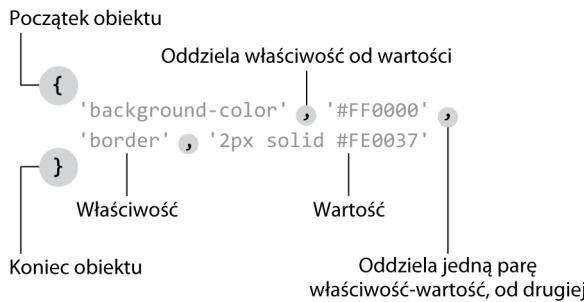
Innym rozwiązaniem jest przekazanie w wywołaniu metody `css()` tak zwanego **literatu obiektowego**. Można go sobie wyobrazić jako listę par, na które składa się nazwa właściwości oraz odpowiadająca jej wartość. Tuż za nazwą właściwości należy zapisać dwukropkę (`:`), a po nim wartość; poszczególne pary 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 literatu zisanego 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 4.5.



Rysunek 4.5. Literaty obiektowe języka JavaScript zapewniają możliwość tworzenia list właściwości i odpowiadających im wartości. JavaScript traktuje każdy literał obiektowy jako jeden blok informacji — podobne jak tablicę stanowiącą listę wartości. Literaty obiektowe, takie jak ten, będziemy często stosować podczas przekazywania opcji do wtyczek jQuery

Aby użyć literatu 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 literatu obiektowego. Z kolei trzy znaki widoczne w ostatnim wierszu przykładu należy interpretować w następujący sposób: zamkający 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 149). 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 `highlightedDiv`, 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 wyświetlonego w znaczniku ``, wystarczy przekazać w jej wywołaniu łańcuch znaków `'src'` (czyli pobrać wartość atrybutu `src` znacznika ``):

```
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 `` umieszczonego wewnętrz 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 ``:

```
$('#banner img').attr('src', 'images/newImage.png');
```

Jeśli trzeba całkowicie usunąć atrybut ze znacznika, można skorzystać z funkcji `removeAttr()`. I tak poniższe wywołanie usuwa z znacznika `<body>` atrybut `bgColor`:

```
$( 'body' ).removeAttr( 'bgColor' );
```

Wykonanie akcji na każdym elemencie kolekcji

Zgodnie z informacjami podanymi na stronie 148, 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 `` 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 założymy, 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 zatytuować „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

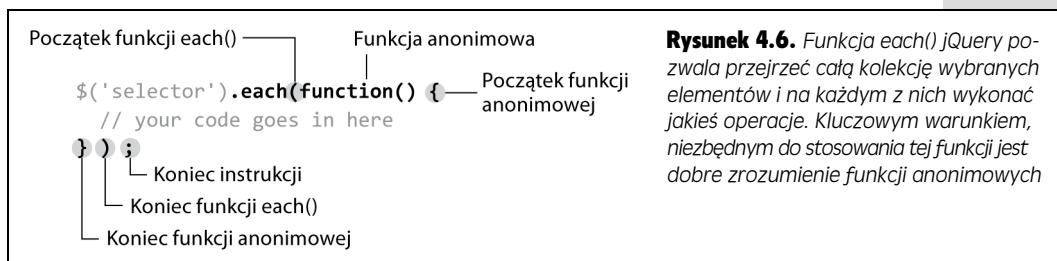
Aby skorzystać z funkcji `each()`, musimy przekazać do niej specjalny rodzaj argumentu — **funkcję anonimową**. Funkcja anonimowa jest zwyczajną funkcją zawierającą wszystkie czynności, jakie chcemy wykonać na pobranym elemencie strony. Określamy ją słowem *anonimowa*, gdyż, w odróżnieniu od zwyczajnych funkcji, które poznaleś na stronie 110, 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 wywoływanego zwykłej 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
});
```

Pozycządowe elementy tej konstrukcji zostały pokazane i opisane na rysunku 4.6. 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.



Skoro zewnętrzna struktura funkcji jest już gotowa, czas umieścić coś wewnętrz funkcji anonimowej, a konkretnie — wszystkie czynności, jakie mają zostać wykonane dla każdego z wybranych elementów strony. Funkcja `each()` działa jak pętla — instrukcje umieszczone wewnętrz funkcji anonimowej zostaną wykonane jeden raz dla każdego z pobranych elementów. Założymy 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 133, 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)

Oczywiście 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="biblList">
</ul>
</div>
```

Pierwszym krokiem jest pobranie listy wszystkich odnośników wskazujących strony spoza witryny. Możemy ją pobrać przy użyciu selektora atrybuty (patrz strona 145):

```
$('.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 `` (został on pokazany w kodzie HTML przedstawionym na stronie 162). 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 rozdziale napiszemy skrypt, który ułatwia tworzenie wyróżnianych cytatów (wyglądających tak jak przedstawione na rysunku 4.7). *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>`, `` bądź jeszcze innych.

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.



Rysunek 4.7. Ręczne dodawanie wyróżnionych cytatów jest bolesne, zwłaszcza gdy weźmiemy pod uwagę fakt, że przy użyciu języka JavaScript można bardzo łatwo zautomatyzować cały ten proces

Opis rozwiązania

Skrypt, który napiszemy, będzie realizował kilka zadań.

1. Odnajdzie na stronie wszystkie znaczniki `` 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 ``, wewnętrznych których umieścimy tekst, jaki ma zostać przekształcony na wyróżnione cytaty. Na przykład założymy, że na stronie znajduje się akapit, zawierający kilka słów, które chcemy pobrać i wyświetlić w formie wyróżnionego cytatu. Wystarczy umieścić te słowa wewnątrz znacznika ``: `...` i właśnie w taki sposób odkryłem potwora z Loch Ness.

2. Powielki każdy ze znaczników .

Każdy wyróżniony cytat jest kolejnym znacznikiem zawierającym dokładnie ten sam tekst, a zatem możemy skorzystać z JavaScriptu, by powieść istniejące na stronie znaczniki .

3. Usunie z powielonych znaczników 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 jest wyłącznie efektem użycia JavaScriptu do zmiany używanej w nich nazwy klasy.

4. Doda powielony znacznik do kodu strony.

W końcu, powielone znaczniki 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 wyświetlane 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 43.

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 zamkajającego znacznika </head> i pisz:

```
<script src="../_js/jquery-1.6.3.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.

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-1.6.3.min.js"><script>
2  <b><script>
3
4  </script>
```

Uwaga: Numery wyświetcone 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-1.6.3.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 występować 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 wróćmy 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, koniecznie się zatem 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-1.6.3.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-1.6.3.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 160, `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-1.6.2.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 162, 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 skopowanego znacznika.

8. Dodaj dwa wyróżnione pogrubieniem wiersze kodu (6. i 7.):

```
1 <script src="../_js/jquery-1.6.3.min.js"></script>
2 <script>
3 $(document).ready(function() {
4     $('span.pq').each(function() {
5         var quote=$(this).clone();
6         quote.removeClass('pq');
7         quote.addClass('pullquote');
8     }); //koniec funkcji each
9 }); //koniec funkcji ready
10 </script>
```

Zgodnie z informacjami podanymi na stronie 155, 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 ``, 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.):

```
1 <script src="../_js/jquery-1.6.3.min.js"></script>
2 <script>
3 $(document).ready(function() {
4     $('span.pq').each(function() {
5         var quote=$(this).clone();
6         quote.removeClass('pq');
7         quote.addClass('pullquote');
```

```
8      $(this).before(quote);
9  });
10 });
11 </script>
```

Wyróżniony pogrubieniem wiersz kodu jest ostatnim elementem funkcji — do tej pory operowaliśmy na kopii znacznika `` przechowywanej w pamięci przeglądarki. Użytkownik nie zobaczyłby funkcji aż do momentu dodania jej do modelu obiektów dokumentu strony.

W tym kroku dodajemy kopię znacznika ``; umieszczamy ją w kodzie HTML strony bezpośrednio przed oryginalnym elementem. W efekcie wynikowa strona będzie zawierać kod HTML w następującej postaci: `... i właśnie w taki sposób odkryłem potwora z Loch Ness.... i właśnie w taki sposób odkryłem potwora z Loch Ness.`.

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 „optał” 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 odzszukaj zdanie i umieść je wewnątrz znaczników ``, na przykład tak:**

```
<span class="pq">Nullam ut nibh sed orci tempor rutrum.</span>
```

Möżesz 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 4.7. Jeśli jednak nie zobaczysz wyróżnionego cytatu, upewnij się, że prawidłowo dodałeś do kodu strony znacznik ``, zgodnie z informacjami podanymi w kroku 10. Dodatkowo przejrzyj podane na stronie 48 porady związane z poprawianiem niedziałających programów. Pełną wersję przykładu możesz znaleźć w pliku `complete_pull-quote.html`.

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 kursora 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.

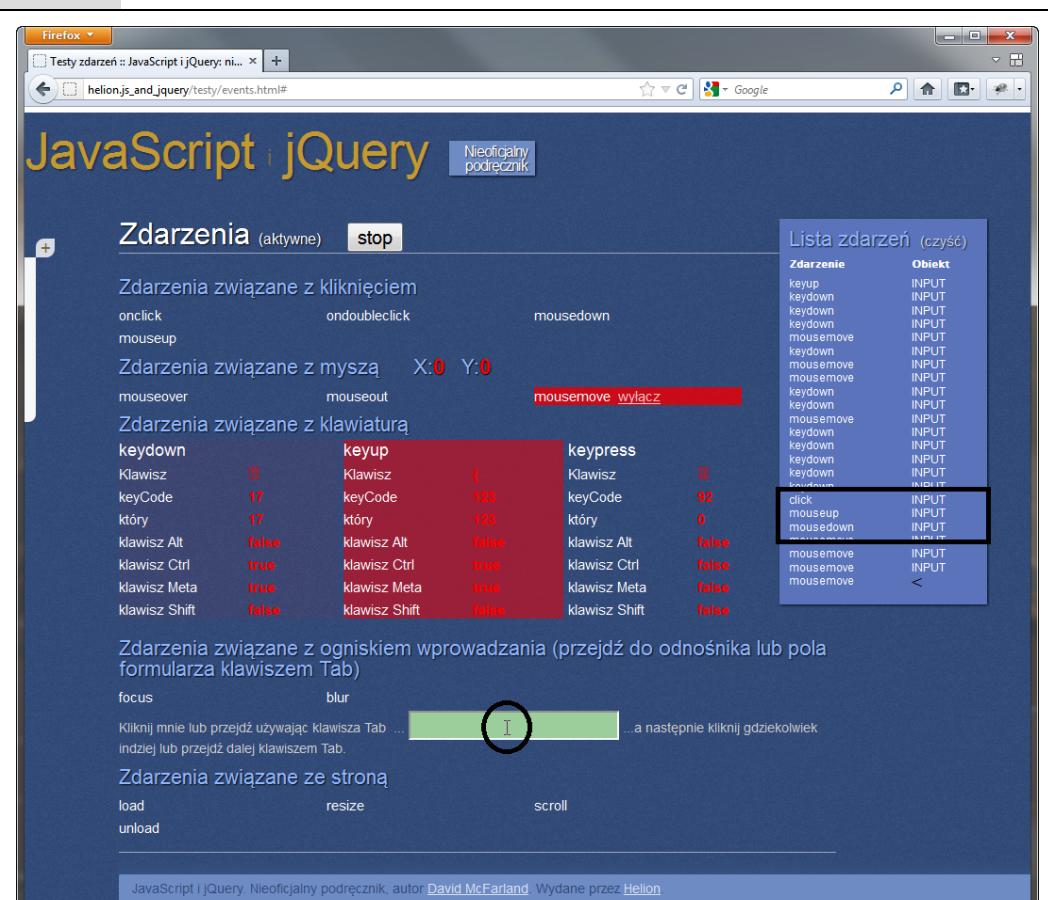
Różne działania użytkowników, na które strona może reagować, to tak zwane *zdarzenia*. JavaScript to język sterowany *zdarzeniami*. Bez tego mechanizmu strona nie może reagować na zachowania internautów ani wykonywać ciekawych operacji. Podobnie działają komputery. Kiedy włączysz system, nic się nie dzieje, dopóki nie zaczniesz uruchamiać programów, kliknąć plików, wybierać opcji z menu i ruszać kursorem po ekranie.

Czym są zdarzenia?

Przeglądarki są zaprogramowane tak, aby wykrywały podstawowe zjawiska, takie jak wczytanie strony, przeniesienie kurSORA, 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 kurSORA 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 zgłaszą kilka zdarzeń. Najpierw, bezpośrednio po wcisnięciu przycisku, informują o zdarzeniu `mousedown`. Następnie, przy zwolnieniu przycisku, zgłaszą zdarzenie `mouseup`, a później — zdarzenie `click` (patrz rysunek 5.1).



Rysunek 5.1. Choć może nie zdajesz sobie z tego sprawy, przeglądarki nieustannie zgłaszą zdarzenia w czasie wpisywania tekstu, poruszania kursorem myszy i klikania. Strona `events.html` (dostępna wśród przykładowych plików) ilustruje działanie wielu zdarzeń. Na przykład kliknięcie zakreślonego na rysunku odnośnika powoduje zgłoszenie zdarzeń `mousedown`, `mouseup` i `click`

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 kursem, kliknij dowolny element i wpisz tekst, aby zobaczyć niektóre z wielu zdarzeń nieustannie zachodzących na stronach WWW (patrz rysunek 5.1). Przykładowo kliknięcie pola tekstowego (zakreślonego na rysunku) powoduje zgłoszenie zdarzeń `mousedown`, `focus`, `mouseup` oraz `click`.

Zdarzenia związane z myszą

Od 1984 roku, kiedy to Steve Jobs zaprezentował komputer Macintosh, mysz stała się podstawowym urządzeniem we wszystkich komputerach osobistych. 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:

- **Zdarzenie click.** To zdarzenie jest zgłaszanego 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 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 dwukrotne kliknięcie i dwa zwykłe kliknięcia są tym samym zdarzeniem, dlatego nie należy przypisywać ich do jednego znacznika. 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.
- **Zdarzenie mouseup.** To zdarzenie odpowiada drugiej części kliknięcia — zwolnieniu przycisku myszy. Pozwala ono wykryć moment upuszczenia przenoszonego elementu.
- **Zdarzenie mouseover.** Kiedy umieścisz kursor nad elementem strony, przeglądarka zgłosi zdarzenie `mouseover`. Przy jego użyciu można przypisać uchwyty zdarzeń do przycisków nawigacyjnych i wyświetlać menu rozwijane po umieszczeniu kurSORA nad danym przyciskiem. Jeśli używałeś kiedyś pseudoklasy `:hover` języka CSS, wiesz już, jak działa to zdarzenie.
- **Zdarzeniemouseout.** Przeniesienie kurSORA poza dany element wyzwala zdarzenie `mouseout`. Można go użyć do zasygnalizowania, że użytkownik przeniósł kurSOR poza stronę, lub do ukrycia menu po opuszczeniu jego obszaru przez kurSOR.

- **Zdarzenie mousemove.** To zdarzenie jest wyzwalane przy każdym ruchu myszą, czyli prawie cały czas. Można go używać do sprawdzania aktualnej pozycji kurSORA 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: Niektóre przeglądarki, na przykład Internet Explorer ([http://msdn2.microsoft.com/en-us/library/ms533051\(VS.85\).aspx](http://msdn2.microsoft.com/en-us/library/ms533051(VS.85).aspx)), obsługują liczne zdarzenia, jednak nie wszystkie mają równie rozbudowane możliwości w tym zakresie.

Zdarzenia związane z dokumentem i oknem

Okno przeglądarki obsługuje liczne zdarzenia, zgłoszane od momentu wczytania strony do czasu jej zamknięcia przez użytkownika:

- **Zdarzenie load.** To zdarzenie jest zgłoszane 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 zwykle używają 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 182).
- **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. Na przykład kiedy użytkownik zmieni rozmiar okna, warto sprawdzić jego szerokość. Jeśli jest duża, można zmodyfikować układ i zapełnić puste miejsce nowymi kolumnami.

Uwaga: Przeglądarki Internet Explorer, Opera i Safari zgłoszają zdarzenie `resize` wielokrotnie w czasie zmiany rozmiaru okna, natomiast Firefox robi to tylko raz, po zakończeniu tej operacji.

- **Zdarzenie scroll.** To zdarzenie jest zgłoszane 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.
- **Zdarzenie unload.** Kiedy klikniesz odnośnik, aby przejść do nowej strony, albo zamkniesz zakładkę 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 zamknięcia strony. Każda próba opuszczenia strony prowadziła do pojawięcia się jej w nowym oknie. Jednak tego zdarzenia

można używać także w wartościowy sposób. Na przykład 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 formularzy opartych na kodzie 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 związań z nimi zdarzeń:

- **Zdarzenie submit.** To zdarzenie jest zgłaszanego przy przesyłaniu formularza. Użytkownik może to zrobić przez kliknięcie przycisku *Wyślij* lub wcisnięcie 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 293 dowiesz się, jak przeprowadzić walidację formularza.
- **Zdarzenie reset.** Choć przycisk *Wyczść* 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 przejdiesz 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*. Założmy, 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 przeciwwieństwo 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 wcisniesz 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 wyświetlana 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 188.

- **Zdarzenie keypress.** To zdarzenie jest zgłaszanego w momencie wciśnięcia klawisza. Następnie przeglądarka wciąż zgłasza je do czasu zwolnienia klawisza.
- **Zdarzenie keydown.** To zdarzenie działa podobnie jak zdarzenie keypress — jest zgłaszanego przy wciśnięciu klawisza. Zachodzi tuż przed zdarzeniem keypress. W Firefoksie i Operze jest zgłaszanego tylko raz. W przeglądarkach Internet Explorer i Safari działa jak zdarzenie keypress — jest zgłaszanego, 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 i starszych, 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, rozwiązuje 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 wykonawczych zdarzeń (funkcji, które reagują na zdarzenie; ang. *event helper*) jest banalnie proste.

Jak mogłeś się przekonać na stronie 136, 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 kurSOR 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. Na przykład jeśli 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 171 do 174. Ponadto dostępnych jest kilka zdarzeń specyficznych dla biblioteki jQuery, omówionych na stronie 183.

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 kurSOR 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 160 — 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 jest 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 rozpoczęcej się od selektora `$(‘a’)`.

Oto prosty przykład. Założymy, że na stronie znajduje się odnośnik o identyfikatorze `menu`. Kiedy użytkownik umieści kursor 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 mouseover
```

Często można zobaczyć kolekcje zamkających nawiasów klamrowych, zwykłych 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 mouseover
```

Wiele osób uważa zbitek znaków specjalnych potrzebny przy używaniu funkcji anonimowych za mało zrozumiałą (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 utrważyć zdobytą wiedzę.

Uwaga: Funkcję `show()` opisano na stronie 198.

Przykład — wyróżnianie wierszy tabeli

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 43 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 zamk傢acego znacznika `</head>`, i wpisz w nim:

```
<script type="text/javascript" src="../js/jquery.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-1.6.3.min.js"></script>
<script>
</script>
```

A teraz dodaj wywołanie funkcji `document.ready()`.

4. Kliknij pusty wiersz pomiędzy otwierającym i zamk傢acem znacznikiem `<script>`, a następnie wpisz poniższy kod wyróżniony pogrubioną czcionką:

```
<script src="../../_js/jquery-1.6.3.min.js"></script>
<script>
$(document).ready(function() {
}); // koniec 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 wykoнаłeś 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 wewnętrz funkcji ready() i wpisz poniższy kod wybrany pogrubioną czcionką:

```
<script src="../_js/jquery-1.6.3.min.js"></script>
<script>
$(document).ready(function() {
    $('html')
}); //koniec 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-1.6.3.min.js"></script>
<script>
$(document).ready(function() {
    $('html').dblclick(); //koniec dblclick
}); //koniec 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. Jednym 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 113).

7. Teraz dodaj funkcję anonimową, czyli wpisz poniższy kod wyróżniony pogrubioną czcionką:

```
<script src="../_js/jquery-1.6.3.min.js"></script>
<script>
$(document).ready(function() {
    ($('html').dblclick(function(){
```

```
        }; //koniec dblclick
    }); //koniec 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 wewnętrz nawiasów klamrowych { }. A tym zajmiemy się w kolejnym kroku.`

8. Teraz dodaj funkcję `alert()`:

```
<script src="../_js/jquery-1.6.3.min.js"></script>
<script>
$(document).ready(function() {
    ($('html').dblclick(function(){
```

```
        alert('ała!');
    }); //koniec dblclick
}); //koniec 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 JavaScriptie grę wyścigową.

A teraz, kiedy poznaleś już podstawy, spróbujemy obsłużyć kilka innych zdarzeń.

9. Dodaj fragment kodu wyróżniony pogrubioną czcionką, by skrypt wyglądał tak, jak na poniższym przykładzie:

```
<script src="../../_js/jquery-1.6.3.min.js"></script>
<script>
$(document).ready(function() {
    $('html').dblclick(function(){
        alert('ała!');
    }); // koniec dblclick
    $('a').mouseover(function(){
        });
    });
}); // koniec 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.

10. Do anonimowej funkcji utworzonej w poprzednim kroku dodaj dwie instrukcje JavaScriptu:

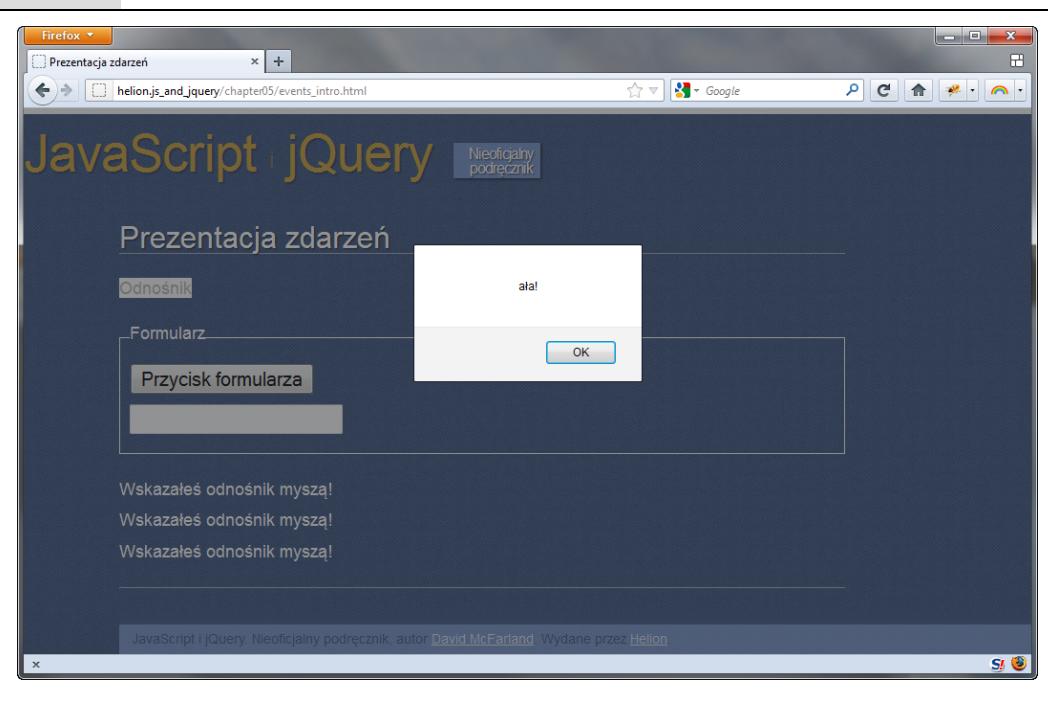
```
<script src="../../_js/jquery-1.6.3.min.js"></script>
<script>
$(document).ready(function() {
    $('html').dblclick(function(){
        alert('ała!');
    }); // koniec dblclick
    $('a').mouseover(function(){
        var message = "<p>Wskazałeś odnośnik myszą!</p>";
        $('.main').append(message);
    });
}); // koniec mouseover
}); // koniec ready
</script>
```

Pierwszy wiersz kodu — `var message = "<p>Wskazałeś odnośnik myszą!"` — definiuje nową zmienną o nazwie `message` i zapisuje w niej łańcuch znaków. łańcuch ten zawiera znacznik akapitu i tekst umieszczony wewnętrzniego. 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 151).

11. **Zapisz stronę, wyświetl ją w przeglądarce i spróbuj przesunąć wskaźnik myszy 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

12. I w końcu dodaj poniższy kod wyróżniony pogrubioną czcionką, żeby skrypt wyglądał tak, jak na poniższym przykładzie:

```
<script src="../../_js/jquery-1.6.3.min.js"></script>
<script>
$(document).ready(function() {
    $('html').dblclick(function(){
        alert('ała!');
    });
    // koniec dblclick
    $('a').mouseover(function(){
```

```
var message = "<p>Wskazałeś odnośnik myszą!</p>";
$('.main').append(message);
}); // koniec mouseover
$('#button').click(function(){
  $(this).val("Przestań!");
});
}); // koniec click
}); // koniec 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 tekstu wyświetlony na przycisku zostanie zmieniony na „Przestań!”. Poniżej znajdziesz wyjaśnienie sposobu, w jaki kod umieszczony wewnątrz funkcji anonimowej pozwala na wprowadzenie takiej zmiany.

Na stronie 162 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 275, 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).

13. 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 278), (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 155), by zmienić kolor tła pola. Odpowiedź (jak również pełną 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 zwiększą 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 118, 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 podstawa 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 używać 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 zamkającym znacznikiem `<script>` (w końcu jest to kod JavaScript) i po elemencie `<script>` dołączającym do strony bibliotekę jQuery.

Dlatego w kontekście kompletnej strony WWW funkcja ta powinna wyglądać następująco:

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN" "http://www.w3.org/TR/  
html4/strict.dtd">  
<html>  
<head>  
<meta http-equiv="Content-Type" content="text/html; charset=UTF-8">  
<title>Tytuł strony</title>  
<script type="text/javascript" src="js/jquery.js"></script>
```

```
<script type="text/javascript">
$(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() {
});
```

Zdarzenia biblioteki jQuery

jQuery udostępnia też specjalne zdarzenia do obsługi dwóch często spotykanych form interakcji — przenoszenia kurSORA nad element i poza niego oraz przełączania ustawień za pomocą kliknięcia.

Zdarzenie hover()

Zdarzenia `mousover` i `mouseout` często są używane razem. Na przykład kiedy użytkownik umieści kurSOR nad przyciskiem, pojawi się menu, a gdy kurSOR znajdzie się poza elementem, menu znika. Ponieważ łączenie tych dwóch zdarzeń jest tak powszechnie, jQuery udostępnia skrótowy zapis uwzględniający oba. Funkcja `hover()` biblioteki jQuery działa jak każde inne zdarzenie, jednak jako argument przyjmuje dwie funkcje zamiast jednej. Pierwsza funkcja jest uruchamiana po umieszczeniu kurSORA nad elementem, a druga — kiedy użytkownik przeniesie kurSOR w inne miejsce. Podstawowa struktura tej funkcji wygląda następująco:

```
$('#selektor').hover(funkcja1, funkcja2);
```

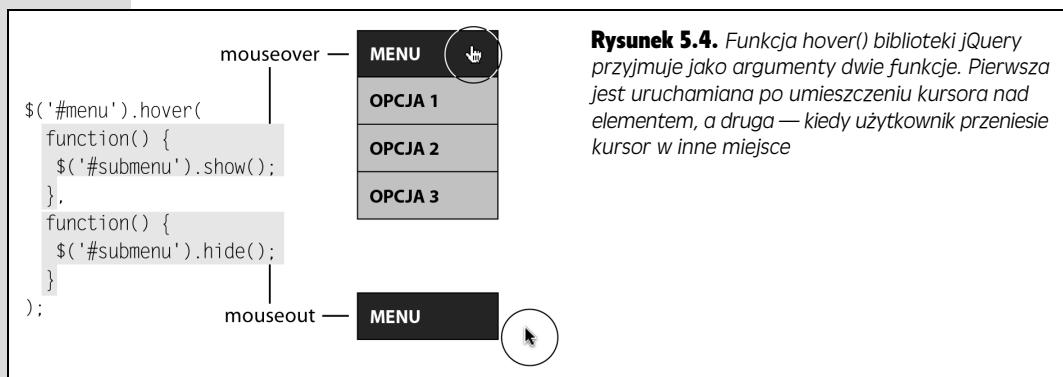
Często napotkasz funkcję `hover()`, używaną wraz z dwiema funkcjami anonimowymi. Taki kod wygląda dziwnie, jednak następny przykład powinien ułatwić jego zrozumienie. Założymy, że po najechaniu kursorem na odnośnik o identyfikatorze `menu` skrypt ma wyświetlać niewidoczny początkowo znacznik `DIV` o identyfikatorze `submenu`. Przeniesienie kurSORa 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 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 mouseover  
    function() {  
        $('#submenu').hide();  
    } // koniec mouseout  
>); // koniec hover
```

Na rysunku 5.4 przedstawiam działanie tego kodu przy wystąpieniu zdarzeń mouseover i mouseout.



Jeśli uważasz, że metoda oparta na funkcjach anonimowych jest zbyt skomplikowana, możesz uzyskać ten sam efekt za pomocą zwykłych funkcji (patrz strona 110). 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 show_submenu() {  
    $('#submenu').show();  
}  
function hide_submenu() {  
    $('#submenu').hide();  
}  
$('#menu').hover(show_submenu, hide_submenu);
```

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.

Zdarzenie `toggle()`

Zdarzenie `toggle()` biblioteki jQuery działa podobnie jak zdarzenie `hover()`, jednak reaguje nie na zdarzenia `mouseover` i `mouseout`, ale na kliknięcia. Pierwsze kliknięcie uruchamia pierwszą, a następne drugą funkcję. To zdarzenie służy do obsługi zmiany stanu w reakcji na kliknięcie. Przykładowo pierwsze kliknięcie przycisku mogłoby uruchamiać animację, a drugie — ją zatrzymywać. Kolejne kliknięcie mogłoby uruchamiać animację od początku.

Załóżmy, że chcesz, aby program wyświetlał element <div> o identyfikatorze submenu (z opisu funkcji hover()) po pierwszym kliknięciu odnośnika, a następnie ukrywał go po kolejnym kliknięciu. Aby osiągnąć ten cel, wystarczy zastąpić we wcześniejszym kodzie słowo „hover” nazwą „toggle”:

```
$('#menu').toggle(  
    function() {  
        $('#submenu').show();  
    }, //pierwsze kliknięcie  
    function() {  
        $('#submenu').hide();  
    } //drugie kliknięcie  
) //koniec toggle
```

Wersja ze standardowymi funkcjami wygląda następująco:

```
function show_submenu() {  
    $('#submenu').show();  
}  
function hide_submenu() {  
    $('#submenu').hide();  
}  
$('#menu').toggle(show_submenu, hide_submenu);
```

Jeśli na przykład chcemy, by pierwsze kliknięcie spowodowało wykonanie jednej czynności, drugie — kolejnej, a trzecie — jeszcze innej, w wywoaniu funkcji toggle() można przekazać więcej niż dwie funkcje jako argumenty. Każda funkcja przekazana w wywoaniu toggle() zostanie wywołana w odpowiedzi na kolejne kliknięcie elementu. I tak pierwsza funkcja zostanie wywołana w odpowiedzi na pierwsze kliknięcie, druga — w odpowiedzi na drugie kliknięcie, trzecia — w odpowiedzi na trzecie i tak dalej. Kiedy jQuery wykona już wszystkie przekazane funkcje, po kolejnym kliknięciu ponownie wykona pierwszą z nich (jeśli na przykład w wywoaniu zostały przekazane trzy funkcje, czwarte kliknięcie spowoduje ponowne, drugie wywołanie pierwszej z nich).

Obiekt reprezentujący zdarzenie

Kiedy przeglądarka zgłasza zdarzenie, rejestruje informacje na jego temat i zapisuje go w *obiekcie zdarzenia*. Ten obiekt zawiera dane zebrane w momencie wystąpienia zdarzenia, na przykład współrzędne kurSORA 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 kurSORA w momencie kliknięcia dowolnego fragmentu strony:

```
$(document).click(function(evt) {  
    var xPos = evt.pageX;  
    var yPos = evt.pageY;  
    alert('X:' + xPos + ' Y:' + yPos);  
) //koniec 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 kurSORA, 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 kurSORA myszy od lewej krawędzi okna przeglądarki.
pageY	Odległość w pikselach kurSORA myszy od górnej krawędzi okna przeglądarki.
screenX	Odległość w pikselach kurSORA myszy od lewej krawędzi monitora.
screenY	Odległość w pikselach kurSORA 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 bind() do przekazania danych do funkcji obsługi zdarzenia (patrz strona 188).

Wskazówka: Za pomocą właściwości which obiektu zdarzenia keypress() można pobrać kod wciśniętego klawisza. Jeśli chcesz ustalić, jaki znak wciśniął 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), dla tego można ją wywołać w funkcji obsługującej dane zdarzenie. Założmy, 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. Domyslnie przeglądarka używa 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 `unbind()`. Aby jej użyć, najpierw należy utworzyć obiekt jQuery, reprezentujący element z usuwanym zdarzeniem. Następnie wystarczy dodać funkcję `unbind()` i przekazać do niej łańcuch znaków z nazwą zadanego zdarzenia. Jeśli chcesz sprawić, aby wszystkie znaczniki `tabButton` nie reagowały na kliknięcie, możesz wywołać poniższą instrukcję:

```
$('.tabButton').unbind('click');
```

Poniższy krótki skrypt ilustruje działanie funkcji `unbind()`:

```
1 $('a').mouseover(function() {  
2     alert('Kursor znajduje się nade mną!');  
3 });  
4 $('#disable').click(function() {  
5     $('a').unbind('mouseover');  
6 });
```

Wiersze od 1. do 3. przypisują funkcję do zdarzenia `mouseover` wszystkich odnośników (znaczników `<a>`). Umieszczenie kurSORA nad odnośnikiem spowoduje wyświetlenie okna dialogowego z tekstem „Kursor 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 odfłączy zdarzenie `mouseover` od wszystkich odnośników, dlatego okno dialogowe przestanie się pojawiać.

Uwaga: Więcej informacji o funkcji `unbind()` biblioteki jQuery znajdziesz na stronie <http://api.jquery.com/unbind/>.

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. Założymy, że przypisales funkcję wykonawczą do 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, wewnętrzny którego jest umieszczony kliknięty element). Dlatego jeśli przypisales też funkcję wykonawczą do 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`.

Biblioteka jQuery udostępnia funkcję `stopPropagation()`, która zatrzymuje przekazywanie zdarzenia do przodków. Jest to metoda obiektu zdarzenia (patrz strona 185), 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 używać funkcji `bind()`.

Uwaga: Jeśli wciąż przyswajasz sobie materiał przedstawiony w poprzednim podrozdziale, możesz pominąć ten fragment i przejść bezpośrednio do przykładu na stronie 191. Zawsze możesz wrócić do tego punktu, kiedy nabierzesz doświadczenia w obsłudze zdarzeń.

Metoda `bind()` 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 kurSORA 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 `bind()` wygląda następująco:

```
$('#selektor').bind('click', myData, functionName);
```

Pierwszy argument to łańcuch znaków zawierający nazwę zdarzenia (na przykład `click`, `mouseover` lub dowolne inne zdarzenie wymienione na stronie 171). Drugi argument to dane przekazywane do funkcji. Może to być literał obiektowy lub zmienna zawierająca taki literał. Literaly obiektowe (patrz strona 157) 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'};
```

Trzeci argument funkcji `bind()` to następna funkcja. Jest ona uruchamiana po zgłoszeniu zdarzenia. Można użyć tu funkcji anonimowej lub standardowej, podobnie jak przy używaniu zwykłych zdarzeń biblioteki jQuery, co opisano na stronie 175.

Uwaga: Przekazywanie danych w funkcji `bind()` nie jest konieczne. Jeśli chcesz jej użyć do dołączenia zdarzenia i funkcji do elementu, możesz pominąć zmienną z danymi:

```
$('.selektor').bind('click', functionName);
```

Ten kod działa tak samo jak poniższa instrukcja:

```
$('.selektor').click(functionName);
```

Założymy, 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 literaly obiektowe, a następnie przekazywać te zmienne do funkcji `bind()`, powiązanej z różnymi elementami:

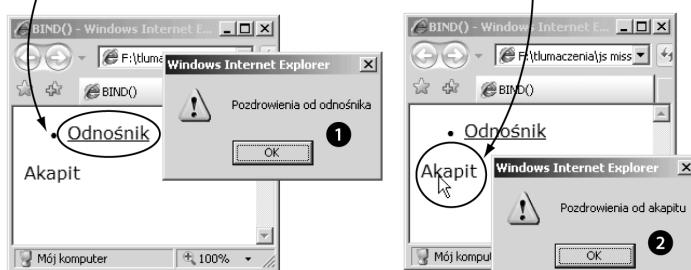
```
var linkVar = { message:'Pozdrowienia od odnośnika'};  
var pVar = { message:'Pozdrowienia od akapitu'};  
function showMessage(evt) {  
    alert(evt.data.message);  
}  
$('a').bind('click',linkVar,message);  
$('p').bind('mouseover',pVar,message);
```

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 185) 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.

Inne sposoby stosowania funkcji bind()

Funkcja `bind()` 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 założmy, że piszemy

```
var ①linkVar = { message:'Pozdrowienia od odnośnika'};  
var ②pVar = { message:'Pozdrowienia od akapitu'};  
③ function showMessage(evt) {  
    alert(evt.data.message);  
}  
$('a').bind('click', ① linkVar, ③ showMessage);  
$('p').bind('mouseover', ② pVar, ③ showMessage);
```



Rysunek 5.5. Funkcja bind() biblioteki jQuery umożliwia przekazywanie danych do funkcji obsługujących zdarzenia. W ten sposób można użyć jednej standardowej funkcji do obsługi kilku elementów (a nawet różnych zdarzeń) i jednocześnie korzystać w tej funkcji z danych specyficznych dla funkcji wykonawczych

program, który w odpowiedzi na kliknięcie miniaturki zdjęcia będzie wyświetlał na ekranie jego powiększoną wersję (jest to popularne rozwiązywanie o nazwie „lightbox”, które możemy znaleźć na tysiącach stron; jak ono działa, dowiesz się na stronie 234). 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).bind('click keypress', function() {  
    $('#lightbox').hide();  
}); // koniec bind
```

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 bind(). 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').bind('click', function() {  
    // tu robimy coś interesującego  
}); // koniec bind  
$('#theElement').bind('mouseover', function() {  
    // tu robimy coś interesującego  
}); // koniec bind
```

Jednak dokładnie to samo można uzyskać, przekazując w wywołaniu metody `bind()` literał obiektowy (patrz strona 157) składający się z nazwy zdarzenia oraz podanej po dwukropku funkcji anonimowej. Poniżej przedstawiona została zmodyfikowana wersja powyższego fragmentu kodu, w której funkcja `bind()` jest wywoływana tylko raz, a przy tym w jej wywołaniu jest przekazywany literał obiektowy (wyróżniony pogrubioną czcionką):

```
$('#theElement').bind({
  'click' : function() {
    // tu robimy coś interesującego
  },
  // koniec funkcji click
  'mouseover' : function() {
    // tu robimy coś interesującego
  };
  // koniec funkcji mouseover
}); // koniec bind
```

Uwaga: Jakby tego wszystkiego było mało, jQuery udostępnia także jeszcze inne sposoby kojarzenia zdarzeń z elementami. Konkretnie rzecz biorąc, funkcja `delegate()` okazuje się niezwykle przydatna w przypadkach, gdy chcemy skojarzyć zdarzenie z elementem, który zostaje dodany do strony dopiero po jej całkowitym wczytaniu (czyli takim, który jest dodawany do strony programowo bądź wczytany przy użyciu techniki AJAX opisanej w czwartej części tej książki). Więcej informacji na temat funkcji `delegate()` można znaleźć na stronie 441.

Przykład — jednostronicowa lista FAQ

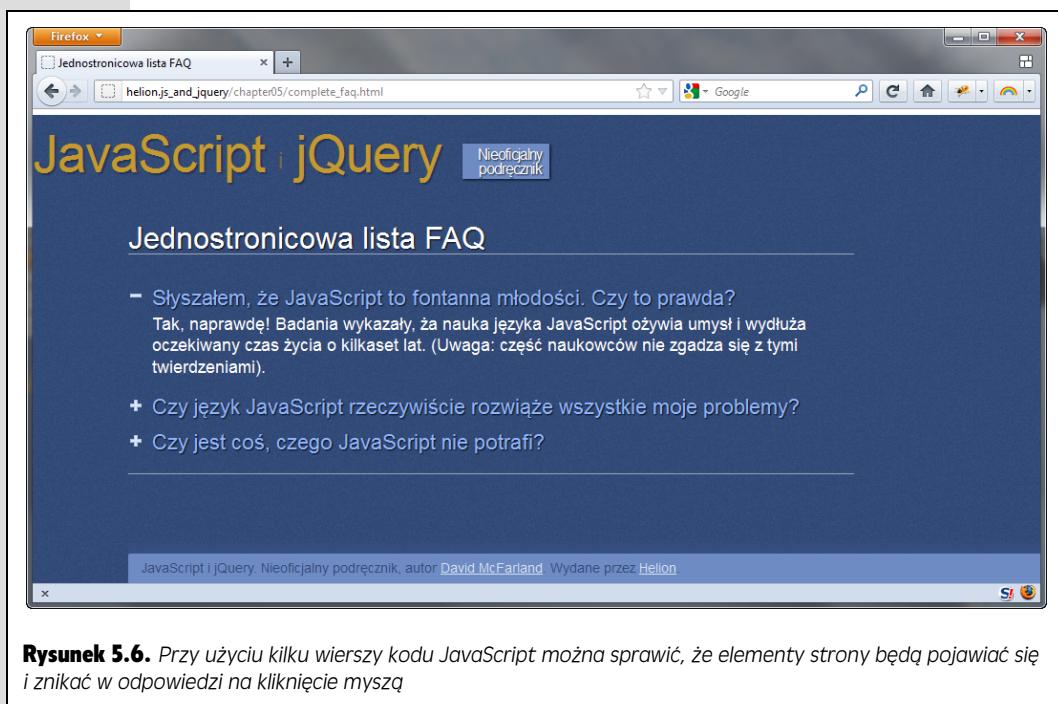
W sieci WWW znajduje się wiele stron z listami często zadawanych pytań (ang. *Frequently Asked Questions* — FAQ). Pozwalają one błyskawicznie uzyskać odpowiedź na pytanie — 24 godziny na dobę 7 dni w tygodniu, co usprawnia obsługę klienta. Niestety, większość list FAQ to albo bardzo długie strony pełne 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 5.6).

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.



Rysunek 5.6. Przy użyciu kilku wierszy kodu JavaScript można sprawić, że elementy strony będą pojawiać się i znikać w odpowiedzi na kliknięcie myszą

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 wyłązoną obsługę języka JavaScript, użytkownik nie zobaczy odpowiedzi ani nie będzie mógł ich wyświetlić. Aby strona była przydatna zarówno w przeglądarkach złączoną, jak i wyłąconą obsługą skryptów, najlepiej ukryć jej zawartość przy użyciu kodu JavaScript.

Uwaga: Informacje o pobieraniu przykładowych plików znajdziesz na stronie 43.

Tworzenie kodu

1. Otwórz w edytorze tekstu plik `faq.html` z katalogu `R06`.

Ten plik zawiera już funkcję `$(document).ready()` (patrz strona 169) i kod dodający 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 198). Zapisz stronę i wyświetl ją w przeglądarce. Wszystkie odpowiedzi powinny być ukryte.

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-1.6.3.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 — wyświetlanie lub ukrywanie odpowiedzi w wyniku kliknięcia — lepiej użyć funkcji `toggle()` (patrz strona 184). Umożliwia ona wywoływanie dwóch różnych funkcji przy kolejnych kliknięciach.

4. Bezpośrednio za kodem wpisanym w kroku 2. (w tym samym wierszu) dodaj fragment `.toggle()`.

Ten kod to początek funkcji `toggle()`, która przyjmuje jako argumenty dwie funkcje anonimowe (patrz strona 160). Pierwsza funkcja będzie uruchamiana przy pierwszym kliknięciu, a druga — przy następnym. Najpierw należy dodać podstawową strukturę obu funkcji anonimowych.

5. Wciśnij klawisz *Enter*, aby dodać nowy wiersz, a następnie wpisz poniższy kod:

```
function() {
}
```

Jest to podstawa struktury funkcji będącej pierwszym argumentem funkcji `toggle()`. Następnie należy dodać strukturę drugiej funkcji anonimowej.

6. Dodaj kod wyróżniony pogrubieniem. Skrypt powinien wyglądać następująco:

```
1 <script src="../../_js/jquery-1.6.3.min.js"></script>
2 <script>
3 $(document).ready(function() {
4     $('.answer').hide();
5     $('#main h2').toggle(
6         function() {
7
8             },
9             function() {
10
11         }
12     ) //Koniec funkcji toggle
13 }); //Koniec funkcji ready
14 </script>
```

Nie zapomnij dodać przecinka na końcu wiersza 8. Pamiętaj, że dwie dodane funkcje anonimowe to argumenty funkcji (patrz strona 113). Przy wywoływaniu funkcji należy rozdzielić argumenty przecinkami, na przykład prompt ('Pytanie', 'Tu wpisz odpowiedź'), dlatego przecinek w wierszu 8. jest niezbędny. Jeśli chcesz, możesz pominąć komentarz z wiersza 12. — // Koniec funkcji toggle. Dodano go jedynie w celu wyjaśnienia, że dany wiersz kończy funkcję toggle().

Pora dodać pożądany efekt. Pierwsze kliknięcie znacznika <h2> ma powodować wyświetlenie właściwej odpowiedzi. Każde pytanie znajduje się w znaczniku <h2>, a powiązana z nim odpowiedź — w znajdująącym się bezpośrednio po nim tagu <div>. Ponadto znaczniki <div> mają przypisaną klasę answer. Dlatego potrzebny jest kod, który pobierze znacznik <div>, znajdujący się po klikniętym tagu <h2>.

7. W pierwszej funkcji (wiersz 6. w kodzie z kroku 5.) dodaj instrukcję `$(this).next('.answer').fadeIn();`.

Na stronie 162 dowiedziałeś się, że konstrukcja `$(this)` wskazuje na element reagujący na dane zdarzenie. Tu jest to określony znacznik <h2>. Biblioteka jQuery udostępnia kilka funkcji, które ułatwiają poruszanie się po strukturze strony. Funkcja `.next()` zwraca znacznik znajdujący się bezpośrednio za podanym tagiem, czyli znacznik następujący po tagu <h2>. Można doprecyzować wyszukiwanie przez podanie w funkcji `.next()` dodatkowego selektora. Wywołanie `.next('.answer')` pozwala znaleźć pierwszy znacznik klasy answer po tagu <h2>. Polecenie `.fadeIn()` stopniowo wyświetla odpowiedź (opis tej funkcji znajdziesz na stronie 200).

Uwaga: Funkcja `.next()` to jedna z wielu funkcji biblioteki jQuery, które są pomocne przy poruszaniu się po modelu DOM strony. Aby poznać więcej takich funkcji, odwiedź stronę <http://docs.jquery.com/Traversing>.

Teraz możesz zapisać stronę i wyświetlić ją w przeglądarce. Kliknij jedno z pytań, a skrypt powinien otworzyć właściwą odpowiedź. Jeśli program nie działa, uważnie sprawdź kod i przypomnij sobie wskazówki ze strony 48, dotyczące rozwiązywania problemów.

Teraz trzeba uzupełnić efekt i sprawić, aby skrypt ukrywał odpowiedź po drugim kliknięciu pytania.

8. Dodaj w 10. wierszu kod wyróżniony pogrubieniem:

```

1  <script src="../../_js/jquery-1.6.3.min.js"></script>
2  <script>
3  $(document).ready(function() {
4      $('.answer').hide();
5      $('#main h2').toggle(
6          function() {
7              $(this).next('.answer').fadeIn();
8          },
9          function() {
10             $(this).next('.answer').fadeOut();
11         }
12     ) //Koniec funkcji toggle
13 }); //Koniec funkcji ready
14 </script>
```

Teraz drugie kliknięcie spowoduje ukrycie odpowiedzi. Zapisz stronę i wypróbuju ją. Choć działa prawidłowo, można ją jeszcze uatrakcyjnić. Obecnie przy każdym pytaniu widoczny jest mały znak plus. Jest to standardowa ikona, często używana jako wskazówka: „Uwaga, tu kryje się coś więcej”. Aby poinformować użytkownika o możliwości ukrycia odpowiedzi, warto zastąpić znak plusa minusem. W tym celu wystarczy dodać i usunąć odpowiednie klasy znacznika <h2>.

9. Dodaj dwa ostatnie wiersze kodu (8. i 12. w poniższym fragmencie). Gotowy skrypt powinien wyglądać następująco:

```
1 <script src="../_js/jquery-1.6.3.min.js"></script>
2 <script>
3 $(document).ready(function() {
4     $('.answer').hide();
5     $('#main h2').toggle(
6         function() {
7             $(this).next('.answer').fadeIn();
8             $(this).addClass('close');
9         },
10        function() {
11            $(this).next('.answer').fadeOut();
12            $(this).removeClass('close');
13        }
14    ) //Koniec funkcji toggle
15 });
16 </script>
```

Nowy kod przy pierwszym kliknięciu dodaje do znacznika <h2> klasę close, a przy drugim — usuwa ją. Ikona ze znakiem minus jest użyta w arkuszu stylów jako rysunek tła. Także tu style CSS ułatwiają programowanie w języku JavaScript.

Zapisz stronę i wypróbuj ją. Teraz po kliknięciu pytania nie tylko pojawia się odpowiedź, ale też zmienia się ikona (patrz rysunek 5.6).

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.

Efekty biblioteki jQuery

Częstym zadaniem realizowanym przy użyciu języka JavaScript jest ukrywanie i wyświetlanie elementów stron WWW. Rozwijane menu nawigacyjne, etykiety 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 ten temat można znaleźć na stronie 209, w podrozdziale pt.: „Wykonywanie operacji po zakończeniu efektu”).

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 — miałyby 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 138). Kod HTML elementu wciąż jest przechowywany w pamięci przeglądarki, jednak nie ma żadnej wizualnej reprezentacji (określonej 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 wartościami liczbowymi 200, 400 oraz 600. A zatem wywołanie:

```
$(‘element’).fadeOut(‘slow’);
```

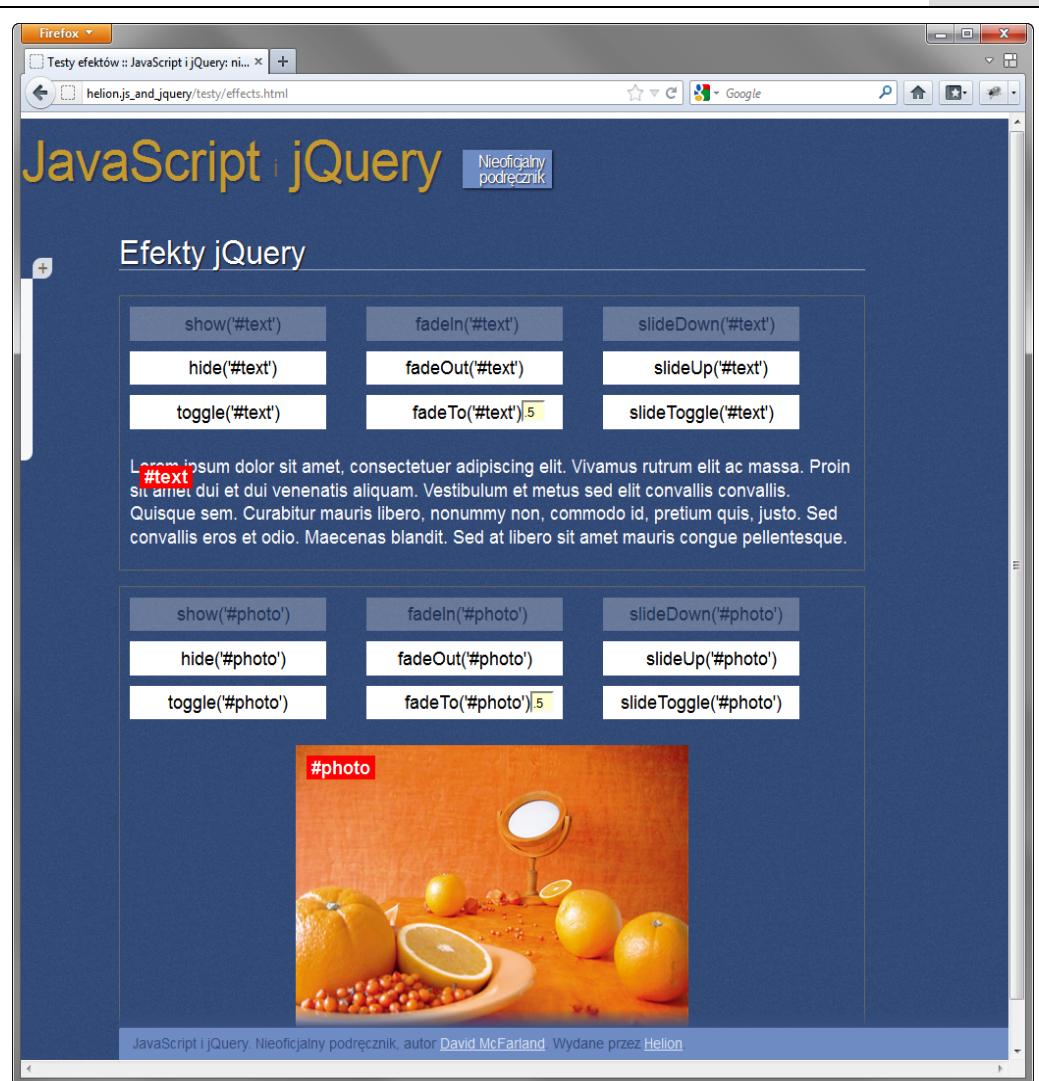
da dokładnie taki sam efekt jak wywołanie:

```
$(‘element’).fadeOut(600);
```

Podstawowe wyświetlanie i ukrywanie

Biblioteka jQuery udostępnia trzy funkcje służące do prostego ukrywania i wyświetlania elementów:

- `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.
- `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.



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

W przykładzie przedstawionym w poprzednim rozdziale na stronie 191 mogłeś zobaczyć metody `hide()` oraz `toggle()` w działaniu. Funkcja `hide()` ukrywała wszystkie odpowiedzi w momencie wyświetlania strony, a następnie funkcja `toggle()` wyświetlała odpowiednią odpowiedź po kliknięciu skojarzonego z nią pytania.

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 pojawi. 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.
- `fadeToggle()` — łączy w sobie funkcje `fadeIn()` oraz `fadeOut()`. Jeśli w momencie wywoływania funkcji element jest niewidoczny, zostanie stopniowo rozjaśniony, jeśli natomiast początkowo jest widoczny — funkcja go wygasí. Funkcji tej można używać, by wyświetlać i ukrywać na stronie ramkę z instrukcjami dla użytkownika. Założymy 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 naprzemienienie wyświetlać i ukrywać element, 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 założymy, 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 wyświetlmy, 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.

NA SZYBKO (UP TO SPEED)

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`. Założmy, 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 wyświetlonym na stronie może zostać umieszczony znak zapytania, którego kliknięcie otwiera niewielką ramkę zawierającą, na przykład, definicję tego słowa. W takim przypadku etykietka ekranowa nie powinna nie być rozmieszczona względem którejś z krawędzi okna przeglądarki, lecz obok odpowiedniego słowa. Wtedy położenie bezwzględnie pozycjonowanego elementu należy określić względem nadzawanego 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 `` zawierający definicję została wyświetlony poniżej słowa, musimy najpierw pozycjonować zewnętrzny znacznik `` 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 *CSS: The Missing Manual*.

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 górną część elementu, a wszelkie inne elementy strony umieszczone poniżej niego zostają przesunięte w dół; następnie pojawia się pozostała część wyświetlalnego elementu. Funkcja ta nie daje żadnego efektu, kiedy element już jest widoczny. Jeśli nie zostanie określona szybkość efektu, funkcja użycie wartości domyślnej 'normal' (odpowiadającej 400 milisekundom).
- `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).

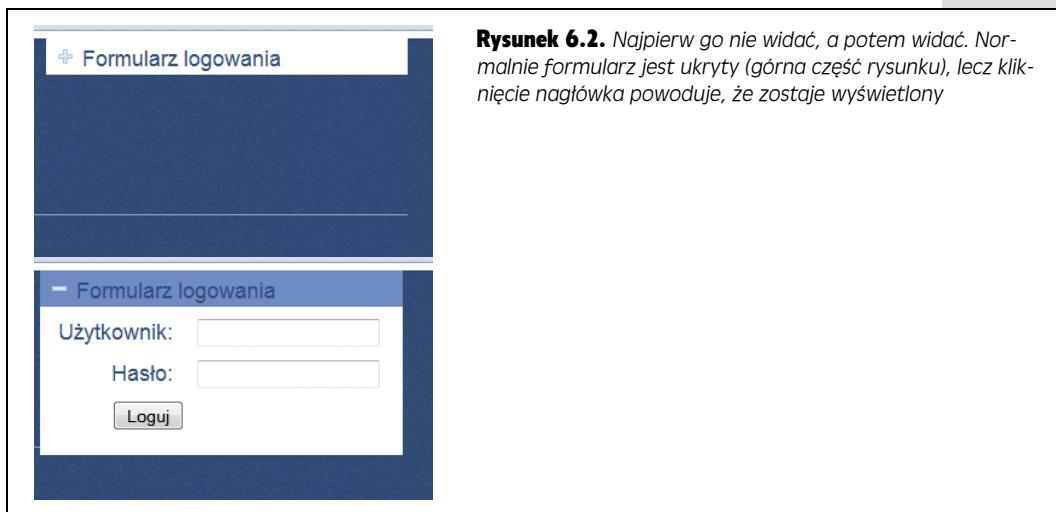
Zasada działania tego przykładu jest prosta:

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", które użytkownicy będą klikali.

2. **Dodajesz do akapitu procedurę obsługi zdarzeń click.**

JavaScript nie zapewnia interaktywności bez wykorzystania zdarzeń: aby coś się stało, użytkownik musi rozpoczęć 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 przezroczystości (z wykorzystaniem funkcji `fadeIn()`).

Uwaga: Informacje na temat pobierania przykładów dołączonych do tej książki można znaleźć na stronie 43.

Tworzenie kodu

1. W edytorze tekstów otwórz plik `signup.html` umieszczony w katalogu R06.

Plik zawiera już odwołanie do pliku jQuery oraz wywołanie funkcji `$(document).ready()` (patrz strona 182). Na początek zajmiesz się wybraniem akapitu zawierającego tekst "Formularz logowania".

2. Kliknij pusty wiersz widoczny poniżej funkcji `$(document).ready()` i wpisz w nim `$('#open')`.

Interesujący nas tekst jest umieszczony wewnątrz akapitu o identyfikatorze `open` — `<p id="open">Formularz logowania</p>`. A zatem wpisane wywołanie jQuery spowoduje pobranie tego akapitu. Kolejnym zadaniem będzie dodanie do niego procedury obsługi zdarzeń.

3. Dodaj kod wyróżniony w poniższym przykładzie pogrubioną czcionką, tak by miał następującą postać:

```
$(document).ready(function() {  
    $('#open').click(function() {
```

```
}); // 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.

4. Kliknij pusty wiersz wewnętrz fragmentu `click()` i wpisz w nim:

```
$('#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.

5. Dodaj poniższy, wyróżniony pogrubieniem fragment kodu, by końcowy skrypt wyglądał tak jak ten:

```
1  $(document).ready(function() {  
2      $('#open').click(function() {  
3          $('#login form').slideToggle(300);  
4          $(this).toggleClass('close');  
5      }); // koniec funkcji click  
6  }); // koniec funkcji ready
```

Po przeczytaniu informacji zamieszczonych na stronie 162 wiesz już, że wewnętrz procedury obsługi zdarzeń można korzystać z wyrażenia `$(this)`, by odwołać się do elementu odpowiadającego na zdarzenie. W naszym przykładzie `$(this)` odwołuje się do akapitu klikniętego 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 `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, wewnętrz sekcji `<head>`).

6. 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_signup.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. powyżej listy nie spełni oczekiwaniń, gdyż funkcja `click()` nie zapewnia możliwości stosowania i wyboru jednej z dwóch różnych akcji. Niemniej jednak, zgodnie z informacjami zamieszczonymi na stronie 184, jQuery udostępnia także specjalne zdarzenie — `toggle()` — opracowane właśnie z myślą o takich sytuacjach. Nie należy go mylić z wizualnym efektem o tej samej nazwie — `toggle()` — który powoduje naprzemienne po-

jawianie się i ukrywanie elementu. Zdarzenie `toggle()` pozwala naprzemiennie wykonywać dwa różne fragmenty kodu podczas parzystych oraz nieparzystych kliknięć. A zatem podczas pierwszego kliknięcia formularz ma się pojawić, a podczas drugiego — zniknąć.

Aby formularz został wsunięty na stronę, a następnie wygaszony podczas kolejnego kliknięcia, powinieneś użyć następującego kodu:

```
$(document).ready(function() {
    $('#open').toggle(
        function() {
            $('#login_form').slideDown(300);
            $(this).addClass('close');
        },
        function() {
            $('#login_form').fadeOut(600);
            $(this).removeClass('close');
        }
    ); //koniec funkcji toggle
}); //koniec funkcji ready
```

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. Zapewnia to wtyczka *Color*. Kopię tej wtyczki można znaleźć w przykładach dołączonych do tej książki, a animowanie kolorów zostało opisane na stronie 211. Najnowszą wersję tej wtyczki można znaleźć na stronie <https://github.com/jquery/jquery-color>. (Najprościej rzecz ujmując, wtyczka jest osobnym plikiem zawierającym kod JavaScript, dodający do biblioteki jQuery nowe możliwości funkcjonalne).

Aby skorzystać z tej funkcji, należy przekazać w jej wywołaniu literał obiektowy (patrz strona 158) zawierający listę właściwości CSS, które mają być animowane, oraz ich wartości docelowych. Założymy, 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`. Jednak biblioteka jQuery pozwala na stosowanie oryginalnych nazw właściwości CSS zawierających łączniki, przy czym muszą one być zapisywane w apostrofach, co pokazano na poniższym przykładzie:

```
{
  'font-size': '24px',
  'border-left-width': '2%'
}
```

Założymy, 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 201).

Istnieje także możliwość określania docelowych wartości animowanych właściwości w odniesieniu do ich wartości bieżących. Założymy 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);
});
```

Uwaga: Nie można animować właściwości `border-width`. Jest to właściwość skrótowa, pozwalającą za jednym zamachem określić szerokości obramowania elementu ze wszystkich czterech stron:

```
border-width: 2px 5px 2px 6px;
```

Gdybyśmy chcieli animować szerokość obramowania elementu, konieczne jest podanie pełnych nazw wszystkich czterech właściwości określających szerokość obramowania. Gdybyśmy na przykład chcieli animować obramowanie elementu, powiększając je do 20 pikseli, konieczne byłoby dodanie do literatu obiektowego właściwości dla wszystkich czterech krawędzi obramowania (`border-width-left`, `border-width-top` i tak dalej):

```
$('#element').animate(
{
    borderTop: 20px,
    borderRight: 20px,
    borderBottom: 20px,
    borderLeft: 20px
}, 1000 );
```

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 nieco 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 206, należało by 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. Stosując wtyczkę jQuery o nazwie *easing* (można ją pobrać ze strony <http://gsgd.co.uk/sandbox/jquery/easing/> oraz znaleźć w przykładach dołączonych do tej książki), możemy w bardzo dużym stopniu zmieniać wygląd tworzonych animacji. Przykładowo metoda `easeInBounce` sprawia, że szybko zmienia się zarówno szybkość, jak i kierunek animacji, co sprawia wrażenie, jakby animowany element odbijał się od czegoś.

Wskazówka: Wtyczka *easing* udostępnia wiele różnych metod określania tempa animacji, jednak w większości przypadków ich nazwy nie pozwolą określić faktycznego efektu działania. Aby na własne oczy przekonać się, jakie efekty wizualne zapewniają poszczególne metody, warto zajrzeć na stronę http://www.robertpenner.com/easing/easing_demo.html.

Aby skorzystać z wtyczki *easing* (będącej zewnętrznym plikiem JavaScript), trzeba dodać jej plik do strony tuż za kodem dołączającym bibliotekę jQuery. Po dołączeniu wtyczki można już będzie korzystać z udostępnianych przez nią metod określania tempa animacji (ich pełną listę można znaleźć na stronie <http://gsgd.co.uk/sandbox/jquery/easing/>). Założymy na przykład, że chcemy, by po kliknięciu elementu `div` umieszczony na stronie został powiększony. Aby dodatkowo animacja była bardziej interesująca, chcemy określić 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-1.6.3.min.js"></script>
2 <script src="js/jquery.easing.1.3.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 biblioteki jQuery oraz wtyczki *easing*. W wierszu 4. znajduje się wszechobecna funkcja `ready()` (patrz strona 182), 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 162), wewnętrz procedury obsługi zdarzeń używane jest wyrażenie `$(this)`, pozwalające odwołać się do elementu odpowiadającego na zdarzenie — w naszym przypadku jest to znacznik `<div>`. 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()` (patrz strona 184), by zastosować w jednym elemencie dwie różne metody określania tempa animacji.

Wykonywanie operacji po zakończeniu efektu

Czasami może się zdarzyć, że będziemy chcieli wykonać jakieś operacje po zakończeniu efektu. Założmy, 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 rozwiązać ten problem, do funkcji generującej efekt można przekazać dodatkową *funkcję zwrotną*. Zostanie ona wykonana dopiero po zakończeniu prezentowania efektu. Zazwyczaj jest ona przekazywana jako drugi argument większości funkcji generujących efekty wizualne jQuery (trzeci argument funkcji `fadeTo()`).

Założmy, 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 — literaliem obiektowym zawierającym animatione 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 literal obiektowy z właściwościami, czas trwania efektu oraz funkcję zwrotną. Przykładowo założmy, 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 wewnętrz 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 założmy, 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 149), 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);
```

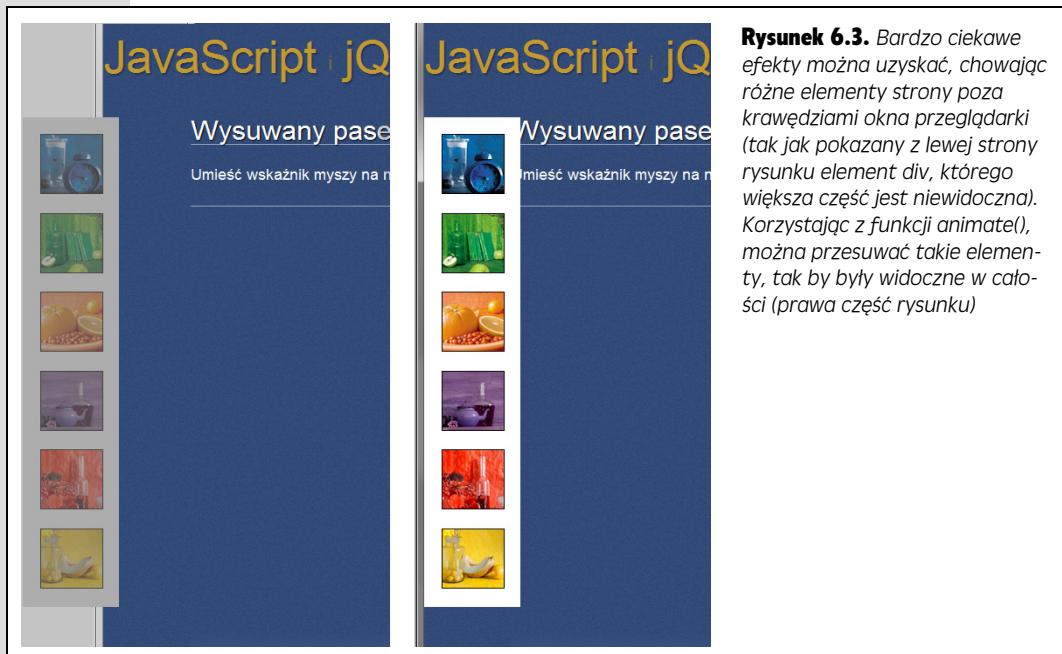
Uwaga: Więcej informacji na temat sposobu działania mechanizmu kolejkowania efektów wizualnych w bibliotece jQuery można znaleźć na stronie <http://api.jquery.com/jQuery.queue/>.

Dodatkową funkcją, która może się przydać podczas tworzenia kolejek efektów wizualnych, jest delay(). Powoduje ona oczekanie określonego okresu czasu (wyrażonego w milisekundach) przed rozpoczęciem kolejnego efektu zapisanego w kolejce. Założymy, że chcemy stopniowo wyświetlić obrazek, oczekać 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 pozyjonowany w sposób bezwzględny (więcej informacji na temat tego sposobu pozyjonowania można znaleźć w ramce na stronie 201), 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 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 43.

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 183) 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 mouseOver oraz mouseOut, opisanych na stronie 171).

3. Dodanie wywołania funkcji animate() w procedurze obsługi zdarzenia mouseover.

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 mouseout.

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 182). 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-1.6.3.min.js"></script>
<script src="../../_js/jquery.easing.1.3.js"></script>
<script src="../../_js/jquery.color.js"></script>
```

Wtyczki jQuery (o których dowieš się znacznie więcej w następnym rozdziale) są zwykłymi zewnętrznymi plikami JavaScript, rozszerzającymi możliwości biblioteki i 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, tak 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 160) — opisujących, co należy zrobić w momencie umieszczenia wskaźnika myszy w obszarze elementu oraz w momencie jego usunięcia. Zamiaszt 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
```

Wewnątrz wywołania funkcji `hover()` umieściliśmy dwie puste funkcje anonimowe, wewnętrznych których wpiszemy pozostały kod programu. Zaczniemy od pierwszej, określającej, co program ma zrobić, gdy użytkownik wskaże element myszą. Drugą funkcję anonimową uzupełnimy nieco później..

Wskazówka: Dobrym pomysłem jest możliwe 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 `alert('mouseOver')`, a w drugiej — `alert('mouseOut')`, a następnie wyświetlić stronę w przeglądarce. W takim przypadku zarówno po wskazaniu znacznika `<div>` myszą, jak i po usunięciu jej wskaźnika z obszaru elementu powinny zostać wyświetlone informacyjne okienka dialogowe. Jeśli w oknie przeglądarki nie zostaną wyświetlane żadne okienka informacyjne, będzie to oznaczało, że gdzieś popełniłeś błąd. W takim przypadku ponownie sprawdź kod strony bądź też wykonaj opisane na stronie 48 czynności pozwalające na odszukanie przyczyny błędu przy użyciu konsoli JavaScriptu.

5. Wewnątrz pierwszej funkcji anonimowej wpisz: `$(this).animate(); // koniec funkcji animate.`

Zgodnie z informacjami podanymi na stronie 162, 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 literal obiektowy określający właściwości CSS, które chcesz animować:

```
$(document).ready(function() {
    $('#dashboard').hover(
        function() {
            $(this).animate(
                {
                    left: '0',
                    backgroundColor: 'rgb(255,255,255)'
                }
            ); // koniec funkcji animate
        },
        function() {
            }
        ); // koniec funkcji hover
    ); // koniec funkcji ready
});
```

Pierwszym argumentem wywołania funkcji `animate()` jest literal obiektowy (patrz strona 158) 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 niebieskiego na biały. 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 wywoaniu 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(255,255,255)'
        },
        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 48. 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(110,138,195)'
  },
  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 niebieskie. 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 dłujo po tym, jak przestałeś poruszać wskaźnikiem myszy. Przyczyną tego problemu jest sposób, w jaki jQuery kolejkuje wykonywane animacje. Zgodnie z informacjami podanymi na stronie 211, 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. Rozwiążaniem 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 mouseover. 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(255,255,255)'
        },
        500,
        'easeInSine'
      ); // koniec funkcji animate
    },
    function() {
      $(this).stop().animate(
        {
          left: '-92px',
          backgroundColor: 'rgb(110,138,195)'
        },
        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`.

III

CZĘŚĆ

Dodawanie mechanizmów do stron WWW

- Rozdział 7. „Efekty związane z rysunkami”
- Rozdział 8. „Usprawnianie nawigacji”
- Rozdział 9. „Wzbogacanie formularzy”
- Rozdział 10. „Rozbudowa interfejsu stron WWW”

Efekty związane z rysunkami

Projektanci stron WWW używają rysunków do uatrakcyjniania projektów stron, dekorowania pasków nawigacyjnych, wyróżniania elementów strony — i pokazywania świata, jak dobrze bawili się w czasie ostatnich wakacji. Dodanie grafiki natychmiast zwiększa zainteresowanie użytkowników i poprawia wygląd witryny. Użycie dodatkowo języka JavaScript pozwala jeszcze bardziej uatrakcyjnić stronę przez dynamiczne dołączenie do niej rysunków, udostępnienie animowanej galerii fotografii lub wyświetlenie serii zdjęć w automatycznym pokazie slajdów. W tym rozdziale poznasz kilka sztuczek związanych z manipulowaniem grafiką i wyświetlaniem jej.

Zamiana rysunków

Prawdopodobnie najczęściej używanym efektem opartym na języku JavaScript jest *efekt rollover z użyciem rysunków*. Polega on na zastąpieniu jednego obrazka innym po umieszczeniu nad grafiką kurSORA myszy. Ta prosta technika od czasu pojawienia się JavaScriptu jest używana do tworzenia interaktywnych pasków nawigacyjnych, których przyciski zmieniają wygląd po najechaniu na nie kursorem.

Jednak od kilku lat coraz większa liczba projektantów używa do uzyskania tego efektu stylów CSS (patrz na przykład strona www.monkeyflash.com/css/image-rollover-navbar/). Nawet jeśli też do nich należysz, warto wiedzieć, 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. Założmy, że na stronie znajduje się obrazek o identyfikatorze `photo`. Przy użyciu biblioteki jQuery można dynamicznie zmienić wartość atrybutu `src` obrazka.

Założymy na przykład, że na stronie znajduje się obrazek o identyfikatorze `photo`. Kod HTML tworzący taki obrazek mógłby mieć następującą postać:

```

```

Aby podmienić obrazek wyświetlany w tym elemencie, wystarczy skorzystać z funkcji `attr()` (opisanej na stronie 159) 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 40) 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 odśylaczy znajdziesz w ramce na stronie 41).

Modyfikacja atrybutu `src` nie ma wpływu na pozostałe atrybuty znacznika ``. Na przykład jeśli 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. Aby zlikwidować problem zniekształcania, należy 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 ``:

```
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 literal obiektowy (patrz strona 158), zawierający nazwy atrybutów i ich nowe wartości. Wcześniejszys kod oparty na bibliotece jQuery można zapisać także w bardziej zwięzlej 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 i zostało dokładniej opisane na stronie 222. 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.

Wstępne wczytywanie rysunków

Zastępowanie rysunków za pomocą omówionych wcześniej technik ma jedną wadę. Kiedy skrypt zmieni w atrybutie `src` ścieżkę do pliku graficznego, przeglądarka musi pobrać nowy obrazek. Jeśli program zacznie pobierać plik dopiero po najechaniu kursemu myszy na rysunek, przed pojawiением 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. Na przykład kiedy użytkownik umieści kurSOR 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 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 var imgs = [];
5 for (var i = 0; i<preloadImages.length;i++) {
6   imgs[i] = new Image();
7   imgs[i].src = preloadImages[i];
8 }
```

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 74 dowiedziałeś się, że kod tablicy jest bardziej czytelny, jeśli każdy element znajduje się w odrębnym wierszu. Wiersz 4. tworzy nową, pustą tablicę — `imgs`. Skrypt zapisze w niej wstępnie wczytane obrazki. Wiersze od 5. do 8. to prosta pętla `for` języka JavaScript (patrz strona 113), uruchamiana raz dla każdego elementu z tablicy `preloadImages`. Wiersz 6. tworzy nowy obiekt rysunku, a wiersz 7. pobiera ścieżkę do pliku z tablicy `preloadImages`. Ta ostatnia operacja powoduje wczytanie rysunku.

Efekt rollover z użyciem rysunków

Efekt rollover z użyciem rysunków to po prostu zastępowanie obrazków (co opisano na stronie 219), uruchamiane po umieszczeniu kurSORA nad grafiką. Aby uzyskać ten efekt, należy przypisać kod zastępujący obrazki do zdarzenia `mouseover`. Założymy, że na stronie znajduje się rysunek o identyfikatorze `photo`. Kiedy użytkownik umieści kurSOR nad tym obrazkiem, skrypt ma wyświetlić nowy plik. Przy użyciu biblioteki jQuery można osiągnąć ten efekt w następujący sposób:

```

1 <script src="jquery-1.6.3.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 `mouseover` z funkcją, która zmienia atrybut `src` obrazka na ścieżkę do nowej fotografii.

W omawianym efekcie przeniesienie kurSORA poza rysunek powoduje zwykle przywrócenie pierwotnego obrazka. Dlatego trzeba dołączyć zdarzenie `mouseout`, aby ponownie wyświetlić pierwszy rysunek. Na stronie 183 dowiedziałeś się, że jQuery udostępnia zdarzenie `hover()`, które łączy zdarzenia `mouseover` i `mouseout`:

```
1 <script src="jquery-1.6.3.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>
```

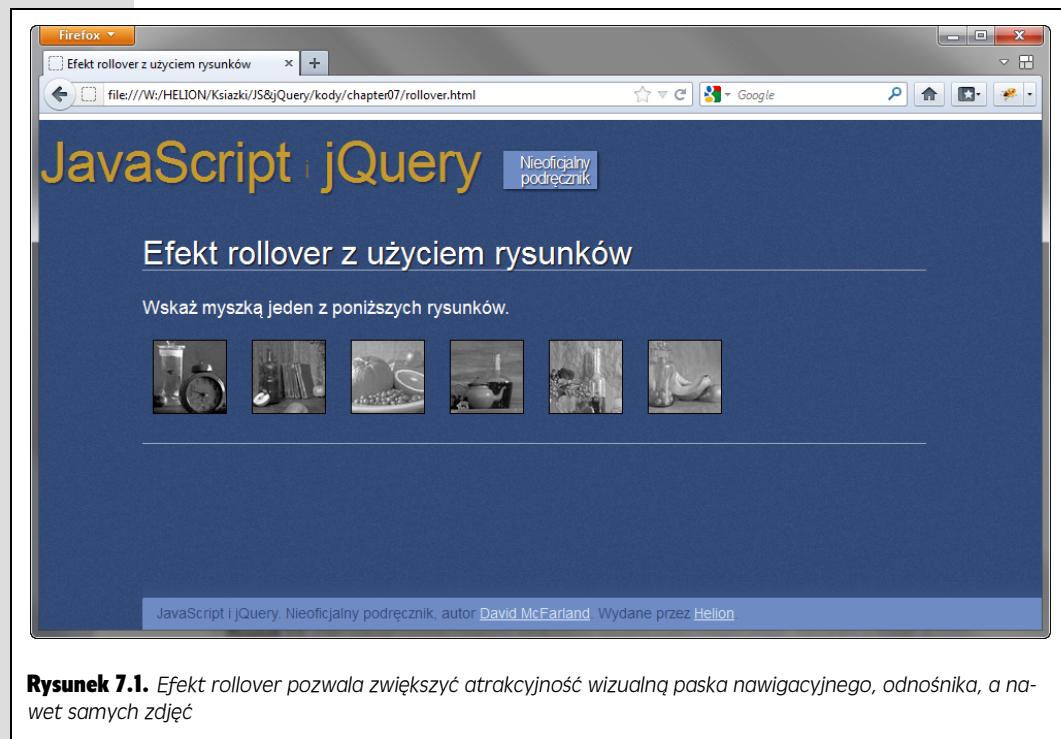
Funkcja `hover()` przyjmuje dwa argumenty. Pierwszym z nich jest funkcja anonimowa, która informuje przeglądarkę o tym, co ma zrobić, kiedy użytkownik umieści kurSOR nad rysunkiem. Drugi argument to funkcja uruchamiana po przeniesieniu kurSora w inne miejsce strony. Nowy kod tworzy zmiennej `oldSrc`, 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 kurSora nad znacznikiem `<h1>` może powodować zastąpienie rysunku nowym obrazkiem. Założymy, że docelowy plik jest taki sam jak we wcześniejszym przykładzie. W poprzednim skrypcie należy wprowadzić zmiany wyróżnione pogrubieniem:

```
1 <script src="jquery-1.6.3.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 kurSora 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

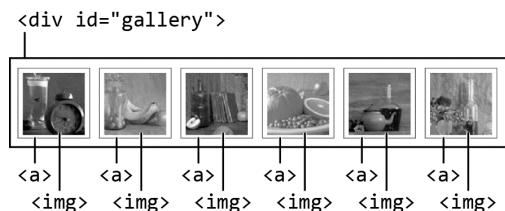
Omówienie zadania

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 kurSORA myszy, a następnie przywracać pierwotną wersję zdjęcia po przeniesieniu kurSORA 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()`.

Uwaga: Więcej informacji o pobieraniu przykładowych plików znajdziesz na stronie 43.



Rysunek 7.2. Podstawowa struktura kodu HTML użytego w tym przykładzie składa się ze znacznika `<div>`, który zawiera zbiór odnośników z rysunkami. Aby ułatwić wyświetlanie nowych zdjęć, ich nazwy utworzono na podstawie nazw pierwotnie widocznych obrazków



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 133). Pierwszy krok polega na pobraniu wszystkich rysunków ze znacznika `<div>` i dodaniu pętli za pomocą funkcji `each()` biblioteki jQuery (patrz strona 160).

2. Kliknij pusty wiersz w funkcji `$(document).ready()` i wpisz kod `$('#gallery img').each(function() {`

Selektor `#gallery img` pobiera wszystkie znaczniki `` z tagu 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 160). W tym momencie warto zamknąć dodaną funkcję anonimową. Dobrym rozwiązaniem jest dodanie zamkającego nawiasu kończącego treść funkcji anonimowej (a także każdej innej), zanim przystąpimy do pisania umieszczonego wewnętrz 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-1.6.3.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 162 dowiedziałeś się, że konstrukcja `$(this)` wskazuje element aktualnie przetwarzany w pętli. Oznacza to, że odpowiada ona po kolei każdemu znacznikowi ``. Funkcja `attr()` biblioteki jQuery (patrz strona 159) 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 poniższe instrukcje:

```
var preloadImage = new Image();
var imgExt = /(\.\w{3,4}\$/);
preloadImage.src = imgFile.replace(imgExt, '_h$1');
```

Na stronie 209 napisano, że w celu wstępnego pobrania rysunku trzeba najpierw utworzyć obiekt `Image`. 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 221) polega na utworzeniu 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 450) 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 463) do zastąpienia znalezionej 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 rozwiązań jest dość skomplikowane, ponieważ wymaga użycia podwzorca wyrażenia regularnego (pełny opis tej techniki znajdziesz w ramce na stronie 464), dlatego na razie nie musisz się przejmować, jeśli w pełni nie rozumiesz, jak ono działa.

Po wstępny 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-1.6.3.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   });
13 }); // Koniec funkcji each
14}); // Koniec funkcji ready
```

Funkcja `hover()` biblioteki jQuery umożliwia szybkie dodanie do elementu zdarzeń `mouseover` i `mouseout` (patrz strona 169). Należy przekazać do niej dwie funkcje. Pierwsza jest uruchamiana w momencie umieszczenia kurSORA nad elementem — tu zmienia rysunek na jego nową wersję. Drugą skrypt wywołuje, kiedy kurSOR znajdzie się w innym miejscu strony. Ta funkcja ta ponownie wyświetla pierwotny obrazek.

7. W pustym wierszu (wiersz 9. 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-1.6.3.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 przywracenia pierwotnej wartości atrybutu `src`. Zapisz stronę, wyświetl ją w przeglądarce i umieść kursor nad czarno-białymi zdjęciami, aby zobaczyć ich kolorowe wersje.

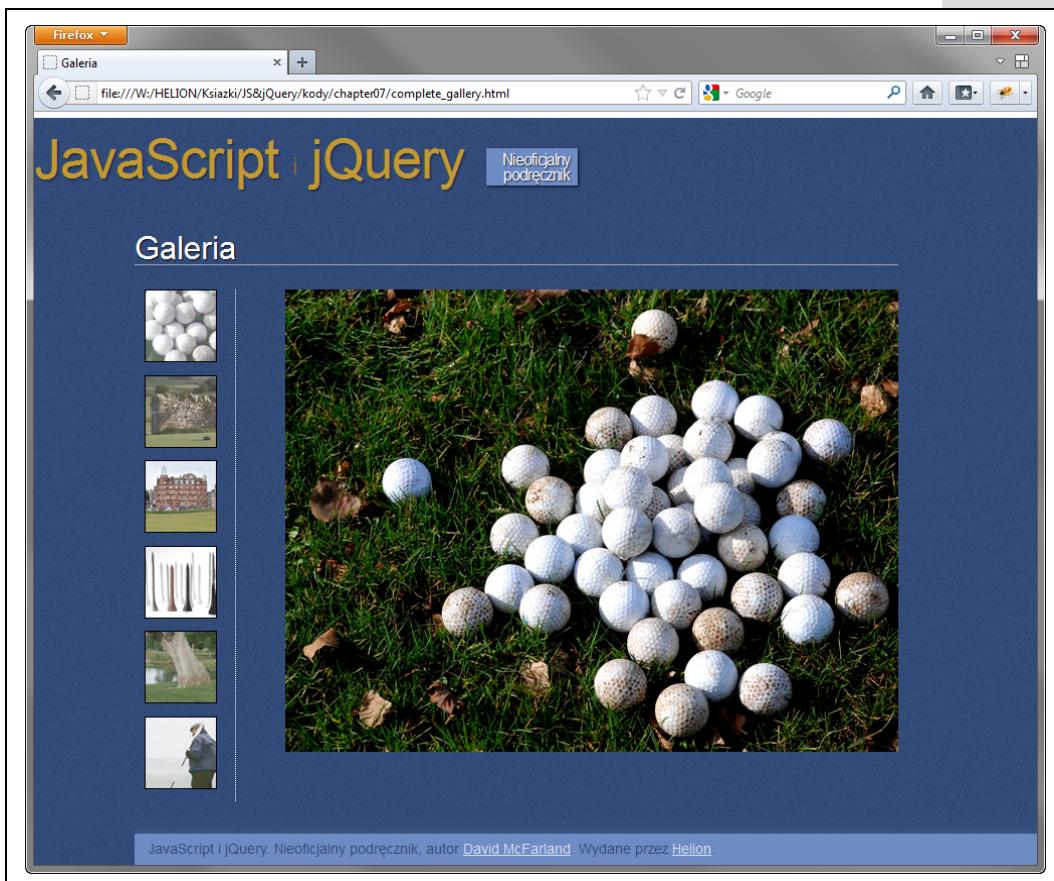
Przykład — galeria fotografii z efektami wizualnymi

W tym przykładzie przekształcisz wcześniejszy program na 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.

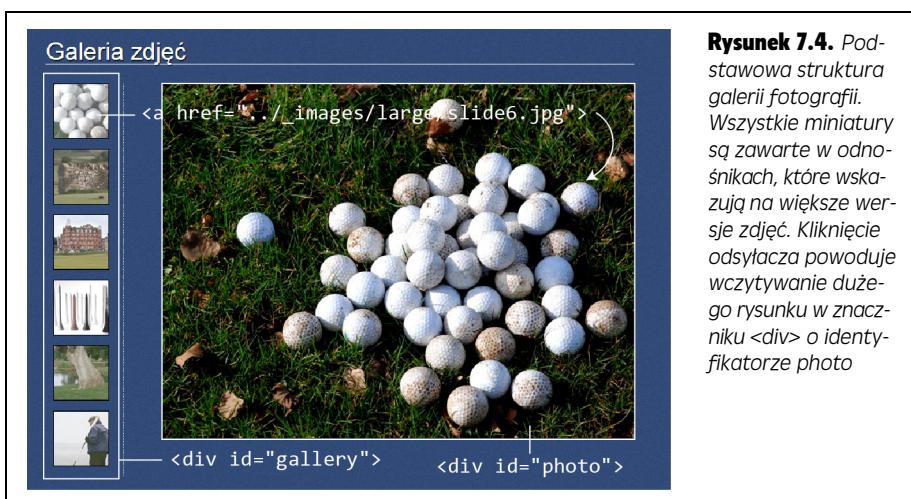
Omówienie zadania

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.

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 7.3. 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



Rysunek 7.4. Podstawowa struktura galerii fotografii. Wszystkie miniatury są zawarte w oznacznikach, które wskazują na większe wersje zdjęć. Kliknięcie odsyłacza powoduje wczytywanie dużego rysunku w znaczniku <div> 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 klikniętej 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 znalazła się znacznik `<div>` o identyfikatorze `photo` (patrz rysunek 7.4).

Uwaga: Informacje o pobieraniu przykładowych plików znajdziesz na stronie 43.

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 174). Skrypt przekazuje do zdarzenia `click` funkcję anonimową. Na stronie 185 dowiedziałeś się, że do funkcji uruchamianych w odpowiedzi na wystąpienie zdarzenia automatycznie prze-

kazywany jest obiekt zdarzenia. Tu do przechowywania tego obiektu służy zmieniona 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 186).

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 63. Wielu programistów dodaje wcięcia o szerokości dwóch odstępów lub jednej tabulacji.

5. Wciśnij klawisz *Enter* i wpisz instrukcję `var oldImage = $('#photo img');`.

Zmienna oldImage przechowuje element pobrany przez bibliotekę jQuery. Jest to znacznik , zawarty w znaczniku <div> o identyfikatorze photo (patrz rysunek 7.4). Teraz należy utworzyć tag, w którym znajdzie się nowe zdjęcie.

6. Ponownie wciśnij klawisz *Enter*, a następnie dodaj do skryptu kod `var newImage = $('');`.

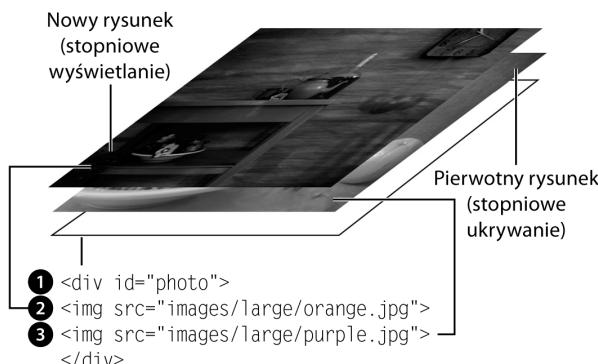
Jest to dość złożony wiersz. JQuery umożliwia pobieranie elementów umieszczonych w kodzie HTML strony, na przykład wyrażenie \$('img') zwraca wszystkie rysunki ze strony. Ponadto za pomocą obiektów jQuery można dodawać do strony nowe elementy. Na przykład wyrażenie \$('<p>Witaj</p>') tworzy nowy znacznik akapitu, zawierający słowo „Witaj”. Dodany wiersz tworzy nowy znacznik 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. Pierwszy fragment tekstu (w apostrofach) to `. Kiedy skrypt przekaże ten argument do obiektu jQuery — `$('')` — 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 198). 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 `<div>` o identyfikatorze `photo` (wiersz 7.). Funkcja `prepend()` (patrz strona 151) dodaje kod HTML na początku znacznika. Na tym etapie znacznik `<div>` 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 `<div>`, 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 obrázki musi zostać umiejscowiony w sposób wzgledny (atribut CSS `position o wartości relative`), aby poszczególne zdjęcia były rozmieszczane w wybranym miejscu strony

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()`. Przyjmuję ona dwa argumenty. Pierwszy określa czas trwania efektu (1000 milisekund, czyli 1 sekundę), a drugi to funkcja zwrotna (patrz strona 209). Usuwa ona znacznik `` danego rysunku, a skrypt wywołuje ją po stopniowym ukrywaniu zdjęcia.

Uwaga: Funkcję `remove()` opisano na stronie 439. Usuwa ona znacznik z modelu DOM, co powoduje skrócenie kodu HTML w pamięci przeglądarki i zwolnienie zasobów komputera. Jeśli pominiiesz tę operację, przy każdym kliknięciu miniatury skrypt doda nowy znacznik `` (krok 7.) i tylko ukryje poprzedni tag — nie usunie go. Dlatego na stronie znajdzie się mnóstwo ukrytych znaczników ``, 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 ``, 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:

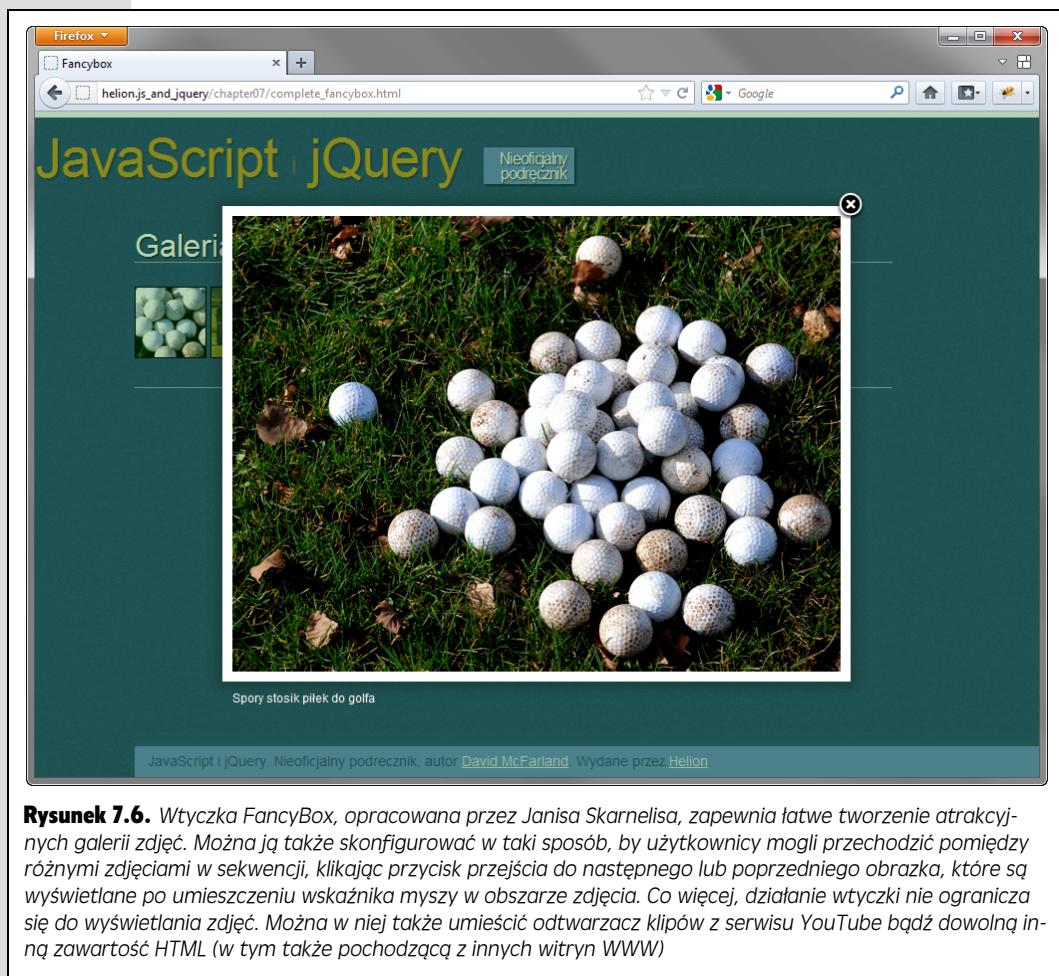
```
1  $('#gallery a').click(function(evt) {
2    evt.preventDefault();
3    var imgPath = $(this).attr('href');
4    var oldImage = $('#photo img');
5    var newImage = $('');
6    newImage.hide();
7    $('#photo').prepend(newImage);
8    newImage.fadeIn(1000);
9    oldImage.fadeOut(1000,function(){
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. Jednak jeśli 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 kurSORA, 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`.

Wzbogacona galeria z wtyczką FancyBox biblioteki jQuery

Wyświetlanie fotografii to tak powszechnie zadanie, że dostępne są dziesiątki sposobów na utworzenie galerii. Jedna z bardzo popularnych technik polega na przyciemnianiu strony i pokazywaniu większej wersji miniatury, unoszącej się nad oknem przeglądarki (patrz rysunek 7.6). Najbardziej znana wersja tej metody to napisany w języku JavaScript program Lightbox (<http://lokesthdkar.com/projects/lightbox2/>). Dostępnych jest wiele imitacji pierwowzoru, w tym także kilka utworzonych jako wtyczki biblioteki jQuery. W tym przykładzie skorzystamy z użytecznej wtyczki FancyBox (<http://fancybox.net/>), która jest łatwa w obsłudze i zapewnia duże możliwości dostosowywania. Za pomocą tylko jednego wiersza kodu FancyBox kreuje spektakularny sposób prezentacji obrazów, wyświetlanych jako portfolio, galeria lub pokaz slajdów.



Rysunek 7.6. Wtyczka FancyBox, opracowana przez Janisa Skarnelsa, zapewnia łatwe tworzenie atrakcyjnych galerii zdjęć. Można ją także skonfigurować w taki sposób, by użytkownicy mogli przechodzić pomiędzy różnymi zdjęciami w sekwencji, klikając przycisk przejścia do następnego lub poprzedniego obrazka, które są wyświetlane po umieszczeniu wskaźnika myszy w obszarze zdjęcia. Co więcej, działanie wtyczki nie ogranicza się do wyświetlania zdjęć. Można w niej także umieścić odtwarzacz klipów z serwisu YouTube bądź dowolną inną zawartość HTML (w tym także pochodzączą z innych witryn WWW)

Uwaga: Przed przejściem do dalszej części podrozdziału warto otworzyć plik *complete_fancybox.html* z katalogu *R07*. Zawiera on działającą wersję demonstracyjną wtyczki FancyBox. Przyjrzenie się jej ułatwi zrozumienie dalszego tekstu.

Podstawy

Wtyczka FancyBox jest bardzo łatwa w użyciu. Wystarczy umieścić na stronie odnośniki do wyświetlanych zdjęć, dołączyć pliki *.css* i *.js*, a następnie dodać wiersz kodu wywołujący wtyczkę.

1. Konfigurowanie strony z galerią.

Ten krok jest prosty. Wystarczy dodać odnośniki do większych rysunków, które chcesz wyświetlić. Te odsyłacze można powiązać z miniaturami, aby kliknięcie małego zdjęcia powodowało wyświetlenie jego dużej wersji (tak działała galeria z poprzedniego przykładu). Trzeba pamiętać, aby odnośniki te wskazywały na plik graficzny (*.png*, *.jpeg* lub *.gif*), a nie na stronę WWW. (Na stronie 259 doiesz się, w jaki sposób można użyć wtyczki FancyBox do wyświetlania treści innych niż zdjęcia, takich jak strony WWW lub klipy wideo prezentowane przy użyciu odtwarzaczy napisanych w technologii Flash).

Potrzebny jest też sposób na pobranie odnośników tylko z galerii (a nie z pozostałych części strony). Można w tym celu umieścić odpowiednie odsyłacze w znaczniku *<div>* o określonym identyfikatorze, na przykład *gallery*, i użyć selektora *'#gallery a'*. Inne rozwiązanie to dodanie nazwy klasy do potrzebnych odnośników (**) i pobranie elementów za pomocą selektora typu *'a.gallery'*. To ostatnie podejście jest wygodne, jeśli odsyłacze są porozrzucane po stronie i nie znajdują się w jednym znaczniku *<div>*.

Wskazówka: Aby dodać do zdjęcia tytuł, wystarczy dołączyć atrybut *title* do znacznika *<a>*, prowadzącego do większej wersji zdjęcia:

```
<a href="images/potato.jpg" title="Dorodny ziemniak">
```

2. Pobieranie plików wtyczki FancyBox i dołączanie ich do strony.

Pliki wtyczki znajdziesz na stronie <http://fancybox.net/>. Zostały także dodane do przykładów dołączonych do tej książki w katalogu *fancybox* umieszczonego w katalogu *R07*. Dobrym rozwiązaniem będzie skopiowanie całego katalogu *fancybox* na własną witrynę WWW (najlepiej do jej katalogu głównego). By wtyczka działała, potrzebujemy kilku plików, takich jak plik JavaScript, arkusz stylów CSS oraz parę plików graficznych:

- Plik JavaScript ma nazwę typu *jquery.fancybox-1.3.4.js*, gdzie „1.3.4” to numer wersji. W katalogu *fancybox* w przykładach do tej książki znajduje się także zminimalizowana (lub spakowana) wersja pliku o nazwie *jquery.fancybox-1.3.4.min.js*. Plik ten można umieścić w dowolnym miejscu witryny, jednak optymalnym rozwiązaniem będzie pozostawienie go w katalogu *fancybox* wraz z innymi plikami niezbędnymi do działania wtyczki.

- Plik z kodem CSS, *jquery.fancybox-1.3.4.css*, zawiera definicje stylów CSS określających różne aspekty wyglądu wtyczki, takie jak wygląd nagłówków, cieni, przycisków nawigacyjnych (poprzedni, następny) oraz przycisku zamkającego. Ponieważ plik ten odwołuje się do wielu plików graficznych (*fancybox.png*, *fancy_title_main.png* i tak dalej), najlepszym rozwiązaniem będzie pozostawienie go w tym samym katalogu, w którym są umieszczone obrazki (to jeden z głównych powodów przemawiających za przechowywaniem wszystkich plików związanych z wtyczką FancyBox w jednym katalogu).
- Wtyczka FancyBox wymaga ponadto kilku plików graficznych. Niektóre z nich są używane wyłącznie do rozwiązywania problemów występujących w przeglądarce Internet Explorer 6, natomiast pozostałe służą do dodawania elementów wizualnych, takich jak cień, ramka wokół tytułu czy też przyciski nawigacyjne. Więcej szczegółowych informacji na temat tych plików można znaleźć na stronie 238.

3. Dołączanie zewnętrznego arkusza stylów.

We wtyczce FancyBox użyto zaawansowanych stylów CSS do uzyskania ciemnej, półprzezroczystej nakładki i wyświetlania rysunku „nad” stroną. Plik z tymi stylami należy dołączyć w standardowy sposób:

```
<link href="fancybox/jquery.fancybox-1.3.4.css" rel="stylesheet" >
```

Większość programistów używających języka JavaScript umieszcza informacje o stylach przed kodem JavaScript. Niektóre skrypty wymagają danych o stylach do prawidłowego działania. Dotyczy to wielu wtyczek biblioteki jQuery, dlatego należy dołączać wszystkie arkusze stylów przed plikami i skryptami JavaScript.

4. Dołączanie plików JavaScript.

Wtyczka FancyBox wykorzystuje wiele mechanizmów biblioteki jQuery, co nie jest niespodzianką. Dlatego najpierw należy dołączyć plik jQuery (opis tej procedury znajdziesz na stronie 132). Plik JavaScript z wtyczką FancyBox (i inne pliki używające jQuery) trzeba dodać po dołączeniu biblioteki jQuery:

```
<script src="../_js/jquery-1.6.3.min.js"></script>
<script src="fancybox/jquery.fancybox-1.3.4.js"></script>
```

Oprócz tego wtyczka FancyBox może także korzystać z modyfikowanego tempa animacji (ang: *easing*) opisanego na stronie 207; animacja jest przez wtyczkę używana do kontroli sposobu wyświetlania większych obrazków. Jeśli zatem chcemy skorzystać z innych metod animacji, a nie domyślnie dostępnych w jQuery (takich jak *swing* oraz *linear*), konieczne jest także dodanie kolejnej wtyczki. Oto przykład:

```
<script src="../_js/jquery-1.6.3.min.js"></script>
<script src="../_js/jquery.easing.1.3.js"></script>
<script src="fancybox/jquery.fancybox-1.3.4.js"></script>
```

5. Dodawanie znacznika **<script>**, funkcji **ready()** biblioteki jQuery i wywołania wtyczki lightBox.

Możesz wierzyć lub nie, ale kroki od 1. do 4. to najtrudniejsza część całego procesu. Uruchomienie wtyczki FancyBox wymaga tylko jednego wiersza kodu JavaScript. Należy go oczywiście umieścić w funkcji **ready()** biblioteki jQuery (patrz strona 182), aby przeglądarka najpierw wczytała kod HTML i była gotowa do manipulowania modelem DOM:

```
<script>
$(document).ready(function() {
  $('#gallery a').fancybox();
});
</script>
```

Funkcję fancybox() trzeba zastosować tylko do odnośników prowadzących do wyświetlanych plików graficznych. Selektor biblioteki jQuery, na przykład \$('#gallery a'), informuje bibliotekę, których odnośników ma użyć. Tu są to znaczniki [wewnętrz elementu o identyfikatorze gallery](#). Jak wspomniano w kroku 1., trzeba przygotować kod HTML, aby użyć jQuery do pobrania odpowiednich odnośników użytych w efekcie FancyBox.

I to już wszystko. Teraz po kliknięciu każdego odnośnika z galerii nad stroną pojawi się półprzezroczyste tło, a w środku okna zobaczysz większą wersję rysunku.

Tworzenie galerii zdjęć

Zazwyczaj podczas dodawania efektu FancyBox do kolekcji odnośników każda miniatura jest traktowana jak niezależnie zdjęcie. Innymi słowy, jeśli użytkownik chce wyświetlić większą wersję zdjęcia, musi kliknąć, by je otworzyć, a następnie ponownie kliknąć (przycisk zamkający lub dowolne inne miejsce strony), by zamknąć to powiększenie, i w końcu ponownie kliknąć inną miniaturkę, żeby wyświetlić jej powiększoną wersję. I tak w kółko. To ciągłe otwieranie i zamknięcie kolejnych zdjęć może być męczące i czasochłonne.

Znacznie lepszym rozwiązaniem jest potraktowanie grupy miniaturek i odnośników jako jednej galerii. W tym przypadku użytkownik kliknie miniaturkę, by wyświetlić powiększone zdjęcie, a następnie może przechodzić pomiędzy kolejnymi powiększonymi zdjęciami przy użyciu przycisków nawigacyjnych (przejście do następnego oraz poprzedniego obrazka). Właśnie w taki sposób działa galeria FancyBox przedstawiona na rysunku 7.6; warto zwrócić uwagę na przycisk *Next* (zakreślony na rysunku).

Aby utworzyć galerię powiązanych ze sobą zdjęć, wystarczy dodać do znacznika [atribut rel i przypisać mu taką samą wartość dla wszystkich obrazków w galerii. Oto przykład:](#)

```
<a href="large_slide1.jpg" rel="gallery">
  
</a>
<a href="large_slide2.jpg" rel="gallery">
  
</a>
<a href="large_slide3.jpg" rel="gallery">
  
</a>
```

Po użyciu powyższego kodu HTML wywołanie funkcji fancybox() dla widocznych na przykładzie trzech odnośników sprawi, że większe obrazki zostaną potraktowane jak elementy jednej galerii; będzie można pomiędzy nimi przechodzić bez konieczności zamknięcia powiększonego zdjęcia i ponownego klikania miniaturki. (Przykład takiego rozwiązania został przedstawiony na stronie 244).

Personalizacja efektu FancyBox

Choć efekt FancyBox wygląda naprawdę atrakcyjnie, można go nieco zmodyfikować. Możesz zmienić wiele różnych aspektów wyglądu galerii, na przykład przyciski służące do zamykania okna i przechodzenia między zdjęciami. Możesz też zmodyfikować kolor i przezroczystość tła nakładanego na stronę lub kolor tła obramowania rysunku i pola z podpisem. Wprowadzenie niektórych zmian wymaga przekazywania dodatkowych opcji w różnych funkcjach wtyczki FancyBox, natomiast w innych przypadkach niezbędne zmiany należy wprowadzać bezpośrednio w arkuszu stylów CSS.

Opcje wtyczki FancyBox

Wtyczka FancyBox przyjmuje zestaw opcji, które wpływają na wygląd efektu. Aby go zmienić, należy przekazać do funkcji `lightbox()` literał obiektowy (patrz strona 158) z nazwami ustawianych opcji i ich wartościami. Aby na przykład zmienić kolor i poziom nieprzezroczystości tła, możesz utworzyć zmienną z pożdanymi ustawieniami i przekazać ją w następujący sposób:

```
$('#gallery a').fancybox({  
    overlayOpacity: .5,  
    overlayColor: '#F64',  
    transitionIn: 'elastic',  
    transitionOut: 'elastic',  
});
```

W tym przykładzie nakładka ma jasnoczerwony kolor (#F65), a jej nieprzezroczystość to 50% (.5). Oprócz tego zostanie zastosowany „elastyczny” ('elastic') efekt pojawiania się i znikania, który ma wpływ na to, jak będą wyświetlane duże wersje obrazów. W tym przypadku zastosowany efekt sprawi, że obrazy będą powiększane podczas wyświetlania i zmniejszane po naciśnięciu przycisku zamykającego bądź po kliknięciu dowolnego miejsca strony. (Zazwyczaj duże wersje obrazów są wyświetlane i ukrywane bez jakichkolwiek dodatkowych efektów wizualnych).

Wtyczka FancyBox obsługuje wiele różnych opcji (pełną ich listę znajdziesz na stronie <http://fancybox.net/api/>), a poniżej możesz zapoznać się z opisem najbardziej przydatnych ustawień.

- Opcja **overlayColor** określa kolor tła nakładanego na stronę przy wyświetlaniu obrazu. Kolor należy podawać w notacji szesnastkowej (na przykład #FF0033). Jeśli nie podamy koloru jawnie, domyślnie zostanie zastosowany kolor szary, konkretnie #666. Opcję tę należy określić w następujący sposób:

```
overlayBgColor: '#FF0033'
```

- Opcja **overlayOpacity** określa nieprzezroczystość tła. Możesz w ten sposób ustalić, jak dobrze będzie widać stronę pod nakładką. Należy podać liczbę z przedziału od 0 do 1. Wartość .5 oznacza nieprzezroczystość na poziomie 50%. Jeśli chcesz, aby tło było nieprzejrzyste i podczas prezentowania obrazka przykrywało całą stronę, ustaw tę opcję na 1. Jeśli wartość tej opcji nie zostanie jawnie określona, FancyBox zastosuje wartość domyślną — 30% (.3). Opcję tę należy określić w następujący sposób:

```
overlayOpacity: .5
```

- Opcja **padding** określa wielkość obszaru wokół obrazu; tworzy wokoło niego widoczną ramkę. Zazwyczaj wtyczka określa jej wielkość na 10 pikseli, jednak można ją dowolnie zmienić. Użycie wartości 0 sprawi, że obramowanie całkowicie zniknie (informacje o tym, w jaki sposób można określić kolor tego obramowania, podałem na stronie 243). Przy ustalaniu wartości tej opcji wystarczy podać liczbę (FancyBox zakłada, że wartość ta jest wyrażona w pikselach, a zatem dodawanie znaków px, konieczne podczas określania wymiarów w arkuszach stylów, w tym przypadku nie jest potrzebne):

```
padding: .5
```

- Opcja **changeSpeed** — podczas przechodzenia pomiędzy poszczególnymi obrazami wyświetlonymi przez wtyczkę FancyBox ramka, w której są one prezentowane, zmienia swoją wielkość i dostosowuje się do wymiaru aktualnie prezentowanego obrazu. Przy użyciu tej opcji można kontrolować szybkość zmiany wymiarów ramki. Domyślnie opcja ta przyjmuje wartość 300, co oznacza 300 milisekund, czyli nieco mniej niż pół sekundy. Wartość tej opcji można określić w następujący sposób:

```
changeSpeed : 500
```

- Opcje **transitionIn** oraz **transitionOut** określają, w jaki sposób będą wyświetlane na ekranie większe obrazy. Użycie wartości `fade` sprawi, że będą się pojawiały stopniowo. W razie użycia wartości `none` obraz będzie się nagle pojawiał na ekranie bez żadnego dodatkowego efektu wizualnego. Najbardziej interesująca wizualnie jest jednak wartość `elastic` — sprawia ona, że obraz będzie powiększany aż do osiągnięcia pełnych wymiarów. W połączeniu z opcjami „easing” (opisanymi w kolejnym punkcie) można stworzyć bardzo dynamiczne (i potencjalnie także bardzo denerwujące) efekty wizualne. Pierwsza z tych opcji, `transitionIn`, określa, w jaki sposób większe obrazy będą się pojawiały na ekranie; natomiast druga, `transitionOut`, w jaki sposób będą one znikawały. Każdej z nich można przypisać inną wartość:

```
transitionIn : 'elastic',  
transitionOut : 'none'
```

- Opcje **easingIn** oraz **easingOut** są stosowane, gdy opcji `transitionIn` lub `transitionOut` zostanie przypisana wartość `'elastic'` (patrz poprzedni punkt listy). Określają one metodę zmieniania tempa animacji, na przykład domyślnie dostępne w jQuery metody `swing` lub `linear`, bądź też, w razie dołączenia specjalnej wtyczki jQuery (patrz strona 207), dowolne inne, udostępniane przez nią metody:

```
easingIn : 'easeInBounce',  
easingOut : 'easeOutSine'
```

- Opcja **titlePosition**. Zazwyczaj przy dodawaniu podpisów do prezentowanych obrazów (patrz uwaga na stronie 243) FancyBox umieszcza je wewnątrz graficznej ramki (o kształcie rombu). Grafikę tę można zmienić (patrz strona 241) bądź też całkowicie z niej zrezygnować — wystarczy przypisać opcji `titlePosition` wartość `outside` (w tym przypadku podpis będzie wyświetlany poniżej ramki prezentującej obraz), `inside` (podpis prezentowany jest wewnątrz ramki zawierającej obraz) bądź `over` (w tym przypadku podpis zostanie umieszczony wzdłuż dolnej krawędzi obrazu). Wartość tej opcji można określić w następujący sposób:

```
titlePosition : 'outside'
```

- Opcja **cyclic**. Kiedy zostanie wyświetlony ostatni obrazek w galerii, dostępny będzie jedynie przycisk przejścia do poprzedniego obrazka, natomiast w momencie wyświetlenia pierwszego obrazu — przycisk przejścia do następnego. Jednak po przypisaniu opcji `cyclic` wartości `true` — `cyclic: true` — użytkownik po wyświetleniu ostatniego obrazu w galerii będzie mógł przeskoczyć do pierwszego, klikając przycisk następnego obrazka, oraz przejść z pierwszego do ostatniego, korzystając z przycisku poprzedniego obrazu. Innymi słowy, opcja ta pozwala cyklicznie przechodzić między kolejnymi obrazami tworzącymi galerię.

```
cyclic : true
```

Oto przykład modyfikowania podanych opcji. Założymy, że chcemy użyć wtyczki FancyBox, by przekształcić grupę odnośników na galerię obrazków. Chcemy, by użytkownik mógł bez końca przechodzić pomiędzy kolejnymi obrazami, czyli gdy znajdzie się na ostatnim i kliknie przycisk przejścia do następnego obrazu, przejdzie na początek galerii. Dodatkowo chcemy ukryć ramkę domyślnie wyświetlana wokół obrazów, by były stopniowo powiększane i zmniejszane podczas wyświetlania i ukrywania, oraz wyświetlać popisy poniżej obrazów. To wszystko możemy zrobić przy użyciu następującego fragmentu kodu:

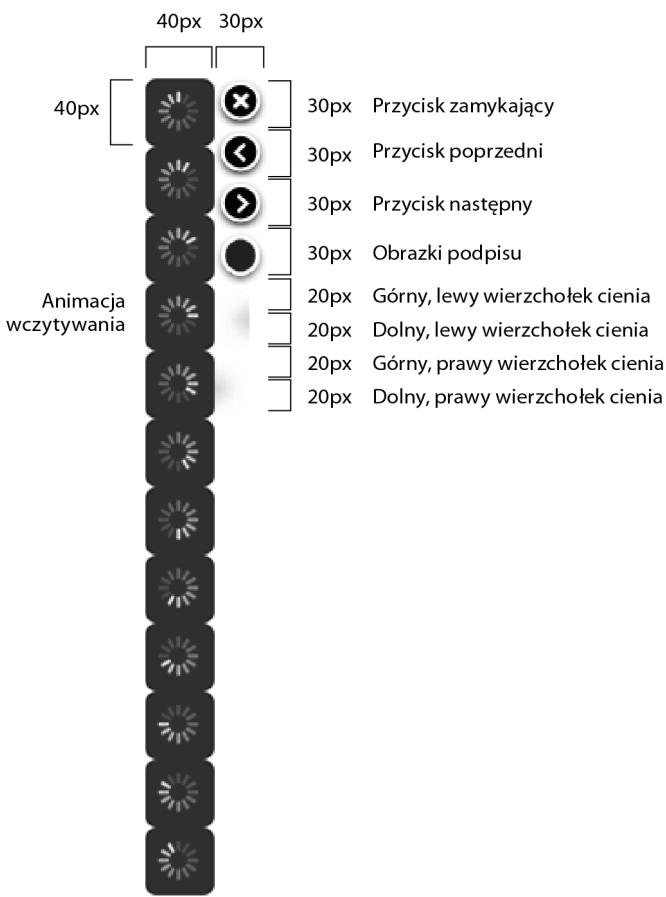
```
$('#gallery a').fancybox ({  
    cyclic : true,  
    padding : 0,  
    transitionIn : 'elastic',  
    transitionOut : 'elastic',  
    titlePosition: 'outside'  
});
```

Określanie wyglądu prezentacji

Wygląd prezentacji tworzonych za pomocą wtyczki FancyBox definiowany jest przy użyciu elementów graficznych oraz kaskadowych arkuszy stylów. Przycisk do zamazywania zdjęć, przyciski nawigacyjne oraz cień wyświetlany pod zdjęciami twozone są z wykorzystaniem obrazków, natomiast ich położenie i wielkość — przy użyciu CSS. Animowana grafika symbolizująca pobieranie jest wyświetlana także podczas wczytywania powiększonej wersji obrazka. Wszystkie te elementy można dostosowywać do własnych potrzeb, jednak najpierw należy zrozumieć, w jaki sposób wtyczka FancyBox organizuje swoje zasoby graficzne.

Przede wszystkim zasoby graficzne używane we wszystkich przeglądarkach, z wyjątkiem Internet Explorera 6, są przechowywane w jednym pliku — `fancybox.png` (patrz rysunek 7.7). FancyBox korzysta z techniki określonej nazwą sprajtów CSS (ang. *CSS sprites*), która pozwala przechowywać wiele obrazków w jednym pliku, jednak (dzięki zastosowaniu kilku właściwości stylów CSS, w tym właściwości `background-image`) wyświetlać wyłącznie wybrany fragment obrazka. Aby na przykład wyświetlić przycisk zamazywający, FancyBox wczytuje plik `fancybox.png` i używa go jako obrazu tła elementu odnośnika umieszczonego na stronie, lecz jednocześnie określa wymiary tego elementu na 30 pikseli szerokości i wysokości i przesuwa go z wykorzystaniem właściwości `background-position` tak, by widoczny był jedynie górny, prawy fragment dużego obrazka. Rozwiążanie to bazuje na założeniu, że podczas wczytywania tylko jednego pliku, `fancybox.png`, zamiast kilku (po jednym dla poszczególnych elementów sterujących aplikacją) poszczególne strony witryny będą

fancybox.png



Rysunek 7.7. Nie trzeba używać opcji wtyczki FancyBox, by zmodyfikować wygląd tworzonej prezentacji. Można ją także lepiej dostosowywać do własnych potrzeb, modyfikując dostarczone wraz z nią pliki graficzne oraz arkusze stylów CSS

się wyświetlały szybciej, a ona sama będzie sprawniej reagować na poczynania użytkownika. (Więcej informacji na temat tej techniki można znaleźć na stronie <http://css-tricks.com/css-sprites/>).

FancyBox, oprócz obrazków, korzysta także ze stylów umieszczonych w zewnętrznym arkuszu, które są używane do określania wyglądu różnych elementów prezentacji. Poniżej zamieściłem opis najczęściej używanych elementów prezentacji oraz sposobów ustalania ich wyglądu.

- **Przycisk zamykający.** FancyBox korzysta z graficznego przycisku zamykającego, wyświetlany w prawym, górnym rogu prezentacji. Przycisk ten ma po 30 pikseli szerokości oraz wysokość i jest umieszczony w prawym, górnym rogu obrazka zapisanego w pliku `fancybox.png`. Jego wygląd można modyfikować na kilka sposobów. Jeśli nie odpowiadają nam jego wymiary, wystarczy otworzyć w programie graficznym plik `fancybox.png` i zastąpić graficzny przycisk umieszczony w jego prawym, górnym wierzchołku własnoręcznie utworzonym zamiennikiem.

Jeszcze większą kontrolę nad wyglądem przycisku zamkającego można uzyskać, modyfikując styl `#fancybox-close` zdefiniowany w arkuszu stylów używanym przez wtyczkę FancyBox [o nazwie `jquery.fancybox-1.3.4.css`]. Za pomocą tego stylu można kontrolować położenie przycisku (zmieniając właściwości `top` oraz `right`) czy jego wysokość (modyfikując właściwości `height` oraz `width`), można nawet wskazać inny plik graficzny określający jego wygląd (w tym celu należy zmienić wartość właściwości `background`).

Uwaga o IE6: W celu rozwiązywania problemów występujących w przeglądarce Internet Explorer 6 we wtyczce FancyBox użyto specjalnych stylów CSS. W tych stylach nie jest stosowany plik `fancybox.png`, a wygląd różnych elementów prezentacji jest określany przy użyciu innych plików graficznych. Internet Explorer 6 jest już praktycznie przeglądarką nieużywaną, jeśli jednak wciąż go uwzględniasz i chcesz zadbać o to, w jaki sposób prezentacja FancyBox w niej wygląda, będziesz mógł to zrobić, wprowadzając odpowiednie modyfikacje do plików `fancy_close.png` (przycisk zamkający prezentację), `fancy_nav_left.png` (przycisk przejścia do poprzedniego obrazu), `fancy_title_right.png` (przycisk przejścia do następnego obrazu), `fancy_loading.png` (animowana ikona wczytywania obrazów), `fancy_title_main.png` (środkowy obszar podpisu obrazu), `fancy_title_right.png` (prawa krawędź obszaru podpisu) oraz `fancy_title_left.png` (lewa krawędź obszaru podpisu). Równie dobrze jednak możesz się nie trudzić i całkowicie zignorować przeglądarkę IE6.

- **Przyciski nawigacyjne.** Podobnie jak opisany powyżej przycisk zamkajający, także przyciski nawigacyjne (przejścia do poprzedniego i następnego obrazka) są obrazkami umieszczonymi w pliku `fancybox.png` (patrz rysunek 7.7). Plik ten można zmodyfikować (należy przy tym pamiętać, by nie przekroczyć wymiarów 30×30 pikseli). Można także zmienić wielkość przycisku albo użyć w nim innego pliku graficznego — wystarczy w tym celu zmienić style obu tych przycisków: style grupowe (zawierające selektory `#fancybox-left-ico` oraz `#fancybox-right-ico`, odnoszące się do obu przycisków i określające takie ustawienia jak ich wysokość i szerokość) oraz style indywidualne (`#fancybox-left-ico`, odnoszący się do przycisku przejścia od poprzedniego obrazu, oraz `#fancybox-right-ico`, odnoszący się do przycisku przejścia do obrazu następnego) określające obrazki używane w obu tych przyciskach.
- **Grafika „wczytywania”.** Kiedy użytkownik kliknie miniaturkę, by wyświetlić powiększoną wersję obrazka, przeglądarka musi wczytać plik. Podczas tej operacji na ekranie wyświetlany jest animowany obrazek informujący, że trwa pobieranie powiększonej wersji miniaturki. Obrazek stanowi w rzeczywistości sekwencję „ramek” umieszczonych w pliku `fancybox.png` (patrz rysunek 7.7). Każda z 12 ramek ma 40 pikseli wysokości i szerokości. Aby zmienić ten obrazek, trzeba otworzyć plik `fancybox.png` i narysować w nim własne wersje 12 klatek animacji. Reguła stylu CSS, która określa, jaki obrazek należy wyświetlić oraz jaka będzie jego szerokość, zawiera selektor `#fancybox-loading`. Jeśli jednak chcemy zmodyfikować położenie przycisku lub wymiary wyświetlonego w nim obrazka, konieczne będzie także wprowadzenie zmian w stylu `#fancybox-loading`. Może to wymagać sporego nakładu pracy. Ponieważ standarowa grafika wyświetlana podczas wczytywania wygląda doskonale, najlepiej nic tutaj nie zmieniać.

- **Cień.** Cienie wyświetlane wokół obrazka prezentowanego przez FancyBox są generowane przy wykorzystaniu kilku plików graficznych, w tym także kilku różnych fragmentów obrazka *fancybox.png*. Ich modyfikacja jest zadaniem trudnym i dlatego lepiej jej unikać. Jeśli elementy graficzne używane we wszystkich kontrolkach opisywanych we wcześniejszych punktach tej listy zostały umieszczone w osobnych plikach, a także wprowadziłeś odpowiednie zmiany w arkuszu stylów, by odwoływał się do tych plików, to i tak nie powinieneś usuwać pliku *fancybox.png* — gdyż w nim znajdują się fragmenty określające postać wierzchołków cienia.
- **Kolor obramowania wokół obrazków wyświetlanych w okienku prezentacji.** Zazwyczaj wokół powiększonych obrazków prezentowanych przez FancyBox wyświetlane jest białe obramowanie. Zgodnie z informacjami podanymi w poprzedniej części rozdziału, szerokość tego obramowania można zmieniać poprzez przekazanie opcji *padding* w wywołaniu funkcji *fancybox()*. Aby jednak zmienić kolor tego obramowania, potrzebne jest wprowadzenie odpowiednich modyfikacji w arkuszy stylów. W tym celu należy odszukać styl z selektorem *#fancybox-content* i zmienić jego właściwość *border-color* z *#FFF* na dowolny inny, wybrany kolor.
- **Kolor tła obrazków.** W galerii, w której użytkownik może kliknąć przyciski w celu poruszania się pomiędzy kolejnymi obrazkami, w czasie gdy dotychczasowy obrazek znika, a pojawia się następny, wyświetlany jest kolor tła. Kolor ten określany jest za pomocą właściwości *background* stylu z selektorem *#fancybox-outer*. Dobrym pomysłem jest zastosowanie tego samego koloru, w którym jest wyświetlone obramowanie wokół obrazów (opisane w poprzednim punkcie).
- **Podpisy.** Wprowadzając odpowiednie zmiany w arkuszu stylów, można modyfikować kolor, czcionkę oraz wielkość podpisów. Czcionkę, jaką są wyświetlane podpisy oraz jej kolor, określa styl *#fancybox-title*; natomiast style *.fancybox-title-inside*, *.fancybox-title-outside* oraz *.fancybox-over* ustalają wygląd podpisu w zależności od ustawienia opcji *titlePosition* (opisanego na stronie 239). Jeśli na przykład opcji tej zostanie przypisana wartość *inside*, podpis będzie wyświetlany wewnątrz ramki obrazka. Po zmianie koloru obramowania (opisanej w poprzednim punkcie listy) należy także zmienić właściwość *background* określoną w stylu z selektorem *.fancybox-title-inside*, tak aby oba kolory były takie same. Jeśli opcja *titlePosition* nie została podana, wtyczka FancyBox będzie wyświetlać obramowanie podpisu przy użyciu fragmentów obrazu *fancybox.png*. Konkretnie rzecz biorąc, używane są trzy fragmenty tego obrazka (jeden dla lewej krawędzi obramowania, drugi dla części środkowej i trzeci dla prawej krawędzi). Aby zmodyfikować używane obrazy, należy wprowadzić zmiany w stylach o następujących selektorach: *#fancybox-title-float-left*, *#fancybox-title-float-main* oraz *#fancybox-titlefloat-right*.

Wskazówka: Wtyczka FancyBox pozwala także na dostosowanie kodu HTML używanego w celu wyświetlania podpisu. Można to robić na wiele sposobów, używając w tym celu własnej funkcji.Więcej informacji i porad dotyczących stosowania wtyczki FancyBox można znaleźć na stronie <http://fancybox.net/blog>.

Przykład — galeria fotografii oparta na wtyczce FancyBox

Choć wtyczka FancyBox jest bardzo łatwa w użyciu, warto zapoznać się z przykładem ilustrującym krok po kroku, jak ją zastosować. W tym podrozdziale użyjesz strony ze zbiorem miniatur i przekształcisz ją na wymyślny pokaz slajdów oparty na wtyczce FancyBox.

Uwaga: Informacje o pobieraniu przykładowych plików znajdziesz na stronie 43.

1. Otwórz w edytorze tekstu plik *fancybox.html* z katalogu R07.

Plik zawiera prostą grupę miniaturek. Każda z nich znajduje się w odnośniku prowadzącym do większej wersji zdjęcia, a cały zbiór zawarto w znaczniku <div> o identyfikatorze gallery.

Pierwszy krok polega na dołączeniu pliku CSS używanego przez wtyczkę FancyBox.

2. W sekcji <head> dokumentu znajdź pusty wiersz pod znacznikiem <link>, który dołącza arkusz stylów *cite.css*, a nad pierwszym znacznikiem <script>. Wpisz w nim następujący kod:

```
<link href="fancybox/jquery.fancybox-1.3.4.css" rel="stylesheet">
```

Plik *jquery.fancybox-1.3.4.css* zawiera wszystkie style używane do formatowania różnych części interfejsu FancyBox, w tym także do określania rozmieszczenia przycisków oraz wyglądu tekstu podpisów wyświetlanych pod obrazkami. Po dołączeniu pliku arkusza stylów należy dodać plik JavaScript. Warto zwrócić uwagę, że na stronie są już pliki biblioteki jQuery oraz wtyczki z metodami do zmiany tempa animacji (więcej informacji na temat tego ostatniego można znaleźć na stronie 207).

3. W pustym wierszu bezpośrednio pod znacznikiem <script> dołączającym plik *jquery.easing.1.3.js* dodaj poniższy kod:

```
<script src="fancybox/jquery.fancybox-1.3.4.min.js"></script>
```

Ten plik JavaScript jest umieszczony w katalogu *fancybox*, w katalogu R07. Ponieważ wtyczka FancyBox wymaga całego zbioru dodatkowych plików — skryptu JavaScript, arkusza stylów CSS oraz kilku obrazków — najlepszym rozwiązaniem jest umieszczenie wszystkich w jednym miejscu. Dzięki temu, jeśli zechcesz użyć wtyczki FancyBox na własnej witrynie, wystarczy, że skopiujesz cały katalog *fancybox* z przykładów dołączonych do tej książki.

Na stronie znajduje się też inny znacznik <script>, zawierający funkcję ready() biblioteki jQuery oraz kod obsługujący wstępne wczytywanie i efekt rollover dodane w pierwszym przykładzie z tego rozdziału. Teraz wystarczy dodać funkcję fancybox() — i efekt będzie gotowy.

4. Kliknij pusty wiersz bezpośrednio pod kodem `$(document).ready(function()` i wpisz następującą instrukcję:

```
$('.#gallery a').fancybox();
```

Wszystkie odnośniki, które prowadzą do większych rysunków, znajdują się w znaczniku `<div>` o identyfikatorze `gallery`, dlatego do pobrania potrzebnych odsyłaczy należy użyć wyrażenia `$('#gallery a')`. Funkcja `.lightbox()` dodaje efekt FancyBox do strony.

Wierz lub nie, ale już skończyłeś! Zapisz stronę i wyświetl ją w przeglądarce. Kliknij jedną z miniatur, aby zobaczyć efekt. Teraz wiesz już, dlaczego wtyczki są tak wartościowe. Prawie nie musisz pisać kodu, aby uzyskać fantastyczne efekty!

Kliknij przycisk zamknięty lub kliknij w dowolnym miejscu strony, by zamknąć powiększone zdjęcie. Kliknij inną miniaturę, żeby wyświetlić powiększenie kolejnego obrazka. Problem z takim rozwiązaniem polega na tym, że każda miniaturka jest traktowana niezależnie. Innymi słowy, aby wyświetlić powiększenie innego obrazka, trzeba zamknąć aktualnie widoczne powiększenie i kliknąć inną miniaturkę. Byłoby znacznie lepiej, gdyby po wyświetleniu powiększenia obrazka można było wygodnie nawigować i wyświetlać powiększenia innych miniatur. Można to zrobić, wprowadzając prostą modyfikację kodu HTML.

5. **Odszukaj pierwszy odnośnik umieszczony wewnętrz znacznika `<div>` — `` — i dodaj do niego atrybut `rel="gallery"`, tak by wyglądał w następujący sposób:**

```
<a href=".../_images/large/slidel.jpg" rel="gallery">
```

Wartość, jaką podasz w atrybucie `rel`, w tym przypadku "gallery", nie ma żadnego znaczenia. Ważne jest natomiast to, by wszystkie obrazki, które mają się znaleźć w jednej grupie, miały tę samą wartość.

6. **Dodaj atrybut `rel="gallery"` do wszystkich pięciu pozostałych odnośników umieszczonych wewnętrz znacznika `<div>` galerii. Zapisz stronę i wyświetl ją w przeglądarce.**

Teraz, po kliknięciu miniaturki możesz umieścić wskaźnik myszy po prawej stronie okienka wyświetlającego powiększoną wersję obrazka, a pojawi się przycisk przejścia do następnego obrazu. Po jego kliknięciu przejdziesz do następnego zdjęcia w galerii. Z kolei umieszczenie wskaźnika myszy po lewej stronie powiększonego obrazka spowoduje wyświetlenie przycisku przejścia do obrazu poprzedniego.

Brakuje jeszcze jednego elementu — podpisów pod wyświetlonymi zdjęciami. Aby je dodać, nie trzeba pisać żadnego kodu JavaScirpt, wystarczy do każdego znacznika `<a>` dodać atrybut `title`.

7. **Ponownie odszukaj pierwszy odnośnik — `` — i dodaj do niego atrybut `title="Masa piłek do golfa"`, tak by wyglądał następująco:**

```
<a href=".../_images/large/slidel.jpg" rel="gallery">
```

Zapisz plik i wyświetl go w przeglądarce. Kliknij pierwszą miniaturkę z lewej strony — i proszę, pojawił się podpis pod obrazkiem. Dodaj podobne atrybuty `title` do wszystkich pozostałych znaczników `<a>` umieszczonych wewnętrz elementu `<div>` galerii. Teraz zmodyfikujemy kilka domyślnych ustawień FancyBox, by odmienić wygląd prezentacji.

8. Zmień kod, który wpisałeś w etapie 4., dodając pomiędzy nawiasami wywołania funkcji fancybox() literał obiektowy. Nowa wersja kodu powinna wyglądać tak:

```
$('#gallery a').fancybox({  
    overlayColor : '#060',  
    overlayOpacity : .3  
});
```

Ten fragment przekazuje literał obiektowy (patrz strona 158) do funkcji fancybox(). Literaly obiektowe składają się z nazwy właściwości, dwukropka i wartości. Przykładowo overlayColor to nazwa opcji wtyczki FancyBox (patrz strona 238), a kod ustawia jej wartość na '#060'. Ta konkretna opcja zmienia domyślny kolor tła wyświetlanego między zawartością strony i okienkiem prezentującym powiększone wersje obrazków. Z kolei opcja overlayOpacity określa poziom nieprzezroczystości tej warstwy — przypisanie jej wartości .3 oznacza, że będzie w 30% nieprzezroczysta.

Uwaga: Literaly obiektowe mają specyficzny wygląd i dziwne reguły. Pamiętaj o dodaniu przecinka po każdej parze właściwość – wartość oprócz ostatniej. Przykładowo ostatni wiersz w powyższym fragmencie, overlayOpacity: .3, nie może kończyć się przecinkiem. Przecinek ten trzeba będzie jednak dodać, jeśli będziemy podawać kolejne opcje. Więcej informacji o literalach obiektowych znajdziesz na stronie 158.

9. Zapisz stronę i wyświetl ją w przeglądarce.

Teraz, kiedy klikniesz miniaturkę, pojawi się zielonkawa warstwa przykrywająca pozostałą zawartość strony. Kolejną czynnością będzie modyfikacja sposobu wyświetlania obrazków.

10. Jeszcze raz zmodyfikuj kod, dodając do niego kilka nowych opcji. Nowa wersja powinna wyglądać tak, jak na poniższym przykładzie:

```
$('#gallery a').fancybox({  
    overlayColor : '#060',  
    overlayOpacity : .3,  
    transitionIn: 'elastic',  
    transitionOut: 'elastic',  
    easingIn: 'easeInSine',  
    easingOut: 'easeOutSine',  
    titlePosition: 'outside' ,  
    cyclic: true  
});
```

Nie zapomnij dodać przecinka na końcu wiersza z tekstem overlayOpacity : .3. Po przypisaniu opcjom transitionIn oraz transitionOut wartości 'elastic' obrazek jest powiększany w momencie wyświetlania i pomniejszany podczas ukrywania. Z kolei opcje easingIn oraz easingOut wykorzystują dodatkową wtyczkę jQuery (opisaną na stronie 207) i używają jej funkcji do określania szybkości animacji. I w końcu opcja cyclic sprawia, że po wyświetleniu ostatniego obrazka w galerii użytkownik będzie mógł kliknąć przycisk *Next*, by przejść z powrotem do pierwszego.

11. Zapisz stronę i wyświetl ją w przeglądarce.

I kto twierdzi, że JavaScript jest trudny? Pełną, działającą wersję tego przykładu można znaleźć w pliku *complete_fancybox.html*.

FancyBox jest zabawną i użyteczną wtyczką. Jak się przekonasz podczas lektury następnego rozdziału, jej możliwości nie ograniczają się do wyświetlania samych obrazów. Można jej także używać do wyświetlania w wyskakującym okienku zwykłego kodu HTML, klipów wideo, a nawet całych stron WWW.

Usprawnianie nawigacji

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 oraz w okienkach zagnieżdżonych na stronie.

Podstawowe informacje o odnośnikach

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 dalszych podrozdziałach zobaczyesz 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 145) 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 w podrozdziale „Otwieranie zewnętrznych odnośników w nowym oknie” na stronie 252).

Po pobraniu odnośników za pomocą jQuery można nimi manipulować przy użyciu funkcji tej biblioteki. Możesz na przykład wywołać funkcję `each()` (patrz strona 160), aby w pętli dodać do każdego odsyłacza klasę używając do tego celu funkcji `addClass()` (patrz strona 154), a nawet zdarzenie (patrz strona 174). W dalszej części rozdziału zobaczysz wiele operacji, które można przeprowadzić na odnośnikach.

Określanie lokalizacji docelowej

Po pobraniu odnośników trzeba czasem sprawdzić, dokąd prowadzą. W pokazie slajdów utworzonym na stronie 234 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 ``. Można też sprawdzić wartość atrybutu `href` i — jeśli odnośnik prowadzi do innej strony — wyświetlić ją „nad” bieżącą, zamiast przechodzić pod nowy adres. Na stronie 259 dowiesz się, jak to zrobić.

Funkcja `attr()` biblioteki jQuery (patrz strona 159) zapewnia łatwy dostęp do atrybutu `href`. Założymy, 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 (patrz strona 159), a następnie uruchamia funkcję `each()` (patrz strona 160), która uruchamia podaną dalej funkcję dla wszystkich znalezionych odsyłaczy (czyli pobiera i przetwarza każdy

z nich w „pętli”). Ta następna funkcja znajduje się 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 463). Wiersz 4. dodaje do tekstu odnośnika zawartość zmiennej href w nawiasach — (<www.barkbeetles.org>). Wiersz 5. kończy funkcję.

To proste rozwiązań 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.

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 228 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 187). Założmy, 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 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 zatrzymać 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 201).

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 przchodź 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 185). Zdarzenia mają specyficzne funkcje i właściwości. Funkcja `preventDefault()` blokuje domyślne działanie powiązane z danym zdarzeniem. Przy kliknięciu 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 witryną. 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 strona 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 w nowej zakładce).

Uwaga: Eksperci z dziedziny użyteczności strony WWW nie są zgodni co do tego, czy otwieranie nowych okien to dobre, czy złe rozwiązanie. Zobacz na przykład artykuł www.useit.com/alertbox/990530.html.

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 w nowej zakładce. 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 140).

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 145) 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 159) w celu przypisania wartości `_blank` do atrybutu `target`. I to już wszystko!

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 nieszczęsnym użytkowników przed otwieraniem każdej strony w nowym oknie, musisz użyć następującego kodu:

```
var myURL = location.protocol + '//' + location.hostname;  
$('.a[href^="http://"]').not('[href^="'+myURL+'"]').attr('target', '_blank');
```

Ten fragment najpierw zapisuje adres URL witryny w zmiennej `myURL`. Ten adres można uzyskać dzięki obiekowi 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 182) i wstawić opisany wcześniej fragment:

```
<script src="js/jquery-1.6.3.min.js"></script>
<script>
$(document).ready(function() {
    var myURL = location.protocol + '//' + location.hostname;

    $('a[href^="http://"]').not('[href^="'+myURL+'"]').attr('target','_blank');
});
</script>
```

Inne rozwiązanie wymaga utworzenia zewnętrznego pliku JavaScript (patrz strona 40). 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.

Na przykład możesz przygotować plik *open_external.js* i umieścić w nim poniższy kod:

```
function openExt() {
    var myURL = location.protocol + '//' + location.hostname;

    $('a[href^="http://"]').not('[href^="'+myURL+'"]').attr('target','_blank')
}
```

Następnie do każdej strony, na której chcesz zastosować tę funkcję, dodaj poniższy kod:

```
<script src="js/jquery-1.6.3.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 (opis tej techniki znajdziesz na stronie 259). 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/_blank` lub `http://www.google.com/_self`). Drugi argument to nazwa okna, którą możesz ustalić dowolnie, przestrzegając przy tym zasad nazywania zmiennych (patrz strona 60). 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 kwadratowym oknie o boku 200 pikseli, użyj następującego kodu:

```
var newWin = open('http://www.google.com/_blank',
  'theWin', 'height=200,width=200');
```

Uwaga: Symbol `_blank` informuje, że całą instrukcję należy zapisać w jednym wierszu. Jednak ponieważ naprawdę długie wiersze kodu JavaScript nie mieszczą się na stronie tej książki, podzielono je na dwie części.

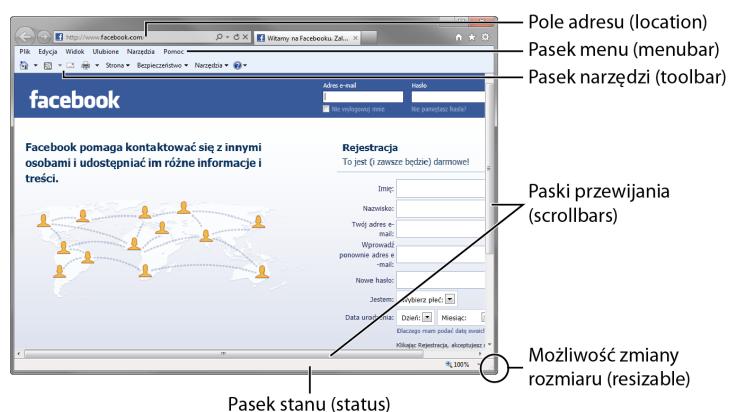
Ten kod otwiera nowe okno i zapisuje referencję do niego w zmiennej `newWin`. W podpunkcie „Używanie referencji do okien” dowiesz się (strona 257), jak użyć takiej referencji do kontrolowania nowego okna.

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 załączaną 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 8.1). Ponadto okna mają szerokość, wysokość i pozycję na ekranie. Większość tych właściwości można ustawić 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);
```



Rysunek 8.1. 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

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 poniżej 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, zakładkami i innymi kontrolkami dostępnymi w danej przeglądarce. W Safari pasek 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://thewildernessdowntown.com/>.

Używanie referencji do okien

Po otwarciu nowego okna można je kontrolować za pomocą powiązanej z nim referencji. Założymy, 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 zachcesz zamknąć okno, użyj metody close():

```
newWin.close();
```

Przeglądarki obsługują wiele metod obiektu okna. Poniżej opisano kilka najczęściej używanych funkcji 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 zapisales 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.
- 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 256) 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.allisnotlost.pl/>.

- 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 określonych liczbach 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 <http://plug-ins.jquery.com/project/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. Założymy, że chcesz, aby niektóre odsyłacze otwierały stronę w nowym, kwadratowym oknie o boku 300 pikseli. Okno to ma mieć paski przewijania i umożliwiać zmianę rozmiaru. Pozostałe elementy „chromowania”, takie jak pasik 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);  
})
```

Otwieranie stron w okienku na pierwotnej stronie

Otwieranie nowych okien może być problematyczne. Wiele przeglądarek blokuje okna wyskakujące, a liczni projektanci są niezadowoleni z braku kontroli nad wyglądem tego elementu. Co zrobić, jeśli potrzebujesz prostego sposobu na wyświetlanie nowych stron bez opuszczania bieżącej strony? Należy oczywiście użyć języka JavaScript! Możesz utworzyć efekt okna na stronie, używając kodu JavaScript do dynamicznego dodania ramki wewnętrzowej (`iframe`) i wyświetlenia w niej nowej strony. Efekt wygląda tak, jakby nowa strona znajdowała się nad pierwotną (patrz rysunek 8.2).

Ramki `<iframe>` (nazwa `iframe` pochodzi od angielskiego „inline frame” — ramka wewnętrzowa) przypominają tradycyjne ramki HTML, jednak można wstawić je w dowolnym miejscu kodu HTML. Za pomocą wymiarów ramki wewnętrzowej i atrybutu `src` (adresu URL) można wczytać inną stronę tak, aby wyglądała jak część bieżącej. Żeby uprościć ten proces, można użyć wtyczki jQuery, która pomoże wykonać żmudne operacje i pozwoli skoncentrować się na projekcie strony.

Już w poprzednim rozdziale poznałeś wtyczkę jQuery, która zapewnia takie możliwości — jest nią FancyBox. W poprzednim rozdziale skorzystaliśmy z niej podczas tworzenia galerii zdjęć, jednak generowanego przez nią sposobu prezentacji można także użyć do wyświetlania całych stron WWW pochodzących zarówno z naszej, jak i z dowolnej innej witryny WWW.



Rysunek 8.2. Przy użyciu wtyczki FancyBox biblioteki jQuery możesz szybko tworzyć „strony na stronie”.

Uwaga: Więcej informacji o ramkach wewnętrzowych znajdziesz na stronie http://www.w3schools.com/tags/tag_iframe.asp. Znacznik <iframe> nie jest poprawny w językach HTML 4.01 Strict i XHTML 1.0 Strict. Jednak wtyczka Greybox dodaje te znaczniki za pomocą języka JavaScript, dlatego sam kod HTML przejdzie validację. Ponadto ponieważ język HTML 5 ma zawierać tag <iframe>, wszystkie popularne przeglądarki będą nadal obsługiwać go w przyszłości.

Sposób użycia wtyczki FancyBox do wyświetlania odnośników na stronie jest nienadal identyczny ze sposobem wykorzystania jej do tworzenia galerii obrazków, opisanym na stronie 234. Poniżej ogólnie opisane zostały czynności, jakie należy w tym celu wykonać:

1. Pobierz pliki wtyczki FancyBox z witryny <http://fancybox.net>.

Informacje o tym, których plików będziesz potrzebował, znajdziesz w punkcie 2. na stronie 235.

2. Dołącz do strony arkusz stylów FancyBox.

We wtyczce FancyBox użyto wyszukanych stylów, by tworzony prezentacji zapewnić odpowiedni wygląd i bez tego arkusza stylów nie będzie ani działać, ani wyglądać prawidłowo.

3. Dołącz do strony pliki JavaScript.

Oczywiście, wtyczka FancyBox wymaga zarówno biblioteki jQuery, jak i swojego własnego pliku JavaScript; a zatem będziesz musiał dołączyć je do strony.

```
<script src="js/jquery-1.6.3.min.js"></script>
<script src="fancybox/jquery.fancybox-1.3.4.js"></script>
```

Zgodnie z informacjami podanymi na stronie 238, można także zdecydować się na wykorzystanie wtyczki jQuery, zawierającej funkcje określające tempo animacji; za jej pomocą można podczas wyświetlania i ukrywania prezentowanych stron utworzyć ciekawsze efekty wizualne.

4. Do każdego odnośnika, którego stronę docelową chcesz wyświetlać w okienku FancyBox, dodaj atrybut `class="iframe"`.

Ponieważ FancyBox musi otwierać zewnętrzne strony WWW wewnętrz znacznika `<iframe>` (co nie jest konieczne w przypadku prezentowania prostej galerii obrazków), potrzebny jest nam zatem jakiś sposób wyróżnienia odnośników wymagających takiego specjalnego traktowania. Najprostszym rozwiązaniem będzie dodanie do każdego z nich atrybut `class="iframe"`:

```
<a href="http://helion.pl" class="iframe">Helion</a>
```

5. Dodaj znacznik `<script>`, funkcję `ready()` jQuery oraz wywołanie funkcji FancyBox.

```
<script>
$(document).ready(function() {
    $('.iframe').fancybox();
}); // koniec funkcji ready
</script>
```

Jak mogłeś się przekonać w poprzednim rozdziale, praktyczne wykorzystanie wtyczki FancyBox jest bardzo proste: wystarczy jeden wiersz kodu. W tym przypadku możemy się posłużyć wywołaniem w postaci `$('.iframe')`, gdyż do każdego odnośnika dodaliśmy klasę `iframe`.

6. W wywołaniu funkcji `fancybox()` umieść literał obiektowy zawierający określenie wysokości i szerokości wyświetlanego okienka.

Wtyczka FancyBox udostępnia wiele różnych ustawień. Zgodnie z informacjami podanymi na stronie 238, można kontrolować na przykład szybkość odtwarzania animacji, stopień nieprzezroczystości nakładki wyświetlonej nad stroną i wiele innych ustawień. Wystarczy określić je w literale obiektowym podawanym w wywołaniu funkcji `fancybox()`. W przypadku prezentowania obrazków wymiary okienka są dostosowywane do wielkości wyświetlonego obrazka. Jednak przy prezentowaniu stron WWW takie wymiary nie są określone, dlatego podanie wysokości i szerokości okienka jest konieczne. Założymy na przykład, że chcemy, by okienko miało szerokość 760 pikseli i wysokość 400 pikseli. W tym celu do kodu z poprzedniego punktu możesz dodać fragment wyróżniony na poniższym przykładzie pogrubioną czcionką:

```
<script>
$(document).ready(function() {
    $('.iframe').fancybox({
        width : 760,
        height : 400
    }); // koniec funkcji fancybox
}); // koniec funkcji ready
</script>
```

Można także zastosować wartości procentowe, aby okienko prezentacji zajmowało określony fragment całego okna przeglądarki. To bardzo dobre rozwiązanie, które pozwala mieć pewność, że okienko wykorzysta bardzo duże monitory, jakie mogą posiadać niektórzy użytkownicy, a jednocześnie będzie dobrze wyglądało także na mniejszych monitorach. W takim przypadku zamiast wartości liczbowych konieczne jest podanie łańcucha znaków zawierającego odpowiednią liczbę zakończoną znakiem procenta (%). Oto przykład:

```
<script>
$(document).ready(function() {
    $('.iframe').fancybox({
        width : '85%',
        height : '75%'
}); //koniec funkcji fancybox
}); //koniec funkcji ready
</script>
```

Przykład — otwieranie strony na stronie

W tym przykładzie przetestujesz wtyczkę FancyBox przez użycie jej na stronie i zmianę wyglądu efektu.

Uwaga: Informacje o pobieraniu przykładowych plików znajdziesz na stronie 43.

1. Otwórz w edytorze tekstu plik *in-page-links.html* z katalogu *R08*.

Po dotarciu do tego miejsca książki musisz już być dosyć zmęczony pisaniem; dlatego też w kodzie tej strony znajdują się już znaczniki dołączające do niej arkusz stylów FancyBox, bibliotekę jQuery, plik JavaScript wtyczki FancyBox, a także znacznik `<script>` zawierający wywołanie metody `$(document).ready()`.

Pierwszą czynnością będzie oznaczenie odnośników, których strony docelowe chcesz wyświetlać w okienku FancyBox.

2. Odszukaj pierwszy odnośnik — `Helion` — i dodaj do niego atrybut `class="iframe"`, by jego kod wyglądał tak:

```
<a href="http://helion.pl/" class="iframe">Helion</a>
```

Podobnie jak w galerii zdjęć generowanych przy użyciu wtyczki FancyBox (której tworzenie zostało opisane na stronie 235), także i w tym przypadku można strony uzupełnić podpisami — wystarczy dodać do znaczników `<a>` atrybut `title` z treścią podpisu.

3. Do odnośnika modyfikowanego w poprzednim punkcie dodaj atrybut `title="Helion"`, by miał następującą treść:

```
<a href="http://helion.pl/" class="iframe" title="Helion">Helion</a>
```

Teraz musisz podobnie zmodyfikować pozostałe odnośniki.

4. Powtórz czynności opisane w dwóch poprzednich punktach dla pozostałych czterech odnośników umieszczonych na stronie.

Jako podpisu (wartości atrybutu `title`) możesz użyć tekstu odnośnika, możesz także sam wymyślić inny podpis. Teraz będziesz już mógł wyświetlać strony docelowe tych odnośników w okienku FancyBox.

5. Kliknij pusty wiersz umieszczony pod wywołaniem `$(document).ready()`, poniżej początku pliku i wpisz kod widoczny na poniższym przykładzie w wierszach do 3. do 7.:

```
1 <script>
2 $(document).ready(function() {
3     $('.iframe').fancybox({
4         width : '90%' ,
5         height : '90%' ,
6         titlePosition : 'outside'
7     }); //Koniec funkcji fancybox
8 }); //Koniec funkcji ready
9 </script>
```

W tym przypadku właściwościom `width` oraz `height` przypisaliśmy wartość 90%, dzięki czemu okienko FancyBox samoczynnie dostosuje się do wielkości okna przeglądarki użytkownika. Z kolei przypisanie wartości 'outside' właściwości `titlePosition` (opisane na stronie 239) sprawi, że podpis będzie wyświetlany poza okienkiem prezentującym stronę.

6. Zapisz stronę i wyświetl ją w przeglądarce. Kliknij odnośniki.

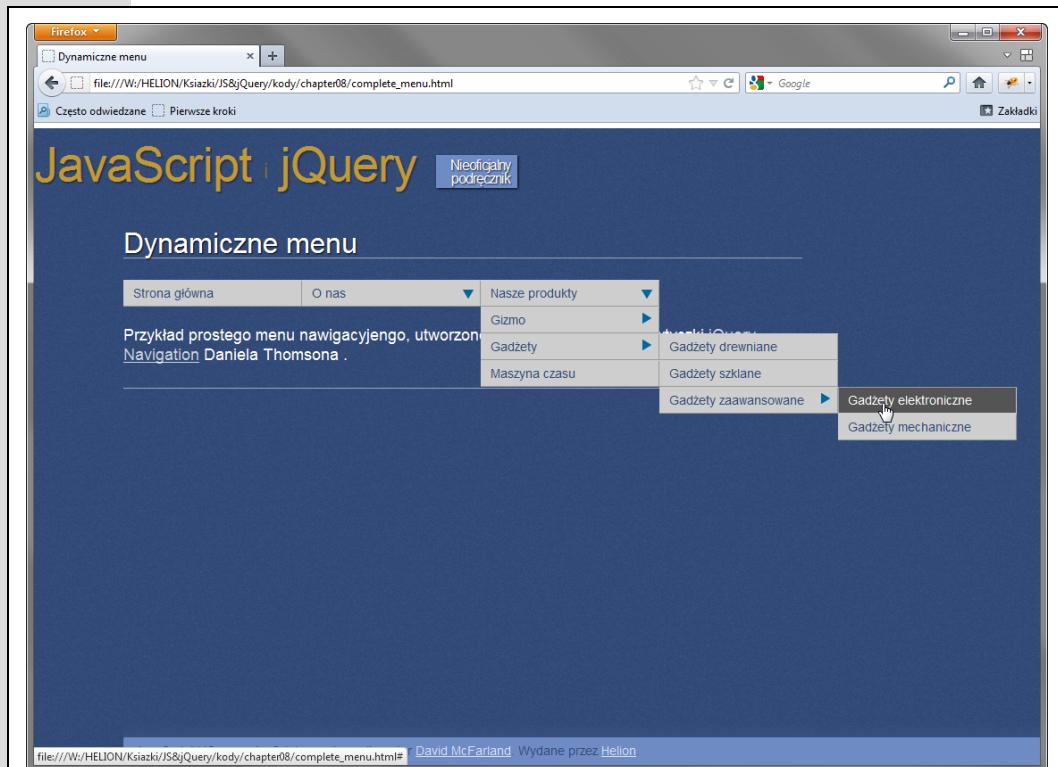
Strony docelowe odnośników są wyświetlane w okienku FancyBox. Super! Możesz także skorzystać z dowolnych innych opcji wtyczki FancyBox (patrz strona 238), by zmodyfikować wygląd strony, na przykład zmienić kolor warstwy, dodać animacje podczas wyświetlania i ukrywania okienka bądź zmienić postać przycisku *Close*. Końcową wersję tego przykładu można znaleźć w pliku *complete_in-page-links.html*.

Uwaga: W poprzednim wydaniu książki dostępny był podrozdział poświęcony tworzeniu „większych odnośników”, czyli odnośników zawierających elementy blokowe, takie jak sekcje (`<div>`), nagłówki i akapity. Technika ta pozwala tworzyć duże, „klikalne” obszary uzupełnione nawet o efekt „rollover”. Jednak aktualnie do budowania takich elementów nie jest już potrzebny JavaScript. Język HTML5 oraz wszystkie nowoczesne przeglądarki umożliwiają umieszczanie wewnętrz elementów `<a>` dowolnych elementów blokowych, w tym także elementów `<div>`.

Animowane menu nawigacyjne

Wraz z powiększaniem się witryny coraz trudniej zapewnić dostęp do wszystkich jej sekcji bez przytaczania użytkowników odnośnikami. Aby ułatwić nawigację, wielu projektantów stron WWW używa systemów menu rozwijanych, które ukrywają odnośniki, jeśli te nie są potrzebne (patrz rysunek 8.3). 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, język CSS nie umożliwia dodawania efektów wizualnych, na przykład stopniowego wyświetlania lub animowanego rozwijania menu.

Na szczęście niewielka ilość kodu JavaScript pozwala utworzyć animowane menu, które działa płynnie we wszystkich przeglądarkach. 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 zagnieżdzonej grupy odnośników, a CSS



Rysunek 8.3. 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. Możesz w ten sposób wyświetlić wiele odsyłaczy i uporządkować stronę

— do upodobnienia odsyłaczy do paska nawigacyjnego oraz rozmieszczenia i ukrywania menu podrzędnych. Następnie dodasz kod JavaScript obsługujący animowane wyświetlanie menu po umieszczeniu kurSORA nad przyciskami paska nawigacji.

Kod HTML

Kod HTML menu nawigacyjnego to prosta lista wypunktowana, utworzona za pomocą znacznika ``. Każdy znacznik `` najwyższego poziomu reprezentuje jeden z głównych przycisków menu. Aby utworzyć menu podrzędne, należy dodać tag `` do znacznika ``, do którego ma należeć dane menu. Kod HTML menu widocznego na rysunku 8.3 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>
</ul>
```

Uwaga: By uprościć prezentowany przykład, zastąpiono w nim faktyczne adresy URL, podawane w atrybutach href odnośników, znakami #, na przykład . W rzeczywistym menu nawigacyjnym odnośniki te wskazywałyby faktyczne strony WWW, na przykład .

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

Pamiętaj, że lista zagnieżdzona znajduje się w znaczniku `` nadziędnego elementu. Przykładowo tag `` zawierający opcje *Gizmo*, *Gadżety* oraz *Maszyna czasu* znajduje się w znaczniku `` elementu *Nasze produkty* (jeśli chcesz przypomnieć sobie informacje o listach HTML, odwiedź stronę 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 Navigation, której twórcą jest Daniel Thomson (<http://www.pollenizer.com/jquery-navigation-plugin/>), wykonuje za nas przeważającą część pracy związanej z odpowiednim rozmieszczaniem elementów list, dzięki czemu nie musimy zwracać sobie głowy tworzeniem arkusza CSS, który wyświetlałby przyciski obok siebie oraz generował rozwijane menu z opcjami podrzędnymi. Zapewne jednak będziemy chcieli określić wygląd poszczególnych przycisków, na przykład ustalić kolor tła, usunąć podkreślenia z odnośników i tak dalej. Poniżej przedstawiłem kilka użytecznych selektorów, których możesz użyć do kontroli różnych elementów menu. (Otwórz na przykład plik *complete_menu.html* umieszczony w katalogu *R08* i przyjrzyj się arkuszowi CSS umieszczonemu w sekcji nagłówka strony).

Uwaga: Podane poniżej selektory zakładają, że główny element `` zawierający całe menu i przyciski ma identyfikator *navigation* — `<ul id="navigation">`. Nic nie stoi na przeszkodzie, byś zastosował zupełnie inny identyfikator, na przykład *mainNav*, *menu* lub jakikolwiek inny, pamiętaj jednak, aby wtedy odpowiednio zmodyfikować podane poniżej selektory. Jeśli na przykład użyjesz identyfikatora *mainNav*, będziesz musiał zmienić selektor `#navigation` a na `#mainNav a`.

- Styl z selektorem `#navigation ul` określa ogólny wygląd każdego menu. Warto usunąć w nim wszystkie marginesy i wypełnienie, przypisując im wartość 0.
- Styl z selektorem `#navigation li` służy do ustalenia wyglądu każdego przycisku (jednak, jak się przekonasz w tym przykładzie, najlepszym rozwiązaniem jest określanie wyglądu w regule odnoszącej się do elementów `<a>` wewnątrz znacznika ``). W najprostszym przypadku w tym stylu należy ustawić wypełnienie i marginesy na 0 oraz przypisać właściwości `list-style-type` wartość `none`, by usunąć znaczniki listy, które inaczej byłyby wyświetlane przy wszystkich przyciskach i opcjach menu. Poniżej zamieściłem dobrą, podstawową wersję stylu dla znaczników `` oraz ``:

```
#navigation ul, #navigation li {  
    margin: 0;  
    padding: 0;  
    list-style-type: none;  
}
```

- Styl z selektorem `#navigation` a określa wygląd odnośników. Można się nim nieźle bawić: zmienić kolor tła, krój czcionki, jej kolor i dodać obramowanie, by przyciski były wyraźnie widoczne. Jednak koniecznie należy uważać, by wartość marginesów i wypełnienia zawsze była równa 0, gdyż w przeciwnym przypadku poszczególne przyciski będą na siebie nachodzić.
- Styl z selektorem `#navigation a:hover` określa wygląd odnośników po umieszczeniu na nich wskaźnika myszy. Można w nim zmienić kolor tła, by przycisk został wyróżniony po wskazaniu go myszą, co stanowi doskonałą, wizualną informację zwrotną dla użytkownika.
- Style z selektorami `#navigation .stack > a` oraz `#navigation .stack > a:hover` kontrolują wygląd odnośników zawierających menu podrzędne. Warto do nich dodać jakiś obraz tła (na przykład strzałkę skierowaną w dół), który poinformuje użytkownika, że użycie tego przycisku wyświetli menu, bądź w jakikolwiek inny sposób odróżni te przyciski od pozostałych, które nie udostępniają kolejnych opcji menu.
- Style z selektorami `#navigation ul .stack > a` oraz `#navigation ul .stack > a:hover` określają postać menu podrzędnych, czyli wyświetlanych po wskazaniu przycisku, który sam stanowi elementu menu. Przykładowo na rysunku 8.4 przyciski *Gizmo* i *Gadżety* zawierają menu podrzędne. Nie trzeba dodatkowo ustalać postaci tych przycisków, jednak wtyczka wyświetla je w sposób nieco inny od pozostałych. Jak widać na rysunku 8.3, po wskazaniu przycisku umieszczonego na głównym poziomie menu (na przykład *Nasze produkty*) kolejny poziom menu wyświetlany jest poniżej niego; natomiast kolejne poziomy menu są wyświetlane *na prawo* od wybranego przycisku (na przykład *Gizmo* oraz *Gadżety*). Jeśli chcemy użyć graficznej strzałki, by oznaczyć menu, dla opcji menu głównego może to być strzałka skierowana w dół, natomiast dla pozostałych opcji — strzałka wskazująca w prawo. Przykłady można znaleźć w pliku *complete_menu.html*.
- Pominięcie elementów pływających. We wtyczce tej użyto właściwości *float* CSS, by umieszczać przyciski w menu głównym, jeden obok drugiego. Z tego powodu tekst oraz inna zawartość strony umieszczona poniżej menu mogą zostać przesunięte ku górze i znaleźć się z prawej strony menu. Aby zapobiec takiej niepożądanej sytuacji, do elementu umieszczonego bezpośrednio poniżej menu należy dodać właściwość stylu *clear:left*. Najprostszym sposobem jest tu utworzenie następującego stylu:

```
.clearLeft { clear : left; }
```

Klasę tę można następnie dodać do elementu umieszczonego bezpośrednio po elemencie `` menu. Założmy na przykład, że za menu umieszczony jest znacznik akapitu. W takim przypadku otwierający znacznik tego akapitu mógłby mieć następującą postać:

```
<p class="clearLeft">
```

Kod JavaScript

Sposób wykorzystania kodu JavaScript do wyświetlania menu jest prosty. Jeśli element jest powiązany z listą zagnieźdzoną (menu rozwijanym), to po umieszczeniu nad nim kurSORA należy wyświetlić tę listę. Kiedy użytkownik przeniesie kurSOR w inne miejsce, należy ukryć wszystkie listy zagnieździone.

Jest jednak kilka drobiazgów, które komplikują rozwiązywanie. Na przykład menu rozwijane, które błyskawicznie znikają w momencie przeniesienia kurSORA poza opcje, wymagają precyzji w korzystaniu z myszy. Łatwo jest przesunąć kurSOR w inne miejsce przy korzystaniu z takiego menu. Jeśli opcje nagle znikną, użytkownik musi ponownie najechać kurSorem na przycisk, aby otworzyć listę odnośników. Jeśli menu rozwijane ma kilka poziomów, bardzo łatwo jest na jednym z nich wyjechać kurSorem 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.

Skorzystanie z wtyczki jQuery Navigation to łatwy sposób na tworzenie prostych rozwijalnych menu i stosowanie efektów wizualnych, takich jak stopniowe pojawianie się elementów oraz ich wysuwanie.

Przykład

Znasz już podstawy tworzenia menu nawigacyjnych. W tym praktycznym przykładzie użyjesz kodu CSS i JavaScript do przekształcenia prostej listy HTML z opcjami ze strony 263 na pasek nawigacyjny.

Uwaga: Informacje o pobieraniu przykładowych plików znajdziesz na stronie 43.

1. Otwórz w edytorze tekstu plik *menu.html* z katalogu *R08*.

Ten plik zawiera wypunktowaną listę odnośników, którą przekształcisz na pasek nawigacyjny. Aby zobaczyć, jak strona wygląda bez skryptów otwórz plik w przeglądarce. Pierwszy krok polega na dołączeniu arkusza stylów.

2. Kliknij pusty wiersz pod znacznikiem `<script src=".../_js/jquery-1.6.3.min.js"></script>` i dodaj następujący kod:

```
<script src=".../_js/nav1.1.min.js"></script>
```

Znacznik ten dołączy do strony wtyczkę jQuery Navigation. Kolejnym krokiem będzie dodanie kodu JavaScript — strona, którą można znaleźć w przykładach do książki, zawiera już znacznik `<script>` z wywołaniem funkcji `$(document).ready()`.

3. Kliknij pusty wiersz wewnętrz wywołania funkcji `$(document).ready()` i wpisz kod wyróżniony pogrubioną czcionką:

```
$(document).ready(function() {  
    $("#navigation").navPlugin({
```

```
'itemWidth': 200,  
'itemHeight': 30  
});  
});
```

Aby uaktywnić menu, najpierw posłużymy się jQuery, by pobrać element `` zawierający główny pasek nawigacyjny — w naszym przykładzie element ten ma identyfikator `navigation`, dlatego też użyliśmy wywołania `$('#navigation')`. Z kolei wywołanie `.navPlugin()` spowoduje użycie wtyczki jQuery Navigation do przygotowania i wyświetlenia menu.

Działanie wtyczki można kontrolować, przekazując w wywołaniu funkcji `.navPlugin()` literał obiektowy. Wtyczka ta nie udostępnia jednak zbyt wielu opcji; dwie najważniejsze to `itemWidth` oraz `itemHeight`, określające — odpowiednio — szerokość i wysokość przycisków menu. Nie można ustalać różnych wymiarów dla poszczególnych przycisków w menu, dlatego trzeba się upewnić, że podane wymiary wystarczą do prawidłowego wyświetlenia przycisku zawierającego *najdłuższy* tekst. Można także zastosować efekty wizualne, które uatrakcyjnią sposób wyświetlania menu.

4. Zmodyfikuj dodany wcześniej fragment kodu tak, by wyglądał on identycznie do tego przedstawionego poniżej:

```
1 $(document).ready(function() {  
2   $('#navigation').navPlugin({  
3     'itemWidth': 200,  
4     'itemHeight': 30,  
5     'navEffect' : 'slide'  
6   });  
7 });
```

Nie zapomnij dodać przecinka na końcu wiersza 4., za liczbą 30. Opcja `navEffect` kontroluje sposób wyświetlania menu. Domyślnie po prostu pojawia się ono na stronie, jednak przy użyciu wartości `'slide'` można spowodować, że będzie wysuwane z góry ku dołowi; druga dostępna wartość — `'fade'` — sprawi, że menu będzie się stopniowo pojawiać. Dostępna jest także możliwość określania szybkości, z jaką menu będzie się pojawiać i znikać.

5. Wprowadź kolejną modyfikację (dodając do kodu wiersz wyróżniony pogrubioną czcionką):

```
1 $(document).ready(function() {  
2   $('#navigation').navPlugin({  
3     'itemWidth': 200,  
4     'itemHeight': 30,  
5     'navEffect' : 'slide',  
6     'speed' : 250  
7   });  
8 });
```

Nie zapomnij o dodaniu przecinka za wartością 'slide' w wierszu 5. Wartość opcji speed jest określana w milisekundach, a zatem 250 oznacza 250 milisekund, czyli 1/4 sekundy i jest to całkiem dobre ustawienie. Zastosowanie większej wartości sprawi, że menu będzie wyświetlane wolno, co utrudni korzystanie z niego.

6. Zapisz stronę i wyświetl ją w przeglądarce.

Teraz dysponujesz już w pełni działającym paskiem nawigacyjnym. To było całkiem łatwe. Zajrzyj do arkusza stylów CSS, by dowiedzieć się, jak są formantowane przyciski, i spróbuj wprowadzić w nim zmiany i zmodyfikować wygląd menu. Pełną wersję przykładu można znaleźć w pliku *complete_menu.html* w katalogu *R08*.

UWAGA NA WTYCZKĘ

Inne wtyczki jQuery usprawniające nawigację

Wtyczka jQuery o nazwie Navigation jest prosta i efektywna, istnieje też bardzo wiele innych wtyczek tej biblioteki, które umożliwiają tworzenie znacznie bardziej zaawansowanych mechanizmów nawigacyjnych.

- ◆ DD Mega Menu (<http://dynamicdrive.com/dynamic-index1/ddmegamenu.htm>) pozwala tworzyć nie tylko rozwijane menu, lecz także całe elementy <div>, które będą wyświetlane na stronie w animowany sposób. Wyobraź sobie pasek nawigacyjny, który po umieszczeniu wskaźnika myszy na przycisku wyświetla dużą ramkę z dowolną zawartością. W ten sposób można ukrywać treści umieszczone na stronie i wyświetlać je, gdy użytkownik wskaże odpowiedni przycisk myszą. Wtyczka ta umożliwia także budowanie zwyczajnych menu rozwijanych.

- ◆ Wtyczka jqDock (<http://www.wizzud.com/jqDock/>) pozwala tworzyć efekt Dock, znany z komputerów Mac: umieszcza w poziomym rzędzie grupę miniatur, które są powiększane w momencie wskazania ich kursem myszy. Daje to bardzo ciekawy efekt wizualny.

Jeśli żadna z tych wtyczek nie przypadnie Ci do gustu, możesz sprawdzić listę 45 innych wtyczek nawigacyjnych oraz samouczków; dostępne są one na stronie <http://www.noupe.com/jquery/45-jquery-navigation-plugins-and-tutorials.html>.

Wzbogacanie formularzy

Od czasu powstania sieci WWW formularze umożliwiają pobieranie informacji od użytkowników witryny. 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 9.1. Najważniejszy jest znacznik `<form>`, który wyznacza początek (otwierający tag `<form>`) i koniec formularza (zamykający tag `</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`. Na przykład aby utworzyć pole tekstowe, należy użyć znacznika `<input>` i przypisać do atrybutu `type` wartość `text`:

```
<input name="user" type="text">
```



Rysunek 9.1. Prosty formularz może zawierać wiele różnych kontrolek, w tym pola tekstowe, przyciski opcji, pola wyboru, listy opcji i tak dalej. Pełny zestaw pól formularzy HTML i ich opis znajdziesz na stronie www.w3schools.com/html/html_forms.asp

Oto kod HTML formularza widocznego na rysunku 9.1. 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="heliskiing"
        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 chcesz otrzymywać od nas iertyjące 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: Znacznik `<label>` to następny tag 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. Założymy, 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 275.

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.

Na szczęście jQuery znacznie ułatwia pobieranie różnych rodzajów pól formularzy (patrz tabela 9.1). Przy użyciu jednego z selektorów formularzy udostępnianych przez tę bibliotekę można łatwo zidentyfikować wszystkie pola konkretnego typu i manipulować nimi. Założymy, 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 9.1. JQuery udostępnia wiele selektorów, które ułatwiają manipulowanie różnymi polami formularzy

Selektor	Przykład	Opis działania
:input	<code>\$('.:input')</code>	Pobiera pola wejściowe, pola tekstowe, znaczniki <select> i przyciski, czyli wszystkie elementy formularzy.
:text	<code>\$('.:text')</code>	Pobiera wszystkie pola tekstowe.
:password	<code>\$('.:password')</code>	Pobiera wszystkie pola z hasłem.
:radio	<code>\$('.:radio')</code>	Pobiera wszystkie przyciski opcji.
:checkbox	<code>\$('.:checkbox')</code>	Pobiera wszystkie pola wyboru.
:submit	<code>\$('.:submit')</code>	Pobiera wszystkie przyciski do przesyłania danych.
:image	<code>\$('.:image')</code>	Pobiera wszystkie przyciski z rysunkami.
:reset	<code>\$('.:reset')</code>	Pobiera wszystkie przyciski do czyszczenia formularza.
:button	<code>\$('.:button')</code>	Pobiera wszystkie pola typu button.
:file	<code>\$('.:file')</code>	Pobiera wszystkie pola do przesyłania plików.
:hidden	<code>\$('.:hidden')</code>	Pobiera wszystkie pola ukryte.

Następnie wystarczy przejść w pętli po pobranych elementach za pomocą funkcji `.each()` (patrz strona 160), aby upewnić się, że każde pole ma określoną wartość. (Dużo więcej informacji o walidacji pól formularzy znajdziesz na stronie 293).

Można połączyć selektory formularzy z innymi selektorami. Założymy, 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, który 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. Przymijmy, 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 poniższego wywołania, w którym użyte zostały selektor atrybutów jQuery (patrz strona 145) oraz filtr :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>. Założymy, 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. Założymy, 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()`. Przymijmy, że strona zawiera formularz do zamawiania towarów, a skrypt ma automatycznie obliczać cenę zakupów na podstawie liczby produktów podanej przez użytkownika (patrz rysunek 9.2). Należy pobrać wpisaną liczbę, pomnożyć ją przez cenę produktu, a następnie ustawić wartość pola z ceną.

```
<input name="quantity" type="text" id="quantity">  
  
Oblicz cenę  
  
LICZBA PRODUKTÓW: 2  
CENA: 9.95  
ŁĄCZNA CENA: 19.90  
Wyślij  
  
<input name="total" type="text" id="total">
```

```
var unitCost=9.95;  
var amount=$('#quantity').val();  
var total=amount * unitCost;  
total=total.toFixed(2);  
$('#total').val(total);
```

Rysunek 9.2. JQuery ułatwia pobieranie i ustawianie wartości pól formularzy

Kod do pobierania liczby produktów i ustawiania łącznej ceny w formularzu z rysunku 9.2 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.
```

Pierwszy wiersz kodu tworzy zmienną, która przechowuje cenę produktu. Drugi 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 468). Wiersz 5. przypisuje łączną cenę zakupów do pola o identyfikatorze `total`. Na następnej stronie 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 pól wyboru wymagają, aby internauta wybrał określoną wartość. Na stronie 274 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 />
```

Ponieważ checked to atrybut języka HTML, można łatwo użyć jQuery do sprawdzania stanu pola:

```
if ($('#news').attr('checked')) {  
    // Pole jest zaznaczone.  
} else {  
    // Pole nie jest zaznaczone.  
}
```

Kod \$('#news').attr('checked') zwróci wartość true, jeśli pole jest zaznaczone. W przeciwnym razie zwróci wartość undefined, którą interpreter traktuje tak samo jak 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 91.

Atrybutu checked i funkcji attr() można używać także do sprawdzania stanu przycisków opcji.

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 wcisnięcie 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 validację 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. Założymy, że chcesz sprawdzić, czy pole z imieniem i nazwiskiem użytkownika, widoczne na rysunku 9.1, zawiera w momencie przesyłania formularza dane. W tym celu należy dodać do formularza zdarzenie `submit` i sprawdzić wartość wspomnianego pola przed przesaniem 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 272 możesz zobaczyć, że formularz ma identyfikator `signup`, a identyfikator pola `name` to `username`. Dlatego można użyć biblioteki jQuery do przeprowadzenia validacji 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 182). Wiersz 2. dodaje funkcję do zdarzenia `submit` formularza (informacje o korzystaniu ze zdarzeń znajdziesz na stronie 169). 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 pełnionym błędzie.

Wiersz 5. jest bardzo istotny, ponieważ wstrzymuje przesyłanie formularza. Jeśli pominięsz 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 185).

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. Założymy, 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.”. Ta technika 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 162, 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. Założmy, 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 atrybut `defaultValue`, który 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.attr('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.

Założymy jednak, że użytkownik wpisał nazwę **helloKitty**, przeszedł do innego pola, a następnie zauważał, 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.

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 validacji po wyjściu poza te elementy. Założymy, 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 467), czy wartość jest liczbą. Jeśli tak nie jest, skrypt uruchamia wiersz 4., który wyświetla okno dialogowe.

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. Założymy, 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 289.

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 klikniętych 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 9.1). Kiedy użytkownik zaznaczy opcję, przeglądarka zgłasza zdarzenie change. Można użyć go do uruchomienia validacji. Wielu projektantów stron WWW często umieszcza jako pierwszą opcję instrukcję, na przykład „Wybierz kraj”. 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. Wybór opcji na jednej liście rozwijanej może uruchamiać funkcję, która zmienia odpowiedzi dostępne na drugiej liście. Rysunek 9.3 przedstawia formularz z dwiema listami rozwijanymi. Zaznaczenie opcji w górnej zmienia listę kolorów dostępnych w dolnej.

Rysunek 9.3 przedstawia trzy klatek z formularzem "Wybierz produkty". W każdej klatce znajdują się dwie listy rozwijane: "PRODUKT" i "KOLORY".

- Klatka 1:** Produkt wybrany: Koszulka. Lista "KOLORY" zawiera: -- Wybierz kolor --, czerwony, zielony, niebieski. Czerwony jest zaznaczony.
- Klatka 2:** Produkt wybrany: Sweter. Lista "KOLORY" zawiera: -- Wybierz kolor --, oliwkowy, indygo, śnieżnobiały. Indygo jest zaznaczony.
- Klatka 3:** Produkt wybrany: Spodnie. Lista "KOLORY" zawiera: -- Wybierz kolor --, khaki, czarny, biały. Biały jest zaznaczony.

Wszystkie listy rozwijane mają przycisk "Wybierz kolor" obok nazwy listy. Kliknięcie w ten przycisk powoduje zmianę listy w dolnej części.

Aby dodać zdarzenie change do listy rozwijanej, należy użyć funkcji change() biblioteki jQuery. Założymy, 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.');
    }
})
```

Inteligentne formularze

Korzystanie z formularzy wymaga od internautów sporo wysiłku. Trzeba wypełnić pola tekstowe, zaznaczyć opcje, włączyć 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 zmusić użytkownika do przenoszenia kurSORA nad pole i klikania go? Czy nie lepiej od razu umieścić kurSORA 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. Założmy, że po wyświetleniu strony kurSOR ma się znajdować w polu `name` formularza widocznego na rysunku 9.1. W kodzie HTML tego formularza (patrz strona 272) pole to ma identyfikator `username`. Dlatego aby w kodzie JavaScript aktywować to pole, czyli umieścić w nim kurSOR, należy użyć poniżej 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 274, 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 buletyn”), możesz doprecyzować selektor przez wskazanie formularza, którego pole skrypt ma aktywować. Założmy, 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 `signup`), 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 *uniemożliwić* użytkownikom wypełnienie pola tekstowego albo zaznaczenie pola wyboru lub opcji listy rozwijanej. Założymy, 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 atrybut disabled na true. Na przykład w celu wyłączenia wszystkich pól input należy użyć następującego kodu:

```
$('.input').attr('disabled', true);
```

Zwykle pola wyłącza się w odpowiedzi na określone zdarzenie. Na przykład w formularzu 1040 można wyłączyć pole na numer ubezpieczenia współmałżonka, jeśli podatnik poinformuje, że jest samotny. Założymy, 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').attr('disabled', true);
});
```

Oczywiście warto też mieć możliwość ponownego włączenia pola. Aby to zrobić, wystarczy przypisać do atrybutu disabled wartość false. Poniższy kod włącza wszystkie pola formularza:

```
$('.input').attr('disabled', false);
```

Uwaga: Przy wyłączaniu pól formularza pamiętaj, aby używać wartości logicznych true i false (patrz strona 58), a nie łańcuchów znaków 'true' i 'false'. Poniższe wywołanie jest błędne:

```
$('.input').attr('disabled', 'false');
```

Prawidłowy jest następujący zapis:

```
$('.input').attr('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').attr('disabled', false);
});
```

Technika ta została wykorzystana w przykładzie przedstawionym na stronie 286.

CZĘSTO ZADAWANE PYTANIA

Blokowanie wielokrotnego przesyłania danych

Czasem te same informacje z formularza są przesyłane kilkakrotnie. Jak temu zapobiec?

Serwery sieciowe 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 atrybutu 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').attr('disabled',true);
});
```

Jeśli na stronie znajduje się tylko jeden formularz, nie trzeba nawet używać identyfikatorów:

```
$('form').submit(function() {
    $('input[type=submit]').
        attr('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.attr('disabled',true);
    subButton.val('Przesyłanie w toku');
});
```

Pamiętaj o umieszczeniu tego kodu w funkcji `$(document).ready(function() {` (patrz strona 182).

Ukrywanie i wyświetlanie opcji formularza

Oprócz wyłączania pól można także w inny sposób zapobiec niepotrzebnemu wypełnianiu formularza — ukrywając zbędne elementy. Na przykład w formularzu podatkowym warto ukryć pole z numerem ubezpieczenia wspólnego żonka, 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 198.

Z perspektywy użytkownictwa 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 (pół, etykiet i innych fragmentów kodu HTML) w tagu `<div>`, określenu identyfikatora tego tagu i schowaniu go. W kolejnym 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 9.4).

The screenshot shows a Firefox browser window with the title bar "Zabawa z formularzami". The main content area displays a web page with a dark blue header featuring the text "JavaScript i jQuery" and "Nieoficjalny podręcznik". Below the header is a section titled "Formularz zamówienia". This section contains several input fields: "IMIĘ I NAZWISKO" (with a circled "1" above it), "ADRES", "MIEJSCOWOŚĆ", "WOJEWÓDZTWO", "KOD POCZTOWY", "PAŃSTWO" (with a dropdown menu "Wybierz państwo"), "SPOSÓB PŁATNOŚCI" (radio buttons for PayPal, Visa, and MasterCard, with PayPal selected), "NUMER KARTY" (with a circled "2" above it), and "DATA WYGAŚNIĘCIA". Below this section is another titled "Dane do wysyłki" containing a checkbox labeled "3" and the text "Takie same, jak przy płatności". At the bottom of the page is a footer with the text "JavaScript i jQuery. Nieoficjalny podręcznik, autor: David McFarland. Wydane przez Helion".

Rysunek 9.4. Za pomocą kodu JavaScript można zwiększyć użyteczność formularzy i dodać do nich interaktywne mechanizmy, na przykład ukrywanie niepotrzebnych elementów i wyłączenie pól, których użytkownik nie powinien wypełniać

Uwaga: Informacje o pobieraniu przykładowych plików znajdziesz na stronie 43.

Aktywowanie pola

Pierwsze pole przykładowego formularza pobiera imię i nazwisko osoby składającej zamówienie (patrz rysunek 9.4). Aby ułatwić wypełnianie formularza, można po wczytaniu strony automatycznie umieścić kurSOR w tym polu.

1. Otwórz w edytorze tekstu plik *form.html* z katalogu *R09*.

Do strony dołączono już plik biblioteki jQuery i funkcję `$(document).ready()` (patrz strona 182). 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 rozpoczęć 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 spojrzyś na rysunek 9.4), 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):

```
1 <div><span class="label">Sposób płatności</span>  
2   <input type="radio" name="payment" id="paypal" value="paypal">  
3   <label for="paypal">PayPal</label>  
4   <input type="radio" name="payment" id="visa" value="visa">  
5   <label for="visa">Visa</label>  
6   <input type="radio" name="payment" id="mastercard" value="mastercard">  
7   <label for="mastercard">MasterCard</label>  
8 </div>  
9 <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() {
  $('#name').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 174, 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   $('#name').focus();
3   $('#paypal').click(function() {
4     $('#creditCard_input').attr('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 uwzględnii (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 283). 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. Aby podkreślić stan pól, należy zmienić kolor

tekstu na jasnoszary. Przy okazji można użyć szarego koloru jako tła pól, aby wyglądały na wyłączone. Większość przeglądarek robi to automatycznie, jednak w Internet Explorerze wyłączone pola wyglądają tak samo jak zwykłe.

6. Dodaj do skryptu kod wyróżniony pogrubieniem:

```
1 $(document).ready(function() {  
2     $('#name').focus();  
3     $('#paypal').click(function() {  
4         $('#creditCard input').attr('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 mieścią się na stronach książki, podzielono je na dwie części. Niemniej jednak, zgodnie z informacjami podanymi na stronie 63, 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').attr('disabled',true)  
    .css('backgroundColor', '#CCC');
```

Warto zwrócić uwagę, że niektórzy programiści w takim kodzie zapisywany 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 `.attr()`.

Skrypt najpierw wywołuje funkcję `css()` biblioteki jQuery, aby zmienić kolor tła pól tekstowych (zauważ, że instrukcja ta jest częścią wiersza 4., dlatego należy wpisać ją w tym samym wierszu, w którym znajduje się funkcja `attr()`). Następnie program zmienia przy użyciu tej samej funkcji kolor czcionki znaczników `<label>` w odpowiednim znaczniku `<div>` (funkcję `css()` opisano na stronie 155).

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').attr('disabled', false)  
    ↵.css('backgroundColor', '');  
    $('#creditCard label').css('color', '');  
}); //koniec funkcji click
```

Selektor `$('#visa, #mastercard')` pobiera oba przyciski opcji (patrz wiersze 4. i 6. kodu HTML ze strony 286). Zauważ, że w celu usunięcia koloru tła i tekstu ustalonego po kliknięciu przycisku *PayPal* wystarczy przekazać jako barwę pusty łańcuch znaków — `$('#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.

WTYCZKI BIBLIOTEKI JQUERY

Łatwiejsze pobieranie dat

Niezależnie od tego, czy rejestrujesz się w witrynie społecznościowej, rezerwujesz miejsca w samolocie, czy przeszukujesz kalendarz wydarzeń, często musisz podać datę. Zazwyczaj stуї do tego zwykłe pole tekstowe, w którym należy wprowadzić łańcuch znaków z datą. Niestety, nie zawsze wiadomo, jaki dzień miesiąca będzie na przykład w piątek za dwa tygodnie. Ponadto puste pole tekstowe sprawia, że użytkownik może wpisać datę w dowolnym formacie: 10-20-2009, 10.20.2009, 10/20/2009, a nawet 20/10/2009.

Najlepszy sposób na ułatwienia pobierania dat i zagwarantowanie ich formatu polega na użyciu kontrolki kalendarza. Jest to wyskakujący kalendarz, który umożliwia użytkownikowi wybór daty przez kliknięcie danego dnia. Język HTML5 definiuje sposób pozwalający na dodawanie kalendarzy do formularzy, a w niektórych przeglądarkach zaimplementowano proste kontrolki tego typu — jednak jeszcze dużo brakuje do tego, byśmy mogli wybierać daty, korzystając jedynie z możliwości zapewnianych przez HTML, na razie cały czas trzeba używać także skryptów JavaScript. Na szczęście możesz pobrać wtyczkę biblioteki jQuery, która sprawia, że dodanie kontrolki kalendarza do formularza stanie się niezwykle proste.

Wtyczka jQuery UI Datepicker jest zaawansowanym wyskakującym kalendarzem służącym do wybierania dat, który można na wiele sposobów dostosować do własnych potrzeb. Aby użyć wtyczki, należy pobrać ją z witryny jQuery UI (<http://jqueryui.com/>). Można skorzystać ze specjalnej strony do pobierania (<http://jqueryui.com/>

`download/`), by pobrać tylko wtyczkę kalendarza. Można także pobrać wtyczki jQuery UI.

Dodatkowo można też utworzyć własny arkusz stylów, korzystając w tym celu z narzędzia jQuery UI Themeroller (<http://jqueryui.com/themeroller/>).

Po pobraniu tych plików należy dołączyć do strony arkusz stylów jQuery UI, samą bibliotekę jQuery (patrz strona 122), a następnie plik jQuery UI (trzeba przy tym postępować zgodnie z instrukcjami dotyczącymi dołączania do stron WWW zewnętrznych plików JS, podanymi na stronie 40). Więcej informacji na temat jQuery UI można znaleźć na stronie 325.

Po wykonaniu podstawowych zadań trzeba wywołać funkcję `datepicker()` dla pola tekowego. Założymy, że formularz zawiera pole tekstowe o identyfikatorze `dateOfBirth`. Aby po kliknięciu tego pola pojawił się kalendarz, należy dodać do strony znacznik `<script>` z podstawową funkcją `$(document).ready()` (patrz strona 182) i wywołać wtyczkę Datepicker w następujący sposób:

```
$('#dateOfBirth').datepicker();
```

Wyrażenie `$('#dateOfBirth')` to oczywiście standarowa technika języka jQuery, która tu pozwala pobrać pole tekstowe. Funkcja `datepicker()` wykonuje wszystkie pozostałe operacje. Wtyczka Datepicker obsługuje opcje, które umożliwiają pobieranie przedziałów dat, otwieranie kalendarza przez kliknięcie jego ikony i tak dalej. Aby dowiedzieć się czegoś więcej o tej przydatnej wtyczce, odwiedź stronę <http://jqueryui.com/demos/datepicker/>.

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 287, 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 umieszczone w tagu `<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').attr('checked')) {
        }
}); // koniec funkcji click
```

Prosta instrukcja warunkowa (patrz strona 91) ułatwia sprawdzenie stanu pola wyboru i ukrycie lub wyświetlenie pól formularza. Konstrukcja `$(this)` wskazuje kliknięty obiekt, którym jest tu pole wyboru. Atrybut `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:

```
1 <script>
2 $(document).ready(function() {
3     $('#name').focus();
4     $('#paypal').click(function() {
5         $('#creditCard input').attr('disabled', true)
6             .css('backgroundColor', '#CCC');
7         $('#creditCard label').css('color', '#BBB');
8     }); // koniec funkcji click
9     $('#visa, #mastercard').click(function() {
10        $('#creditCard input').attr('disabled', false)
11            .css('backgroundColor', '');
12        $('#creditCard label').css('color', '');
13    }); // koniec funkcji click
14    $('#hideShip').click(function() {
15        if ($('#this').attr('checked')) {
16            $('#shipping').slideUp('fast');
```

```

17      } else {
18          $('#shipping').slideDown('fast');
19      }
20  }); // koniec funkcji click
21}); // Koniec funkcji ready
22</script>

```

Wyrażenie `$('#shipping')` pobiera znacznik `<div>` z polami formularza, natomiast funkcje `slideUp()` i `slideDown()` (patrz strona 202) 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 205.

Gotową wersję przykładu, `complete_form.html`, znajdziesz w katalogu *R09*. Jeśli Twoja wersja nie działa, porównaj kod z gotowym rozwiązaniem i przypomnij sobie etapy rozwiązywania problemów, które opisałem na stronie 48.

Walidacja formularzy

Frustrujące jest przeglądanie informacji zwrotnych przesyłanych 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 wcisnięcia 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 9.5).

Sprawdzanie, czy pole tekstowe zawiera dane, jest proste. Na stronie 275 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

Dzięki kontu Yahoo! dostajesz darmowy e-mail i inne wiodące usługi internetowe.

Imię i nazwisko: imię Nazwisko ⚠ Ta informacja jest wymagana

Płeć: - wybierz - ⚠ Ta informacja jest wymagana

Data urodzin: Dzień - wybierz miesiąc - Rok ⚠ Data jest nieprawidłowa

Mieszkam w: Polska

Kod pocztowy: ⚠ Ta informacja jest wymagana

Wybierz identyfikator i hasło

Identyfikator i adres e-mail: Yahoo! @ yahoo.pl Sprawdź ⚠ Ta informacja jest wymagana

Hasło: Stopień szyfrowania hasła ⚠ Ta informacja jest wymagana

Ponownie wpisz hasło ⚠ Ta informacja jest wymagana

W razie zapomnienia identyfikatora lub hasła...

Dodatkowy adres e-mail (opcjonalnie): ⚠ Ta informacja jest wymagana

Pytanie kontrolne 1: - wybierz - ⚠ Ta informacja jest wymagana

Twoja odpowiedź: ⚠ Ta informacja jest wymagana

Pytanie kontrolne 2: - wybierz - ⚠ Ta informacja jest wymagana

Twoja odpowiedź: ⚠ Ta informacja jest wymagana

Kod wizualny | Kod audio Pomoc

dVA~~d~~827L

Wpisz pokazany kod: Użyj nowego kodu

Kliknięcie poniższego przycisku „Utwórz moje konto” jest równoznaczne z potwierdzeniem
przeczytania dokumentu Regulamin Yahoo!, Zasady ochrony danych osobowych Yahoo!,
Warunki świadczenia usług komunikacji elektronicznej, i zgodą na nie oraz zgodę na
otrzymywanie informacji dotyczących konta do Yahoo! w formie elektronicznej. Aby
zapewnić korzystanie z ciekich produktów, odpowiedni dobór reklam oraz ochronę przed
atakami automatycznych systemów Yahoo! mogą skanować i analizować treść poczty,
komunikatora i innych środków komunikacji.

Otwórz moje konto

Copyright © 2012 Yahoo! Wszelkie prawa zastrzeżone. [Pravo autorskie/Zasady własności intelektualnej](#) | [Regulamin](#) | [Przewodnik dotyczący bezpieczeństwa w Internecie](#)
UWAGA: W tym serwisie gromadzimy dane osobowe.Więcej informacji o tym, w jaki sposób korzystamy z twoich danych, można znaleźć w dokumencie [Zasady ochrony danych osobowych](#). [\[Zaktualizowane\]](#)

Rysunek 9.5. Jeśli użytkownik w trakcie rejestrowania konta w serwisie Yahoo! nie wypełni prawidłowo formularza, zobaczy liczne komunikaty o błędach (w kółku)

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 z nich jest wtyczka biblioteki jQuery.

Wtyczka Validation

Validation (<http://bassistance.de/jquery-plugins/jquery-plugin-validation/>) to rozbudowana, ale łatwa w użyciu wtyczka biblioteki jQuery, utworzona przez Jórnego 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.

Proces korzystania z wtyczki Validation wygląda następująco:

- 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 129). 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://bassistance.de/jquery-plugins/jquery-plugin-validation/>. Dostępny pakiet obejmuje wiele dodatkowych elementów, w tym wersje demonstracyjne, testy i tak dalej. Niezbędny jest jednak tylko plik *jquery.validate.min.js*. Wtyczkę znajdziesz też w katalogu *_js* w przykładach dołączonych do książki, umieszczony tam plik nosi nazwę *jquery.validate.min.js* (patrz przykład na stronie 303). Jest to zwyczajny zewnętrzny plik JavaScript, dlatego aby go dołączyć, zastosuj się do instrukcji ze strony 40.

- 3. Dołączanie reguł validacji.**

Reguły validacji 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 294), a bardziej elastyczny, ale też skomplikowany sposób wymaga zastosowania kodu JavaScript (patrz strona 297).

- 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 296, a bardziej elastyczną metodę poznasz na stronie 300.

Uwaga: Na stronie 302 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. Założmy, 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), można też podać je w funkcji `validate()` (metodę tę poznasz na stronie 297).

Cały kod JavaScript potrzebny do przeprowadzenia podstawowej walidacji formularza (z uwzględnieniem dwóch wcześniej opisanych kroków) może być bardzo prosty:

```
<script src="../../js/jquery-1.6.3.min.js"></script>
<script src="../../js/jquery.validate.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 182).

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 funkcję `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 291.

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 9.2. 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 9.2. 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 10/30/2009 to poprawny zapis, natomiast 10-30-2009 — 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 całkowitą. 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 technika ta służy do formatowania elementów. 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 9.2.

Formularz może zawierać pole na datę urodzenia. Założymy, 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 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 01/25/2009”.

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. Założymy, 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 najczęściej 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. Na przykład jeżeli 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 01/28/2009.">
```

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 158) z odrębnymi obiektami opisującymi każdą opcję. Na przykład aby określić regułę walidacji, należy przekazać obiekt z jej kodem. Najpierw trzeba wpisać otwierający nawias klamrowy po pierwszym nawiasie funkcji walidacyjnej, a następnie zamkujący 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 9.6.



Rysunek 9.6. 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 294) 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 303 zastosowano oba podejścia do walidacji jednego formularza.

Kod HTML formularza z rysunku 9.6 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://docs.jquery.com/Plugins/Validation/validate#options>.

Zaawansowane reguły

Zaawansowany sposób określania reguł walidacji polega na przekazaniu obiektu (patrz strona 158) 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 9.6, 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 9.6, 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 157.

Wiersze od 4. do 7. (wyróżnione pogrubieniem) określają reguły dla pola `email`. Nazwa pola to `email` (patrz kod HMTL na stronie 297), 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 9.2. Założmy, 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 zwiększe 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* określona 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* określona 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. Założmy, 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. Założmy, 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">
```

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 342 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. Założmy, 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.

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 niezbędnym 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 do strony działającej po stronie serwera (na przykład strony PHP, JSP, ASP lub Cold Fusion). 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 opisałem na poprzedniej stronie), 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 dziający przykład zastosowania tej techniki, odwiedź stronę <http://jquery.bassistance.de/validate/demo/captcha/>.

Ten proces przypomina tworzenie zaawansowanych reguł, co opisano w poprzednim punkcie. Podstawowa struktura obiektu `messages` wygląda następująco:

```
messages: {
    nazwa_pola: {
        typ_walidacji: 'Komunikat o błędzie'
    }
}
```

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.

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 dadasz 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">
  This field is required.</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. Na przykład aby czcionka tekstu wiadomości była pogrubiona i czerwona, należy umieścić w arkuszu poniższy styl:

```
label.error {
  color: #FO0;
  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 9.7).

Rysunek 9.7. 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 43.

Prosta walidacja

W tym przykładzie najpierw zastosujesz prostą metodę walidacji za pomocą wtyczki Validation, opisaną na stronie 294. Następnie użyjesz bardziej złożonych technik opartych na metodzie zaawansowanej, omówionej na stronie 297. Zobaczysz, że można swobodnie łączyć oba podejścia w jednym formularzu.

1. Otwórz w edytorze tekstu plik validation.html z katalogu R09.

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 dodączę do strony plik jquery.js, dodaj następujący kod:

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

Wtyczka Validation znajduje się w podkatalogu _js w głównym katalogu z przykładami, wraz z plikiem JavaScript biblioteki jQuery.

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.php" 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" ↵ "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 9.8).

Formularz rejestracji

IMIĘ I NAZWISKO	<input type="text"/>	Wpisz imię i nazwisko.
ADRES E-MAIL	<input type="text"/>	
HASŁO	<input type="password"/>	

Rysunek 9.8. Na razie nie przejmuj się wyglądem komunikatu o błędzie. Na stronie 311 zobaczysz, jak go sformatować

Gratulacje! Właśnie dodałeś do formularza walidację za pomocą prostej metody opisanej na stronie 294. Następnie należy dodać regułę walidacji dla pola z datą urodzenia.

Uwaga: Jeśli zamiast komunikatu o błędzie zobaczyś 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 dodajesz 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 wypróbuje 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 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 należy wypróbować zaawansowaną technikę walidacji.

Walidacja zaawansowana

Na stronie 296 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 zamkujący 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 zauważyc, 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 email. „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 przesyłać 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.

5. Ponownie przejdź do edytora i wpisz przecinek po zamkającym 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 przesyłać 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 „witat”, i spróbuj przesyłać 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 wyróżnione pogrubieniem):

```
1  rules: {  
2      email: {  
3          required: true,  
4          email: true  
5      },  
6      password: {  
7          required: true,  
8          rangelength:[8,16]  
9      },  
10     confirmPassword: {  
11         equalTo: '#password'  
12     }  
13 }, // 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 300).

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 wyróżnione pogrubieniem):

```
1   messages: {
2     email: {
3       required: "Podaj adres e-mail.",
4       email: "To nie jest prawidłowy adres e-mail."
5     },
6     password: {
7       required: 'Wpisz hasło.',
8       rangelength: 'Hasło musi mieć od 8 do 16 znaków.'
9     },
10    confirmPassword: {
11      equalTo: 'Podane hasła nie pasują do siebie.'
12    }
13 } //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 wyróżnione 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 294. 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.

- 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 dodajesz 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 móc wysłać formularz.

- 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 9.9. 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().

- Znajdź dodany wcześniej skrypt walidacji i wpisz przecinek po zamkającym 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.
```

Wtyczka Validation przyjmuje opcjonalny obiekt errorPlacement, zawierający funkcję anonimową (patrz strona 160), która określa lokalizację komunikatu o błędzie. Każdy błąd jest przesyłany przez tę funkcję, gdy zatem chcesz zmienić położenie jedynie wybranych komunikatów, konieczne będzie dodanie

Firefox

Zabawa z formularzami

file:///W:/HELION/Księgi/Javascript/JQuery/kody/chapter09/validation.html

JavaScrip t jQuery

Nieoficjalny podręcznik

Formularz rejestracji

IMIĘ I NAZWISKO Wpisz imię i nazwisko.

ADRES E-MAIL Podaj adres e-mail.

HASŁO Wpisz hasło.

POTWIERDŹ HASŁO

HOBBY Zaznacz przynajmniej 1 hobby. Helskiing Jedzenie korniszonów Produkcja masła orzechowego

DATA URODZENIA

PLANETA URODZENIA Wybierz planetę.

KOMENTARZE

CZY CHCESZ OTRZYMYWAĆ OD NAS IRYTUJĄCE LISTY ELEKTRONICZNE?

Zaznacz jedno z pól. Tak Zdecydowanie A czy mam jakiś wybór?

Wyślij

JavaScript i jQuery. Nieoficjalny podręcznik, autor David McFarland. Wydane przez Helion

Rysunek 9.9. 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

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 91) 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 151), aby dodać komunikat o błędzie bezpośrednio przed zamkającym znacznikiem `</div>`.

Kod JavaScript formularza jest już gotowy. Oto kompletny skrypt (wraz z funkcją `$(document).ready()`):

```
1  $(document).ready(function() {  
2      $('#signup').validate({  
3          rules: {  
4              email: {  
5                  required: true,  
6                  email: true
```

```
7      },
8      password: {
9          required: true,
10         rangelength:[8,16]
11     },
12     confirm_password: {equalTo:'#password'},
13     spam: "required"
14 }, //Koniec obiektu rules.
15 messages: {
16     email: {
17         required: "Podaj adres e-mail.",
18         email: "To nie jest prawidłowy adres e-mail."
19     },
20     password: {
21         required: 'Wpisz hasło.',
22         rangelength: 'Hasło musi mieć od 8 do 16 znaków.'
23     },
24     confirm_password: {
25         equalTo: 'Podane hasła nie pasują do siebie.'
26     }
27 }, //Koniec obiektu messages.
28 errorPlacement: function(error, element) {
29     if ( element.is(":radio") || element.is(":checkbox") ) {
30         error.appendTo( element.parent() );
31     } else {
32         error.insertAfter(element);
33     }
34 } //Koniec obiektu errorPlacement.
35 }); //Koniec funkcji validate
36 }); //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 zamkającym 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: #FO0;
    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 302). 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 margin, 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 do pliku `form.css` ostatnią regułę:

```
#signup input.error, #signup select.error {  
    background: #FFA988;  
    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 CSS 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 jak ten z rysunku 9.7. Jego gotową wersję znajdziesz w pliku `complete_validation.html` w katalogu *R09*.

Rozbudowa interfejsu stron WWW

Bywa, że strony WWW przypominają długie, jednostronicowe broszury. Odwiedzające je osoby mogą się czuć przytłoczone przez wiele tekstu i znaczną liczbę obrazków, które muszą dłucho przewijać, zwłaszcza gdy nie są w stanie szybko znaleźć poszukiwanych informacji. To naszym zadaniem, zadaniem twórców stron jest zapewnienie użytkownikom narzędzi, które ułatwią im znalezienie tego, czego szukają. Przy użyciu JavaScriptu oraz biblioteki jQuery można usprawniać tworzone strony i ułatwiać użytkownikom korzystanie z nich — na przykład poprzez ukrywanie zawartości, aż do momentu gdy będzie potrzebna, oraz zapewnianie łatwiejszego dostępu do informacji.

W tym rozdziale poznasz techniki służące do poprawiania czytelności i łatwości korzystania ze stron WWW. Karty pozwalają na umieszczanie znacznych ilości informacji na stosunkowo niewielkim obszarze i zapewniają możliwość kliknięcia wybranej karty w celu uzyskania dostępu do mniejszej porcji danych. Etykietki ekranowe — niewielkie, wyskakujące okienka pokazywane po wskazaniu jakiegoś elementu strony wskaźnikiem myszy — umożliwiają wyświetlanie dodatkowych informacji. Coraz bardziej popularną formą kontroli zawartości strony są tak zwane slidery (od angielskiego słowa *slide* — przesuwać) — można je porównać do okna, którego zawartość da się przesuwać, by ukryć jedne, a wyświetlić inne elementy tejże strony. Slidery pozwalają na prezentowanie znacznych ilości informacji i są bardzo często stosowane na stronach głównych witryn.

W tym rozdziale poznasz także kilka przydatnych technik pozwalających na tworzenie własnych komponentów interfejsu użytkownika, takich jak określanie wymiarów okna przeglądarki, konkretnego elementu strony oraz położenia elementu na stronie.

Organizowanie informacji przy użyciu kart

Umieszczenie na stronie zbyt wielu informacji może przyciągnąć użytkownika i sprawić, że strona będzie wyglądała na przepełnioną. Język JavaScript zapewnia wiele możliwości prezentowania znaczących ilości informacji na niewielkim obszarze. Jedną z technik jest stosowanie *kart*. Panel kart składa się z rzędu zakładek wyświetlonych u góry oraz jednej, widocznej karty. Kiedy użytkownik kliknie zakładkę, aktualnie prezentowana karta znika, a na jej miejscu pojawia się inna (patrz rysunek 10.1).

Dane techniczne	Pełen opis	Dostawa
Przetwornik - szczegóły	CMOS 23,1 x 15,4 mm	
Liczba efektywnych pikseli	14,2 mln	
Mocowanie obiektywu	Mocowanie F firmy Nikon (ze stykami AF)	
System ustawiania ostrości	Autofokus (AF): pojedynczy AF (AF-S), tryb ciągły AF (AF-C), automatyczny wybór AF-S/AF-C (AF-A), wyprzedzające śledzenie ostrości włączane automatycznie przy zmianie stanu fotografowanego obiektywu. Ręczne ustawianie ostrości (MF): można korzystać ze wskaźnika ustawienia ostrości	
Wybór punktu AF	Jednopolowy AF, AF z dynamicznym wyborem pola, automatyczny wybór pola AF, śledzenie 3D (11 punktów)	
Blokada AF	Ustawienie ostrości można zablokować naciśkając do połowy spust migawki (pojedynczy AF) lub naciśkając przycisk AE-L/AE-L	
Ręczne ustawianie ostrości	Tak	
Metody pomiaru światła	Matrycowy: Matrycowy pomiar ekspozycji 3D Color Matrix II (obiektywy typu G i D); matrycowy pomiar ekspozycji color matrix II (pozostałe obiektywy z procesorami); Centralnie ważony: przypisanie 75% wagi pomiaru do obszaru o średnicy 8 mm w środku kadru; Punktowy: pomiar z obszaru o średnicy 3,5 mm (około 2,5% powierzchni kadru) na środku wybranego pola AF	
Blokada ekspozycji światła AE-L	Blokada zmierzonej wartości przyciskiem AE-L/AE-L	
Kompensacja ekspozycji	Od -5 do +5 EV w krokach co 1/3 EV	
Czułość ISO	Od ISO 100 do ISO 3200 w krokach co 1 EV; Można również ustawić wartość mniejszą o około 2 EV powyżej wartości ISO 3200 (odpowiednik ISO 12800), dostępny również automatyczny dobór ISO	
Czas otwarcia migawki	Od 1/4000 do 30 s w krokach co 1/3 EV, czas B (bulb). Sterowana elektronicznie szczelinowa o pionowym przebiegu w płaszczyźnie ostrości	
Stabilizator obrazu	wg parametrów obiektywu	
Balans bieli	Automatyczny (balans bieli TTL z przetwornika obrazu i 420-pikselowego czujnika RGB), żarowe, fluorescencyjne (7 rodzajów), światło słoneczne, lampa błyskowa, chmury, cień, zmierzona wartość manualna, wszystkie z wyjątkiem zmierzonej wartości ręcznej z dokładną regułą.	

Rysunek 10.1. Panele kart są często stosowane na witrynach zajmujących się handlem elektronicznym, na których informacje są prezentowane na osobnych kartach. W przedstawionym tu przykładzie (będącym jedynie fragmentem całej strony) przedstawiono karty zawierające opis produktu, jego specyfikację oraz informacje o sposobie dostawy; przy takim rozwiążaniu użytkownik może kliknąć kartę, by wyświetlić te informacje, które go interesują

Panele kart, tak jak wszystkie komponenty interfejsu użytkownika stron WWW, są tworzone przy użyciu kodów napisanych w językach HTML, JavaScript oraz za pomocą CSS. Każdy z tych elementów paneli kart można pisać na wiele sposobów, poniżej przedstawione zostało bardzo proste rozwiązanie.

Kod HTML

Panel kart składa się z dwóch podstawowych elementów: zakładek (czyli przycisków umieszczonych jeden obok drugiego w wierszu wyświetlonym na górze lub na dole komponentu) oraz kart (będących znacznikami `<div>` zawierającymi informacje, które chcemy prezentować). Dodatkowo komponent może zawierać kilka innych znaczników służących do zapewnienia właściwej jego organizacji oraz mających na celu ułatwienie kodu JavaScript, który go obsługuje. Oto one.

- **Element pojemnika.** Choć właściwie nie jest to niezbędne, jednak zastosowanie dodatkowego znacznika `<div>`, w którym będą umieszczone wszystkie zakładki i karty, może w wyraźny sposób oznaczyć początek i koniec komponentu oraz ułatwić tworzenie kodu JavaScript zwłaszcza wtedy, kiedy na jednej stronie ma znaleźć się więcej takich paneli karty. Oto podstawowy kod HTML takiego elementu:

```
<div class="tabbedPanels">  
  </div>
```

Dodanie do tego znacznika atrybutu `class` identyfikuje go i ułatwia tworzenie stylów określających postać elementów wewnętrz panelu oraz tworzenie selektorów jQuery odwołujących się do poszczególnych zakładek i kart. Jeśli na naszej stronie ma się znajdować tylko jeden panel kart, zamiast klasy można by określić identyfikator tego znacznika.

- **Zakładki.** Zazwyczaj tworzy się je w postaci listy wypunktowanej, zawierającej odnośniki:

```
<ul class="tabs">  
  <li><a href="#panel1">Informacje ogólne</a></li>  
  <li><a href="#panel2">Specyfikacja</a></li>  
  <li><a href="#panel3">Dostawa</a></li>  
</ul>
```

Odnośniki umieszczone w poszczególnych punktach listy odwołują się do identyfikatorów przypisanych poszczególnym kartom (opisanym poniżej). Utworzenie odnośnika od karty sprawia, że użytkownicy, którzy wyłączyli w swoich przeglądarkach obsługę języka JavaScript, będą mogli przeskoczyć prosto od wybranego fragmentu treści — kliknięcie takiego odnośnika powoduje przewinięcie strony do określonego miejsca.

Uwaga: Jeśli nie wiesz, jak tworzy się takie odnośniki, krótkie wyjaśnienie możesz znaleźć na stronie <http://www.yourhtmlsource.com/text/internallinks.html>.

- **Pojemnik kart.** Znacznik `<div>` zawierający wszystkie karty może przydać się do określania ich postaci w stylach CSS oraz odwoływania do nich przy użyciu jQuery:

```
<div class="panelContainer">  
  </div>
```

- **Karty.** To właśnie w nich umieszczane są właściwe informacje. Każda karta jest reprezentowana przez znacznik `<div>` i może zawierać dowolne treści: nagłówki, akapity, obrazki oraz inne znaczniki `<div>`. Każdy z tych znaczników powinien mieć unikalny identyfikator, odpowiadający identyfikatorowi podanemu w atrybutie `href` odnośników tworzących zakładki (patrz drugi punkt listy):

```
<div class="panel" id="panel1">  
    <!-- tu będzie treść karty -->  
</div>  
<div class="panel" id="panel2">  
    <!-- tu będzie treść karty -->  
</div>  
<div class="panel" id="panel3">  
    <!-- tu będzie treść karty -->  
</div>
```

Dodanie do każdego z tych znaczników jakieś klasy — na przykład `class="panel"` — także jest dobrym pomysłem, gdyż zapewnia dodatkowy sposób określania ich wyglądu i pobierania przy użyciu jQuery.

Wszystkie znaczniki `<div>` poszczególnych kart są umieszczane wewnętrz nadziennego znacznika `<div>` pełniącego rolę pojemnika. Kompletna struktura kodu HTML panelu kart ma zatem następującą postać:

```
<div class="tabbedPanels">  
    <ul class="tabs">  
        <li><a href="#panel1">Informacje ogólne</a></li>  
        <li><a href="#panel2">Specyfikacja</a></li>  
        <li><a href="#panel3">Dostawa</a></li>  
    </ul>  
    <div class="panelContainer">  
        <div class="panel" id="panel1">  
            <!-- tu będzie treść karty -->  
        </div>  
        <div class="panel" id="panel2">  
            <!-- tu będzie treść karty -->  
        </div>  
        <div class="panel" id="panel3">  
            <!-- tu będzie treść karty -->  
        </div>  
    </div>  
</div>
```

Kod CSS

Arkusz stylów pozwoli nadać nagłówkom kart wygląd zakładek (przycisków umieszczonych tuż obok siebie), a także sprawi, że same karty będą wyglądały jako spójna całość, w której treść będzie się łączyć z zakładką.

- **Pojemnik.** Nie musimy w żaden sposób określić wyglądu znacznika `<div>`, wewnętrz którego są umieszczone wszystkie karty (w rzeczywistości ten znacznik w ogóle nie jest potrzebny). Jednak może się przydać, gdybyśmy chcieli ograniczyć szerokość całego panelu na przykład po to, by umieścić go obok jakiegoś innego elementu strony bądź umieścić obok siebie dwa takie panele. W takim przypadku w stylu odnoszącym się do tego znacznika moglibyśmy określić jego szerokość w następujący sposób:

```
.tabbedPanels {  
    width: 50%;  
}
```

- **Lista wypunktowana oraz jej elementy.** Ponieważ listy wypunktowane są zazwyczaj nieco wcięte, zatem musimy z niej usunąć wszelkie wypełnienia zarówno z lewej, jak i z prawej strony. Co więcej, aby zakładki były rozmieszczone bok siebie, a nie jedna nad drugą, w elementach listy musimy zastosować właściwość `float`. I w końcu nie możemy zapomnieć o usunięciu punktorów, które są standardowo wyświetlane z lewej strony każdego punktu listy. Wszystkie te zadania realizują dwa poniższe style:

```
.tabs {  
    margin: 0;  
    padding: 0;  
}  
.tabs li {  
    float: left;  
    list-style: none;  
}
```

Uwaga: Przedstawiony tu kod CSS odnosi się do kodu HTML z poprzedniego punktu rozdziału. Innymi słowy, reguła `.tabs` odwołuje się do listy wypunktowanej — `<ul class="tabs">` — natomiast reguła `.tabs li` — do znaczników `` umieszczonych wewnętrz tej listy.

- Same **zakładki** są reprezentowane przez znaczniki `<a>` umieszczone wewnętrz punktów listy, czyli znaczników ``. W stylu określającym ich postać na pewno trzeba będzie odpowiednio ustawić kilka właściwości. Przede wszystkich chcemy usunąć podkreślenie, którym zazwyczaj są oznaczane wszystkie odnośniki, oprócz tego ich właściwości `display` przypiszemy wartość `block`, by można było określić ich marginesy i wypełnienia. Oto styl określający postać zakładek:

```
.tabs a {  
    display: block;  
    text-decoration: none;  
    padding: 3px 5px;  
}
```

Oczywiście, można przypuszczać, że będziesz chciał uzupełnić tę regułę stylem o dodatkowe właściwości, by zakładki wyglądały naprawdę wspaniale. Móglbyś na przykład ożywić je, określając kolor tła, zmienić czcionkę, jej kolor i wielkość, by tekst zakładek wyróżniał się wśród pozostałej treści kart.

- **Aktywna zakładka.** Bardzo dobrym pomysłem jest wyróżnienie zakładki skojarzonej z aktualnie widoczną kartą. To rodzaj sygnału „jesteś tutaj”, który wizualnie identyfikuje informacje prezentowane na karcie. Popularnym rozwiązaniem stosowanym w tym celu jest utworzenie stylu, który przy użyciu jQuery zostanie dodany do zakładki po jej kliknięciu. Nie ma żadnych obowiązkowych właściwości, które musielibyśmy umieszczać w tym stylu, jednak warto nadać zakładce taki sam kolor tła, który ma powiązana z nią karta (a jednocześnie zapewnić, by pozostałe zakładki miały inny kolor tła), gdyż dzięki temu zakładka oraz karta będą tworzyły wizualną całość:

```
.tabs a.active {  
    background-color: white;  
}
```

Wskazówka: Często stosowanym rozwiązańiem jest dodawanie obramowań wokół zakładek i kart. Po kliknięciu zakładki ukrywamy jej dolne obramowanie, co sprawia wrażenie, jakby zakładka została zespłoniona z kartą (patrz rysunek 10.1). Aby powstało takie rozwiązanie, na początek w regule `.tabs a` należy dodać właściwość `border` oraz przypisać dolnemu marginesowi (`margin-bottom`) wartość `-1px`. Zastosowanie wartości ujemnej spowoduje przesunięcie zakładki o jeden piksel w dół, co sprawi, że będzie ona zachodzić na kartę. Dodatkowo w regule `.tabs a.active` należy nadać dolnej krawędzi obramowania kolor odpowiadający kolorowi tła kart. W ten sposób, choć krawędź obramowania wciąż będzie wyświetlane, ze względu na to, że będzie ono mieć ten sam kolor, co tło karty, a dodatkowo będzie zachodzić na jej obramowanie, będzie się wydawało, że zakładka i karta stanowią jedność. (Aby takie rozwiązanie działało w przeglądarce Internet Explorer 8 oraz jej wcześniejszych wersjach, konieczne jest także dodanie do reguły stylu właściwości `position:relative`). W końcu, możemy także dodać obramowanie do pojemnika zawierającego karty — powinno ono mieć taki sam styl, grubość i kolor, co obramowanie użyte w stylu `.tabs a`. Ostateczny efekt zastosowania takich stylów można zobaczyć w przykładzie zamieszczonym na stronie 320.

- **Pojemnik kart.** Bardzo ważny jest styl określający postać znacznika `<div>`, wewnątrz którego są umieszczone poszczególne karty i ich zawartość. Ponieważ w stylu dla zakładek użyliśmy właściwości `float:left` (aby przeglądarka wyświetliła je jedną obok drugiej), zatem musimy zadbać, by dalsza zawartość naszego komponentu była prawidłowo wyświetlana poniżej zakładek. W przeciwnym razie przeglądarka spróbuje wyświetlić ją z ich prawej strony.

```
.panelContainer {  
    clear: left;  
}
```

Dodatkowo tego stylu można użyć w celu określenia postaci kart. Ponieważ pojemnik ten tworzy prostokąt wokół wszystkich kart, można w nim określić kolor tła, obramowanie, wypełnienie i tak dalej.

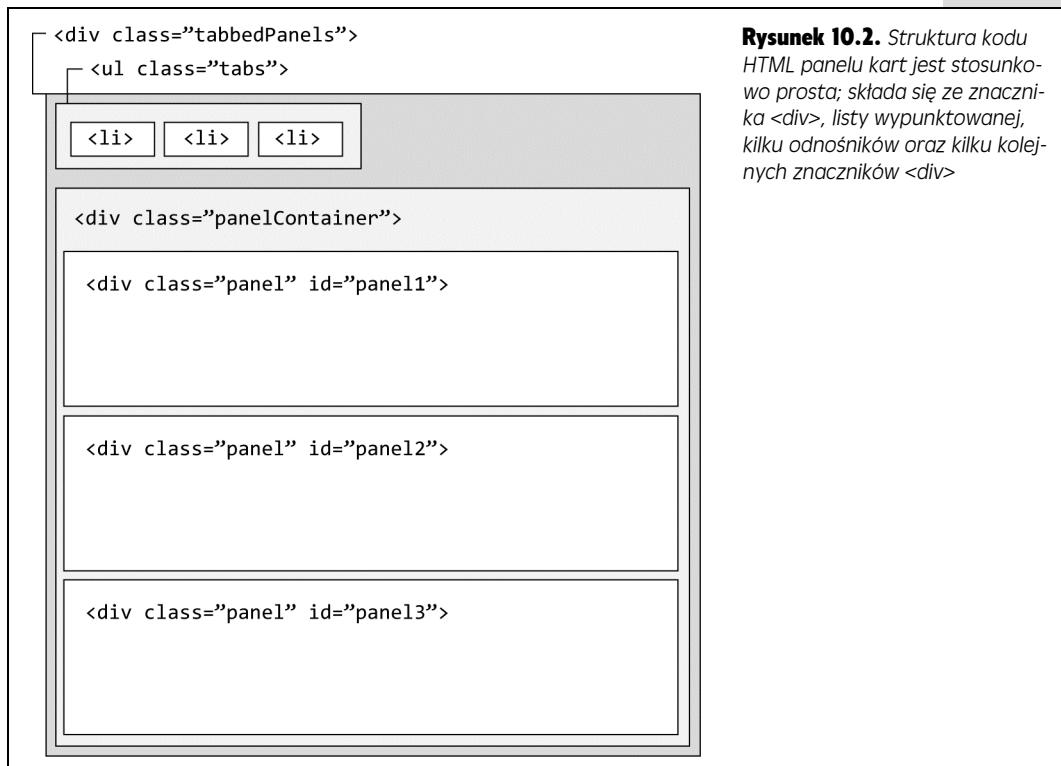
- **Karty.** Zgodnie z informacjami podanymi w poprzednim punkcie listy, do określenia podstawowych aspektów wyglądu kart, takich jak obramowanie, kolor tła i podobne, można użyć reguły stylów odnoszącej się do pojemnika, w którym są one umieszczone. Gdy jednak będziemy chcieli, możemy także określić regułę stylów odnoszącą się do poszczególnych znaczników `<div>` tworzących karty. Wystarczy w tym celu zdefiniować regułę stylu z selektorem `.panel`.
- **Zawartość kart.** Do określenia postaci zawartości umieszczonej na kartach można zastosować selektory elementów potomnych, które pozwalają odwoływać się do znaczników wewnętrz elementów `<div>` tworzących karty. Aby na przykład określić postać znaczników `<h2>` umieszczonych wewnętrz kart i wyświetlić ich zawartość czcionką Arial, w kolorze pomarańczowym, moglibyśmy użyć następującego stylu:

```
.panel h2 {  
    color: orange;  
    font-family: Arial, Helvetica, sans-serif  
}
```

Podobnie, by określić postać akapitów na kartach, należałoby użyć selektora w postaci `.panel p`.

Kod JavaScript

Przygotowaliśmy już kod HTML oraz arkusz stylów CSS i możemy zobaczyć zakładki wyświetcone w rzędzie u góry komponentu oraz trzy znaczniki <div> (karty), umieszczone poniżej, jeden nad drugim (patrz rysunek 10.2). Podstawowy wygląd komponentu jest zatem zgodny z naszymi zamierzeniami. Musimy jeszcze napisać kod JavaScript, który będzie obsługiwał otwieranie i zamykanie kart, zmieniał klasy oraz pozwalał na wyróżnianie aktywnej zakładki i przywracanie pozostałych do standardej postaci. Oto czynności, jakie trzeba wykonać.



Rysunek 10.2. Struktura kodu HTML panelu kart jest stosunkowo prosta; składa się ze znacznika <div>, listy wypunktowanej, kilku odnośników oraz kilku kolejnych znaczników <div>

1. Dodać obsługę zdarzeń click do odnośników w zakładkach.

Paneli kart są nierozerwalnie związane z iteracją użytkownika z zakładkami — kliknięcie pierwszej zakładki powoduje wyświetlenie pierwszej karty, kliknięcie innej — wyświetlenie odpowiadającej jej karty.

2. Dodać funkcję anonimową obsługującą zdarzenia click, która:

- Ukryje aktualnie widoczną kartę,
- Usunie klasę active z wybranej wcześniej zakładki,
- Doda klasę active do klikniętej zakładki,
- Wyświetli kartę skojarzoną z klikniętą zakładką.

3. Zgłoś zdarzenie `click` dla pierwszej zakładki.

Ten krok jest konieczny, gdyż w momencie wyświetlania strony widoczne są wszystkie karty, a żadna zakładka nie jest wyróżniona. Oczywiście, można napisać kod, który wyróżni pierwszą zakładkę oraz ukryje wszystkie karty z wyjątkiem pierwszej, jednak takie rozwiązanie nie jest konieczne — mamy do dyspozycji funkcję anonimową obsługującą zdarzenia `click` (patrz krok 2.), która zrobi to za nas. Możemy się zatem ograniczyć do programowego „kliknięcia” pierwszej zakładki, co spowoduje wykonanie tej funkcji.

Tak ogólnie wygląda kod, który musimy napisać. Napiszesz go krok po kroku w ramach przykładu, przedstawionego w następnym punkcie rozdziału.

Przykład — panel kart

Skoro już rozumiesz podstawowe założenia związane z tworzeniem panelu kart, tu znajdziesz opis czynności, jakie musisz wykonać, by go ostatecznie uruchomić. W tym przykładzie dodasz kody CSS oraz JavaScript, które przekształcą prostą listę odnośników przedstawioną na stronie 316 w interaktywny pasek nawigacyjny.

Uwaga: Informacje dotyczące pobierania przykładów do książki można znaleźć na stronie 43.

1. W edytorze tekstów otwórz plik `tabs.html` umieszczony w katalogu **R10**.

Plik `tabs.html` zawiera kod HTML opisany na stronie 315: nadzędny znacznik `<div>` całego panelu karty, wypunktowaną listę odnośników pełniących role zakładek, kolejny znacznik `<div>` zawierający karty oraz po jednym znaczniku `<div>` dla każdej karty. Znajdziesz w nim także podstawowe style CSS. Jeśli wyświetlisz ten plik w przeglądarce, zobaczysz trzy zakładki i zawartość trzech kart (wszystkie te karty są rozmieszczone w pionie, jedna nad drugą).

Uwaga: Starając się w możliwie jak największym stopniu poprawić przejrzystość przykładu, używany kod CSS umieściliśmy bezpośrednio w kodzie HTML strony, w formie arkusza wpisanego. Jeśli masz zamiar wielokrotnie korzystać z niego podczas tworzenia własnych paneli kart, umieść go w zewnętrznym pliku CSS.

Plik biblioteki jQuery został już dołączony do strony, a w sekcji nagłówka znajduje się wywołanie funkcji `$(document).ready()`. Kolejnym krokiem, jaki wykonasz, będzie ukrycie kart.

2. Kliknij puste miejsce wewnętrz funkcji `$(document).ready()` i wewnętrz niej dodaj poniższy kod wyróżniony pogrubieniem:

```
$(document).ready(function() {  
    $('.tabs a').click(function() {  
        // koniec funkcji click  
    }); // koniec funkcji ready
```

Wywołanie `$('.tabs a')` pobiera wszystkie znaczniki `<a>` umieszczone wewnątrz elementu klasy `tabs` (czyli naszej wypunktowanej listy). (Funkcja `click()`

została opisana na stronie 175). Aktualnie dysponujesz pustą funkcją anonimową, musisz zatem uzupełnić jej kod. Zaczniesz od prostej instrukcji, która pozwoli poprawić wydajność działania kodu.

3. Wewnątrz funkcji anonimowej wpisz poniższy, pogrubiony kod:

```
$('.tabs a').click(function() {  
    $this = $(this);  
}); // koniec funkcji click
```

Zgodnie z informacjami podanymi na stronie 162, wyrażenie `$(this)` stosowane wewnątrz funkcji anonimowej obsługującej zdarzenia pozwala odwołać się do elementu, do którego zdarzenie zostało skierowane — w naszym przypadku odwołuje się ono do zakładki klikniętej przez użytkownika. Za każdym razem, gdy używamy wyrażenia `$()` do pobrania elementu, wywołujemy funkcję jQuery, zmuszając tym samym przeglądarkę do wykonania wielu wierszy kodu JavaScript. Jeśli wewnątrz jakiejś funkcji będziemy wielokrotnie używali jakiegoś selektora jQuery, doskonałym pomysłem będzie zapisanie go w zmiennej. W powyższym przykładzie `this` jest zwyczajną zmienną zdefiniowaną przez programistę (czyli przez Ciebie).

Zapisanie wartości wyrażenia `$(this)` w zmiennej oznacza, że gdy tylko będziesz chciał odwołać się do odnośnika, wystarczy skorzystać ze zmiennej `this` — nie będziesz musiał ponownie pobierać go przy użyciu selektora jQuery. Innymi słowy, jeśli w kodzie funkcji dwukrotnie pojawi się wywołanie `$(this)`, będzie to oznaczać, że przeglądarka musi dwukrotnie wykonać funkcję jQuery w celu pobrania tego samego elementu. Jeśli za pierwszym razem zapiszesz wartość `$(this)` w zmiennej — `this` — będziesz mógł z niej wielokrotnie korzystać bez zmuszania przeglądarki do wykonywania jakichkolwiek dodatkowych czynności (bardziej szczegółowe informacje o zaletach, jakie daje zapisywanie selektorów jQuery w zmiennych, można znaleźć na stronie 422).

Teraz zajmiesz się ukryciem kart i aktywacją klikniętej zakładki.

4. Wpisz kod z wierszy 3. i 4. poniższego fragmentu:

```
1  $('.tabs a').click(function() {  
2      $this = $(this);  
3      $('.panel').hide();  
4      $('.tabs a.active').removeClass('active');  
5  }); // koniec funkcji click
```

Wiersz 3. powoduje ukrycie wszystkich kart. Ponieważ każda z nich jest znacznikiem `<div>` należącym do klasy `panel`, selektor `$('.panel')` pobiera je wszystkie, a wywołanie funkcji `.hide()` (patrz strona 198) powoduje ich ukrycie. Musisz to zrobić, gdyż w przeciwnym razie po otwarzeniu jednej karty poprzednia pozostałaaby widoczna.

Wiersz 4. usuwa klasę `active` ze wszystkich zakładek — odnośników umieszczonych w elemencie należącym do klasy `tabs`. Na stronie 317 wyjaśniliśmy, że utworzenie klasy `active` pozwoli zmienić wygląd zakładki klikniętej przez użytkownika (by wyglądała jak wizualny sygnał „jestes tutaj”). Oczywiście, kiedy użytkownik kliknie zakładkę, by ją aktywnić, trzeba usunąć klasę `active` z zakładki, która do tej pory była aktywna. I właśnie to robi kod z wiersza 4., używając przy tym funkcji `removeClass()` (opisanej na stronie 155). Teraz zajmiesz się wyróżnieniem klikniętej zakładki.

5. Dodaj kod umieszczony w wierszu 5.:

```
1  $('.tabs a').click(function() {  
2      $this = $(this);  
3      $('.panel').hide();  
4      $('.tabs a.active').removeClass('active');  
5      $this.addClass('active').blur();  
6  }); // koniec funkcji click
```

Pamiętasz zapewne, że `$this` jest zmienną utworzoną w wierszu 2., która zawiera odwołanie do klikniętego odnośnika. A zatem wywołanie `$this.addClass('active')` dodaje do tego odnośnika klasę active — przeglądarka użyje jej do określenia postaci klikniętej zakładki. Umieszczone na końcu wiersza wywołanie funkcji `.blur()` korzysta z techniki tworzenia sekwencji wywołań, jaką daje biblioteka jQuery (która została opisana na stronie 149). To zwyczajna funkcja wywoływanego po wykonaniu funkcji `addClass()`. Funkcja ta usuwa ognisko wprowadzania z wybranego elementu (odnośnika lub pola formularza). W naszym przypadku jej wywołanie sprawi, że przeglądarka nie będzie wyświetlać wokół tekstu klikniętego odnośnika cienkiej, przerwywanej linii. Gdybyśmy jej nie wywołali, zakładka nie wyglądałaby równie dobrze.

I to już prawie wszystko... jeszcze tylko musisz wyświetlić kartę.

6. Dodaj kod umieszczony w wierszach 6. i 7.:

```
1  $('.tabs a').click(function() {  
2      $this = $(this);  
3      $('.panel').hide();  
4      $('.tabs a.active').removeClass('active');  
5      $this.addClass('active').blur();  
6      var panel = $this.attr('href');  
7      $(panel).fadeIn(250);  
8  }); // koniec funkcji click
```

Każda zakładka jest w rzeczywistości odnośnikiem wskazującym powiązaną z nią kartę. Pamiętasz zapewne, że kod HTML karty wygląda w następujący sposób: `<div id="panel1" class="panel">`; natomiast kod HTML odpowiadającej mu zakładki to: ``. Zwrót uwagę, że zawartość atrybutu `href` odnośnika wygląda dokładnie tak samo jak selektor identyfikatora CSS. Ponieważ jQuery korzysta z selektorów CSS do pobierania elementów stron, zatem wystarczy pobrać wartość atrybutu `href` i użyć jej do pobrania karty, którą chcemy wyświetlić. W wierszu 6. tworzymy nową zmienną — `panel` — w której zapisujemy wartość atrybutu `href` odnośnika (używana przy tym funkcja `attr()` jQuery została opisana na stronie 159).

W wierszu 7. korzystamy z odczytanej wcześniej wartości do pobrania karty i stopniowego jej wyświetlenia (używamy przy tym funkcji `fadeIn()` opisanej na stronie 200). Moglibyśmy ją zastąpić jakkolwiek inną funkcją jQuery generującą efekty wizualne, taką jak `show()`, `slideDown()` bądź `animate()`.

Teraz, gdy już prawie cały kod jest gotowy, musimy programowo wygenerować zdarzenie `click` podczas wczytywania strony, spowoduje to wywołanie funkcji, ukrycie kart, wyróżnienie pierwszej zakładki i wyświetlenie skojarzonej z nią karty. Na szczęście, jQuery pozwala w bardzo prosty sposób symulować zgłoszenie zdarzenia.

7. Poniżej funkcji `click()` dodaj jeszcze jeden wiersz, powodujący zgłoszenie zdarzenia `click` dla pierwszej zakładki.

```

1  $('.tabs a').click(function() {
2      $this = $(this);
3      $('.panel').hide();
4      $('.tabs a.active').removeClass('active');
5      $this.addClass('active').blur();
6      var panel = $this.attr('href');
7      $(panel).fadeIn(250);
8  }); // koniec funkcji click
9  $('.tabs li:first a').click();

```

Jak widać, w wybranym wierszu zastosowaliśmy złożony selektor — `.tabs li:first a` — który służy do pobrania pierwszej zakładki. Selektor ten (podobnie jak wszystkie inne selektory elementów potomnych) należy analizować od strony prawej do lewej. Litera `a` umieszczona z prawej strony selektora oznacza, że chodzi o pobranie znacznika `<a>`. Umieszczony w środkowej części łańcucha `li:first` korzysta z pseudoelementu `first` pobierającego pierwszy element potomny. W naszym przypadku cały ten fragment selektora pozwala pobrać znacznik `` będący pierwszym dzieckiem innego elementu. Ponieważ zakładki zostały utworzone przy użyciu listy, zatem `li:first` odpowiada pierwszemu elementowi tej listy (czyli pierwszej zakładce). I w końcu fragment `.tabs` pozwoli mieć pewność, że pobierzemy odnośnik umieszczony na liście będącej częścią naszego panelu kart. Zabezpiecza on przed przypadkowym pobraniem odnośnika zapisanego na innej liście wypunktowanej (na przykład pełniącej rolę paska nawigacyjnego), umieszczonej w innym miejscu strony.

Po pobraniu interesującego nas elementu wywołujemy funkcję `click()`, jednak w tym przypadku nie służy ona do określenia funkcji anonimowej obsługującej zdarzenia `click`, lecz do jego zgłoszenia. Innymi słowy, wywołanie umieszczone w wierszu 9. oznacza: „hej, przeglądarko — kliknij pierwszą zakładkę”. Zgłoszenie tego zdarzenia powoduje całą sekwencję czynności: ukrycie kart, wyróżnienie zakładki i stopniowe wyświetlenie odpowiedniej karty. O rany! To już prawie wszystko. Gdybyś jednak stronę w tej postaci wyświetlił w przeglądarce, zauważałbyś pewnie jeden, niewielki problem. Ponieważ zakładki są odnośnikami, gdy zatem okno przeglądarki będzie niewielkie, można zaobserwować, że po kliknięciu zakładki przeglądarka nie tylko wyświetli odpowiednią kartę, lecz także do niej przeskoczy. Musisz zatem poinstruować przeglądarkę, by nie przechodziła do miejsca docelowego odnośnika.

8. Na końcu anonimowej funkcji obsługującej zdarzenia `click` (patrz wiersz 9.) dodaj instrukcję `return false;`. A tak powinna wyglądać ostateczna wersja kodu:

```

1  $(document.ready(function() {
2     ($('.tabs a').click(function() {
3          $this = $(this);
4          $('.panel').hide();
5          $('.tabs a.active').removeClass('active');
6          $this.addClass('active').blur();
7          var panel = $this.attr('href');
8          $(panel).fadeIn(250);
9          return false;
10     }); // koniec funkcji click
11     $('.tabs li:first a').click();
12 )); // koniec funkcji ready

```

9. Zapisz plik i wyświetl stronę w przeglądarce.

Dokończona przykładowa strona powinna wyglądać tak, jak przedstawiona na rysunku 10.3. Można ją poszerzyć o więcej zakładek i kart, dodając do listy kolejne punkty z odnośnikami wskazującymi kolejne znaczniki <div> reprezentujące nowe karty.

The screenshot shows a web page titled "JavaScript i jQuery" with a subtitle "Nieoficjalny podręcznik". Below the title, there's a heading "Panel z kartami". A horizontal navigation bar contains three tabs: "Zakładka 1" (which is active), "Zakładka 2", and "Zakładka 3". The main content area is titled "Panel 1 content". It contains a paragraph of Latin text and two small images: one of a building and another of a person. Below this, there's another section with a building image and more Latin text. At the bottom of the page, there's a footer bar with the text "JavaScript i jQuery. Nieoficjalny podręcznik, autor David McFarland. Wydane przez Helion".

Zakładka 1 **Zakładka 2** **Zakładka 3**

Panel 1 content

At vero eos et accusamus et iusto odio dignissimos ducimus qui blanditiis praesentium voluptatum deleniti atque corrupti quos dolores et quas molestias excepturi sint occaecati cupiditate non provident, similius sunt in culpa qui officia deserunt mollitia animi, id est laborum et dolorum fuga.

At vero eos et accusamus et iusto odio dignissimos ducimus qui blanditiis praesentium voluptatum deleniti atque corrupti quos dolores et quas molestias excepturi sint occaecati cupiditate non provident, similius sunt in culpa qui officia deserunt mollitia animi, id est laborum et dolorum fuga.

At vero eos et accusamus et iusto odio dignissimos ducimus qui blanditiis praesentium voluptatum deleniti atque corrupti quos dolores et quas molestias excepturi sint occaecati cupiditate non provident, similius sunt in culpa qui officia deserunt mollitia animi, id est laborum et dolorum fuga. Et harum quidem rerum facilis est et expedita distinctio. Nam libero tempore, cum soluta nobis est eligendi optio cumque nihil impedit quo minus id quod maxime placeat facere possimus. omnis voluptas assumenda est, omnis dolor repellendus. Temporibus autem quibusdam et aut officiis debitis aut rerum necessitatibus saepe eveniet ut et voluptates repudiandae sint et molestiae non recusandae. Itaque earum rerum hic tenetur a sapiente delectus, ut aut reiciendis voluptatibus maiores alias consequatur aut perferendis doloribus asperiores repellat.

JavaScript i jQuery. Nieoficjalny podręcznik, autor David McFarland. Wydane przez Helion.

Rysunek 10.3. Panele zakładek stanowią elegancki sposób zapewnienia użytkownikom łatwego dostępu do wielu informacji przy jednoczesnym oszczędzaniu miejsca na stronie

Uwaga: Pełną wersję tego przykładu — *complete_tabs.html* — można znaleźć w katalogu R10. Dodatkowo umieściliśmy tam także bardziej złożoną wersję tego samego przykładu, pozwalającą na umieszczanie na jednej stronie kilku paneli kart. Znajdziesz ją w pliku *complete_complex_tabs.html*. Zostały w niej zastosowane zaawansowane funkcje jQuery służące do poruszania się po drzewie DOM strony, opisane w dalszej części książki, na stronie 432.

WIEDZA W PIŁUŁCE

Projekt jQuery UI

Bardziej zaawansowaną wersję panelu kart można znaleźć w projekcie jQuery UI. Jest to oficjalny projekt zespołu jQuery, którego celem jest pisanie wtyczek rozwiązujących popularne zadania związane z tworzeniem interfejsu użytkownika; są to „akordeony” (ang. *accordion*), karty, okna dialogowe, kalendarze oraz elementy stron, które można przeciągać. Uczestnicy projektu dążą do opracowania jednej wtyczki, która zapewniałaby możliwość rozwiązania większości problemów napotykanych podczas tworzenia interfejsu użytkownika aplikacji internetowych. Projekt ten ma swoją własną witrynę WWW (<http://jqueryui.com/>), na której można znaleźć najnowszą wersję kodu, przykłady oraz odnośnik do dokumentacji umieszczonej na głównej witrynie jQuery.

Projekt jQuery UI udostępnia wiele narzędzi dla projektantów stron, a nawet obsługuje tematy CSS — pozwalające na zapewnienie wspólnego, spójnego wyglądu wszystkich elementów jQuery UI. Projekt ten jest stosunkowo złożony i w jego skład wchodzi wiele różnych elementów. Pliki projektu można także modyfikować i dostosowywać do własnych potrzeb — usuwać z nich

komponenty, których nie będziemy potrzebować, ograniczając tym samym wielkość pliku i skracając czas jego pobierania. Można nawet tworzyć własne tematy CSS dopasowujące wygląd komponentów jQuery UI do wyglądu naszej witryny. Cały ten proces ułatwia specjalne narzędzie służące do przygotowywania pliku jQuery UI, dostępne na stronie <http://jqueryui.com/download>.

W poprzednim wydaniu tej książki wtyczka jQuery UI była używana w kilku rozdziałach, jednak w międzyczasie zespół jej twórców zdecydował się na całkowite przepisanie jej kodu i dodanie wielu nowych, fascynujących komponentów oraz możliwości. Niestety, w czasie pisania tej książki najnowsza wersja wtyczki jQuery UI (numer 1.9) nie była jeszcze dostępna, dlatego też nie warto tracić czasu na poznawanie grupy komponentów, które i tak niebawem zostaną zastąpione. Dlatego w tym wydaniu książki jQuery UI i jej komponenty nie zostały opisane.

Jednak jQuery UI zapowiada się doskonale i zdecydowanie warto się nią interesować. Zajrzyj zatem na witrynę jQuery UI, by przekonać się, czy wersja 1.9 została już opublikowana; jeśli tak, sprawdź ją koniecznie.

Dodawanie sliderów

Kolejnym narzędziem używanym przez projektantów stron do walki ze zbyt wielką liczbą prezentowanych informacji są slidery (ang. *content slider*) — proste komponenty interfejsu użytkownika, prezentujące jedno wybrane zdjęcie lub fragment treści, wybrane z większej grupy. Wiele witryn zawierających bardzo duże ilości informacji, takich jak witryna firmy Microsoft, korzysta ze sliderów w celu prezentowania zdjęć, tekstów oraz odnośników w niewielkich fragmentach, które są przesuwane po ekranie i zastępowane innymi (patrz rysunek 10.4). Slider przypomina nieco panel kart, ale w jego przypadku poszczególne karty mają zazwyczaj tę samą wielkość, są wyświetlane i chowane z wykorzystaniem animacji, które je przesuwają na ekranie, a ich pojawianie się i znikanie jest zazwyczaj obsługiwane przy użyciu liczników czasu. Są one powszechnie stosowane na stronach głównych, gdyż mają bardzo atrakcyjną postać, a jednocześnie pozwalają na zachowanie prostoty strony. Często używa się ich także jako zwiastunów reklamujących treści lub produkty opisane na innych stronach witryny. Kliknięcie karty takiego slidera zazwyczaj powoduje przejście na inną stronę.



Rysunek 10.4. Aby zminimalizować natłok informacji na ekranie, witryny, takie jak Microsoft.com, korzystają ze sliderów (na rysunku jeden został zaznaczony czarną ramką), wewnątrz których prezentowane są obrazki lub fragmenty treści, po jednym w danej chwili. W przedstawionym przykładzie zaznaczony obrazek może zostać wysunięty na bok, odsłoni się wtedy inny obrazek z kolejnymi informacjami

Utworzenie slidera wymaga opanowania kilku możliwości języka JavaScript i biblioteki jQuery, a konkretnie tworzenia animacji, korzystania z liczników czasu oraz manipulacji kodem HTML i CSS. Choć — oczywiście — można utworzyć swój własny slider, jednak istnieje sporo wtyczek jQuery udostępniających wiele przydatnych możliwości. Jedną z najbardziej wszechstronnych wtyczek tego typu jest AnythingSlider (kod można pobrać ze strony <https://github.com/ProLoser/AnythingSlider>).

Stosowanie slidera AnythingSlider

Do działania wtyczki AnythingSlider potrzebujemy kilku plików; są to jQuery (co chyba oczywiste), zewnętrzny plik JavaScript z kodem obsługującym slider, plik CSS ze stylami określającymi podstawowy wygląd slidera i stosowanych w nim efektów oraz obrazek z kontrolkami slidera (przyciskami następny i poprzedni). Pliki te można pobrać ze strony <https://github.com/ProLoser/AnythingSlider/>. (Dodaliśmy je także do przykładów dołączonych do tej książki i umieściliśmy w katalogu z przykładami do tego rozdziału). Aby skorzystać z tej wtyczki, należy wykonać następujące, bardzo proste czynności.

1. Dołączyć do strony plik CSS *anythingslider.css*.

Ten zewnętrzny arkusz stylów CSS określa sposób formatowania przycisków nawigacyjnych slidera, jak również ukryte style służące do właściwego rozmieszczania poszczególnych kart. Istnieje możliwość określania postaci podstawowych elementów slidera poprzez modyfikowanie reguł CSS umieszczonej w tym pliku.

2. Dołączyć do strony plik biblioteki jQuery.

Biblioteka jQuery udostępnia wszystkie podstawowe narzędzia niezbędne do utworzenia i obsługi slidera AnythingSlider. Podobnie jak podczas korzystania ze wszystkich innych wtyczek jQuery, także i teraz na początek należy wczytać sam plik biblioteki. Jeśli wtyczka zostanie wczytana przed biblioteką jQuery, slider nie będzie działał.

3. Dołączyć plik JavaScript z kodem wtyczki AnythingSlider.

Ten plik zawiera cały kod odpowiadający za przekształcenie kodu HTML w interaktywny slider.

4. Dodać kod HTML.

AnythingSlider nie wymaga żadnego skomplikowanego kodu HTML. Należy tylko określić pojemnik — znacznik `<div>` z identyfikatorem slidera: `<div id="slider">` — a wewnętrznie umieścić po jednym znaczniku `<div>` dla każdej z kart. Jak widać, rozwiążanie to w znacznym stopniu przypomina panel kart (opisany na stronie 315).

5. Dodać znacznik `<script>`, a wewnątrz niego umieścić wywołanie funkcji `$(document).ready()`, w której z kolei wywołana zostanie funkcja slidera.

Jedną z ogromnych zalet korzystania z wtyczek jQuery jest to, że zazwyczaj kod, jaki w tym celu musimy napisać, jest bardzo krótki i prosty. W naszym przypadku wszystkim, czego potrzebujemy do utworzenia slidera, jest dopisanie poniżej odwołania dodanego w kroku 3. poniższego fragmentu kodu:

```
<script>
$(document).ready(function) {
    $('#slider').anythingSlider();
}); // koniec funkcji ready
</script>
```

Istnieje wiele różnych sposobów określania postaci slidera AnythingSlider, o czym przekonasz się dalej w tym rozdziale. Jednak najpierw sprawdź, jak taki slider działa.

Przykład — AnythingSlider

Utworzenie prostego slidera jest bardzo proste. Przekonasz się o tym, wykonując przedstawiony tu przykład. Możesz w tym celu wykorzystać dowolny edytor HTML.

Uwaga: Informacje dotyczące pobierania przykładów dołączonych do tej książki można znaleźć na stronie 43.

1. W edytorze tekstów otwórz plik `slider.html` umieszczony w katalogu R10.

Pierwszym krokiem będzie dodanie do strony pliku CSS wtyczki.

2. Kliknij pusty wiersz umieszczony poniżej wiersza `<link href=".../_css/site.css" rel="stylesheet">` i wpisz w nim:

```
<link rel="stylesheet" href="anythingSlider/anythingslider.css">
```

Ten wiersz kodu wczytuje arkusz stylów `anythingslider.css`, zawierający style określające postać slidera. Zawartości tego pliku przyjrzymy się dokładniej nieco później, kiedy będziemy zajmowali się aktualizacją wyglądu naszego komponentu. Kolejnym krokiem będzie dodanie do strony niezbędnych plików JavaScript.

3. W kolejnych dwóch wierszach wpisz poniższy kod:

```
<script src="../../_js/jquery-1.6.3.min.js"></script>
<script src="anythingSlider/jquery.anythingslider.min.js"></script>
```

Pierwszy z powyższych znaczników <script> powoduje wczytanie pliku biblioteki jQuery, natomiast drugi wczytuje plik wtyczki AnythingSlider. Kolejnym krokiem będzie dodanie do strony kodu HTML.

Wskazówka: W tym przykładzie zakładamy, że pliki JavaScript oraz CSS wtyczki są umieszczone w odłączonym katalogu *anythingSlider*, umieszczonym wewnątrz katalogu z przykładami do rozdziału — R10. Umieszczenie wszystkich plików wymaganych do prawidłowego działania wtyczki w osobnym katalogu jest doskonałym sposobem zagwarantowania, że wszystkie będą się znajdowały tam, gdzie powinny; takie rozwiązanie ułatwia także wykorzystanie wtyczki na innych witrynach. Jeśli podobały Ci się efekty działania wtyczki AnythingSlider, wystarczy, że skopiujesz katalog *anythingSlider* na swoją witrynę (umieść go w katalogu głównym bądź wewnątrz katalogu, gdzie przechowywane są pliki JavaScript).

4. W kodzie strony odszukaj znacznik nagłówka poziomu pierwszego — <h1>Anything Slider</h1> — i w wierszu poniżej niego wpisz:

```
<div id="slider">
</div>
```

Jak widać, użyliśmy tu znacznika <div> (czyli znacznika HTML służącego do definiowania regionów stron WWW). Ten konkretny znacznik będzie reprezentować sam komponent slidera. Wewnątrz niego umieścisz kolejne znaczniki <div> — po jednym dla każdej z kart slidera.

5. Wewnątrz elementu <div> slidera (czyli pomiędzy otwierającym znacznikiem <div> i zamkającym znacznikiem </div>) wpisz:

```
<div>
<a href="page1.html"></a>
</div>
```

Ten drugi znacznik <div> zawiera odnośnik i obrazek. Kliknięcie obrazka spowoduje przejście na inną stronę: takie rozwiązanie jest powszechnie stosowane w sliderach, które bardzo często spełniają rolę animowanych reklam na stronie. Każda karta slidera jest swoistym zwiastunem reklamującym inne treści, a następnie użytkownicy mogą ją kliknąć, by przejść do artykułu lub innego fragmentu witryny.

Gdy wykorzystamy slider AnythingSlider, w kartach będziemy mogli umieszczać dowolny kod HTML — nasze możliwości nie są ograniczone do jednego dużego zdjęcia. Można w nich umieszczać tekst, obrazki oraz inne znaczniki <div> — niemal wszystko, co tylko zechcemy.

6. Wewnątrz elementu <div> slidera dodaj kolejne dwa znaczniki <div>:

```
<div>
<a href="page2.html"></a>
</div>
<div>
```

```
<a href="page3.html"></a>
</div>
```

Te dwa zagnieżdżone znaczniki `<div>` reprezentują kolejne karty. Możesz ich dodać tyle, ile zechcesz. Teraz nadszedł czas, by zabrać się za pisanie kodu JavaScript.

7. W górnej części pliku, poniżej drugiego znacznika `<script>`, lecz przed zamkającym znacznikiem `</head>`, dodaj pusty wiersz i wpisz w nim:

```
<script>
$(document).ready(function() {
    $('#slider').anythingSlider();
});
</script>
```

Możesz w to uwierzyć lub nie, jednak wszystko, co musisz zrobić, to pobranie znacznika `<div>` slidera — `$('#slider')` — i wywołanie funkcji `anythingSlider()`. Całą resztą zajmie się już sama wtyczka.

8. Zapisz plik i wyświetl stronę w przeglądarce.

Strona powinna wyglądać tak, jak przedstawiona na rysunku 10.5. (Jeśli jednak tak nie wygląda, powinieneś ponownie sprawdzić jej kod. Możesz go porównać z plikiem `complete_slider.html`, zawierającym końcową, pełną wersję kodu przykładowu). Wypróbuj działanie elementów sterujących: strzałka w prawo powoduje wyświetlenie następnej karty, a strzałka w lewo — karty poprzedniej, przyciski z cyframi pozwalają przejść do konkretnej karty, a przycisk *Start* uruchamia automatyczny pokaz slajdów.

Modyfikowanie wyglądu slidera

Jak widać, stosowanie wtyczki AnythingSlider jest bardzo proste. Oczywiście, domyślny wygląd slidera nie musi wcale pasować do projektu naszej witryny i może się zdarzyć, że nie będziemy chcieli bądź potrzebowali wszystkich jego możliwości (takich jak automatyczny pokaz slajdów lub przyciski pozwalające przechodzić do następnej lub poprzedniej karty). Wygląd slidera AnythingSlider można zmieniać na kilka sposobów: modyfikując pliki graficzne, wprowadzając zmiany w arkuszach stylów oraz ustalając opcje wtyczki (ten sposób został opisany w następnym punkcie rozdziału).

Dzięki zastosowaniu techniki „sprajtów CSS” jeden plik graficzny spełnia wiele zadań — określa normalny i „wyróżniony” stan strzałek do przodu i do tyłu, zawiera tło przycisków z numerami kart oraz ustala postać przycisku *Start* (więcej informacji na temat tej techniki można znaleźć na stronie <http://css-tricks.com/css-sprites/>). Możesz otworzyć ten plik w dowolnym programie graficznym i zmodyfikować wygląd przycisków strzałek, każdy z nich ma wymiary 45×140 pikseli.

Można także modyfikować arkusz stylów i w ten sposób wypływać na postać pokazu slajdów. Poniżej zamieszczono listę najczęściej stosowanych zmian, z których, być może, będziesz chciał skorzystać.

JavaScript i jQuery

Nieoficjalny podręcznik

Wtyczka AnythingSlider



Rysunek 10.5. Przy użyciu wtyczki AnythingSlider można szybko utworzyć interaktywny pokaz slajdów, by zwrócić uwagę użytkownika na wybrane strony witryny lub prezentowane na niej produkty

- **Wysokość oraz szerokość slidera.** Pierwsza reguła stylu zapisana w pliku `anythingslider.css` — `#slider` — określa ogólną szerokość i wysokość komponentu. Szerokość (`width`) można zmieniać, by tworzyć szersze bądź węższe prezentacje; a wysokość (`height`) regulować, jeśli prezentowana zawartość ma inną wysokość niż domyślne 390 pikseli.
- **Kolor przycisków nawigacyjnych.** Przyciski z cyframi wyświetlane u dołu slidera są zazwyczaj zielone. Jeśli jednak nie podoba Ci się ten kolor, możesz zmodyfikować regułę stylu z bardzo złożonym selektorem `div .anythingSlider .activeSlider .anythingControls ul a.cur, div.anythingSlider.active .Slider .anythingControls ul a`. Zmień podany w niej kolor o wartości `#7C9127` na dowolny inny, lepiej odpowiadający projektowi Twojej witryny. Jeśli chciałbyś także określić kolor czcionki, dodaj do reguły właściwość `color`, na przykład:
`color: #F44439;`
- **Kolor przycisków wskazanych myszą.** Możesz także zmienić kolor tła przycisków nawigacyjnych, używaną w nich czcionkę oraz dowolne inne aspekty ich wyglądu; wystarczy w tym celu zmodyfikować regułę stylu z selektorem `div .anythingSlider .anythingControls ul a:hover`. W domyślnej postaci usuwa ona z przycisku obraz tła (cień).

- **Aktualnie wybrany przycisk nawigacyjny.** Aby określić styl wyróżniający przycisk skojarzony z aktualnie wybraną kartą, należy dodać do arkusza regułę z selektorem `div.anythingSlider .activeSlider .anythingControls ul a.cur` i określić w nim kolor tła, czcionkę i tak dalej. Ważne jest, by reguła ta była umieszczona w arkuszu stylów poniżej reguły opisanej w punkcie „Kolor przycisków nawigacyjnych”; alternatywnie możesz także zmodyfikować opisaną tam regułę stylu, usuwając z niej selektor `div.anythingSlider .activeSlider .anythingControls ul a.cur`. Ponieważ w stylu podanym wcześniej jest już określony kolor tła, zatem przesłoni on kolor podany w tej regule, chyba że zostanie ona umieszczona bliżej końca arkusza stylów.
- **Kolory przycisków rozpoczynających i zatrzymujących prezentację.** Postać przycisków służących do rozpoczętania i zatrzymywania automatycznej prezentacji kart jest kontrolowana przy użyciu dwóch stylów. Aby zmienić zielone tło oraz czcionkę przycisku rozpoczynającego prezentację, należy zmodyfikować regułę stylu z selektorem `div.anythingSlider .start-stop`. Z kolei modyfikując regułę z selektorem `div.anythingSlider .start-stop.playing`, można zmienić czerwony kolor przycisku przerywającego automatyczną prezentację.
- **Usunięcie cieni oraz inne zmiany wyglądu przycisków nawigacyjnych.** Jeśli nie podobają Ci się cienie widoczne przy przyciskach nawigacyjnych oraz obsługujących automatyczną prezentację, powinieneś zmodyfikować reguły stylów z selektorami `div.anythingSlider .anythingControls ul a` oraz `div.anythingSlider .start-stop`. Konkretnie rzecz biorąc, musisz usunąć z nich właściwość `border-image`. Możesz także zmodyfikować właściwości `border-radius`, `-moz-border-radius` oraz `-webkit-border-radius`, by całkowicie usunąć lub zmienić promień okrągłych wierzchołków tych przycisków. Ogólnie mówiąc, style te określają podstawowe aspekty wyglądu przycisków slidera, zatem warto z nimi eksperymentować, by przekonać się, jakie efekty można za ich pomocą uzyskać.
- **Zielone obramowania powyżej i poniżej prezentacji.** Powyżej oraz poniżej slidera wyświetlane jest zielone obramowanie o szerokości trzech pikseli. Aby je zmienić, należy zmodyfikować styl z selektorem `div.anythingSlider .anythingWindow`. Jeśli chcesz całkowicie usunąć obramowanie, usuń właściwości `border-top` oraz `border-bottom`; ewentualnie zmodyfikuj podane w nich wartości, by zmienić kolor bądź szerokość obramowania.
- **Położenie przycisków strzałek.** Położenie przycisków pozwalających na przejście do poprzedniej i następnej karty można kontrolować, modyfikując odpowiednio regułę stylu z selektorem `div.anythingSlider .back` (strzałka w lewo) oraz `div.anythingSlider .right` (strzałka w prawo). Oprócz tego, reguła z selektorem `div.anythingSlider .arrow` określa pewne wspólne aspekty wyglądu obu tych przycisków, w tym ich położenie pośrodku obszaru slidera. Gdybyś na przykład chciał wyświetlić te przyciski bliżej górnej krawędzi slidera, wystarczy, że zmodyfikujesz styl `div.anythingSlider .arrow`, zmieniając właściwość `top: 50%` na `top: 20%` bądź nawet na wartość bezwzględną wyrażoną w pikselach — `top: 45px`.

Modyfikacja działania slidera

Już sama modyfikacja arkusza stylów CSS pozwala na wprowadzenie wielu zmian w wyglądzie slidera. By jednak wprowadzić fundamentalne zmiany w sposobie działania tej wtyczki, konieczne jest określenie wartości kilku jej właściwości. W tym celu trzeba przekazać do niej odpowiedni literał obiektowy (patrz strona 158):

```
{
    buildArrows : false,
    startText : "Uruchom prezentację",
    stopText : "Zatrzymaj prezentację"
}
```

W tym przykładzie `buildArrows` jest właściwością wtyczki, natomiast `false` — przypiswaną jej wartością. Zastosowanie tej konkretnej wartości sprawi, że wtyczka nie wyświetli w sliderze strzałek do przechodzenia pomiędzy kolejnymi kartami. Za każdą parą nazwa-wartość, z wyjątkiem ostatniej, należy umieścić przecinek (zwróć uwagę, że nie ma go za parą `stopText : "Zatrzymaj prezentację"`).

Ten literał obiektowy należy następnie przekazać w wywołaniu funkcji `anythingSlider()`. Oto przykład:

```
$('#slider').anythingSlider({
    buildArrows : false,
    startText : "Uruchom prezentację",
    stopText : "Zatrzymaj prezentację"
});
```

Poniżej przedstawiono kilka najbardziej przydatnych opcji.

- **Ukrycie przycisków nawigacyjnych.** Aby ukryć przyciski ze strzałkami, należy przypisać wartość `false` właściwości `buildArrows`:

```
buildArrows : false
```

- **Zmiana etykiet przycisków.** Aby zmienić teksty wyświetlane po wskazaniu przycisków rozpoczęjącego i przerywającego automatyczną prezentację, należy je podać we właściwościach `startText` oraz `stopText`:

```
startText : "Uruchom prezentację",
stopText : "Zatrzymaj prezentację"
```

- **Wyłączenie automatycznego odtwarzania.** Być może nie będziesz chciał wyświetlać przycisków do rozpoczęcia i przerwania automatycznej prezentacji, bo preferujesz zapewnienie użytkownikowi możliwości samodzielnego wyboru karty, która ma być widoczna. W takim przypadku powinieneś przypisać wartość `false` właściwości `buildStartStop`:

```
buildStartStop : false
```

- **Animacja w pionie.** Aby karty w sliderze były przesuwane w kierunku pionowym, a nie w poziomie, przypisz wartość `true` właściwości `vertical`:

```
vertical : true
```

- **Automatyczne odtwarzanie.** Jeśli chcesz, by w momencie wyświetlenia strony rozpoczęła się automatyczna prezentacja kart slidera, przypisz wartość `true` właściwości `autoPlay`:

```
autoPlay : true
```

Automatyczne rozpoczęwanie prezentacji jest bardzo popularnym rozwiązaniem, często wykorzystywanym na witrynach ze sliderami — pozwala wyświetlić więcej treści bez konieczności zmuszania użytkownika do klikania przycisku „rozpocznij pokaz”.

- **Karty o różnej wielkości.** Jeśli zawartości poszczególnych kart są różnej wielkości, możesz zażądać, by okno slidera zmieniało wielkość i dostosowywało się do wymiarów zawartości aktualnie prezentowanej karty. Założymy na przykład, że pierwsza karta zawiera znacznik `<div>`, wewnątrz którego został umieszczony pojedynczy akapit tekstu; natomiast na drugiej karcie mamy znacznik `<div>` zawierający nagłówek, dwa duże zdjęcia oraz trzy akapity tekstu. Jeśli w takim przypadku właściwości `resizeContents` przypiszesz wartość `true`, slider będzie automatycznie zmieniał swoją wielkość, dostosowując się do wielkości zawartości prezentowanej karty. W naszym przypadku w sliderze prezentowana jest początkowo karta z jednym akapitem, a zatem jego wysokość będzie niewielka. Jednak po kliknięciu przycisku i przejściu do następnej karty — o znacznie większej zawartości — wysokość slidera zostanie powiększona. Aby zapewnić takie działanie komponentu, umieść w literale obiektowym następującą właściwość:

```
resizeContents : true
```

Aby przekonać się, jakie efekty daje zastosowanie niektórych spośród tych właściwości, wyświetl w przeglądarce przykładową stronę `complete_slider2.html`.

Jak mogłeś się przekonać, zarówno umieszczanie komponentu AnythingSlider na stronie, jak i dostosowywanie jego wyglądu i działania do własnych potrzeb jest całkiem proste. Przedstawiliśmy tu jednie drobny ułamek wszystkich możliwości tej wtyczki. Pozwala ona także na prezentowanie klipów wideo, dodawanie efektów specjalnych oraz stosowanie własnego kodu JavaScript, który zapewni, że prezentowany slider będzie działał dokładnie tak, jak chcemy. Więcej informacji na ten temat można znaleźć w Wiki wtyczki AnythingSlider, dostępnej na stronie <https://github.com/ProLoser/AnythingSlider/wiki>.

Określanie wielkości i położenia elementów strony

Podczas dynamicznego modyfikowania zawartości stron lub dodawania do nich nowych treści z wykorzystaniem języka JavaScript i biblioteki jQuery często bardzo przydatna może się okazać znajomość rozmiaru oraz położenia elementów strony. Przykładowo może się zdarzyć, że będziesz chciał wyświetlić nad zawartością strony dodatkową warstwę, tak zwaną *nakładkę* (ang. *overlay*), tworząc ona efekt, w którym zawartość strony zostaje „wyszarzona”, podobny do tego, jaki daje wtyczka FancyBox opisana na stronie 244). W tym celu konieczne jest dodanie do strony bezwzględnie umieszczonego znacznika `<div>`, który przykryje całą zawartość okna przeglądarki. Aby to zrobić, trzeba mieć pewność, że znacznik ten będzie miał dokładnie takie same wymiary jak okno programu, a to oznacza, że trzeba będzie w jakiś sposób je określić.

Gdybyśmy chcieli utworzyć etykietki ekranowe — niewielkie okienka wyświetlane, gdy użytkownik umieści wskaźnik myszy w obszarze jakiegoś elementu — konieczne będzie określenie współrzędnych wskaźnika myszy, co pozwoli umieścić etykietkę w odpowiednim miejscu.

Określanie wysokości i szerokości elementów

Biblioteka jQuery udostępnia funkcje `.height()` oraz `.width()`, które zwracają odpowiednio wysokość i szerokość wybranego elementu strony. Podając odpowiedni selektor, można określić wymiary dowolnego znacznika wchodzącego w skład strony, a nawet całego okna przeglądarki lub całej zawartości dokumentu.

- **Wysokość i szerokość okna przeglądarki.** Jeśli chcesz poznać wysokość i szerokość okna przeglądarki, musisz użyć selektora `$(window)`, a następnie wywołać funkcję `height()` lub `width()`:

```
var winH = $(window).height();
var winW = $(window).width();
```

Powyższy fragment kodu pobiera wysokość oraz szerokość okna przeglądarki i zapisuje je w dwóch zmiennych. Pobieranie wymiarów okna przeglądarki jest przydatne, kiedy chcemy mieć pewność, że jakiś element nie zostanie umieszczony poza widocznym obszarem strony.

- **Wysokość i szerokość dokumentu.** Dokument to nie to samo, co okno przeglądarki, i w większości przypadków ma zupełnie inne wymiary. Dokument reprezentuje naszą stronę WWW; jeśli umieścimy na niej tylko niewielki fragment tekstu — na przykład jeden akapit — dokument będzie miał wysokość tego akapitu (powiększoną dodatkowo o marginesy górny i dolny). Na dużym monitorze wysokość takiego dokumentu będzie mniejsza od wysokości okna przeglądarki.

W odwrotnej sytuacji — gdy zawartość strony jest bardzo obszerna i użytkownik musi ją przewijać, by dotrzeć do jej końca — wysokość dokumentu będzie większa od wysokości okna. Podobnie, gdybyś zmodyfikował styl określający postać znacznika `<div>`, wewnątrz którego umieszczona jest cała zawartość strony, i nadał mu szerokość 1500 pikseli, okazałoby się, że szerokość dokumentu jest większa od szerokości okna przeglądarki. Aby określić wysokość i szerokość dokumentu, wywołania metod `height()` i `width()` należy poprzedzić selektorem `$(document)`:

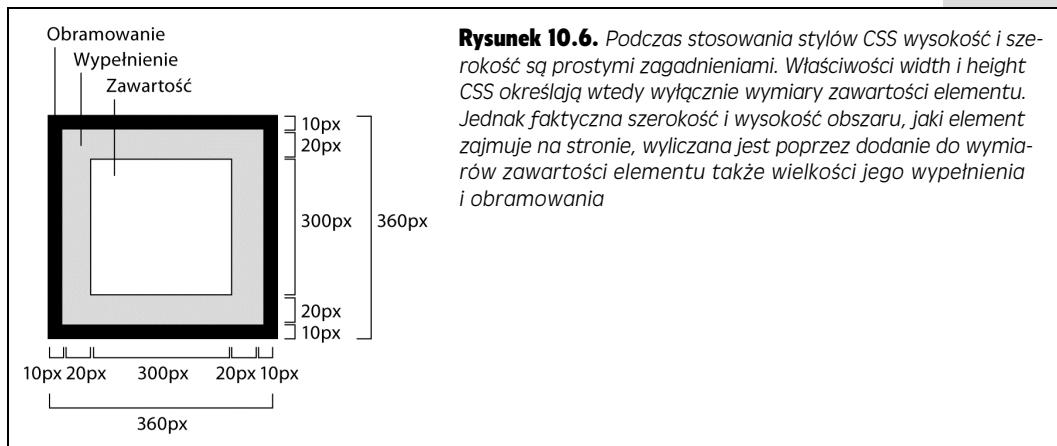
```
var docH = $(document).height();
var docW = $(document).width();
```

Funkcji `height()` oraz `width()` można także używać do określania wymiarów zwyczajnych elementów HTML, takich jak akapity, sekcje (znaczniki `<div>`) czy też obrazki, jednak w takich przypadkach nie zawsze będą one zwracać te informacje, których poszukujemy. Funkcje te zwracają wartości właściwości `height` oraz `width` CSS, a te nie zawsze są takie same jak faktyczne wymiary elementów strony. Właściwości te określają wymiary przydzielane zawartości znacznika, czyli na przykład tekstowi umieszczonemu wewnątrz akapitu. Kiedy jednak dodamy do elementu marginesy, wypełnienie oraz obramowanie, całkowity obszar, jaki zajmuje na stronie, będzie większy od tego, co wskazują właściwości `height` i `width`.

Aby zrozumieć, jak to działa, przeanalizujmy poniższy, przykładowy kod CSS określający postać znacznika `<div>`:

```
div {
    width : 300px;
    height : 300px;
    padding : 20px;
    border : 10px solid black;
}
```

Schemat tego elementu został przedstawiony na rysunku 10.6. Rzeczywista wysokość i szerokość tego elementu na stronie będzie wynosić 360 pikseli, gdyż stanowi sumę wysokości (lub szerokości) wypełnienia oraz obramowania. A zatem, faktyczna szerokość elementu jest sumą grubości jego lewego obramowania, szerokości lewego wypełnienia, szerokości określonej w stylu CSS, szerokości prawego wypełnienia oraz grubości prawego obramowania; analogicznie, faktyczna wysokość jest sumą grubości górnego obramowania, wysokości górnego wypełnienia, wysokości określonej w stylu CSS, wysokości dolnego wypełnienia oraz grubości dolnego obramowania.



Ze względu na te wszystkie wymiary jQuery udostępnia trzy zestawy funkcji służących do określania różnych szerokości i wysokości elementów strony. Oto one.

- `width()` i `height()` — funkcje te zwracają szerokość i wysokość podane w stylu CSS. Założymy na przykład, że strona zawiera znacznik `<div>`, którego postać określona powyżej reguła CSS.

```
var divW = $('div').width(); //300
var divH = $('div').height(); //300
```

Po wykonaniu powyższego fragmentu kodu zmiennym `divW` oraz `divH` zostanie przypisana wartość 300 — czyli szerokość i wysokość określone w stylu CSS.

- Funkcja `innerWidth()` zwraca szerokość elementu podaną w stylu CSS powiększoną o szerokość prawego i lewego wypełnienia, a funkcja `innerHeight()` — wysokość elementu podaną w stylu CSS powiększoną o wysokość górnego i dolnego wypełnienia:

```
var divW = $('div').innerWidth(); //340
var divH = $('div').innerHeight(); //340
```

W tym przypadku zmienne `divW` oraz `divH` przyjmą wartość 340, odpowiadającą szerokości (i wysokości) podanej w regule stylu, powiększonej o wymiary wypełniania z obu stron elementu.

- Funkcja `outerWidth()` zwraca szerokość podaną w stylu CSS, powiększoną o szerokość prawego i lewego wypełnienia oraz grubość prawego i lewego obramowania; analogicznie, funkcja `outerHeight()` zwraca wysokość elementu podaną w stylu CSS, powiększoną o wysokość górnego i dolnego wypełnienia oraz grubość górnego i dolnego obramowania.

```
var divW = $('div').outerWidth(); //360
var divH = $('div').outerHeight(); //360
```

Po wykonaniu powyższego fragmentu kodu w zmiennych `divW` oraz `divH` zostanie zapisana wartość 360, czyli szerokość (i wysokość) podana w regule CSS, powiększona o wielkość wypełnienia i grubość obramowania z obu stron elementu.

Funkcje `outerWidth()` oraz `outerHeight()` pobierają także jeden dodatkowy argument — wartość `true`, której przekazanie sprawi, że funkcje te będą uwzględniały w obliczeniach także wielkość marginesów elementu. Przykładowo założmy, że postać znacznika `<div>` jest określana przy użyciu następującej reguły CSS:

```
div {
    width : 300px;
    height : 300px;
    padding : 20px;
    border : 10px solid black;
    margin: 20px;
}
```

Warto zwrócić uwagę na właściwość `margin: 20px`. Jeśli chcesz, by w wyliczanej szerokości i wysokości elementu zostały uwzględnione te marginesy, musisz wywołać funkcje `outerWidth()` oraz `outerHeight()` w następujący sposób:

```
var divW = $('div').outerWidth(true); //400
var divH = $('div').outerHeight(true); //400
```

To, których funkcji należy użyć, zależy od tego, co chcemy osiągnąć. Założymy, że chcemy zasłonić czarnym prostokątem pewien tekst wyświetlony na stronie — na przykład odpowiedź na pytanie quizowe — a później go wyświetlić. Jednym z potencjalnych rozwiązań będzie zasłonięcie odpowiedzi prostokątem z czarnym tłem. W takim przypadku możemy użyć funkcji `width()` oraz `height()`, by określić wymiary samego tekstu (z pominięciem wypełnień i obramowań), i na podstawie podanych wartości określić wymiary prostokąta, który następnie wyświetlimy nad tekstem.

A teraz dla odmiany założymy, że tworzymy własną wersję znanej gry Pong, w której niewielka piłeczka odbija się od krawędzi pola gry. Oczywiście, będziemy chcieli, by piłeczka cały czas pozostawała wewnątrz wyznaczonego obszaru (znacznika `<div>`, który najprawdopodobniej będzie miał wyświetcone obramowanie). W tym przypadku potrzebna jest znajomość wymiarów całego obszaru wewnątrz obramowań, co pozwoli upewnić się, że animowana piłeczka nie „wyleci” poza element oraz jego krawędzie. W takim przypadku powinniśmy skorzystać z funkcji `innerHeight()` oraz `innerWidth()`, gdyż piłeczka może się znaleźć w dowolnym miejscu wewnątrz pudełka elementu, nawet jeśli będzie on miał wypełnienie.

Uwaga: Funkcji `innerHeight()`, `innerWidth()`, `outerHeight()` oraz `outerWidth()` nie należy stosować podczas określania wymiarów okna przeglądarki (z selektorem `$(window)`) czy dokumentu (z selektorem `$(document)`). W tych dwóch przypadkach można korzystać wyłącznie z funkcji `height()` i `width()`.

Określanie położenia elementu na stronie

Znajomość położenia elementu na stronie przydaje się bardzo często, na przykład chcemy wyświetlić nad obrazkiem etykietę ekranową, kiedy użytkownik wskaże go myszą. Położenie tej etykiety powinno być zależne od położenia obrazka na stronie — oznacza to, że musimy najpierw określić położenie obrazka, a dopiero potem na jego podstawie wyliczyć współrzędne miejsca, gdzie ma się pojawić etykieta. Biblioteka jQuery udostępnia kilka funkcji ułatwiających określanie położenia elementów na stronie. Oto one.

- Funkcja `offset()`. Jej wywołanie zwraca obiekt zawierający właściwości `top` i `left`, określające odpowiednio położenie lewego, górnego wierzchołka elementu od górnej oraz lewej krawędzi dokumentu. Przykładowo założmy, że kiedy użytkownik wskaże myszą obrazek, chcemy wzduż jego górnej krawędzi wyświetlić opis. W takim przypadku musimy znać położenie obrazka. Założymy dodatkowo, że obrazek ten ma identyfikator `captionImage`. Współrzędne określające jego położenie można pobrać przy użyciu następującego wywołania:

```
var imagePosition = $('#captionImage').offset();
```

W efekcie, w zmiennej `imagePosition` zostaną zapisane współrzędne obrazka. Są one zapisane w obiekcie JavaScript, z którego można je odczytać, stosując zapis z kropką, opisany na stronie 82. Współrzędna pozioma jest zapisana we właściwości `left`, natomiast współrzędna pionowa — we właściwości `top`:

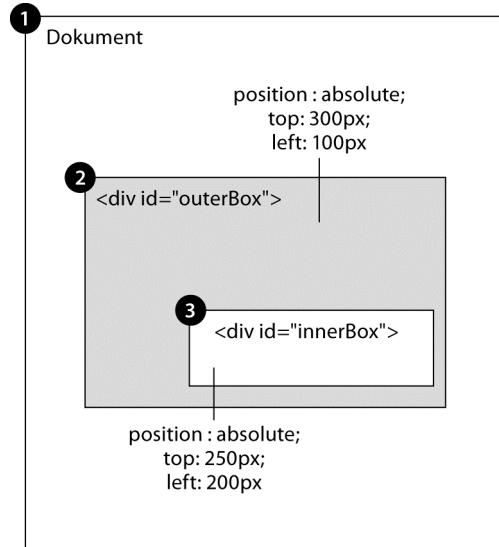
```
imagePosition.top // liczba pikseli od górnej krawędzi dokumentu  
imagePosition.left // liczba pikseli od lewej krawędzi dokumentu
```

Założymy teraz, że chcemy użyć tych informacji, by wyświetlić na stronie znacznik `<div>` o identyfikatorze `caption`. Możemy użyć funkcji `.css()` jQuery (patrz strona 155), by określić jego właściwości CSS `top`, `left` oraz `position`, i tym samym wyświetlić go w odpowiednim miejscu strony:

```
$('#caption').css({  
  'position' : 'absolute',  
  'left' : imagePosition.left,  
  'top' : imagePosition.top  
});
```

Uwaga: Funkcje `offset()` oraz `position()` zwracają współrzędne wyrażone w pikselach nawet wtedy, kiedy położenie elementu na stronie zostanie określone przy użyciu takich jednostek jak em lub wartości procentowe.

- Funkcja `position()`. Wywołanie tej funkcji zwraca obiekt zawierający współrzędne elementu liczone względem jego pierwszego przodka, w którego stylach CSS została określona wartość właściwości `display`. Zrozumienie tego wcale nie jest łatwe, posłużymy się zatem przykładem dwóch elementów `div` przedstawionych na rysunku 10.7. Oba te elementy zostały umieszczone w sposób bezwzględny; położenie elementu `outerBox` jest określone względem dokumentu, natomiast położenie elementu `innerBox`, którego kod HTML jest umieszczony wewnątrz elementu `outerBox`, względem zewnętrznego znacznika `<div>`. Położenie elementu zewnętrznego jest określone względem dokumentu, gdyż nie jest on umieszczony wewnątrz żadnego innego znacznika HTML, w którym



Rysunek 10.7. Biblioteka jQuery udostępnia dwie funkcje pozwalające na określanie współrzędnych elementu na stronie. Gdy na stronie jeden element umiejscowiony bezwzględnie (#innerBox) zostanie umieszczony wewnątrz innego elementu umiejscowionego bezwzględnie (#outerBox), funkcje te zwrócią różne wyniki

właściwości CSS `display` została przypisana wartość `absolute`, `relative` bądź `fixed`. W przypadku tego elementu funkcja `position()` zwróci dokładnie takie same wyniki, co funkcja `offset()`, a zatem wywołanie:

```
$('#outerBox').position() // { left : 100, top : 300 }
```

zwróci obiekt, którego właściwość `left` będzie miała wartość 100, a właściwość `top` wartość 300. Takie wartości zostały podane w regule stylów dla tego elementu.

Jednak w przypadku wewnętrznego znacznika `<div>` — o identyfikatorze `innerBox` — którego położenie jest określone względem elementu zewnętrznego, wywołania funkcji `offset()` i `position()` zwrócią inne wyniki:

```
$('#innerBox').offset() // { left : 300, top : 550 }  
$('#innerBox').position() // { left : 200, top : 250 }
```

Teraz funkcja `offset()` zwróci współrzędne liczne względem całego dokumentu, czyli 300 pikseli na prawo od lewej krawędzi dokumentu i 550 pikseli poniżej jego górnej krawędzi. Natomiast funkcja `position()` zwróci wartości podane w regule stylów CSS dla tego elementu.

Zazwyczaj bardziej przydatna z tych dwóch funkcji jest `offset()`, gdyż pozwala określić położenie elementu w odniesieniu do całej strony i dostarcza informacje potrzebne do wyznaczenia współrzędnych elementu względem innego elementu strony.

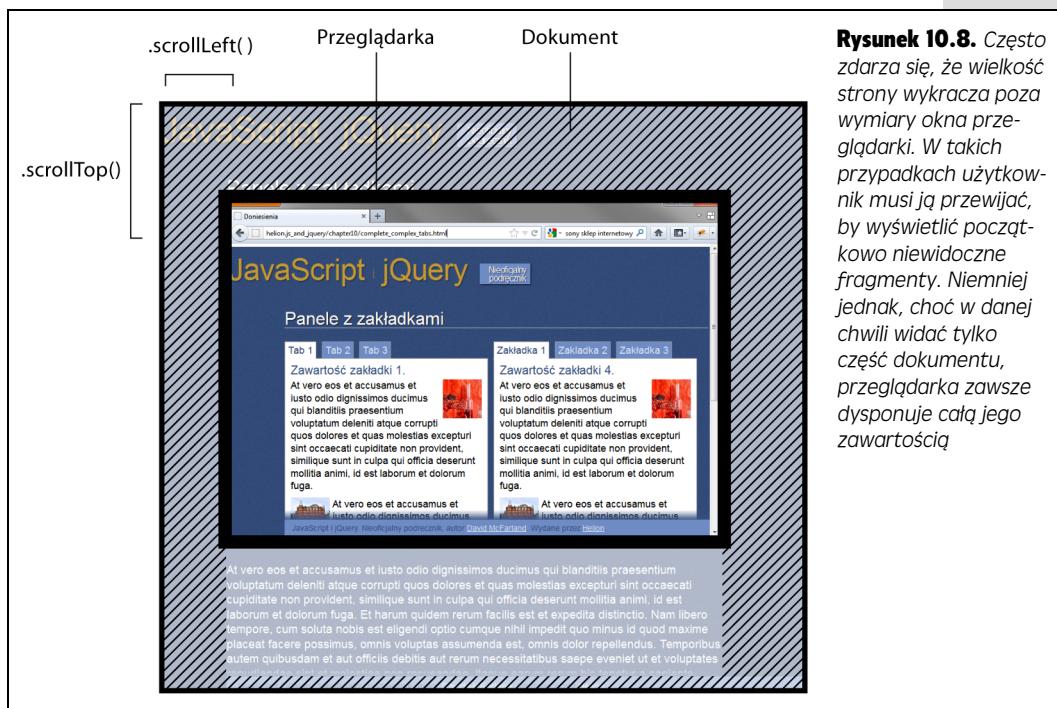
Wskazówka: Funkcji `offset()` można także używać do ustawiania położenia elementu na stronie. Wystarczy w tym celu przekazać w jej wywołaniu obiekt, co pokazano na poniższym przykładzie:

```
$('#element').offset({  
    left : 100,  
    top : 200  
});
```

W tym przypadku współrzędne muszą być określone w pikselach — nie można stosować innych jednostek, takich jak `em` (na przykład `20em`) lub wartości procentowe (na przykład `20%`).

Uwzględnianie przewinięcia strony

Strony WWW bardzo często są większe od okna przeglądarki, w którym są prezentowane: dokumenty HTML zawierające bardzo obszerną zawartość niejednokrotnie są wyższe, a zdarza się także, że i szersze od okna przeglądarki. Aby wtedy przejrzeć całą zawartość strony, użytkownik musi ją przewijać (patrz rysunek 10.8). Kiedy użytkownik przewija stronę, część jej treści przestaje być widoczna. Przykładowo strona przedstawiona na rysunku 10.8 została przewinięta nieco w dół i w prawo, co sprawia, że jej fragmenty przy górnej i lewej krawędzi nie są widoczne. Oznacza to, że góry, lewy wierzchołek okna przeglądarki nie będzie się pokrywał z górnym, lewym wierzchołkiem dokumentu. Jeśli w takim przypadku spróbujemy wyświetlić jakiś element, na przykład animowany pasek reklamowy, u góry ekranu, po przypisaniu właściwościom `top` i `left` wartości 0 pojawią się problemy. Będą one spowodowane tym, że element został umieszczony w lewym, górnym wierzchołku dokumentu, lecz poza fragmentem strony, który jest aktualnie widoczny w oknie przeglądarki.



Na szczęście, biblioteka jQuery udostępnia dwie funkcje pozwalające na określenie, o jaki dystans strona została przewinięta w pionie oraz w poziomie (inaczej mówiąc, zwracają one liczby określające w pikselach, jaki fragment dokumentu znajduje się powyżej górnej krawędzi okna przeglądarki oraz poza jego lewą krawędzią). Poniższy fragment kodu pozwala określić wysokość fragmentu dokumentu umieszczonego nad górną krawędzią okna przeglądarki:

```
$(document).scrollTop()
```

By natomiast określić szerokość obszaru dokumentu umieszczonego poza lewą krawędzią okna, można użyć następującego wywołania:

```
$(document).scrollLeft()
```

Obie te funkcje zwracają wielkości liczbowe wyrażone w pikselach, z których można skorzystać podczas wyliczania współrzędnych elementów na stronie. Jeśli na przykład chcemy wyświetlić okienko na środku strony, nawet jeśli została ona nieco przewinięta w pionie, trzeba będzie określić, o jaki dystans została przewinięta, i przesunąć wyświetlane okienko o odpowiedni odcinek w dół strony. Dużą ostrożność należy także zachować w przypadku wyświetlania etykietek ekranowych na przewiniętych stronach — bardzo łatwo doprowadzić do sytuacji, w której etykietka będzie wyświetlana w obszarze strony, lecz poza jej fragmentem, który w danej chwili jest widoczny w oknie przeglądarki. Dlatego też w 12. kroku opisu kolejnego przykładu, zamieszczonym na stronie 349, zobaczysz, jak skorzystać z funkcji `scrollTop()`, by chronić się przed wyświetlaniem etykietek nad górną krawędzią prezentowanego w przeglądarce obszaru strony.

Dodawanie etykietek ekranowych

Etykietki są często stosowanym sposobem prezentowania informacji uzupełniających. Są to niewielkie, wyskakujące okienka, wyświetlane po wskazaniu myszą wybranego elementu strony — odnośnika, słowa, obrazka i tym podobnych. Często są one stosowane do wyświetlania definicji słowa, podpisu pod zdjęciem, a nawet bardziej szczegółowych informacji, takich jak czas, koszt oraz lokalizacja jakiegoś zdarzenia.

Podstawowa zasada działania etykietek ekranowych jest bardzo prosta: wskazujemy wybrany element myszą i wyświetlamy inny element (zazwyczaj będzie nim znacznik `<div>`) w pobliżu wskazanego; po usunięciu wskaźnika myszy z obszaru elementu etykietka znika. Poznałeś już sposoby tworzenia kodu JavaScript niezbędnego do zaimplementowania takiego rozwiązania, zatem bez przeszkód możemy opisać cały proces tworzenia etykietek krok po kroku.

Wykorzystamy w nim kody JavaScript, CSS oraz HTML w celu uzyskania ostatecznego efektu przedstawionego na rysunku 10.9. Kod HTML posłuży do zdefiniowania zarówno elementu wyzwalającego wyświetlenie etykietki (czyli tego, który należy wskazać myszą), jak i samej etykietki. Podstawowe aspekty wyglądu etykietki ustalimy przy użyciu arkusza stylów CSS, natomiast kod JavaScript pozwoli ukryć etykietę w momencie wczytywania strony. Dodatkowo określmy także procedurę obsługi zdarzeń `hover` i dodamy ją do wszystkich elementów strony, dla których chcemy wyświetlać etykietki.

Kod HTML

Etykietka ekranowa składa się z dwóch elementów: samej etykietki, czyli elementu wyświetlanego na stronie, gdy użytkownik wskaże myszą element wyzwalający, oraz tegoż elementu wyzwalającego, którym może być dowolny inny element strony, taki jak obrazek, odnośnik, nagłówek bądź znacznik ``.

JavaScript i jQuery

Nieoficjalny podręcznik

Etykiety

At vero eos et odio dignissimos accusamus et iusto ducimus qui blanditiis præsentium voluptatum delenit atque corrupti quos dolores et quas molestias excepturi sint occaecati cupiditate non provident, similique sunt in culpa qui officia deserunt mollitia animi, id est laborum et dolorum fuga.

Kolejna etykieta

A oto jest kolejna etykieta. Spójrzcie, umieściliśmy w niej nawet małe zdjęcie!



odio dignissimos ducimus qui blanditiis præsentium voluptatum et quas molestias excepturi sint occaecati cupiditate non qui officia deserunt mollitia animi, id est laborum et dolorum fuga.

odio dignissimos ducimus qui blanditiis præsentium voluptatum et quas molestias excepturi sint occaecati cupiditate non qui officia deserunt mollitia animi, id est laborum et dolorum fuga.

At vero eos et accusamus et iusto odio dignissimos ducimus qui blanditiis præsentium voluptatum delenit atque corrupti quos dolores et quas molestias excepturi sint occaecati cupiditate non provident, similique sunt in culpa qui officia deserunt mollitia animi, id est laborum et dolorum fuga.

At vero eos et accusamus et iusto odio dignissimos ducimus qui blanditiis præsentium voluptatum delenit atque corrupti quos dolores et quas molestias excepturi sint occaecati cupiditate non provident, similique sunt in culpa qui officia deserunt mollitia animi, id est laborum et dolorum fuga.

Rysunek 10.9. Etykiety ekranowe są niewielkimi okienkami zawierającymi dodatkowe informacje. Są one wyświetlane, kiedy użytkownik wskaże myszą określony element strony, tak zwany element wyzwalający (został zakreślony na rysunku), oraz ukrywane, gdy wskaźnik myszy zostanie usunięty z obszaru tego elementu

Etykieta jest znacznikiem `<div>` należącym do klasy `tooltip` i posiadającym unikalny identyfikator. Wewnątrz tego znacznika można umieścić dodatkowy kod HTML, chociażby nagłówki, akapity tekstu i obrazki. Nie należy jednak umieszczać w nich odnośników, gdyż nie będą działały prawidłowo — przesunięcie wskaźnika myszy do obszaru etykiety w celu kliknięcia odnośnika spowoduje usunięcie go z obszaru elementu wyzwalającego, co sprawi, że etykieta zniknie.

Oto bardzo prosty kod etykiety ekranowej:

```
<div class="tooltip" id="aardvarkTooltip">
  <h2>Mrówkojad</h2>
  <p>Średniej wielkości ssak ryjący, prowadzący nocny tryb życia; występuje głównie w Afryce.</p>
</div>
```

Choć znaczniki `<div>` etykiet można umieścić w dowolnym miejscu kodu HTML strony (w końcu przez znaczna większość czasu i tak są one niewidoczne), jednak najlepszym rozwiązaniem jest umieszczenie ich tuż przed zamkającym znacznikiem `</body>`. To optymalne miejsce, gdyż pozwala uniknąć wszelkich dziwnych problemów związanych ze sposobem prezentacji treści, które mogłyby powstać po umieszczeniu etykiet wewnętrz innego elementu, pozycjonowanego względnie lub bezwzględnie.

Elementem wyzwalającym może być dowolny element strony — znacznik `<h1>`, `<div>` lub ``. Jeśli chcemy, by elementem wyzwalającym było słowo lub grupa słów, trzeba je umieścić wewnątrz znacznika ``. Do prawidłowego działania elementu wyzwalającego niezbędne są dwie informacje.

- **Nazwa klasy.** Wszystkie elementy wyzwalające muszą należeć do tej samej klasy, na przykład trigger. Nazwa klasy jest niezbędną, by kod JavaScript mógł odzyskać wszystkie te elementy i dodać do nich procedury obsługi zdarzeń odpowiadające za wyświetlanie i ukrywanie etykiet.
- **Dane identyfikujące etykietę.** Każdy element wyzwalający jest skojarzony z jedną etykietą. Etykieta ta jest znacznikiem `<div>`, który zazwyczaj jest niewidoczny, lecz można go wyświetlić, gdy użytkownik umieści wskaźnik myszy na odpowiednim elemencie wyzwalającym. Każda etykietka musi posiadać unikalny identyfikator, a my musimy dysponować jakimś sposobem skojarzenia elementu wyzwalającego z odpowiednim znacznikiem `<div>` etykiety, dzięki któremu będziemy wiedzieli, jaki znacznik `<div>` wyświetlić, kiedy użytkownik wskaże myszą element wyzwalający. Prostym rozwiązaniem jest umieszczenie identyfikatora etykiety w jakimś atrybutie znacznika elementu wyzwalającego (warto dodać przed nim znak `#`, dzięki czemu pobranie etykiety przy użyciu jQuery będzie naprawdę bardzo proste). Język HTML5 pozwala na dodawanie danych do znaczników HTML, jeśli nazwa atrybutu rozpoczyna się od ciągu znaków `data-`.

Przykładowo założymy, że elementem wyzwalającym dla naszej etykiety jest słowo `mrówkojad`. Wskazanie go myszą powinno spowodować wyświetlenie etykiety (czyli, w rzeczywistości, znacznika `<div>` o identyfikatorze `aardvarkTooltip`). Element wyzwalający możemy utworzyć, umieszczając wybrane słowo wewnątrz znacznika ``, co pokazano na poniższym przykładzie:

```
<span class="trigger" data-tooltip="#aardvarkTooltip">mrówkojad</span>
```

Niestandardowe atrybuty danych standardu HTML5 są naprawdę rewelacyjne. Pozwalają projektantom na umieszczanie w znacznikach przeróżnych informacji, które później można pobierać przy użyciu kodu JavaScript. Szczegółowy opis tych atrybutów można znaleźć na stronie <http://html5doctor.com/html5-custom-data-attributes/>.

Jeśli korzystasz z języków XHTM 1 bądź HTML 4.01 i obawiasz się problemów zgodności ze standardami, nie będziesz mógł używać takich atrybutów danych. Zamiast tego możesz wykorzystać jeden z prawidłowych atrybutów języka HTML 4, na przykład `title`:

```
<span class="trigger" title="#aardvarkTooltip">mrówkojad</span>
```

Takie zastosowanie atrybutu `title` nie jest — co prawda — zgodne z jego przeznaczeniem i niektórzy projektanci stron mogą nie pochwalać takiego rozwiązania. Najprościej będzie skorzystać z języka HTML5 i użyć atrybutów danych.

Na jednej stronie można umieścić dowolną liczbę elementów wyzwalających oraz skojarzonych z nimi etykiet.

Kod CSS

Każdy znacznik `<div>` etykiety należy do klasy `tooltip`, a zatem dodanie do używanego na stronie arkusza stylów reguły z selektorem `.tooltip` pozwoli określić ich ogólny wygląd (na przykład kolor tła, obramowanie, szerokość i tak dalej). Oto wersja tej reguły umieszczona w przykładowym pliku dołączonym do książki:

```
.tooltip {
    width: 25%;
    padding: 5px;
    background-color: white;
    border: 3px solid rgb(195,151,51);
    border-radius : 5px;
}
```

Bez zastosowania dodatkowych stylów użytkownicy nie będą w stanie określić, że elementy wyzwalające pełnią szczególną rolę. Ma to szczególne znaczenie w przypadkach, gdy będziemy dodawać etykietę ekranową do wybranego słowa należącego do większego akapitu tekstu. Można utworzyć specjalny styl CSS, który wyróżni elementy wyzwalające — doda do nich obramowanie, kolor tła i tym podobne. Poniższa, prosta reguła dodaje dolną krawędź do wszystkich elementów należących do klasy trigger:

```
.trigger {
    border-bottom: 1px dashed white;
    cursor : help;
}
```

Szczególnie użyteczna jest właściwość CSS cursor — kontroluje ona postać wskaźnika myszy, w czasie gdy będzie się znajdował w obszarze elementu. Kiedy wskaźnik myszy zostanie umieszczony w obszarze tekstu, wygląda jak kurSOR do zaznaczania, jednak tę postać wskaźnika można zmienić — użycie wartości help sprawi, że będzie wyglądał jak znak zapytania (co jest dobrym rozwiązaniem, gdy etykieta zawiera definicję jakiegoś terminu), a wartości pointer — że będzie wyglądał jak dłoń z wyprostowanym palcem wskazującym, czyli w sposób standardowy dla wskaźnika umieszczonego na odnośniku. Informacje o pozostałych dostępnych kształtach wskaźnika myszy można znaleźć na stronie www.w3schools.com/cssref/pr_class_cursor.asp.

Oprócz tego, do arkusza stylów można dodać regułę z pseudoelementem :hover, określającą postać elementu wyzwalającego w przypadku umieszczenia w jego obszarze wskaźnika myszy; oto przykład takiej reguły:

```
.trigger:hover {
    color: rgb(255,0,0);
}
```

Kod JavaScript

Najprościej rzecz ujmując, etykieta ekranowa powinna zostać wyświetlona, gdy użytkownik umieści wskaźnik myszy w obszarze elementu wyzwalającego, i ma zniknąć, kiedy wskaźnik zostanie z tego elementu usunięty. Już wcześniej, w rozdziale 6., dowiedziałeś się, jak można wyświetlać i ukrywać elementy. Jednak w tym przykładzie zwyczajne wyświetlenie i ukrycie elementu to za mało. Kluczową czynnością związaną z wyświetlaniem każdej etykiety jest umieszczenie jej w pobliżu elementu wyzwalającego. To z kolei wiąże się z koniecznością użycia funkcji jQuery w celu określenia szerokości, wysokości oraz wymiarów tego elementu. I to jest najtrudniejsze zadanie. Aby nieco lepiej wyjaśnić wykonywane czynności, w tym przykładzie zdecydowaliśmy się podzielić opis tworzonego kodu na trzy części.

1. Ukrycie etykiet.

W momencie wczytywania strony wszystkie znajdujące się na niej etykiety (czyli znaczniki `<div>` umieszczone na samym końcu jej treści) powinny być ukryte. Oczywiście, można by to zrobić przy użyciu odpowiedniego stylu CSS jeszcze przed wczytaniem strony, jednak w takim przypadku żaden użytkownik, przeglądający stronę za pomocą przeglądarki, w której została wyłączona obsługa języka JavaScript, nie byłby w stanie uzyskać dostępu do treści etykiet. Jeśli informacje zamieszczone w etykietach nie są bardzo ważne i można zaakceptować fakt, że niektórzy użytkownicy (w tym także mechanizmy wyszukiwawcze) ich nie zobaczą, to proszę bardzo — możesz ukryć etykiety poprzez zastosowanie odpowiedniego stylu CSS:

```
.tooltip {  
    display: none;  
}
```

2. Dodanie do elementów wyzwalających procedury obsługi zdarzeń `hover`.

Czynność ta ma kluczowe znaczenie dla działania etykiet. Kiedy użytkownik wskaże myszą element wyzwalający, muszą zostać wykonane dwie operacje.

- Musi zostać wyświetlony znacznik `<div>` odpowiedniej etykiety.
- Znacznik ten należy umieścić w pobliżu elementu wyzwalającego. W tym celu trzeba określić bieżące położenie tego elementu. Dodatkowo trzeba się upewnić, że etykieta nie przesłoni tego elementu oraz że nie będzie wystawać poza widoczny obszar okna przeglądarki.

3. Dodanie do elementów wyzwalających procedury obsługi zdarzeń `mouseover`.

To bardzo proste zadanie — wystarczy ukryć znacznik `<div>`, kiedy użytkownik usunie wskaźnik myszy z jego obszaru.

Aby przekonać się, jak ten program działa, czas przejść do przykładu, w którym utworzysz swoje własne etykiety ekranowe.

Przykład — etykiety ekranowe

Utworzenie prostych etykiet ekranowych naprawdę nie jest trudnym zadaniem. W tym przykładzie szczegółowo opiszymy cały ten proces. Do pracy nad tym przykładem możesz wykorzystać dowolny edytor HTML.

Uwaga: Informacje dotyczące pobierania przykładów do książki można znaleźć na stronie 43.

1. W edytorze HTML otwórz plik `tooltip.html` umieszczony w katalogu R10.

W tym pliku został już umieszczony wewnętrzny arkusz stylów CSS, zawierający kilka reguł określających wygląd elementów wyzwalających oraz etykiet. Są to dokładnie te same style, które zostały przedstawione we wcześniejszej części rozdziału, na stronie 342. Jednak w kodzie strony nie ma jeszcze żadnych etykiet — ich kod będziesz musiał dodać.

2. Odszukaj zamykający znacznik </body> umieszczony na samym końcu pliku i powyżej niego dodaj poniższy kod HTML tworzonej etykiety:

```
<div class="tooltip" id="tip1">
    <h2>Etykieta</h2>
    <p>To jest tekst etykiety. Został on umieszczony wewnątrz
        →znacznika div,
        dzięki temu można tu umieścić prawie wszystko.</p>
</div>
```

Najważniejszym elementem tego kodu jest zewnętrzny znacznik `<div>`. Użyliśmy w nim klasy `tooltip`, co jest niezbędnego zarówno po to, by określić postać etykiety, jak i ze względu na kod programu, który utworzyłeś już niebawem. Dodatkowo w znaczniku umieściliśmy unikalny identyfikator, który pozwoli zidentyfikować daną etykię i skojarzyć ją z elementem wyzwalającym, jaki utworzyłeś w następnym kroku. Wewnątrz etykiety możesz umieścić dowolny kod HTML — w tym przypadku jest to nagłówek oraz jeden akapit tekstu.

3. Odszukaj znacznik <p> umieszczony w kodzie bezpośrednio poniżej znacznika <h1>Etykiety ekranowe</h1>, mniej więcej w połowie wielkości pliku. Wybierz kilka słów i zapisz je pomiędzy znacznikami , co pokazano na poniższym przykładzie:

```
<span class="trigger" data-tooltip="#tip1">accusamus et iusto</span>
```

Zastosowanie klasy `trigger` identyfikuje ten znacznik `` jako element wyzwalający etykiety. Jedna z reguł arkusza stylów umieszczonego w sekcji nagłówka strony formatuje dowolny znacznik należący do tej klasy w szczególny sposób. Dodatkowo umieszczony w kodzie znacznika atrybut `data-tooltip` identyfikuje etykię, z którą dany element wyzwalający jest skojarzony.

W następnym kroku dodasz do strony kolejną etykię.

4. Tuż poniżej znacznika <div> dodanego w kroku 2. (lecz wciąż przed zamykającym znacznikiem </body>) dodaj kolejny <div>:

```
<div class="tooltip" id="tip2">
    <h2>Kolejna etykieta</h2>
    <p>
        A oto jest kolejna etykieta. Spójrzcie, umieściliśmy w niej nawet
        →małe zdjęcie!</p>
</div>
```

Dodałeś właśnie drugą etykię. Zwróć uwagę, że użyliśmy w niej tej samej nazwy klasy, co w poprzedniej, czyli `tooltip`. Natomiast identyfikator tego znacznika jest unikalny — `tip2`. Dodatkowo wewnątrz etykiety umieściliśmy zdjęcie. Teraz musisz utworzyć element wyzwalający dla tej etykiety.

5. Wybierz kolejnych kilka słów z jakiegoś akapitu tekstu i ponownie umieść je wewnątrz znacznika :

```
<span class="trigger" data-tooltip="#tip2">At vero eos</span>
```

Zwróć uwagę, by podać identyfikator drugiej etykiety — `#tip2`. Nic nie stoi na przeszkodzie, abyś dodał kolejne etykiety i elementy wyzwalające; pamiętaj jednak, żeby każda z etykiet miała unikalny identyfikator i podaj ten identyfikator w atrybucie `data-tooltip` elementu wyzwalającego skojarzonego z daną etykietą.

Teraz nadszedł czas, by zająć się pisaniem kodu JavaScript. Do strony został już dołączony plik biblioteki jQuery oraz wywołanie funkcji `$(document).ready()`.

Kolejnym zadaniem będzie zatem ukrycie wszystkich etykiet w momencie wczytywania strony.

6. Kliknij pusty wiersz wewnętrz funkcji `$(document).ready()` i wpisz w nim:

```
$('.tooltip').hide();
```

Ten wiersz kodu jest bardzo prosty. Wywołanie funkcji `hide()` (opisanej na stronie 198) powoduje ukrycie wszystkich etykiet, dzięki czemu użytkownik nie zobaczy ich zaraz po wyświetleniu strony. Oczywiście, chcemy, by konkretne etykiety pojawiały się, kiedy użytkownik wskaże myszą odpowiednie elementy wyzwalające, a zatem kolejnym krokiem będzie pobranie wszystkich elementów wyzwalających i dodanie do nich procedury obsługi zdarzeń `mouseover`.

7. Poniżej kodu dodanego w poprzednim kroku dodaj kolejny fragment:

```
$('.trigger').mouseover(function() {  
}); // koniec funkcji mouseover
```

To szkielet kodu procedury obsługi zdarzeń, podobny do tego, który został opisany na stronie 174. W tym przypadku pobieramy wszystkie elementy należące do klasy `trigger` i dodajemy do nich procedurę obsługi zdarzeń `mouseover`. Funkcja ta stanowi kluczowy element naszego kodu obsługującego etykiety ekranowe, gdyż to właśnie ona będzie kontrolować wyświetlanie oraz odpowiednie rozmieszczanie etykiet na ekranie. Precyzyjne określenie miejsca, w którym ma zostać wyświetlona etykieta, jest dosyć złożone i będzie wymagało pobrania wielu różnych informacji. Dlatego też na samym początku tej funkcji zdefiniujesz kilka zmiennych.

8. Wewnątrz funkcji anonimowej dodanej w kroku 7. wpisz poniższy, wyróżniony pogrubieniem fragment kodu:

```
1  $('.trigger').mouseover(function() {  
2      var ttLeft,  
3          ttTop,  
4  }); // koniec funkcji mouseover
```

Zaczynamy od utworzenia dwóch zmiennych — `ttLeft` zawiera poziomą współrzędną etykiety, natomiast `ttTop` — współrzędną pionową. Początkowo obie te zmienne są puste, ponieważ jeszcze nie wiemy, jakie mają być ich wartości.

Ten sposób tworzenia zmiennych może Ci się wydawać nieco dziwny; gdyż zapewne jesteś przyzwyczajony do tworzenia dwóch zmiennych przy wykorzystaniu dwóch słów kluczowych `var`, w sposób pokazany poniżej:

```
var ttLeft;  
var ttTop;
```

Takie rozwiązanie jest całkowicie prawidłowe, jednak podczas tworzenia większej liczby zmiennych często stosuje się technikę wykorzystującą tylko jedno słowo kluczowe `var`, za którym są podawane nazwy wszystkich zmiennych oddzielone przecinkami. Dzięki temu możemy uniknąć konieczności wielokrotnego wpisywania słowa `var`. Przecinek umieszczony na końcu wiersza 3. nie jest żadnym błędem — już zaraz dodasz do kodu kolejne zmienne.

9. Do kodu programu dodaj kolejną zmianą (umieszczoną w wierszu 4.):

```
1  $('.trigger').mouseover(function() {  
2      var ttLeft,
```

```

3      ttTop,
4      $this=$(this),
5  }); //koniec funkcji mouseover

```

W tym przypadku wyrażenie `$(this)` odwołuje się do elementu wyzwalającego, natomiast cała instrukcja przypisania `$this = $(this)` pozwala zapisać odwołanie do tego elementu w zmiennej. Dokładnie to samo zrobiliśmy w przykładzie pokazującym sposób tworzenia panelu kart, w jego 3. kroku (opisanym na stronie 321). Dalej w kodzie tej funkcji będziemy wielokrotnie odwoływali się do elementu wyzwalającego i gdybyśmy za każdym razem używali wywołania `$(this)`, zmuszalibyśmy interpreter JavaScriptu przeglądarki do wielokrotnego wykonywania kodu funkcji jQuery, co stanowiłoby duże marnotrawstwo czasu i mocy procesora. Gdy zamiast tego zapiszemy wynik wywołania `$(this)` w zmiennej, funkcja jQuery konieczna do pobrania elementu wyzwalającego zostanie wykonana tylko raz, przez co nasz program stanie się bardziej efektywny (więcej informacji dotyczących zalet zapisywania elementów pobieranych przy użyciu jQuery w zmiennych można znaleźć na stronie 422).

Kolejnym krokiem będzie pobranie etykiety skojarzonej z danym elementem wyzwalającym.

Wskazówka: W przypadku zapisywania elementów pobieranych przy użyciu jQuery w zmiennych często stosowaną praktyką jest umieszczenie na początku nazwy zmiennej znaku `:`

```
var $banner = $('#banner');
```

Oczywiście, nie jest to konieczne; zmienna `var banner = $('#banner')` będzie działać równie dobrze. Jednak znak `$` przypomina o tym, że zmienna zawiera selekcję jQuery, a nie jakiekolwiek inne, zwyczajne wartości, takie jak łańcuchy znaków lub liczby.

10. Dodaj kolejną zmienną (umieszczoną w wierszu 5.):

```

1  $('.trigger').mouseover(function() {
2      var ttLeft,
3          ttTop,
4          $this=$(this),
5          $tip = $($this.attr('data-tooltip')),
6  }); //koniec funkcji mouseover

```

Zmienna `$tip` zawiera pobrany przy użyciu jQuery znacznik etykiety. Wywołanie `$($this.attr('data-tooltip'))` spełnia kilka zadań, więc rozbiujemy je na elementy i dokładniej przeanalizujemy. Fragment umieszczony wewnętrz wywołania jQuery `$()` — `$this.attr('data-tooltip')` — korzysta z funkcji `attr()`, by pobrać wartość atrybutu `data-tooltip` elementu wyzwalającego (pamiętaj, że to właśnie do niego odwołuje się zmienna `$this`). Innymi słowy, całe to wywołanie odwołuje się do aktualnego elementu wyzwalającego, odnajduje jego atrybut `data-tooltip` i pobiera jego wartość. Dla elementu wyzwalającego dodanego w kroku 3. wywołanie to zwróci łańcuch znaków '`#tip1`'; natomiast dla elementu dodanego w kroku 5. byłaby to wartość '`#tip2`'.

Po pobraniu wartości atrybutu `data-tooltip` jest ona używana w wywołaniu funkcji jQuery `$()` (zewnętrzna funkcja wywołania zapisanego w wierszu 5. powyższego kodu). Innymi słowy, w rzeczywistości kod ten sprowadza się do wywołania o postaci `$('#tip1')` lub `$('#tip2')`. Hej, ale przecież Ty to znasz! To jest zwyczajny sposób pobierania elementów przy użyciu biblioteki jQuery!

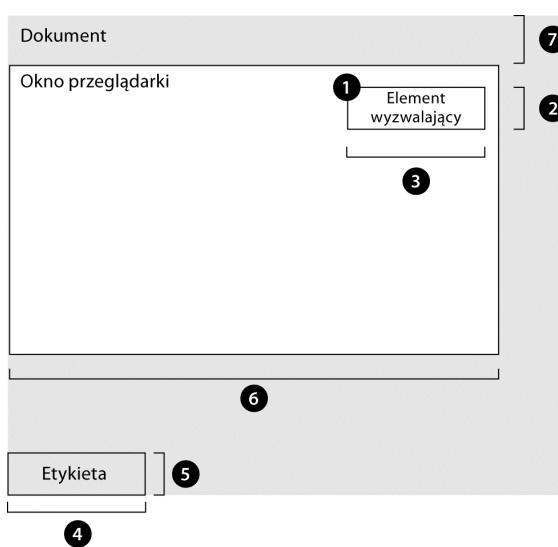
Po wykonaniu 5. wiersza powyższego fragmentu kodu w zmiennej o nazwie \$tip będzie zapisany obiekt jQuery z pobraną odpowiednią etykietą. Możesz go użyć, by wyświetlić, animować lub określić położenie etykiety na stronie.

Kolejnym zadaniem jest zgromadzenie wszystkich informacji niezbędnych do określenia prawidłowego położenia etykiety na stronie.

11. Do funkcji mouseover dodaj kod z wierszy od 6. do 12. poniższego fragmentu kodu:

```
1  $('.trigger').mouseover(function() {  
2      var ttLeft,  
3          ttTop,  
4          $this=$(this),  
5          $tip = $($this).attr('data-tooltip'),  
6          triggerPos = $this.offset(),  
7          triggerH = $this.outerHeight(),  
8          triggerW = $this.outerWidth(),  
9          tipW = $tip.outerWidth(),  
10         tipH = $tip.outerHeight(),  
11         screenW = $(window).width(),  
12         scrollTop = $(document).scrollTop();  
13  }); //koniec funkcji mouseover
```

Dodane wiersze kodu pobierają i zapisują w zmiennych informacje o położeniu oraz wymiarach kilku elementów. Schemat przedstawiony na rysunku 10.10 pomoże Ci wyobrazić sobie i zrozumieć znaczenie każdej z tych wartości. Widać na nim całą stronę WWW (przedstawioną jako szary prostokąt), która jest większa od okna przeglądarki (zaznaczonego jako czarna ramka). Strona została nieco przewinięta w dół, zatem pewien jej fragment znalazł się powyżej górnej krawędzi okna przeglądarki. Co więcej, ponieważ strona jest zarówno dłuższa, jak i szersza od okna przeglądarki, zatem pewne jej fragmenty są także ukryte poza prawą oraz dolną krawędzią okna przeglądarki.



Rysunek 10.10. Ten schemat prezentuje stronę WWW (szary obszar), która jest zarówno wyższa, jak i szersza od okna przeglądarki (oznaczonego jako czarny prostokąt). Użytkownik przeglądający tę stronę przewinął ją nieco ku dołowi, a zatem jej górna część jest niewidoczna, podobnie zresztą jak fragmenty położone przy jej prawej oraz dolnej krawędzi. Etykieta została przedstawiona jako niewielki prostokąt położony na samym dole strony, gdyż właśnie tam znajduje się bezpośrednio po wczytaniu strony. Dopiero w kroku 15. dokonasz pisania kodu odpowiedzialnego za określenie właściwego położenia etykiety oraz umieszczenie jej w tym miejscu

Wiersz 6. powyższego fragmentu kodu pobiera współrzędne elementu wyzwalającego (oznaczonego na rysunku 10.10 cyfrą 1). Ich znajomość jest niezbędna, gdyż to właśnie względem tego elementu musimy określić położenie etykiety.

W wierszach 7. i 8. wywoływane są funkcje `outerHeight()` (patrz strona 335) oraz `outerWidth()` (patrz strona 335), które pozwalają pobrać odpowiednio wysokość (cyfra 2 na rys. 10.10) oraz szerokość (cyfra 3) elementu wyzwalacza (uwzględniając przy tym wielkości wypełnienia i obramowania). Kolejne wiersze — 9. i 10. — pobierają odpowiednio szerokość (cyfra 4) oraz wysokość (cyfra 5) etykiety. Ponieważ nie chcemy, by etykieta była wyświetlana poza oknem przeglądarki, zatem musimy także znać jego szerokość (pobieramy ją w wierszu 11. powyższego fragmentu kodu, a na rysunku 10.10 została ona oznaczona cyfrą 6) i wiedzieć, czy użytkownik nie przewinął strony w dół (a jeśli przewinął, to o ile, wiersz 12., cyfra 7). Nie możesz także zapomnieć o dodaniu średnika na końcu wiersza 12., gdyż właśnie w nim kończy się instrukcja `var` rozpoczęta w wierszu 2.

Być może zastanawiasz się, do czego są Ci potrzebne te wszystkie informacje. Czy nie byłoby łatwiej określić położenie elementu wyzwalającego, a następnie wyświetlić etykietę bezpośrednio nad nim? W większości przypadków takie rozwiązanie zdałoby egzamin, jednak istnieje kilka przypadków, w których nie działałoby prawidłowo. I tak w przypadku zilustrowanym na rysunku 10.11 element wyzwalający znajduje się w prawym, górnym wierzchołku okna przeglądarki, a fragment strony jest przewinięty i ukryty poza górną krawędzią okna. Gdybyśmy teraz umieścili etykietę bezpośrednio nad elementem wyzwalającym, jej znaczna część byłaby niewidoczna. Innymi słowy, nasz kod musi działać inteligentnie — powinien określić, czy umieszczenie etykiety nad elementem wyzwalającym nie sprawi, że jej część znajdzie się poza oknem przeglądarki. Gdyby faktycznie tak miało się stać, nasz kod musi wyznaczyć nowe położenie etykiety.

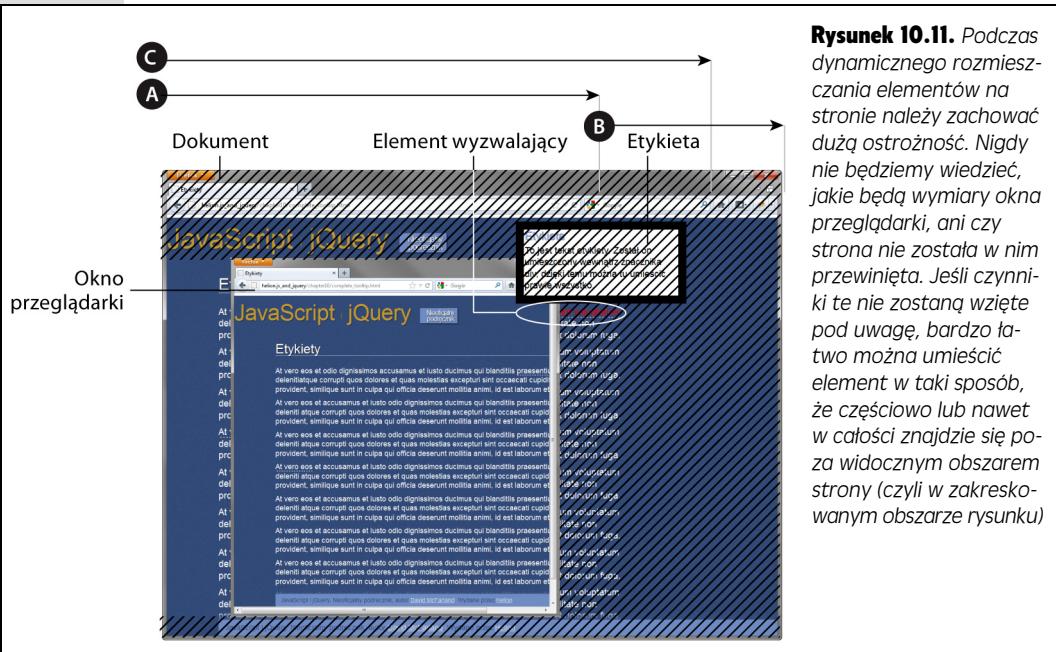
Trzeba zacząć od sprawdzenia, czy etykieta umieszczona bezpośrednio nad elementem wyzwalającym zmieści się w górnej części okna przeglądarki.

12. Do tej samej funkcji dodaj wiersze od 13. do 17. z poniższego fragmentu kodu:

```

1  $('.trigger').mouseover(function() {
2      var ttleft,
3          ttTop,
4          $this=$(this),
5          $tip = $($this.attr('data-tooltip')),
6          triggerPos = $this.offset(),
7          triggerH = $this.outerHeight(),
8          triggerW = $this.outerWidth(),
9          tipW = $tip.outerWidth(),
10         tipH = $tip.outerHeight(),
11         screenW = $(window).width(),
12         scrollTop = $(document).scrollTop();
13         if (triggerPos.top - tipH - scrollTop > 0 ) {
14             ttTop = triggerPos.top - tipH - 10;
15         } else {
16             ttTop = triggerPos.top + triggerH +10 ;
17         }
18     }); //koniec funkcji mouseover

```



Rysunek 10.11. Podczas dynamicznego rozmieszczania elementów na stronie należy zachować dużą ostrożność. Nigdy nie będziemy wiedzieć, jakie będą wymiary okna przeglądarki, ani czym strona nie została w nim przewinięta. Jeśli czynniki te nie zostaną wzięte pod uwagę, bardzo łatwo można umieścić element w taki sposób, że częściowo lub nawet w całości znajdzie się poza widocznym obszarem strony (czyli w zakresku-wanym obszarze rysunku)

W tym krótkim fragmencie kodu dzieje się całkiem sporo, jednak jego analizę warto zacząć od przyjrzenia się, gdzie dokładnie chcemy umieścić etyktę w stosunku do elementu wyzwalającego. Normalnie wyświetlimy etykietę 10 pikseli ponad elementem wyzwalającym, aby go nie przesłaniała. Aby określić współrzedną pionową etyktę, należy zacząć od pobrania pionowej współrzędnej elementu wyzwalającego, następnie odjąć od niej wysokość etyktety, a wynik pomniejszyć o dodatkowe 10 pikseli. W ramach przykładu założmy, że element wyzwalający jest umieszczony 150 pikseli pod górną krawędzią strony, a etykteta ma wysokość 100 pikseli. Chcemy umieścić etykietę tak, aby nie przesłaniała elementu wyzwalającego, należy zatem związać jego współrzędną pionową — 150 — odjąć 100, a od uzyskanego wyniku (50) odjąć jeszcze 10 (zostawiając w ten sposób niewielki odstęp pomiędzy etyktetą i elementem wyzwalającym). W rezultacie okazuje się, że etykteta powinna być umieszczona 40 pikseli poniżej górnej krawędzi dokumentu.

A co by się stało, gdyby element wyzwalający był umieszczony 10 pikseli poniżej górnej krawędzi dokumentu, a etykteta miała wysokość 100 pikseli? Gdybyśmy bezmyślnie skopiowali powyższe równanie, okazałoby się, że współrzędna pionowa etyktety wynosi -90 pikseli ($10 - 100 = -90$); innymi słowy, znalazłaby się ona ponad górną krawędzią dokumentu, czyli byłaby niewidoczna!

I właśnie w tym miejscu do akcji wkracza warunek umieszczony w wierszu 13. Od wartości pionowej współrzędnej elementu wyzwalającego odejmujemy wysokość etyktety oraz wielkość przewinięcia strony. Następnie sprawdzamy, czy uzyskany wynik jest większy od zera (gdyby był mniejszy, etykteta zostałaby umieszczona poza górną krawędzią okna przeglądarki). W tych obliczeniach musimy uwzględnić także przewinięcie strony, gdyż może się zdarzyć, że etykteta zmieści się na stronie powyżej elementu wyzwalającego, jeśli jednak strona zo-

stanie przewinięta, może się okazać, że tak umieszczona etykieta znalazła się poza obszarem strony widocznym w przeglądarce (właśnie taka sytuacja została przedstawiona na rysunku 10.11).

Jeśli ten warunek będzie spełniony, etykieta zostanie umieszczona nad elementem wyzwalającym (a wartość współrzędnej pionowej — `ttTop` — zostanie wyliczona w wierszu 14.). Jeśli jednak warunek nie zostanie spełniony, wykonany będzie wiersz 16., a etykieta pojawi się 10 pikseli poniżej dolnej krawędzi elementu wyzwalającego (jej współrzędną pionową wyliczamy poprzez pobranie współrzędnej pionowej lewego, górnego wierzchołka elementu wyzwalającego — `triggerPos.top` — i dodanie do niej jego wysokości — `triggerH`).

Koleijną czynnością będzie obliczenie poziomej współrzędnej etykiety.

Uwaga: W opisywanym tu przykładzie etykieta jest umieszczana ponad elementem wyzwalającym, jednak nie znaczy to wcale, że tak ma być. Nic nie stoi na przeszkodzie, byś zmienił kod skryptu i sprawdził, czy potrafisz wyświetlić etykietę poniżej elementu wyzwalającego bądź z jego prawej lub lewej strony.

13. Do tej samej funkcji dodaj wiersz 18. poniższego fragmentu:

```

1   $('.trigger').mouseover(function() {
2     var ttLeft,
3         ttTop,
4         $this=$(this),
5         $tip = $($this.attr('data-tooltip')),
6         triggerPos = $this.offset(),
7         triggerH = $this.outerHeight(),
8         triggerW = $this.outerWidth(),
9         tipW = $tip.outerWidth(),
10        tipH = $tip.outerHeight(),
11        screenW = $(window).width(),
12        scrollTop = $(document).scrollTop();
13        if (triggerPos.top - tipH - scrollTop > 0 ) {
14            ttTop = triggerPos.top - tipH - 10;
15        } else {
16            ttTop = triggerPos.top + triggerH +10 ;
17        }
18        var overFlowRight = (triggerPos.left + tipW) - screenW;
19    }); //koniec funkcji mouseover

```

Wyliczenie współrzędnej poziomej etykiety jest nieco bardziej złożone niż współrzędnej pionowej. W tym przypadku nie tylko musimy wiedzieć, czy fragment etykiety jest umieszczony poza prawą krawędzią okna przeglądarki, lecz także o ile poza nią wystaje. Przykładowo założmy, że współrzędna pozioma elementu wyzwalającego wynosi 850 pikseli (na rysunku 10.11 została oznaczona literą A), etykieta ma 250 pikseli szerokości (została oznaczona literą B), a okno przeglądarki ma szerokość 1000 pikseli (oznaczono ją literą C). Jeśli w takim przypadku etykieta zostanie wyświetlona w punkcie o współrzędnej poziomej 850 pikseli, jej prawa krawędź znajdzie się w miejscu o współrzędnej poziomej 1100 pikseli (A + B). A to oznacza, że prawy fragment etykiety o szerokości 100 pikseli nie będzie widoczny! By wyeliminować ten problem, musimy wiedzieć, jaki fragment etykiety wystaje poza prawą krawędź okna przeglądarki, i odpowiednio skorygować jej współrzędną poziomą.

Kod zapisany w wierszu 18. oblicza całkowitą szerokość fragmentu etykiety wystającego poza prawą krawędź okna przeglądarki (oczywiście, o ile w ogóle jest taki fragment). W tym celu wyliczamy współrzendną poziomą prawej krawędzi

etykiety, zakładając, że byłaby ona umieszczona w tym samym miejscu (w poziomie), co element wyzwalający — triggerPos.left + tipA (A + B, na rysunku 10.11). Od uzyskanego wyniku odejmujemy następnie szerokość okna przeglądarki (C). Jeśli ostateczny wynik jest wartością dodatnią, jakiś fragment etykiety znajdzie się poza oknem przeglądarki. Jeśli jednak wynik będzie ujemny, będzie to znaczyć, że w oknie przeglądarki jest na tyle dużo miejsca, by etykieta się w nim zmieściła w całości.

14. Poniżej wiersza dodanego w poprzednim kroku (czyli 18. wiersza) dodaj następujący fragment kodu:

```
if (overFlowRight > 0) {  
    ttLeft = triggerPos.left - overFlowRight - 10;  
} else {  
    ttLeft = triggerPos.left;  
}
```

Najprościej rzecz ujmując, ten fragment kodu sprawia, że jeśli wartość zmiennej overFlowRight jest większa od zera (czyli etykieta nie zmieści się w całości w oknie przeglądarki), współrzędna pozioma etykiety zostanie wyliczona jako współrzędna pozioma elementu wyzwalającego pomniejszona o wielkość, o jaką etykieta wystaje poza prawą krawędź okna przeglądarki. Pomniejszenie wyniku o dodatkowe 10 pikseli sprawia, że etykieta nawet nie będzie dotykać krawędzi okna przeglądarki. Jeśli jednak wartość zmiennej overFlowRight jest mniejsza od zera, współrzędna pozioma etykiety może być taka sama jak współrzędna pozioma elementu wyzwalającego — ttLeft = triggerPos.left;.

O rany — cała masa arytmetyki! Na szczęście, to już koniec. Teraz, kiedy już wyliczyliśmy współrzędne etykiety, możemy ją wyświetlić. W końcu!

15. Do funkcji mouseover dodaj wiersze od 24. do 28. poniższego fragmentu kodu:

```
1  $('.trigger').mouseover(function() {  
2      var ttLeft,  
3          ttTop,  
4          $this=$(this),  
5          $tip = $($this.attr('data-tooltip')),  
6          triggerPos = $this.offset(),  
7          triggerH = $this.outerHeight(),  
8          triggerW = $this.outerWidth(),  
9          tipW = $tip.outerWidth(),  
10         tipH = $tip.outerHeight(),  
11         screenW = $(window).width(),  
12         scrollTop = $(document).scrollTop();  
13         if (triggerPos.top - tipH - scrollTop > 0 ) {  
14             ttTop = triggerPos.top - tipH - 10;  
15         } else {  
16             ttTop = triggerPos.top + triggerH +10 ;  
17         }  
18         var overFlowRight = (triggerPos.left + tipW) - screenW;  
19         if (overFlowRight > 0) {  
20             ttLeft = triggerPos.left - overFlowRight - 10;  
21         } else {  
22             ttLeft = triggerPos.left;  
23         }  
24         $tip.css({  
25             left : ttLeft ,  
26             top : ttTop,  
27             position: 'absolute'  
28         }).fadeIn(200);  
29     }); // koniec funkcji mouseover
```

W końcu nadszedł moment prawdy. Korzystając z techniki tworzenia sekwencji wywołań funkcji jQuery (patrz strona 149), najpierw wywołujemy funkcję `.css()` (patrz strona 155) i określamy w ten sposób współrzędne (właściwości `left` oraz `top`) oraz sposób rozmieszczenia znacznika etykiety (właściwość `position`, której przypisujemy wartość `absolute`, gdyż chcemy go umieścić w sposób bezwzględny), a następnie funkcję `fadeIn()` (patrz strona 200), która sprawi, że etykieta stopniowo pojawi się na ekranie. Na szczęście, ukrycie etykiety, kiedy wskaźnik myszy zostanie usunięty z obszaru elementu wyzwalającego, jest znacznie łatwiejsze.

16. Dokoncz tworzenie kodu, dodając wiersze od 30. do 32.; poniżej przedstawiona została pełna, końcowa wersja kodu.

```
1  $('.trigger').mouseover(function() {  
2      var ttLeft,  
3          ttTop,  
4          $this=$(this),  
5          $tip = $($this.attr('data-tooltip')),  
6          triggerPos = $this.offset(),  
7          triggerH = $this.outerHeight(),  
8          triggerW = $this.outerWidth(),  
9          tipW = $tip.outerWidth(),  
10         tipH = $tip.outerHeight(),  
11         screenW = $(window).width(),  
12         scrollTop = $(document).scrollTop();  
13         if (triggerPos.top - tipH - scrollTop > 0 ) {  
14             ttTop = triggerPos.top - tipH - 10;  
15         } else {  
16             ttTop = triggerPos.top + triggerH +10 ;  
17         }  
18         var overFlowRight = (triggerPos.left + tipW) - screenW;  
19         if (overFlowRight > 0) {  
20             ttLeft = triggerPos.left - overFlowRight - 10;  
21         } else {  
22             ttLeft = triggerPos.left;  
23         }  
24         $tip.css({  
25             left : ttLeft ,  
26             top : ttTop,  
27             position: 'absolute'  
28         }).fadeIn(200);  
29     }); //koniec funkcji mouseover  
30     $('.trigger').mouseout(function () {  
31         $('.tooltip').fadeOut(200);  
32     }); //koniec funkcji mouseout
```

Procedura obsługi zdarzeń `mouseover` jest bardzo prosta: w odpowiedzi na usunięcie wskaźnika myszy z obszaru elementu wyzwalającego wystarczy zaciemnić wszystkie etykiety. I to wszystko. Teraz zapisz plik i wyświetl go w przeglądarce. Pełną wersję kodu przykładu można znaleźć w pliku *complete_tooltip.html* umieszczonym w katalogu *R10*.

ALARM! WTYCZKA!

Etykiety ekranowe w nieco łatwiejszy sposób

Próba utworzenia własnego narzędzia do obsługi etykiet jest doskonałym sposobem opanowania funkcji jQuery służących do określania wymiarów i położenia elementów. Jeśli jednak poszukujesz dodatkowych możliwości, takich jak pięknie wyglądające etykiety, komiksowe dymki, pobieranie zawartości dymków przy użyciu AJAX-a bądź precyzyjne umieszczanie etykiet w wybranych miejscach strony WWW, musisz wiedzieć, że istnieje wiele wtyczek jQuery udostępniających znacznie więcej możliwości niż prosty skrypt utworzony w tym rozdziale.

◆ **qTip2** (<http://craigsworks.com/projects/qtip2/>) jest bardzo rozbudowaną wtyczką obsługującą etykiety. Pozwala nie tylko na tworzenie prostych etykiet, takich jak przedstawione w naszym przykładzie, lecz także etykiet przypominających dymki z komiksowymi rozmowami; pozwala także na śledzenie ruchu wskaźnika myszy przesuwanego po ekranie, na pobieranie zawartości etykiet z serwera oraz udostępnia wiele innych możliwości, między innymi tworzenie okien dialogowych oraz rozwijalnych menu. Jak widać, wtyczka ta to prawdziwy, wielofunkcyjny zestaw narzędziowy.

◆ **jQuery Tools Tooltip** (<http://jquerytools.org/demos/tooltip/index.html>) to kolejna doskonała wtyczka do tworzenia etykiet. Generowane przez nią etykiety są bardzo atrakcyjne i mają ogromne możliwości dostosowywania. Skoro już mowa o tej wtyczce, warto zjrzeć także na stronę kolekcji narzędzi jQuery Tools (<http://jquerytools.org/tools/>). Jest ona reklamowana jako „zaginiona biblioteka interfejsu użytkownika dla stron WWW” i mimo że slogan ten jest nieco napiszony, to jednak skrypt ten zapokaja wiele potrzeb twórców stron WWW. Udogodnia między innymi narzędzia do tworzenia kart, nakładek, formularzy oraz suwaków (podobnie jak wtyczka AnythingSlider opisana na stronie 325).

◆ **Wtyczka jQuery UI Tooltip** (<http://wiki.jqueryui.com/w/page/12138112/Tooltip>). Dowiedziałeś się już o bibliotece jQuery UI w ramce zamieszczonej na stronie 325. Zawiera ona sporo komponentów interfejsu użytkownika oraz innych narzędzi przeznaczonych dla projektantów i twórców stron WWW. Choć w czasie pisania tej książki twórcy tego projektu jeszcze nie udostępnili oficjalnie wtyczki do tworzenia etykiet, jednak prace nad nią są już całkiem zaawansowane, a według planu ma ona zostać udostępniona w wersji 1.9 biblioteki jQuery UI. Wszystkie wtyczki wchodzące w skład biblioteki jQuery UI są doskonale.

IV

CZĘŚĆ

AJAX — komunikacja z serwerem sieciowym

- Rozdział 11. „Wprowadzenie do AJAX-a”
- Rozdział 12. „Flickr oraz Google Maps”

Wprowadzenie do AJAX-a

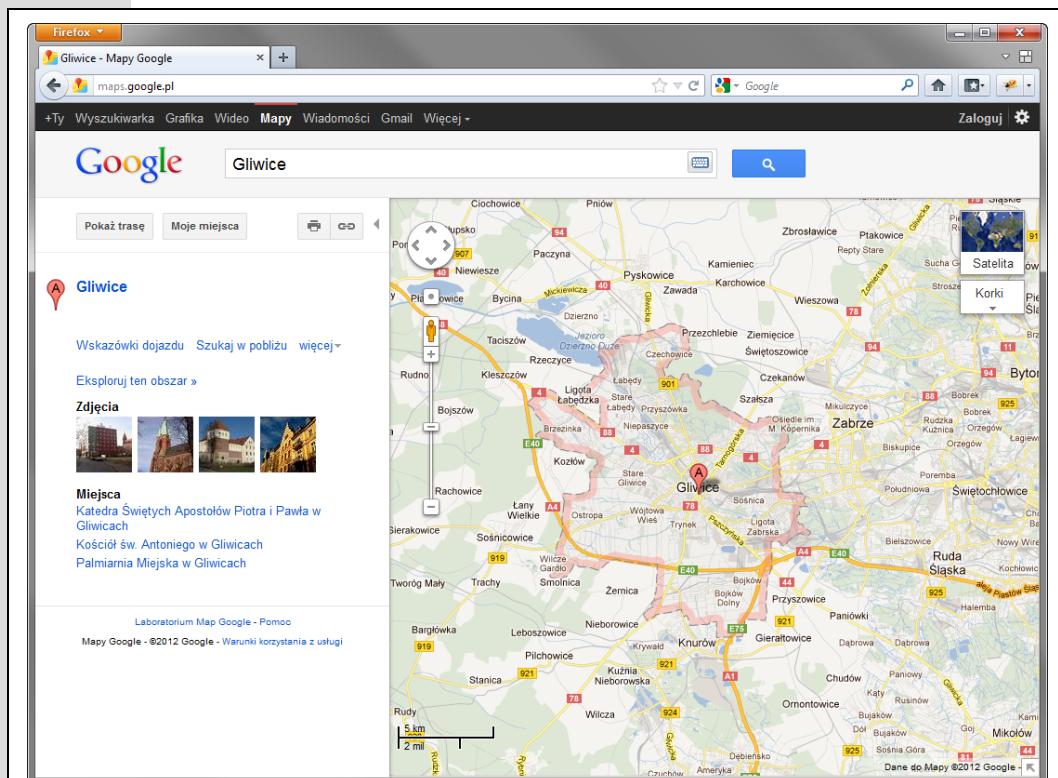
JavaScript to wspaniały język, ale nawet jego możliwości są ograniczone. Jeśli chcesz wyświetlić informacje z bazy, wysłać e-mail z danymi z formularza lub wczytać dodatkowy kod HTML, musisz nawiązać komunikację z serwerm sieciowym. Zwykle odbywa się to przez pobranie nowej strony WWW. Na przykład przy wyszukiwaniu informacji w bazie przeglądarka zwykle opuszcza pierwotną stronę i wyświetla wyniki w nowej.

Oczekiwanie na wczytanie nowej strony oczywiście zajmuje czas. Ponadto po głębszym zastanowieniu się można dojść do wniosku, że znikanie i ponowne pojawianie się stron todziwna technika. Wyobraź sobie, że przy otwieraniu każdego nowego pliku w edytorze Microsoft Word wszystkie menu, panele i okna nagle znikają, a następnie pojawiają się wraz z otwartym plikiem. Niektóre witryny, na przykład Facebook, Flickr, Twitter, Google Maps i Gmail, burzą ściśly podział na witryny WWW i tradycyjne programy komputerowe. Użytkownicy chcą, aby witryny działały i reagowały szybciej, podobnie jak standardowe aplikacje. AJAX to technologia, która umożliwiła powstanie aplikacji sieciowych nowej generacji.

AJAX umożliwia żądanie i pobieranie odpowiedzi od serwera sieciowego oraz aktualizowanie wyświetlanych materiałów bez konieczności wczytywania całej nowej strony. Efektem są witryny, które dużo szybciej reagują na działania użytkowników. W witrynie Google Maps (patrz rysunek 11.1) 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>), Gmail (www.gmail.com) i Google Suggest (www.google.com/webhp?complete=1@h1=en). AJAX to akronim od zwrotu *Asynchronous JavaScript and XML*, jednak w przeciwieństwie do języków HTML, JavaScript i CSS nie jest to „oficjalna”



Rysunek 11.1. 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

technologia. To po prostu określenie, które opisuje współdziałanie kilku technologii — języka JavaScript, mechanizmów przeglądarek i serwerów sieciowych — przy pobieraniu i wyświetlaniu nowych materiałów bez konieczności wczytywania całych stron WWW.

Uwaga: Jeśli chcesz przeczytać pierwszy artykuł w blogu, w którym użyto nazwy AJAX, odwiedź stronę <http://www.adaptivepath.com/ideas/ajax-new-approach-web-applications>.

Współczesne przeglądarki umożliwiają korzystanie z języka JavaScript do przesyłania żądań na serwer sieciowy, który z kolei zwraca odpowiednie dane do przeglądarki. Skrypty JavaScript przyjmują te dane i używają ich. Jeśli klikniesz w witrynie Google Maps strzałkę skierowaną w góre, kod JavaScript zażąda od serwera Google nowych danych, a następnie użyje ich do wyświetlenia odpowiedniego fragmentu mapy.

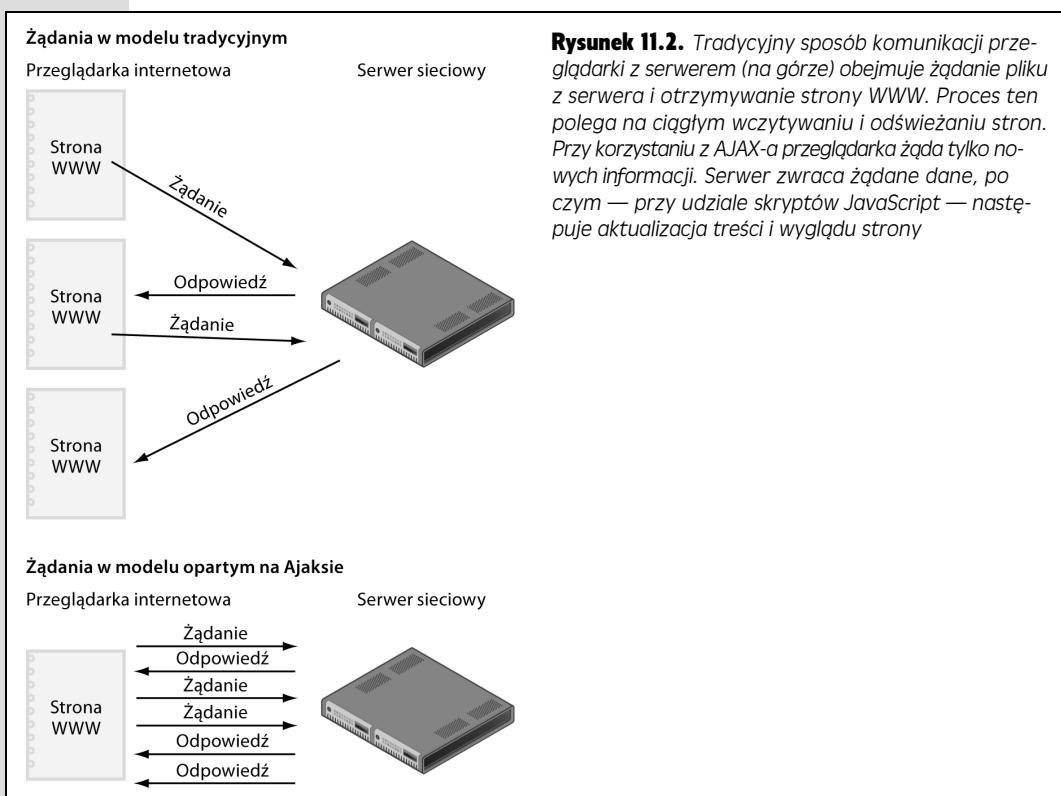
Choć pewnie nie pracujesz nad nową wersją witryny Google Maps, AJAX umożliwia też wykonywanie wielu prostych operacji. Są to na przykład:

- **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 pozostały zawartości strony, którą w przeciwnym razie przeglądarka musiałaby wczytywać. Jak utworzyć takie rozwiązanie, dowiesz się na stronie 384.
- **Przesyłanie formularzy i natychmiastowe wyświetlanie wyników.** Wyobraź sobie formularz rejestracji 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 388 zobaczysz, jak utworzyć taki formularz za pomocą AJAX-a.
- **Logowanie się bez opuszczania strony.** Strona z małym formularzem logowania to następnie 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 AJAXowi 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, dzięki czemu usprawnia korzystanie z witryny. Dzięki AJAXowi można tworzyć witryny, które bardziej przypominają tradycyjne aplikacje niż strony WWW.

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. Rysunek 11.2 przedstawia różnicę między komunikacją serwera sieciowego z tradycyjnymi stronami HTML i ze stronami opartymi na AJAX-ie.



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 istotny składnik umożliwiający działanie AJAX-a. Jest to **obiekt XMLHttpRequest**. Ten element o dziwnej nazwie sprawia, że kod JavaScript może nawiązać komunikację z serwerem sieciowym 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 362.

- **Język JavaScript** wykonuje większość skomplikowanych zadań w AJAX-ie. 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 138) 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 sieciowy** 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 381) lub dane w formacie JSON (patrz strona 386). 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 kolejnych rekordów z bazy, a serwer powinien wtedy zwrócić informacje zawierające te dane.

Ten element układanki bywa skomplikowany i zwykle wymaga użycia kilku technologii: serwera sieciowego, serwera aplikacji i serwera bazodanowego. Serwer sieciowy 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 bazodanowy*. 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 bazodanowy 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 sieciowy* i *serwer bazodanowy* oznaczają oprogramowanie, które może działać na tym samym komputerze (jest to często stosowane rozwiązanie).

Różne serwery sieciowe, serwery aplikacji i serwery bazodanowe można łączyć na wiele sposobów. Na przykład można korzystać z serwera sieciowego IIS Microsoftu, ASP.NET (serwera aplikacji) i narzędzia SQL Server (serwera bazodanowego). Inny zestaw to Apache (serwer sieciowy), PHP (serwer aplikacji) i MySQL (serwer bazodanowy).

Uwaga: Zestaw Apache, PHP i MySQL (często nazywany AMP) jest bezpłatny i bardzo popularny. Większość dostawców usług hostingowych korzysta z tych serwerów. Także przykłady przedstawione w tej książce oparto na tym zestawie (patrz ramka poniżej).

WIEDZA W PIGUŁCE

Konfigurowanie serwera sieciowego

AJAX współpracuje z serwerem sieciowym. W końcu podstawowym zadaniem tej technologii jest umożliwienie wysyłania i pobierania informacji z serwera za pomocą kodu JavaScript. Choć prawie wszystkie (z jednym wyjątkiem) przykłady w tym i następnym rozdziale działają na komputerze lokalnym bez konieczności używania serwera sieciowego, prawdopodobnie będziesz chciał go uruchomić, aby lepiej poznać świat AJAX-a. Jeśli masz już witrynę dostępną w internecie, możesz przetestować programy ajaksowe przez przeniesienie plików na używany serwer sieciowy. 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 sieciowy na własnym komputerze, aby móc 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 WAMPP (<http://www.wampserver.com/en/>). Jest to bezpłatny

pakiet instalacyjny, który konfiguruje wszystkie elementy potrzebne do zasymulowania działania prawdziwych witryn dostępnych w internecie.

Na stronie <http://uptospeedguides.com/wamp/> znajdziesz klip wideo demonstrujący sposób instalacji pakietu.

Miłośnicy komputerów Mac mogą skorzystać z łatwego w użyciu programu MAMP (www.mamp.info/en/download.html), który obejmuje serwery Apache, PHP i MySQL. Także ten pakiet jest bezpłatny. Klip demonstrujący sposób jego instalacji można znaleźć na stronie <http://uptospeedguides.com/mamp/>.

Przykład opisany na stronie 381 wymaga zestawu AMP. 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. Jest wiele materiałów opisujących instalację serwera IIS. Jeśli chcesz używać go w systemie Vista, odwiedź stronę <http://learn.iis.net/page.aspx/85/installing-iis7/>. Użytkownicy systemu Windows XP Pro powinni zajrzeć pod adres <http://www.webwiz.co.uk/kb/asp-tutorials/installing-iis-winxp-pro.htm>.

Komunikacja z serwerem sieciowym

Podstawą każdego programu ajaksowego jest obiekt XMLHttpRequest (czasem nazywany też XHR). Jest on wbudowany we współczesne przeglądarki, które umożliwiają przesyłanie informacji na serwer sieciowy 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 sieciowy. W najprostszej postaci instrukcja tworząca obiekt XMLHttpRequest w kodzie JavaScript wygląda następująco:

```
var newXHR = new XMLHttpRequest();
```

Niestety, AJAX powoduje tyle problemów związanych z brakiem zgodności między przeglądarkami, że do zgłoszania ajaksowych żądań najlepiej użyć biblioteki JavaScript, na przykład jQuery. Sposób obsługi AJAX-a za pomocą tej biblioteki poznasz na stronie 365.

2. Używanie metody `open()` obiektu `XHR` do określenia rodzaju przesyłanych danych i ich docelowej lokalizacji.

Dane można przesyłać na dwa sposoby — używając metody GET lub POST (te same możliwości masz przy wysyłaniu formularzy HTML). Metoda GET przesyła dane na serwer sieciowy w adresie URL, na przykład `show.php?productID=34`. Tu dane to informacje po znaku ?, czyli `productID=34`. Jest to para 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.

Uwaga: Adres URL podany w metodzie `open()` musi prowadzić do dokumentu z tej samej witryny, w której znajduje się strona zgłaszająca żądanie. Z uwagi na zabezpieczenia przeglądarki nie pozwalają kierować żądań ajaksowych do innych domen.

Metoda POST przesyła dane niezależnie od adresu URL. Programiści zwykle używają metody GET do pobierania danych z serwera, a metody POST — do aktualizowania informacji na serwerze (na przykład do dodawania, modyfikowania lub usuwania rekordów w bazie danych). Na stronie 372 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:

```
newXHR.open('GET', 'shop.php?productID=34');
```

3. Tworzenie funkcji obsługującej pobrane dane.

Kiedy serwer sieciowy 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 klikniętą przez użytkownika przy ocenie produktu).

Potrzebne są jeszcze pewne dodatkowe operacje, jednak do zarządzania szczegółami posłuży biblioteka jQuery, dlatego musisz jedynie pamiętać, że wywoływana zwróci funkcja zawiera kod JavaScript, który obsługuje odpowiedź zwróconą przez serwer.

4. Wysyłanie żądania.

Aby przesyłać informacje na serwer sieciowy, 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:

```
newXHR.send(null);
```

Słowo `null` określa, że skrypt nie przesyła żadnych dodatkowych danych. Pamiętaj, że w metodzie `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()`:

```
newXHR.send('q=javascript');
```

Także tu nie musisz martwić się szczegółami. W kolejnym punkcie zobacysz, 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 oczekивать na odpowiedź od serwera.

5. Pobieranie odpowiedzi.

Kiedy serwer przetworzy żądanie, przesyła odpowiedź do przeglądarki. Za obsługę odpowiedzi odpowiada wywoływana zwróci 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).

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 386). 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 zwróci 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

Miedzy przeglądarkami występuje wiele różnic, dlatego aby ajaksowe programy działały w Internet Explorerze, Firefoksie, Safari i Operze, trzeba utworzyć wiele dodatkowego kodu. Ponadto choć podstawowy proces korzystania z obiektu XMLHttpRequest nie jest skomplikowany, to przy każdym zgłoszeniu trzeba powtarzać wszystkie operacje, dlatego użycie biblioteki JavaScript pozwala przyspieszyć tworzenie ajaksowych aplikacji.

Biblioteka jQuery udostępnia kilka funkcji, które znacznie upraszczają cały proces używania AJAX-a. Po przyjrzeniu się pięciu krokom obsługi żądań ajaksowych (patrz strona 363) można zauważać, że fragmenty wykonujące istotne dla skryptu operacje (czyli kod przetwarzający odpowiedź zwróconą przez serwer) trzeba dodać tylko w jednym, 3. kroku. Biblioteka jQuery upraszcza wszystkie pozostałe etapy, dlatego można skoncentrować się na pisaniu ciekawego kodu.

Używanie funkcji load()

Najprostsza funkcja ajaksowa biblioteki jQuery to `load()`. Wczytuje ona plik HTML do określonego elementu strony. Założmy, 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!).

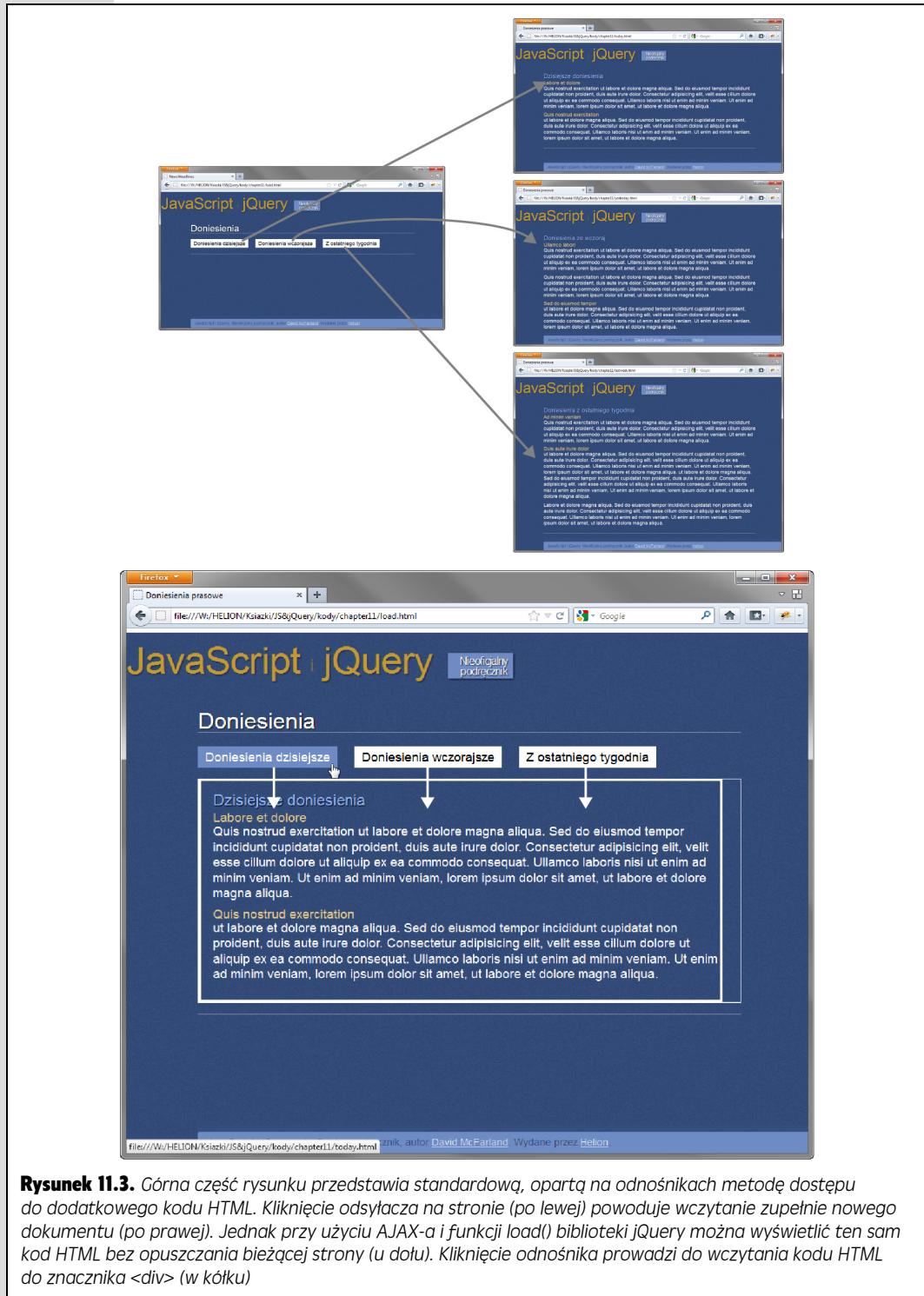
Inne podejście polega na wczytaniu wybranych wiadomości do pola z artykułami na aktualnie widocznej stronie. Oznacza to, że kiedy użytkownik wybierze nową kategorię informacji, przeglądarka zażąda z serwera nowego pliku HTML, a następnie umieści go w obszarze przeznaczonym na wiadomości, nie przechodząc przy tym do następnej strony (patrz rysunek 11.3).

Aby wywołać funkcję `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. Założmy, że na stronie znajduje się znacznik `<div>` o identyfikatorze `headlines` i chcesz zapisać w tym elemencie kod HTML z pliku `today_news.html`. Można to zrobić w następujący sposób:

```
$('#headlines').load('today_news.html');
```

Kiedy skrypt uruchomi ten kod, przeglądarka zażąda pliku `today_news.html` z serwera sieciowego. Po pobraniu go przeglądarka zastąpi bieżącą zawartość znacznika `<div>` o identyfikatorze `headlines` kodem nowego pliku.

W żadanym 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 funkcja `load()` tylko dołącza jego kod do aktualnej (kompletnej) strony WWW.



Rysunek 11.3. Góra 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 `<div>` (w kółku)

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 funkcji `load()` biblioteki jQuery (opisanej we wcześniejjszej części rozdziału), 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 sieciowym znajduje się inny skrypt, który wykonuje zadania niemożliwe do obsługicia 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, .NET, JSP, ASP lub Cold Fusion (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, and JavaScript* (wydawnictwo O'Reilly), *Head First PHP & MySQL* (wydawnictwo O'Reilly) lub *PHP Solutions: Dynamic Web Design Made Easy* (wydawnictwo Friends of Ed). Są do wartościowe podręczniki dla początkujących.

Dostępnych jest też wiele bezpłatnych materiałów do nauki języka PHP. Seria samouczków PHP 101 (<http://devzone.zend.com/node/view/id/627>) 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 (www.w3schools.com/PHP).

Uwaga: Można wczytywać pliki HTML pochodzące tylko z tej samej witryny, w której działa bieżąca strona. Nie możesz użyć funkcji `load()` na przykład do wczytania strony głównej witryny Google do elementu `<div>` strony z własnej witryny. Możesz jednak wyświetlić stronę z innej witryny za pomocą ramki wewnętrzwerszowej. Technikę tę zastosowano we wtyczce FancyBox, którą opisałem na stronie 259.

Kiedy używasz funkcji `load()`, musisz zwrócić szczególną uwagę na ścieżki prowadzące do plików. Adres URL przekazywany do tej funkcji 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 wywołująccej funkcję `load()`. Jeśli używasz ścieżek podawanych względem dokumentu (patrz ramka na stronie 41), 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 rozwiążanie tego problemu — wystarczy używać ścieżek podawanych względem katalogu głównego lub upewnić się, że wczytywany plik znajduje się w tym samym katalogu co strona wywołująca funkcję `load()`.

Funkcja `load()` pozwala nawet określić, która część pobranego pliku HTML ma znaleźć się na stronie. Założmy, 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. Jeśli chcesz wstawić tylko zawartość elementu `<div>` o identyfikatorze news z pliku `todays_news.html`, możesz użyć 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 zobaczyś, jak zastosować tę technikę.

Przykład — korzystanie z funkcji load()

W tym przykładzie użyjesz biblioteki jQuery, aby zamiast tradycyjnej metody otwierania stron HTML techniką „kliknij i wczytaj” (patrz rysunek 11.3, u góry) zastosować bardziej interaktywnym podejście, które zastępuje treść bieżącej strony nowym kodem HTML (patrz rysunek 11.3 — 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 11.4. Strona zawiera listę wypunktowaną odnośników, z których każdy wskazuje na inną stronę z różnymi wiadomościami. Lista ta znajduje się w znaczniku `` 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 funkcji `load()` biblioteki jQuery, kliknięcie jednego z odnośników spowoduje umieszczenie informacji w tym elemencie `<div>`.



Rysunek 11.4. Przy używaniu języka JavaScript do dodawania treści do strony programiści często dodają pusty znacznik `<div>` 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 `<div>` (`<div id="headlines">`). Przy użyciu AJAX-a można w łatwy sposób umieścić w nim zawartość dowolnego pliku, do którego prowadzą odnośniki ze środkowej części strony

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 dadasz, 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: Funkcji `load()` można używać bezpośrednio z poziomu własnego dysku twardego, bez korzystania z serwera sieciowego, dlatego aby uruchomić ten przykład, nie musisz instalować na komputerze serwera (patrz ramka na stronie 362).

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. Aby zastosować stopniowe wzbogacanie, należy zablokować standardowe działanie odnośnika, pobrać z niego adres URL, a następnie wczytać odpowiedni plik na stronę i umieścić jego zawartość w pustym znaczniku `<div>`. To naprawdę proste.

Uwaga: Informacje o pobieraniu przykładowych plików znajdziesz na stronie 43.

Tworzenie kodu

- Otwórz w edytorze tekstu plik `load.html` z katalogu `R11`.

Należy zacząć od przypisania zdarzenia `click` do każdego odnośnika z listy wypunktowanej z głównej części strony. Ta lista (znacznik ``) ma identyfikator `newslinks`, dlatego można użyć jQuery do łatwego pobrania wszystkich odsyłaczy i wywołania dla nich funkcji `click()`.

- 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 174).

Następny krok polega na pobraniu adresu URL każdego odnośnika.

- 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 148 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. Funkcja `attr()` (patrz strona 159) służy do pobierania i ustawiania atrybutów tych odnośników. Tu funkcja ta pobiera wartość atrybutu `href`, co pozwala ustalić adres URL strony, do której prowadzi dany odsyłacz. W kolejnym kroku użyjesz tego adresu wraz z funkcją `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 (możesz to zrobić teraz), zauważysz, że kliknięcie odnośników nie powoduje wczytywania 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 187).

- 6. Zapisz plik i wyświetl go w przeglądarce. Kliknij jeden z odnośników.**

Teraz pojawi się następny problem, widoczny na rysunku 11.5. Funkcja `load()` działa, jednak pobrany plik zawiera dużo zbędnego kodu HTML, między innymi baner, kod układu strony, ramkę boczną i stopkę. Wszystkie te elementy są już widoczne na bieżącej stronie. Potrzebny jest tylko fragment pobranej strony — obszar z nowymi elementami. Na szczęście także tu można skorzystać z funkcji `load()`.



Rysunek 11.5. Funkcja `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 powtórzone nagłówki, ramkę boczną lub stopkę, będzie wyglądał jak strona wewnętrz innej strony

7. Znajdź wiersz z funkcją `load()` i dodaj fragment `+ ' #newsItem'` po argumentie `url`. Gotowy kod powinien wyglądać następująco:

```
$('#links a').click(function() {
    var url=$(this).attr('href');
    $('#headlines').load(url + ' #newsItem');
    return false;
});
```

Na stronie 368 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 65.

8. Zapisz plik i wyświetl go w przeglądarce. Kliknij odnośniki, aby je przetestować.

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 kilku tylko wierszy kodu! Gotową wersję tego przykładu znajdziesz w pliku `complete_load.html` w katalogu `R11`.

Funkcje `get()` i `post()`

Funkcja `load()` (patrz strona 365) to prosty sposób na pobranie z serwera sieciowego kodu HTML i dodanie go do strony. Jednak serwer nie zawsze zwraca zwykły kod HTML. Może też przesyłać 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 381) lub obiekt w formacie JSON (patrz strona 386). Nie należy po prostu dodawać takich danych do strony. Najpierw trzeba je przetworzyć i wygenerować potrzebny kod HTML.

Funkcje `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 363, 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 374.

Podstawowa składnia tych funkcji wygląda następująco:

```
$.get(url, data, callback);
```

Lub:

```
$.post(url, data, callback);
```

W odróżnieniu od większości innych funkcji biblioteki jQuery przed metodami `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 zwrocone przez serwer (szczegółowe informacje o funkcjach wywoływanych zwrotnie znajdziesz na stronie 209).

W momencie wywołania funkcji `get()` lub `post()` przeglądarka wysyła 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 w jakiś sposób aktualizuje zawartość strony. Będziesz miał okazję zobaczyć takie rozwiązanie w działaniu na stronie 381.

Formatowanie danych przesyłanych na serwer

Programy JavaScript używające AJAX-a zwykle przesyłają dane na serwer. Na przykład aby pobrać informacje na temat produktu zapisanego w bazie danych, trzeba przesłać numer reprezentujący dany towar. Kiedy serwer otrzyma liczbę w żądaniu od obiektu XMLHttpRequest, 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 literała 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');
```

CZĘSTO ZADAWANE PYTANIA

Metoda GET czy POST?

Dwie metody przesyłania danych na serwer sieciowy, *GET* i *POST*, wyglądają bardzo podobnie. Której z nich powinieneś używać?

Trudno udzielić jednoznacznej odpowiedzi na to pytanie. W niektórych sytuacjach programista nie ma wyboru. Założymy, że przesydasz 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 zatrzymać się na chwilę 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, dla tego 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 przesydasz na serwer duży zbiór danych z formularza (na przykład składający się z setek słów artykułu w blogu), użyj właśnie tego sposobu. Metoda *GET* ma wbudowane ograniczenie ilości przesyłanych danych. Jest ono różne w poszczególnych przeglądarkach, jednak zwykle wynosi kilka tysięcy znaków. Do przesyłania danych z formularzy zawierających więcej niż kilka pól programiści zazwyczaj używają metody *POST*.

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.

Jeśli chcesz przesyłać 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 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' // Bląd.
```

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 odstęp powieli 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);
```

Literaty 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 157 dowiedziałeś się, że literaty obiektowe języka JavaScript umożliwiają przechowywanie par nazwa – wartość. Podstawowa struktura takiego literatu wygląda następująco:

```
{
  nazwa1: 'wartość1',
  nazwa2: 'wartość2'
}
```

Literat obiektowy można przekazać bezpośrednio do funkcji get() lub post(). W poniższym kodzie użyto łańcucha znaków z zapytaniem:

```
$.post('rateMovie.php', 'rating=5');
```

Aby użyć literatu obiektowego, należy wprowadzić następujące zmiany:

```
$.post('rateMovie.php', { rating: 5 });
```

Literaty 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);
```

Literat obiektowy można także przekazać bezpośrednio w wywołaniu funkcji post():

```
var data = $.post('rankMovie.php',
{
  rating: 5,
  user: 'Robert'
});
// koniec post
```

Funkcja serialize() biblioteki jQuery

Tworzenie łańcucha znaków z zapytaniem lub literalu 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 literal 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()`. Założymy, 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. Kiedy serwer ją prześle, wywoływana zwrotnie funkcja 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 381, a omówienie formatu JSON znajdziesz na stronie 386).

Drugi argument wywoływanej zwróciście funkcji 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 zwróciście funkcja otrzyma jako status komunikat o błędzie.

Wywoływana zwróciście funkcja przetwarza pobrane informacje i zazwyczaj aktualizuje stronę WWW, na przykład następuje przesyłany 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 149). Inne metody manipulowania modelem DOM strony opisano w rozdziale 4.

Aby zrozumieć cały cykl zgłoszania żądania i przetwarzania odpowiedzi, przyjrzyj się prostemu przykładowi oceniania filmu (patrz rysunek 11.6). 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.



``

Oceń film

Kliknij, aby oddać głos:

- 1 gwiazdka
- 2 gwiazdki
- 3 gwiazdki
- 4 gwiazdki
- 5 gwiazdek



Rysunek 11.6. Na tej stronie użytkownik może kliknąć odnośnik, aby ocenić film (na górze). Przy użyciu AJAX-a można przesyłać ocenę na serwer bez opuszczania strony. Na podstawie odpowiedzi zwróconej przez serwer można następnie zaktualizować zawartość strony (na dole)



`<div id="message">`

Oceń film

Twój głos został dodany.
Średnia ocena filmu to 3 gwiazdki.



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 film (`movie=123`). Można użyć kodu JavaScript do przechwytywania kliknięć tych odnośników i przekształcania ich na ajaksowe wywołania kierowane na serwer:

```

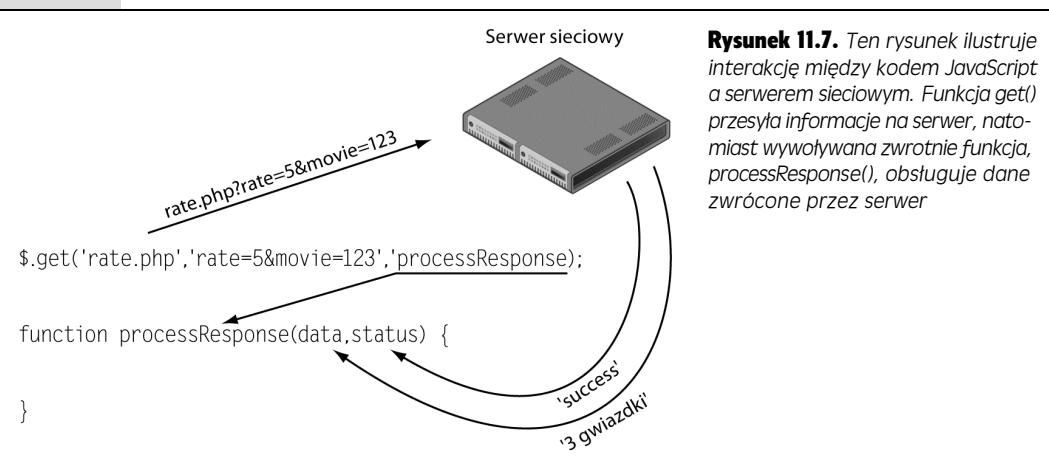
1 $('a').click(function() {
2   var href=$(this).attr('href');
3   var querystring=href.slice(href.indexOf('?')+1);
4   $.get('rate.php', querystring, processResponse);
5   return false; //Blokowanie działania odnośnika.
6 });

```

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 449), która pobiera fragment łańcucha znaków, i metoda `indexOf()` (patrz strona 448), 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 11.7), 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.



Uwaga: Jeśli chcesz przypomnieć sobie działanie funkcji i sposoby ich tworzenia, wróć do strony 110.

Pora utworzyć wywoływaną zwrotnie funkcję. Przyjmuje ona dane i łańcuch znaków ze statusem odpowiedzi (jeśli serwer zwrócił informacje, ma ona wartość 'success'). Pamiętaj, że nazwę wywoływanej zwrotnie funkcji należy określić w żądaniu (wiersz 4. kodu z poprzedniej strony). 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 149). Skrypt zastępuje zawartość znacznika `<div>` o identyfikatorze `message` nowym kodem HTML. Przykładowy efekt przedstawia dolna część rysunku 11.6.

W tym przykładzie wywoływaną zwrotnie funkcję zdefiniowano poza funkcją `get()`. Jednak jeśli chcesz umieścić cały kod związany z AJAX-em w jednym miejscu, możesz użyć funkcji anonimowej (patrz strona 160):

```
$.get('file.php', data, function(data,status) {
    // Tu kod wywoływanej zwrotnie funkcji.
});
```

Niżej pokazałem, w jaki sposób można zmienić 4. wiersz kodu ze strony 378, 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 funkcji `.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 378 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.

Przykład — korzystanie z funkcji `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 sieciowego, w którym będziesz mógł przetestować strony. W ramce na stronie 362 znajdziesz informacje o instalowaniu serwera na potrzeby testów.

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 funkcji `get()` lub `post()`, a serwer zwraca informacje w formacie XML, argument `data` przekazywany do wywoływanej zwróci funkcji (patrz strona 209) 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 wywoływanej zwróci funkcji. 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 143), a także filtrów biblioteki jQuery (patrz strona 146).

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`. Ponieważ zmienna ta zawiera plik XML, kod powoduje, że jQuery znajdzie znacznik `<content>` w danych XML.

Aby lepiej poznać format XML, odwiedź stronę www.w3schools.com/XML. Jeśli chcesz dowiedzieć się, jak generować dane XML po stronie serwera, zajrzyj pod adres www.w3schools.com/XML/xml_server.asp. Dodatkowe informacje o funkcji `find()` biblioteki jQuery znajdziesz na stronie <http://api.jquery.com/find/>.

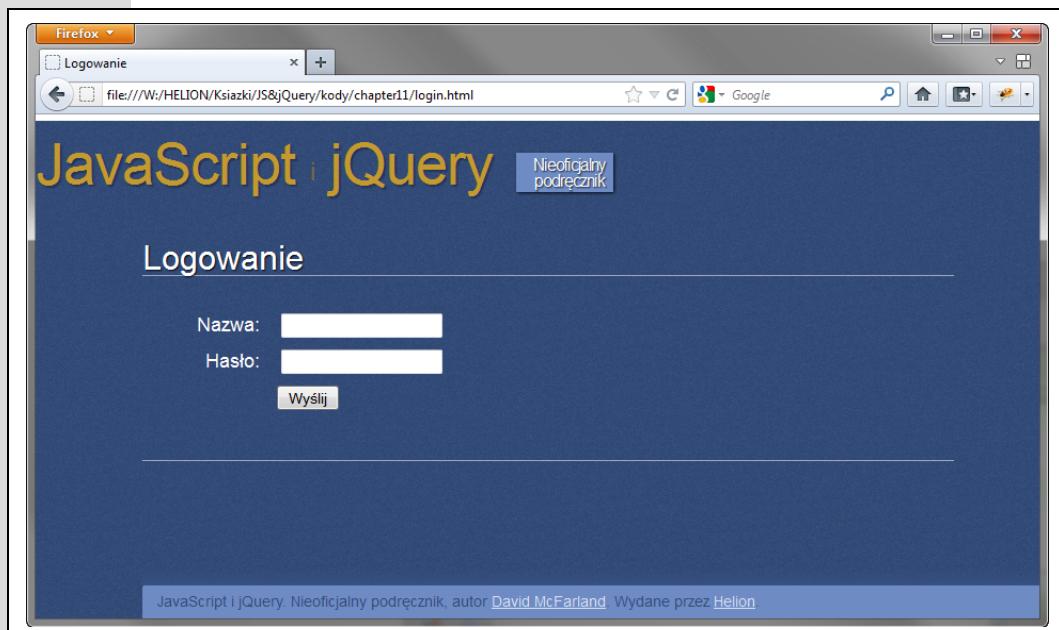
Omówienie przykładu

Rozpoczynasz pracę od formularza widocznego na rysunku 11.8. 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.

Aby obsługiwać formularz przy użyciu AJAX-a, należy przesyłać dane uwierzytelniające za pomocą obiektu `XMLHttpRequest`. Serwer zwróci następnie komunikat do wywoływanej zwróci funkcji, która usunie formularz i wyświetli informację o udanym logowaniu, jeśli dane były poprawne, lub komunikat o błędzie, jeśli wystąpiły problemy.

Tworzenie kodu

W uchwadze na stronie 43 znajdziesz informacje o pobieraniu przykładowych plików. Wyjściowy dokument zawiera kod HTML formularza. Należy do niego dodać kod oparty na jQuery i AJAX-ie.



Rysunek 11.8. Strona logowania jest całkiem prosta — zawiera kilka pól i przycisk *Wyślij*. Nie ma powodu, aby opuszczać tą stronę po zalogowaniu się użytkownika. Za pomocą AJAX-a można przesyłać dane uwierzytelniające, a następnie poinformować internautę o tym, czy logowanie zakończyło się powodzeniem, czy porażką

1. Otwórz w edytorze tekstu plik *login.html* z katalogu *R11*.

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 przesyłać na serwer. Można to zrobić przez znalezienie każdego pola, określenie wartości wpisanej przez użytkownika i utworzenie łańcucha przez połączenie poszczególnych informacji. Na szczeble funkcja `serialize()` biblioteki jQuery pozwala wykonać wszystkie te operacje w jednym kroku.

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

```
var formData = $(this).serialize();
```

Ten wiersz tworzy nową zmienną na dane z formularza, a następnie wywołuje dla tego formularza funkcję `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 162). Funkcja `serialize()` (patrz strona 376) pobiera nazwy i wartości pól formularza, a następnie przekształca je przed przesłaniem na serwer na odpowiedni format.

Teraz należy użyć funkcji `post()` do skonfigurowania obiektu XMLHttpRequest.

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

```
$.get('login.php', formData, processData);
```

Ta instrukcja przekazuje do funkcji `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 wywoływanej zwrotnie funkcji, która będzie przetwarzać odpowiedź serwera. Teraz należy przygotować tę funkcję.

5. Dodaj następny pusty wiersz i wpisz w nim poniższy kod:

```
1 function processData(data) {  
2  
3 } // Koniec funkcji processData.
```

Te wiersze tworzą szkielet wywoywanej zwrotnie funkcji. 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`.

Skrypt na podstawie odpowiedzi serwera wyświetla odpowiedź z informacją o udanym lub nieudanym logowaniu. Do obsługi takiego rozwiązania doskonale nadaje się instrukcja warunkowa.

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 witryny w podobny sposób, jednak większość z nich wymaga przygotowania bazy danych lub skonfigurowania różnych ustawień serwera sieciowego. Opis tych zagadnień wykracza poza zakres tej książki. Kompletny, oparty na języku PHP skrypt logowania znajdziesz na stronie <http://www.html-form-guide.com/php-form/php-login-form.html>. W klipie wideo na stronie <http://www.youtube.com/watch?v=4oSCuEtxRK8> pokazano, w jaki sposób napisać niezbędny skrypt PHP.

6. W funkcji `processData()` (w wierszu 2. z etapu 5.) wpisz poniższy kod:

```
1 if (data=='pass') {  
2   $('#content').html('<p>Logowanie zakończyło się  
3   powodzeniem!</p>');
```

Wiersz 1. sprawdza, czy serwer zwrócił łańcuch znaków 'pass'. Jeśli tak, logowanie zakończyło się powodzeniem i skrypt wyświetli informujący o tym komunikat (wiersz 2.). Formularz znajduje się w znaczniku `<div>` o identyfikatorze `content`, dlatego instrukcja `$('#content').html('...')`

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     $('#content').html('<p>Logowanie zakończyło się
powodzeniem!</p>');
4   } else {
5     $('#formwrapper').prepend('<p id="fail">Nieprawidłowe dane
6   →uwierzytelniające. Spróbuj ponownie.</p>');
7   }
8 } //Koniec funkcji processData.
```

Wiersz 5. wyświetla informację o nieudanym logowaniu. Zauważ, że użyto tu funkcji `prepend()` (patrz strona 151). 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 sieciowego znajdziesz na stronie 362.

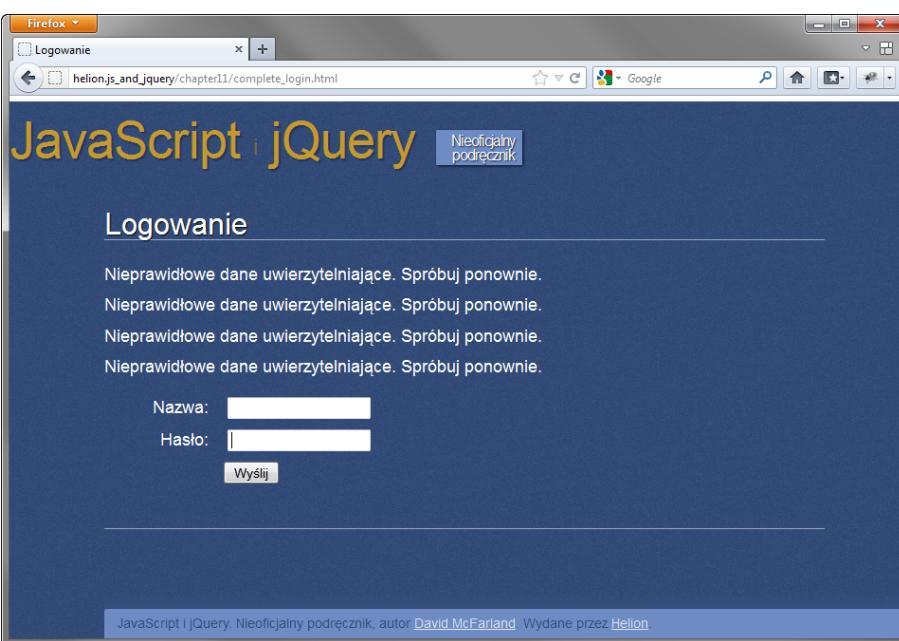
9. Spróbuj zalogować się w witrynie.

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 11.9). Funkcja `prepend()` nie usuwa pierwszego komunikatu o błędzie, a jedynie dodaje po raz wtóry ten sam tekst. Nie jest to dobre rozwiązanie.

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 umieszczone 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     $('#content').html('<p>Logowanie zakończyło się
powodzeniem!</p>');
4   } else {
5     if ($('#fail').length==0) {
6       $('#formwrapper').prepend('<p id="fail">Nieprawidłowe dane
```



Rysunek 11.9. Funkcja `prepend()` biblioteki jQuery dodaje kod HTML do istniejącego elementu. Metoda ta nie usuwa żadnych danych, dlatego skrypt może wielokrotnie dodawać ten sam komunikat

```

7      ↗uwierzytelniające. Spróbuj ponownie.</p>');
8  }
9 }
10 } // 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 15. 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          $('#content').html('<p>Logowanie zakończyło się
8              ↵powodzeniem!</p>');
9      } else {
10         if ($('#fail').length==0) {
11             $('#formwrapper').prepend('<p id="fail">Nieprawidłowe dane
12                 ↵uwierzytelniające. Spróbuj ponownie.</p>');
13         }
14     } // Koniec funkcji processData.
15     return false;
16 }); // Koniec funkcji submit
17}); // 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 R11.

Uwaga: Jak wspomniano na stronie 372, 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). Format ten jest oparty na języku JavaScript i — podobnie jak XML (patrz ramka na stronie 381) — jest technologią przesyłania danych. W aplikacjach ajaksowych JSON jest lepszy od XML-a — JSON to JavaScript, więc w aplikacjach pisanych w tym języku będzie działał szybciej, a korzystanie z niego będzie łatwiejsze. Kod XML musi zostać przetworzony przez program JavaScript, co zazwyczaj jest wolniejsze i wymaga znacznie bardziej rozbudowanego kodu.

Uwaga: Inny rodzaj formatu JSON, określany jako JSONP, pozwala na pobieranie informacji z innych domen. Pozwala to na przykład zażądać rysunku z witryny Flickr (www.flickr.com) i wyświetlić go na stronie własnej witryny (w uwadze na stronie 367 dowiedziałeś się, że żądania ajaksowe można kierować tylko do stron z danej witryny). Przykład wykorzystania formatu JSONP można znaleźć w następnym rozdziale.

Sposób tworzenia obiektów JSON poznałeś już na stronie 386. Taki obiekt to w swej istocie literał obiektowy języka JavaScript, czyli kolekcja par 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'. Poszczególne pary trzeba rozdzielać przecinkami, nie należy jednak umieszczać takiego znaku po ostatniej parze (Internet Explorer zgłosi wtedy błąd).

Uwaga: Nazwy w parach można też podawać w apostrofach:

```
{
  'firstName': 'Franciszek',
  'lastName': 'Nowak',
  'phone': '503-555-121'
}
```

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 sieciowy reaguje na żądania ajaksowe, może zwrócić łańcuch znaków w formacie JSON. Serwer 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) i przekształca ją za pomocą skomplikowanego kodu JavaScript na gotowy do użytku obiekt JSON.

Uwaga: Język PHP 5.2 ma wbudowaną funkcję — `json_encode()` — ułatwiającą tworzenie obiektów JSON na podstawie standardowych tablic języka PHP. A zatem podczas tworzenia aplikacji ajaksowych można w prosty sposób przekształcić tablice PHP na obiekt w formacie JSON i przesłać go do skryptu działającego w przeglądarce. Więcej o tej funkcji dowiesz się na stronie www.php.net/manual/en/function.json-encode.php.

Funkcja `getJSON()` działa podobnie jak `post()` i `get()`, ale dane przekazane do wywoływanej zwrotnie funkcji to obiekt JSON. Aby użyć metody `getJSON()`, trzeba tylko zrozumieć, jak przetwarzanie takie obiekty w wywoywanych zwrotnie funkcjach. Założymy, 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 wywoływanej zwrotnie funkcji `processContacts`. Podstawowa struktura wywoływanej zwrotnie funkcji wygląda następująco:

```
function processContacts(data) {
}
```

Funkcja `processContacts()` przyjmuje jeden argument, `data`, który zawiera obiekt JSON 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 82) służy do wskazywania właściwości obiektu. Kropkę należy umieścić między nazwą danego obiektu a potrzebną właściwością. Używasz 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 literal obiektowy:

```
var bday = {
    person: 'Roman',
    date: '10/27/1980'
};
```

Zmienna `bday` zawiera literal obiektowy, dlatego aby pobrać wartość właściwości `person` tego obiektu, należy użyć składni z kropką:

```
bday.person // 'Roman'
```

Poniższy kod pobiera datę urodzenia:

```
bday.date // '10/27/1980'
```

Tak samo można korzystać z obiektów JSON zwracanych przez serwer sieciowy. Przyjrzyj się poniższej instrukcji z metodą `getJSON()` i wywoływanej zwrotnie funkcji:

```
$.getJSON('contacts.php', 'contact=123', processContacts);
function processContacts(data) {
}
```

Jeśli serwer zwróci dane JSON ze strony 387, obiekt JSON zostanie przypisany do zmiennej `data` (to argument wywoływanej zwrotnie funkcji `processContacts()`), co odpowiada uruchomieniu poniższego kodu:

```
var data = {
    firstName: 'Franciszek',
    lastName: 'Nowak',
    phone: '503-555-121'
};
```

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: 503-555-121
```

Złożone obiekty JSON

Aby tworzyć także bardziej złożone kolekcje informacji, można używać literałów obiektowych jako wartości w obiekcie JSON. Technika ta polega na zagnieżdżaniu literałów obiektowych w innych strukturach tego typu (nie odkładaj jeszcze książek!).

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: '415-555-523'
}
```

Serwer sieciowy może zwrócić łańcuch znaków z pojedynczym obiektem JSON, zawierającym oba opisane zbiory danych:

```
{
    contact1: {
        firstName: 'Franciszek',
        lastName: 'Nowak',
        phone: '503-555-121'
    },
    contact2: {
        firstName: 'Małgorzata',
        lastName: 'Kowal',
        phone: '415-555-523'
    }
}
```

Wywoływana zwrotnie funkcja przyjmuje jeden argument 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: '503-555-121'
    },
    contact2: {
        firstName: 'Małgorzata',
        lastName: 'Kowal',
        phone: '415-555-523'
    }
};
```

W wywoływanej zwrotnie funkcji 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) {
});
```

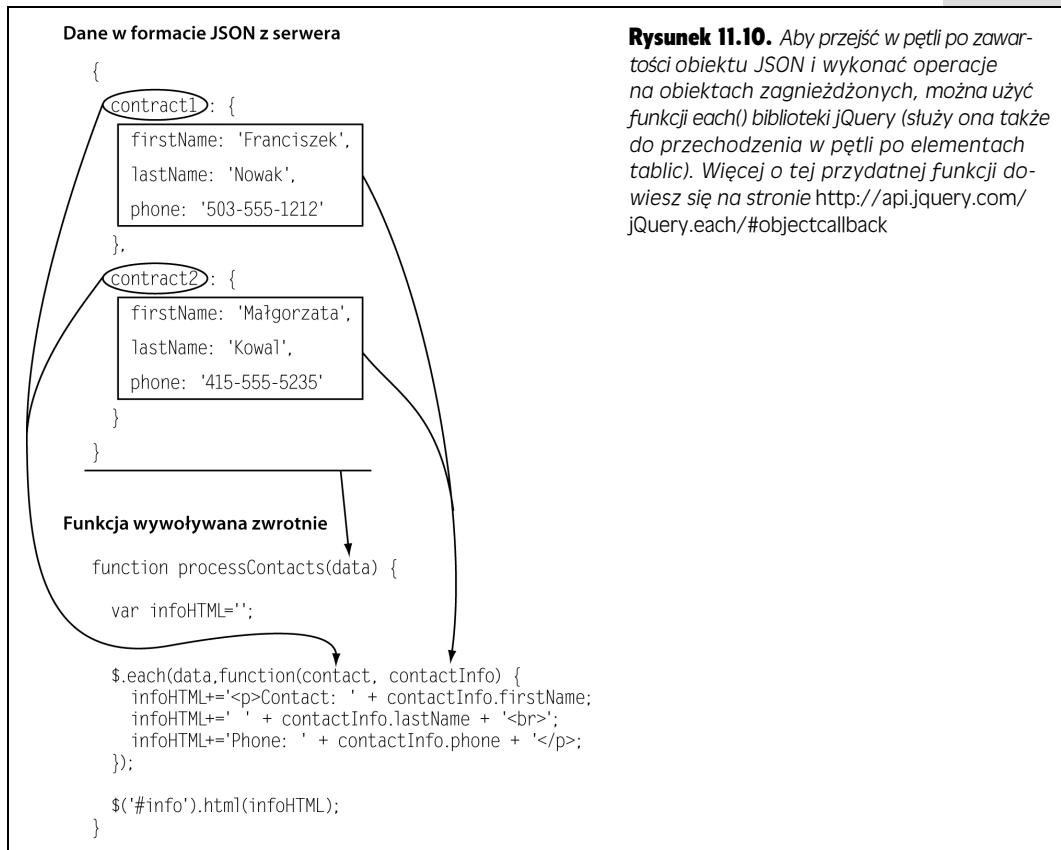
Do funkcji each() należy przekazać obiekt JSON i funkcję anonimową (patrz strona 160). Ta funkcja anonimowa przyjmuje nazwę i wartość każdego elementu z obiektu JSON. Poniższy kod używa przedstawionego wcześniej obiektu JSON:

```
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+='\n<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 w nim wywoływaną zwrotnie funkcję (`processContacts`).
2. Wiersz 2. tworzy wywoływaną zwrotnie funkcję, 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 funkcja `each()`, która przechodzi po obiektach w danych w formacie JSON.

Funkcja each() przyjmuje obiekt JSON jako pierwszy argument (data) i funkcję anonimową jako argument drugi. Ilustruje to rysunek 11.10. 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 contact → Info przechowuje literał obiektowy z danymi kontaktowymi.



5. Wiersze od 8. do 10. pobierają dane kontaktowe jednej osoby.

Pamiętaj, że funkcja 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: 503-555-121</p>
<p>Imię i nazwisko: Małgorzata Kowal<br>
Telefon: 415-555-523</p>
```


Flickr oraz Google Maps

W poprzednim rozdziale poznaleś podstawy technologii AJAX: czym jest, jak działa oraz w jaki sposób biblioteka jQuery może ułatwić proces tworzenia programów, które korzystają z tej technologii. Ponieważ najważniejszym aspektem tej technologii jest dwukierunkowa komunikacja pomiędzy przeglądarką i serwerem, aby zatem można było w pełni wykorzystać możliwości, jakie daje, konieczna jest umiejętność pisania programów działających po stronie serwera. Jednak wcale nie trzeba być światowym autorytetem w dziedzinie pisania programów działających po stronie serwera, by z powodzeniem pracować z AJAX-em. Okazuje się, że można skorzystać z usług udostępnianych przez popularne witryny WWW, takie jak Flickr, Twitter czy też Google Maps, by pobierać z nich obrazki, komunikaty bądź mapy i wyświetlać je na własnych stronach.

Prezentacja JSONP

W poprzednim rozdziale dowiedziałeś się, że 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 XMLHTTP 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. Rozwiążanie to przypomina nieco dołączanie do stron zewnętrznych plików JavaScript z serwerów firmy Google.

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 12.1).



Rysunek 12.1. 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

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 przesyłanych przez wiele różnych osób i dotyczących konkretnego zagadnienia, takiego jak projektowanie stron WWW, widoki i tym podobne).

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 funkcji `$.getJSON()` biblioteki jQuery, by pobierać dane JSONP z innej witryny.

Uwaga: Jeśli będziesz chciał uzyskać większą kontrolę nad zdjęciami pobieranymi z serwisu Flickr, istnieje wiele wtyczek jQuery, które mogą Ci w tym pomóc. Szczególnie użyteczna jest wtyczka jQuery Flickr Photo Gallery (<http://johnpatrickgiven.com/jquery/flickr-gallery/>).

Tworzenie adresu URL

Flickr udostępnia kilka różnych adresów URL pozwalających na pobieranie odmiennych rodzajów zdjęć (patrz strona <http://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 Flickr (na przykład ze swojego własnego konta, jeśli je posiadamy), a adresu http://api.flickr.com/services/feeds/groups_pool.gne — by pobierać zdjęcia z określonej grupy (takiej jak *Web Design*, zawierającej zdjęcia i obrazki mające inspirować projektantów do tworzenia pięknych stron WWW).

Kiedy już określmy, 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 373, łańcuch zapytania jest umieszczany na końcu adresu URL i składa się ze znaku zapytania (?) oraz kilku parametrami; oto przykład: http://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:

`http://api.flickr.com/services/feeds/groups_pool.gne?id=37996591093@N01`

W przypadku kanałów ze zdjęciami pochodząymi z kont konkretnych użytkowników 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:

`http://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:

`http://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 *R12*). W rezultacie w przeglądarce zostaną wyświetlane dane, a konkretnie literał obiektowy zawierający dane z kanału Flickr. To są właśnie te dane, które pobierzesz z serwisu przy użyciu funkcji `$.getJSON()` (opisanej na stronie 387). 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 398, natomiast na stronie 400 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 przesyłać żądań 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 funkcji \$.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 wywoaniu funkcji \$.getJSON() należy podać następujący adres URL:

`http://api.flickr.com/services/feeds/photos_public.gne?ids=25053835@N03&format=json&jsoncallback=?`

Kilka innych opcji publicznych kanałów Flickr

Kiedy korzystamy z kanałów zawierających publicznie dostępne zdjęcia opublikowane na serwisie Flickr, do adresu URL można dodać kilka dodatkowych parametrów, mających wpływ na zawartość pobieranego kanału. Przykładowo założymy, że Ty oraz Twoi znajomi lubicie robić zdjęcia wiewiórkom i publikować je na Flickrze, a teraz chcesz pobrać 20 najnowszych, opublikowanych przez was zdjęć wiewiórek. Możesz to zrobić, filtrując pobierany kanał przy użyciu jednego znacznika lub kilku z nich (ang. *tags*).

Serwis Flickr daje możliwość oznaczenia każdego publikowanego zdjęcia, 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".

Uwaga: Konkretnych znaczników można poszukiwać wyłącznie w kanałach publicznych (http://www.flickr.com/services/feeds/docs/photos_public/). Nie można natomiast szukać znaczników w kanałach grup (takich jak *Web Design*).

Usługi obsługujące kanały 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. Założmy, że Twój identyfikator Flickr ma postać 8623220@N02, a identyfikator Twojego znajomego — 25053835@N03. W takim razie możesz pobrać zdjęcia z obu kanałów i dodatkowo wyszukać w nich opatrzone znacznikiem squirrel (wiewiórka), używając następującego adresu URL:

`http://api.flickr.com/services/feeds/photos_public.gne?ids=25053835@N03,8623220@N02&tags=squirrel&format=json&jsoncallback=?`

- **tagmode.** Zazwyczaj podczas poszukiwania grupy znaczników Flickr zwraca tylko te zdjęcia, w których zostały zastosowane wszystkie podane znaczniki. Przykładowo założymy, że do adresu dodaliśmy parametr ?tags=squirrel,winter,→city. W takim przypadku znajdziemy wyłącznie zimowe zdjęcia wiewiórek

w mieście. Gdybyśmy jednak chcieli wyszukać zdjęcia wiewiórek lub miasta 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:

`http://api.flickr.com/services/feeds/photos_public.gne?ids=25053835@N03,8623220@N02tags=squirrel,winter,city&format=json&jsoncallback=?`

Stosowanie funkcji `$.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 = "http://api.flickr.com/services/feeds/ +  
2      photos_public.gne?ids=25053835@N03&format=json&jsoncallback=?"  
3  $.getJSON(flickrURL, function(data) {  
4      //przetwarzamy pobrane dane w formacie JSON  
5  }); //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 386, 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`' — zwykłym łańcuchem znaków. Jednak taką wartością może być także kolejny obiekt (patrz rysunek 11.10, na stronie 391), 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": "250538350N03",  
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": "250538350N03",  
28  "tags": ""  
29 } // ...  
30 }
```

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 82. Założymy, że użyliśmy kodu przedstawionego w poprzednim punkcie rozdziału (na stronie 398): funkcji anonimowej, przetwarzającej dane JSON przekazane do niej w zmiennej data (patrz 2. wiersz kodu na stronie 398). 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 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 ``, 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 miniaturki pobrańnych 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 43.

1. W edytorze tekstów otwórz plik `flickr.html` umieszczony w katalogu **R12**.

Zaczniesz od utworzenia kilku zmiennych, w których będą zapisane komponenty adresu URL koniecznego do pobrania danych kanału z serwisu Flickr.

2. Kliknij pusty wiersz wewnętrz funkcji `$(document).ready()` i wpisz:

```
var URL = "http://api.flickr.com/services/feeds/photos_public.gne";  
var ID = "25053835@N03";  
var jsonFormat = "&format=json&jsoncallback=?";
```

Każda z widocznych tu zmiennych jest jednym z elementów długiego adresu URL, opisanego na stronie 395. Zapisanie każdego z elementów tego adresu w osobnej zmiennej ułatwia wprowadzanie ewentualnych zmian w kodzie. Gdybyś na przykład chciał pobrać zdjęcia z kanału innego użytkownika, wystarczyłoby zmienić wartość zmiennej `ID` (jeśli dysponujesz własnym kontem w serwisie Flickr, możesz wpisać w niej własny identyfikator — jeżeli go nie znasz, zajrzyj na stronę <http://idgettr.com/>).

Wskazówka: Gdybyś chciał pobrać zdjęcia z kanału grupy, takiej jak *Web Desing*, zmień adres podany w kroku 2. na http://api.flickr.com/services/feeds/groups_pool.gne, a wartość zmiennej `ID` — na identyfikator interesującej Cię grupy.

W następnym kroku połączysz te zmienne, tworząc pełny adres URL.

3. Dodaj kolejny wiersz poniżej tych, które wpisałeś w poprzednim kroku, i wpisz w nim:

```
var ajaxURL = URL + "?id=" + ID + jsonFormat;
```

Instrukcja ta łączy wszystkie zmienne i dodaje do nich także początkowy ciąg znaków łańcucha zapytania — ?id= — tworząc w ten sposób pełny adres URL, taki jak przedstawiony na stronie 396: http://api.flickr.com/services/feeds/photos_public.gne?id=25053835@N03&format=json&jsoncallback=?. Teraz nadszedł czas, by zająć się AJAX-em i użyć funkcji `$.getJSON()`.

4. Dodaj następny wiersz i wpisz w nim:

```
$.getJSON(ajaxURL, function(data) {  
}); // koniec funkcji getJSON
```

To ogólna struktura wywołania funkcji `$.getJSON()`: prześle ona żądanie na adres URL utworzony w krokach 2. i 3. 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.

5. Do kodu z poprzedniego kroku dodaj wiersz wyróżniony pogrubioną czcionką:

```
$.getJSON(ajaxURL, function(data) {  
    $('h1').text(data.title);  
}); // koniec funkcji getJSON
```

Zastosowaliśmy tutaj 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 82), 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*.

Aby wyświetlić na stronie zdjęcia, trzeba w pętli przejrzeć całą zawartość obiektu zapisanego we właściwości `items` (patrz strona 399).

6. Dodaj kolejne dwa wiersze wyróżnione pogrubioną czcionką, umieszczając je poniżej kodu wpisanego w poprzednim kroku:

```
$.getJSON(ajaxURL, function(data) {  
    $('h1').text(data.title);  
    $.each(data.items, function(i,photo) {  
    }); // koniec funkcji each  
}); // koniec funkcji getJSON
```

Funkcja `.each()` została opisana na stronie 160 — służy ona do przeglądnięcia całej zawartości kolekcji zwróconej przez selektor jQuery. Funkcja `$.each()` jest podobna, choć działa nieco inaczej. Jest to ogólna pętla pozwalająca przejrzeć całą zawartość tablicy (patrz strona 72) lub grupy obiektów. W jej wywołaniu przekazywana jest tablica lub literal obiektowy oraz funkcja anonimowa. Funkcja `$.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 75), 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 399).

W kanałach Flickr element `data.items` reprezentuje tablicę obiektów z danymi o poszczególnych zdjęciach (patrz strona 399), a zatem funkcja `$.each()` będzie po kolej 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 miniatuerek 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/">

</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 miniaturka zdjęcia.

7. Wewnątrz funkcji `$.each()` dodaj poniższy kod wyróżniony pogrubioną czcionką:

```
$ .each(data.items, function(i, photo) {
    var photoHTML = '<span class="image">';
    photoHTML += '<a href="' + photo.link + '">';
    photoHTML += '</a>';
}); // koniec funkcji each
```

Powyższy kod rozpoczyna się od deklaracji zmiennej `photoHTML`, w której zapisywany jest łańcuch znaków zawierający otwierający znacznik ``. Każda kolejna instrukcja dodaje do tego łańcucha kolejny fragment (aby sobie przypomnieć znaczenie operatora `+=`, zajrzyj na stronę 68). Kluczowymi elementami tych instrukcji są odwołania `photo.link` oraz `photo.media.m`. Jeśli zajrzesz do danych zapisanych w formacie JSON i przedstawionych na stronie 399, 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.

8. Dodaj kolejny fragment kodu wyróżniony pogrubioną czcionką:

```
$ .each(data.items, function(i, photo) {  
    var photoHTML = '<span class="image">';  
    photoHTML += '<a href="' + photo.link + '">';  
    photoHTML += '</a>';  
    $('#photos').append(photoHTML);  
}); // koniec funkcji each
```

Wywołanie `$('#photos')` pobiera istniejący na stronie znacznik `<div>`, natomiast funkcja `append()` (opisana dokładniej na stronie 151) dodaje przekazany kod HTML na samym końcu tego znacznika. Innymi słowy, podczas każdej iteracji pętli na końcu znacznika `<div>` zostanie dodany kolejny fragment kodu HTML.

9. 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 48, 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 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 zamianianie fragmentów łańcuchów znaków.

10. 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 = "http://api.flickr.com/services/feeds/photos_public.gne";  
    var ID = "25053835@N03";  
    var jsonFormat = "&format=json&jsoncallback=?";  
    var ajaxURL = URL + "?id=" + ID + jsonFormat;  
    $.getJSON(ajaxURL, function(data) {  
        $('h1').text(data.title);  
        $.each(data.items, function(i, photo) {  
            var photoHTML = '<span class="image">';  
            photoHTML += '<a href="' + photo.link + '">';  
            photoHTML += '</a>';  
            $('#photos').append(photoHTML);  
        });  
    });  
});
```

```
photoHTML += '<img src=' + photo.media.m.replace('_m','_s')  
+ '></a>';  
$( '#photos' ).append(photoHTML);  
}); // koniec funkcji each  
}); // koniec funkcji getJSON  
}); // koniec funkcji ready
```

Metoda `replace()` języka JavaScript (opisana na stronie 463) 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`').

11. Zapisz stronę i wyświetl ją w przeglądarce.

Teraz powinieneś zobaczyć na stronie 20 równo rozmieszczonych, kwadratowych miniaturek (patrz rysunek 12.1). Możesz kliknąć jedną z nich, by wyświetlić stronę danego zdjęcia. Działającą wersję tego przykładu można znaleźć w pliku `complete_flickr.html`, w katalogu R12.

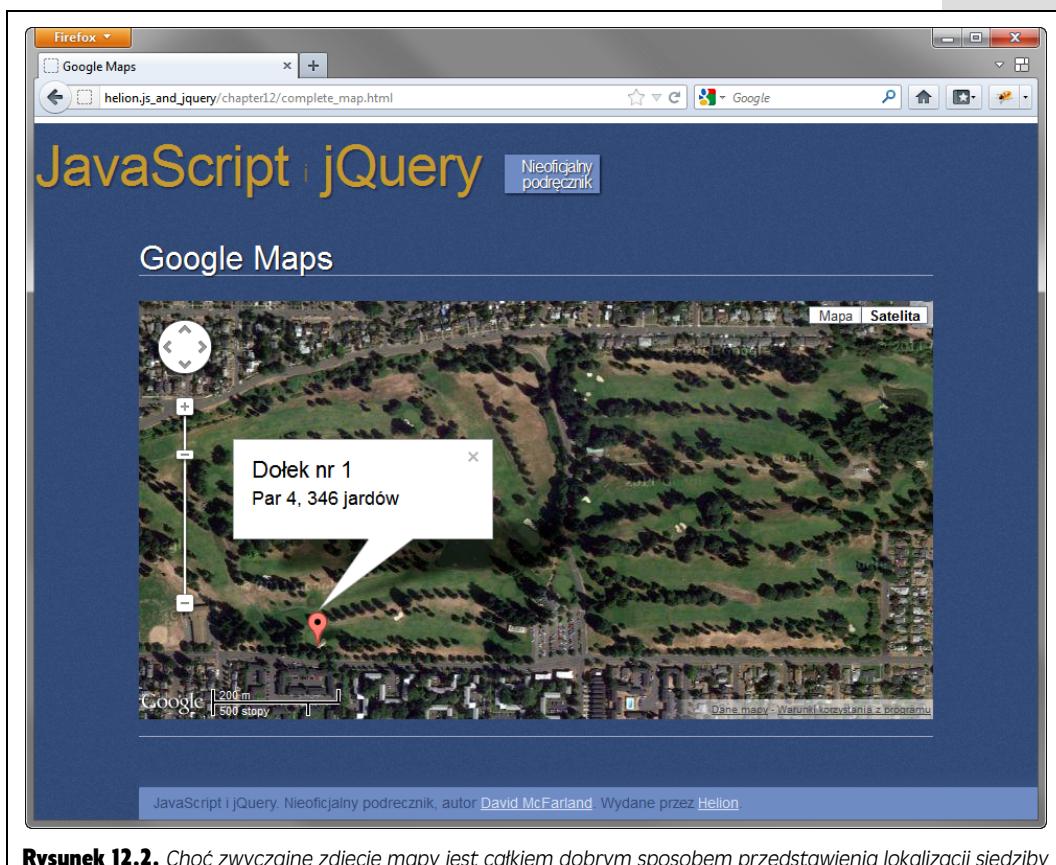
Uwaga: Kanały Flickr udostępniają co najwyżej 20 zdjęć. Z każdego kanału można pobrać więcej niż 20 zdjęć. A co zrobić, jeśli będziemy chcieli pobrać z kanału 10 zdjęć? Przykład rozwiązania takiego problemu możesz znaleźć w pliku `complete_flickr_limit_phostos.html` dostępnym w katalogu R12.

Wyświetlanie na własnej stronie map Google Maps

Google Maps (<http://maps.google.pl>) jest jedną z najbardziej znanych wizytówek rewolucji, jaką przyniosło szersze stosowanie języka JavaScript. Możliwości powiększania i pomniejszania mapy, przesuwania się wzduż ulic i błyskawicznego pobierania informacji o sposobie dojazdu w wybrane miejsce sprawiają, że witryna ta jest niesamowicie użyteczna. A dzięki inteligentnemu zastosowaniu technologii AJAX sposób jej reagowania na poczynania użytkownika sprawia, że w ogromnym stopniu przypomina ona tradycyjne programy komputerowe.

Jednak usługa Google Maps udostępnia projektantom stron WWW jeszcze więcej możliwości — pozwala na wyświetlanie mapy na własnych stronach. Jeśli prowadzisz usługi budowlane (lub tworzysz witryny WWW dla firm), możliwość prezentacji zrozumiałych i łatwych w użyciu map może przyciągnąć wielu klientów. Przy użyciu Google Maps i wtyczki jQuery w prosty sposób umieścisz na tworzonych stronach interaktywne mapy (patrz rysunek 12.2).

Opracowana przez Jevgenijsa Shtraussa wtyczka `GoMap` (<http://www.pittss.lv/jquery/gomap/>) pozwala na dodawanie map Google Maps na dowolnej stronie, pobieranie instrukcji dojazdu z jednego miejsca mapy do drugiego, dodawanie znaczników we wskazanych miejscach mapy oraz na wykonywanie wielu innych operacji. Poniżej opisane zostały podstawowe czynności, jakie trzeba wykonać, by skorzystać z tej wtyczki.



Rysunek 12.2. Choć zwyczajne zdjęcie mapy jest całkiem dobrym sposobem przedstawienia lokalizacji siedziby naszej firmy (bądź firmy naszego klienta), jednak znacznie lepszym rozwiązaniem będzie zastosowanie mapy interaktywnej, takiej jak dostępne na witrynie Google Maps. Zapewniają one użytkownikom możliwość powiększania i zmniejszania oraz przesuwania. Za pomocą wtyczki jQuery o nazwie GoMap wyświetlanie map Google Maps na własnej stronie jest bardzo proste

1. Dołącz zewnętrzny plik JavaScript Google Maps.

Aby skorzystać z usługi obsługującej mapy, konieczne jest dodanie do strony pliku JavaScript pobieranego z serwera firmy Google. Znacznik `<script>`, jakiego należy w tym celu użyć, ma następującą postać:

```
<script src="http://maps.google.com/maps/api/js?sensor=false"></script>
```

Znacznik wczytuje plik JavaScript, niezbędny do dostępu oraz wykorzystania usługi wyświetlającej i obsługującej mapy. Wtyczka gmap3 bazuje na kodzie zawartym w tym pliku, by udostępnić bardziej wygodny sposób tworzenia map.

Uwaga: Wtyczka GoMap korzysta z trzeciej wersji Google Maps API (interfejsu programowania aplikacji). Wcześniejsze wersje API wymagały zarejestrowania się na witrynie Google Maps i pobrania unikalnego klucza, który pozwalał wyświetlać mapy na własnej witrynie. Na szczęście, już nie trzeba rejestrować się, by korzystać z map na swojej witrynie.

PORADNIA DLA ZAAWANSOWANYCH

Większe możliwości dzięki wykorzystaniu jQuery i technologii AJAX

Istnieje bardzo wiele wtyczek biblioteki jQuery, które pozwalają znacząco przyspieszyć tworzenie aplikacji wykorzystujących technologię AJAX. W niektórych przypadkach konieczne jest także pisanie programów działających po stronie serwera — wtyczka obsługuje jedynie komunikację po stronie przeglądarki. Mamy do dyspozycji kilka programów działających po stronie serwera. Oto niektóre z nich.

- ◆ **Wtyczka Form.** Wtyczka jQuery Form zapewnia możliwość prostego przesyłania danych z formularzy przy użyciu technologii AJAX. Wykraca ona znacznie poza podstawowe możliwości opisane w poprzednim rozdziale i pozwala na przesyłanie plików. Jest także wyposażona w mechanizm weryfikacji poprawności danych. Więcej informacji na jej temat można znaleźć na stronie <http://jquery.malsup.com/form>.
- ◆ **Automatyczne uzupełnianie.** Projekt jQuery UI udostępnia wtyczkę obsługującą mechanizm automatycznego uzupełniania (ang. *autocomplete*) — <http://jqueryui.com/demos/autocomplete>. Pozwala ona dodawać do swoich stron świetne możliwości, w które są wyposażone takie witryny jak Google czy Amazon. Wystarczy zacząć wpisywać słowo w polu tekstowym, by poniżej została wyświetlona lista sugestii. W ten sposób możemy ograniczyć ilość tekstu, jaki użytkownicy muszą wpisać, i jednocześnie dostarczamy im przydatne sugestie. Wykorzystując dodatkowo program działający po stronie serwera i komunikację przy użyciu technologii AJAX, można opracować formularze wyszukiwawcze, które niemalże będą czytać użytkownikowi w myślach.

◆ **Przesyłanie plików z użyciem AJAX-a.** Jeśli chcesz wyposażyć swoją witrynę w możliwość przesyłania plików na serwer, postaraj się, by przy zastosowaniu technologii AJAX była ona bardzo wygodna i użyteczna. Wtyczka *Uploadify* (<http://www.uploadify.com/>) pozwala uprościć cały proces przesyłania plików na serwer.

- ◆ **Taconite.** Przy użyciu technologii AJAX można pobierać informacje z serwera i aktualizować zawartość strony. Jednak może się zdarzyć, że będziemy chcieli zaktualizować kilka różnych obszarów strony. Jeśli przykładowo użytkownik zaloguje się, korzystając z formularza obsługiwanego przy użyciu AJAX-a, pewnie zechcemy zaktualizować jego status wyświetlony w jednym miejscu strony, listę stron odwiedzonych podczas poprzedniej wizyty na witrynie wyświetlzoną w innym miejscu oraz koszyk z zakupami widoczny jeszcze gdzie indziej. Wtyczka Taconite pozwala pobierać z serwera zapisane w prostym formacie XML instrukcje określające, które obszary strony chcemy zaktualizować i jakich informacji należy przy tym użyć. Więcej informacji na temat tej wtyczki można znaleźć na stronie <http://jquery.malsup.com/taconite/>.
- ◆ **Twitter.** Jeśli poszukujesz prostego sposobu umieszczenia na swojej stronie strumienia komunikatów z Twittera, sprawdź wtyczkę *Tweet!* (<http://tweet.seaofclouds.com/>). Pozwala ona umieścić na stronie własny strumień komunikatów, poszukiwać komunikatów w strumieniach innych użytkowników, a nawet poszukiwać na Twitterze dowolnych słów kluczowych.

2. Dołącz dwa pliki jQuery.

Oczywiście, konieczne jest dołączenie do strony zarówno pliku biblioteki jQuery, jak i wtyczki *GoMap*. Plik wtyczki można pobrać ze strony <http://www.pittss.lv/jquery/gomap>. Zawiera on cały kod, który sprawia, że wyświetlanie map Google Maps na własnej stronie WWW staje się proste. A zatem, oprócz znacznika `<script>` dodanego w poprzednim kroku, musisz dodać do kodu strony kolejne dwa znaczniki:

```
<script src="js/jquery-1.6.3.min.js"></script>
<script src="js/jquery.gomap-1.3.2.min.js"></script>
```

Uwaga: Kopię pliku JavaScript wtyczki GoMap można znaleźć w katalogu _js w przykładach dołączonych do książki (na stronie 43 znajdziesz informacje, jak można je pobrać).

3. Dodaj do kodu strony znacznik `<div>` z określonym identyfikatorem.

Wtyczka wyświetli mapę w tym pustym znaczniku, zatem umieść go w wybranym miejscu strony, tam gdzie chcesz, by była umieszczona mapa. Dodatkowo konieczne będzie zapewnienie możliwości identyfikacji tego elementu, co oznacza, że trzeba podać w nim identyfikator. Kod HTML tego elementu może wyglądać następująco:

```
<div id="map"></div>
```

Nie można zapomnieć o dodaniu do arkusza stylów używanego na stronie reguły określającej wysokość i szerokość tego elementu. Oto przykład:

```
#map {  
    width: 760px;  
    height: 400px;  
}
```

4. Wywołaj funkcję `goMap()`.

Ostatnim etapem jest dodanie do kodu strony znacznika `<script>` zawierającego wywołanie funkcji `$(document).ready()` i umieszczenie wewnętrz niej wywołania funkcji `goMap()`. Aby wywołać funkcję `goMap()`, musisz pobrać znacznik `<div>`, używając selektora jQuery w postaci `$('#map')`, a następnie dodać do niego wywołanie funkcji `goMap()`.

```
<script>  
$(document).ready(function() {  
    $('#map').goMap();  
});  
</script>
```

Jednak takie proste wywołanie funkcji `goMap()` spowoduje wyświetlenie mapy wyśrodkowanej na miejscu położonym gdzieś na Łotwie. W większości przypadków (z wyjątkiem osób mieszkających na Łotwie) będziemy chcieli wyśrodkować mapę na jakimś innym, określonym miejscu (takim jak lokalizacja firmy naszego klienta) i powiększyć ją w stopniu pozwalającym na wyświetlenie bardziej szczegółowych informacji, na przykład nazw ulic. W kolejnym punkcie dowiesz się, jak można to zrobić.

Określanie lokalizacji na mapie

Wyświetlana na stronie mapa Google Maps ma swój punkt centralny, zdefiniowany za pomocą współrzędnych określających szerokość i długość geograficzną danego miejsca. Jeśli chcemy wyśrodkować mapę na konkretnym miejscu — takim jak lokalizacja firmy klienta lub miejsce, w którym organizujemy imprezę urodzinową — konieczne będzie określenie jego współrzędnych geograficznych. Można to zrobić całkiem łatwo.

1. Wejdź na stronę <http://itouchmap.com>, wpisz adres w polu **Address** i kliknij przycisk **Go**.

Ta użyteczna witryna podaje długość i szerokość geograficzną każdego podanego adresu. Wyświetlone informacje zapisz lub skopiuj — będziesz ich potrzebował w następnym kroku.

Wskazówka: Współrzędne geograficzne można także określić, korzystając bezpośrednio z mapy Google Maps. Wejdź na stronę <http://maps.google.pl>, odszukaj potrzebne miejsce, a następnie w pasku adresu przeglądarki wpisz javascript:void(prompt(' ',gApplication.getMap().getCenter()));. Na ekranie zostanie wyświetlone okienko dialogowe z szerokością i długością geograficzną miejsca znajdującego się na środku wyświetlonej mapy.

2. Zaktualizuj wywołanie funkcji goMap(), dodając do niego poniższy, wyraźniony pogrubieniem kod:

```
1  <script>
2  $(document).ready(function() {
3      $('#map').goMap({
4          latitude : 45.53940,
5          longitude : -122.59025
6      }); //koniec funkcji goMap
7  }); //koniec funkcji ready
8 </script>
```

W tym przypadku przekazujemy do funkcji literał obiektowy — rozpoczynający się nawiasem { umieszczonym na końcu 3. wiersza kodu i kończący nawiasem } na początku wiersza 6. — zawierający opcje wtyczki. Pierwsza właściwość tego obiektu — latitude — określa szerokość geograficzną środka mapy, natomiast druga — longitude — ustala długość geograficzną tego miejsca. Dwie liczby podane w wierszach 4. i 5. powinieneś zastąpić współrzędnymi geograficznymi miejsca, na którym chcesz wyśrodkować swoją mapę.

Na szczęście, wtyczka GoMap jest na tyle elastyczna, że pozwala zamiast współrzędnych geograficznych podać adres miejsca, na którym chcemy wyśrodkować mapę. W tym celu wystarczy zastąpić właściwości latitude i longitude właściwością address zawierającąłańcuch znaków z adresem wybranego miejsca. Oto przykład:

```
$('#map').goMap({
    address : 'Wiejska 6, Warszawa, Polska'
}); //koniec funkcji goMap
```

Jeśli interesuje nas bardziej ogólne określenie obszaru, na którym będzie wyśrodkowana mapa, na przykład chcemy wyświetlić konkretne miasto, także mamy taką możliwość:

```
$('#map').goMap({
    address : 'Warszawa, Polska'
}); //koniec funkcji goMap
```

Możemy użyć dowolnego łańcucha znaków, który trzeba wpisać w polu wyszukiwania na witrynie Google Maps (<http://maps.google.pl>) — adresu, a nawet nazwy jakiegoś znanego miejsca lub obiektu, na przykład „Kennedy Space Center” lub „Eiffel Tower”. Jednak usługa Google Maps nie zawsze odnajdzie podany adres, a poza tym adresy nie są najlepsze, gdy chcemy wskazać określone miejsce na dużym obszarze — na przykład ulubione miejsce w parku. W takich przypadkach trzeba skorzystać ze współrzędnych geograficznych.

Uwaga: Jeszcze jedną wtyczką, o której warto wspomnieć, jest *GMAP3* (<http://gmap3.net/>). Udoszcznia ona znacznie więcej możliwości niż wtyczka *GoMap*, jednak jest także bardziej złożona, a jej plik większy. Jedną z interesujących cech wtyczki *GMAP3* jest możliwość generowania informacji o sposobie dojazdu bezpośrednio na mapie.

Inne opcje wtyczki GoMap

Podczas tworzenia mapy można, oprócz współrzędnych geograficznych i adresu, podać także wiele innych opcji. Każdą z nich należy dodać do literatu obiektowego przekazywanego w wywołaniu funkcji *goMap()*. Aby na przykład wyśrodkować mapę na współrzędnych 45/-122 i ustawić takie powiększenie, by były widoczne szczegóły, wywołanie funkcji *goMap()* powinno wyglądać następująco:

```
$( '#map' ).goMap(  
{  
    latitude : 45,  
    longitude : -122,  
    zoom : 15  
}); //koniec funkcji goMap
```

Opisywane tu opcje są przekazywane jako elementy literatu obiektowego.

Oto kilka opcji, które można wykorzystać.

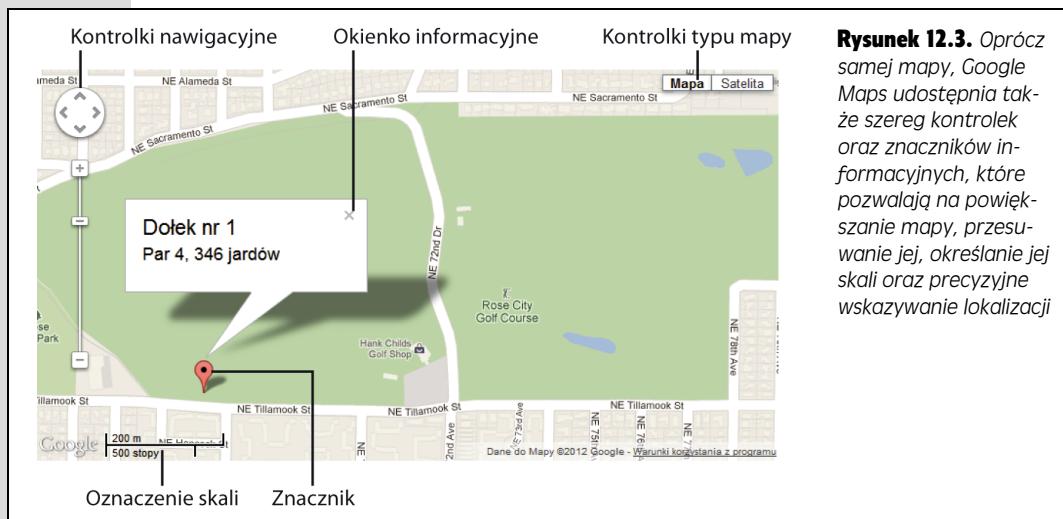
- **Kontrola skali mapy.** Czasami będziesz chciał powiększyć mapę w takim stopniu, by były widoczne najdrobniejsze szczegóły, takie jak nazwy ulic. W innych przypadkach możesz preferować wyświetlenie na mapie całego miasta lub nawet województwa. Stopień powiększenia mapy można kontrolować poprzez podanie odpowiedniej liczby w opcji *zoom*. Wartość 0 powoduje, że zostanie zastosowane najmniejsze powiększenie (czyli będzie widoczna mapa całego świata), każda kolejna, większa wartość sprawi, że powiększenie będzie większe. Ogólnie rzecz biorąc, powiększenie o wartości 15 jest dobrym ustawieniem, jeśli chcemy wyświetlać takie szczegóły jak nazwy ulic, a powiększenie 13 — jeśli zależy nam na bardziej ogólnym widoku. Największe możliwe powiększenie zależy do tego, jak szczegółową mapą danego regionu dysponuje Google, zazwyczaj są to wartości z zakresu od 17 do 23. Ustaw zatem powiększenie w następujący sposób:

```
zoom:15
```

- **Określenie typu prezentowanej mapy.** Zazwyczaj wtyczka GoMap wyświetla tak zwaną mapę **hybrydową** — zdjęcia satelitarne z naniesionymi ulicami oraz nazwami najważniejszych miejsc. Jednak możemy zdecydować się na wyświetlenie zwyczajnej mapy drogowej, samych zdjęć satelitarnych bądź też mapy terenowej pokazującej lasy, poziomice i tak dalej. Rodzaj wyświetlonej mapy można zmieniać, przypisując jedną z kilku predefiniowanych wartości opcji *maptype*. Dostępne wartości to: 'HYBIRD', 'ROADMAP', 'SATELLITE' oraz 'TERRAIN'. Oto przykład:

```
maptype: 'TERRAIN'
```

- **Dodanie skali mapy.** Drukowane mapy zawierają zazwyczaj jakąś skalę, taką jak 1 — 25000 (1cm odpowiada 250 metrom). Taką skalę można także wyświetlać na mapach Google Maps (patrz rysunek 12.3). Aby dodać niewielki



Rysunek 12.3. Oprócz samej mapy, Google Maps udostępnia także szereg kontrollek oraz znaczników informacyjnych, które pozwalają na powiększanie mapy, przesuwanie jej, określanie jej skali oraz precyzyjne wskazywanie lokalizacji

znacznik skali, wyświetlany w lewym, dolnym rogu mapy, należy przypisać opcji `scaleControl` wartość `true`:

```
scaleControl : true
```

Jeśli nie chcemy, by znacznik skali był widoczny, nie musimy nic robić — w mapach tworzonych przy użyciu wtyczki *GoMap* jest on domyślnie niewidoczny.

- **Usunięcie elementów nawigacyjnych.** Jeśli nie chcemy, by na mapie były wyświetlane kontrolki do określania powiększenia oraz przesuwania widocznego obszaru mapy, możemy to ustalić, przypisując opcji `navigationControls` wartość `false`:

```
navigationControl : false
```

Choć powyższa opcja pozwala ukryć elementy sterujące mapą, jednak użytkownik wciąż może powiększać ją za pomocą kółka myszy lub podwójnego kliknięcia. Jeśli zatem naprawdę zależy nam na tym, by uniemożliwić użytkownikom nawigowanie po mapie, konieczne jest przekazanie w funkcji `goMap()` dwóch dodatkowych opcji:

```
navigationControl: false,  
scrollwheel: false,  
disableDoubleClickZoom: true,
```

Wtyczka *GoMap* nie pozwala na zablokowanie możliwości zmiany prezentowanego fragmentu mapy przy użyciu przeciągania. A zatem użytkownicy i tak będą mogli przesuwać mapę, choć odbierzemy im możliwość jej powiększania i pomniejszania.

- **Kontrolki typu mapy.** Mapy Google Maps mogą wyglądać jak zwyczajne mapy drogowe, zdjęcia satelitarne, połączenie obu tych map bądź też jak mapy terenowe, prezentujące takie szczegóły jak poziomice. Zazwyczaj mapa udostępnia kontrolki pozwalające na przełączanie aktualnie używanego typu mapy. Można je jednak ukryć, dodając do literatu obiektywego opcję `mapTypeControl` z wartością `false`:

```
mapTypeControl : false
```

Jeśli natomiast lubimy te kontrolki, możemy ustalić zarówno ich wygląd, jak i położenie. Do tego celu używana jest opcja `mapTypeControlOptions` zawierająca kolejny literał obiektowy z opcjami określającymi styl i położenie kontrolek. Dostępne wartości związane z określaniem położenia kontrolek typu mapy to: `'TOP'`, `'TOP_LEFT'`, `'TOP_RIGHT'`, `'BOTTOM'`, `'BOTTOM_LEFT'`, `'BOTTOM_RIGHT'`, `'LEFT'` i `'RIGHT'`. A to kolejna grupa wartości, które pozwalają określać styl tych kontrolek: `'DEFAULT'`, `'DROPDOWN_MENU'` oraz `'HORIZONTAL_BAR'`. Założymy na przykład, że chcielibyśmy wyświetlić te kontrolki w formie rozwijanego menu i umieścić w prawym, dolnym rogu mapy. W tym celu do literaku obiektowego przekazywanego w wywołaniu funkcji `goMap()` należałoby dodać następującą opcję:

```
mapTypeControlOptions: {  
    position: 'BOTTOM_RIGHT',  
    style: 'DROPDOWN_MENU'  
}
```

Dodawanie znaczników

Aby zaznaczyć konkretne miejsce na mapie, można umieścić na niej specjalny czerwony znacznik, taki jak widoczny na rysunku 12.3. Takie znaczniki są doskonałym sposobem wskazania lokalizacji firmy naszego klienta lub innych interesujących miejsc na mapie. Aby podać szczegółowe informacje dotyczące konkretnego znacznika, można do niego dodać wyskakujące okienko zawierające kod HTML (patrz rysunek 12.3). Dzięki zastosowaniu wtyczki *GoMap* dodawanie takich znaczników jest bardzo proste.

Wtyczka *GoMap* udostępnia dwa sposoby dodawania znaczników. Można utworzyć znacznik oraz mapę i jednocześnie wyświetlić je na stronie. Można także wyświetlić mapę, a następnie dodać do niej jeden znacznik lub większą ich liczbę. To drugie rozwiązanie pozwala na dynamiczną kontrolę dodawanych znaczników — na przykład dodawanie znaczników, gdy użytkownik kliknie przycisk umieszczony na stronie.

Pierwsze z przedstawionych rozwiązań wiąże się z wywołaniem funkcji `goMap()` i przekazaniem do niej literaku obiektowego, w którym została podana opcja `markers`. Oto przykład takiego wywołania:

```
$('#map').goMap({  
    markers : [  
        {  
            latitude : 45.53940,  
            longitude : -122.59025,  
            title : 'Marker 1'  
        }  
    ]  
}); //koniec funkcji goMap
```

Właściwość `markers` powoduje powstanie na mapie jednego lub kilku znaczników. Jej wartością jest tablica (patrz strona 72), taka jak lista zakupów, zawierająca jeden lub kilka elementów. Aby utworzyć zmienną o nazwie `arrayItems` zawierającą tablicę z trzema łańcuchami znaków, należałoby użyć następującej instrukcji:

```
var arrayItems = ['element1', 'element2', 'element3'];
```

Także znaczniki są określane przy użyciu podobnej tablicy, z tym że zamiast łańcuchów znaków musi ona zawierać literały obiektowe (patrz strona 157). Literał definiujący jeden znacznika ma następującą, przykładową postać:

```
{  
    latitude : 45.53940,  
    longitude : -122.59025,  
    title : 'Znacznik nr 1.'  
}
```

Powyższy kod reprezentuje jeden znacznik i ma najprostszą możliwą postać — zawiera jedynie właściwości określające szerokość i długość geograficzną oraz tytuł znacznika (będzie on wyświetlany, gdy użytkownik wskaże znacznik myszą).

Dodając do tablicy kolejne literały łańcuchowe, możemy umieścić na mapie większą liczbę znaczników. Oto przykład dodania trzech znaczników:

```
$('#map').goMap({  
    markers : [  
        {  
            latitude : 45.53940,  
            longitude : -122.59025,  
            title : 'Znacznik nr 1'  
        },  
        {  
            latitude : 45.53821,  
            longitude : -122.59796,  
            title : 'Znacznik nr 2'  
        },  
        {  
            latitude : 45.53936,  
            longitude : -122.58159,  
            title : 'Znacznik nr 3'  
        }  
    ]  
}); // koniec funkcji goMap
```

Tworząc mapę w powyższy sposób, wciąż możemy korzystać z opcji opisanych w poprzednim punkcie rozdziału, poświęconym funkcji `goMap()`. Aby przykładowo wyświetlić mapę drogową z jednym znacznikiem, o stopniu powiększenia 15, z wyświetlzoną skalą, należałoby użyć następującego wywołania (zwróć uwagę, że dodatkowe opcje są zapisane poza tablicą `markers`):

```
$('#map').goMap({  
    markers : [  
        {  
            latitude : 45.53940,  
            longitude : -122.59025,  
            title : 'Znacznik nr 1'  
        }  
    ],  
    zoom : 15,  
    maptype : 'ROADMAP',  
    scaleControl : true  
}); // koniec funkcji goMap
```

Można także podać adres, by określić miejsce znacznika bez korzystania ze współrzędnych geograficznych; oto przykład:

```
$('#map').goMap({  
    markers : [  
        {  
            address : 'Wiej ska 6, Warszawa, Polska',  
        }  
    ]  
}); // koniec funkcji goMap
```

```

        title : 'Sejm RP'
    },
],
zoom : 15,
}); //koniec funkcji goMap

```

Znaczniki można także dodawać już po wyświetleniu mapy. Technika ta jest przydatna, gdy nie chcemy, by znacznik był umieszczony w środku mapy; na przykład mapa mógłaby prezentować park, a znaczniki — różne interesujące miejsca w tym parku. W takim przypadku chcielibyśmy zapewne, by to park został wyśrodkowany w obszarze mapy, a nie jeden z prezentowanych znaczników (a właśnie tak się dzieje w przypadku jednocięsnego tworzenia mapy i znaczników w sposób opisany na stronie 409).

Czasami będziemy także chcieli mieć możliwość dynamicznego dodawania znaczników — na przykład w odpowiedzi na kliknięcie jakiegoś elementu strony. Kontynuując przykład mapy parku, moglibyśmy obok niej umieścić przycisk *Pokaż wszystkie ławki w parku*. Jego kliknięcie spowodowałoby wyświetlenie na mapie znaczników pokazujących lokalizację wszystkich ławek w parku.

Przy takim sposobie dodawania znaczników zaczynamy od utworzenia samej mapy, a później dodajemy znaczniki, wywołując w tym celu specjalną funkcję wtyczki *GoMap — createMarker()*. Aby na przykład wyświetlić mapę, a następnie dodać do niej jeden znacznik, należy użyć następującego fragmentu kodu:

```

$('#map').goMap(
{
    latitude : 45,
    longitude : -122,
    zoom : 15
}); //koniec funkcji goMap
$.goMap.createMarker({
    latitude : 45.53940,
    longitude : -122.59025,
    title : 'Znacznik nr 1'
});

```

Warto zwrócić uwagę, że podczas dodawania znaczników do mapy nie jest używany żaden selektor jQuery. Innymi słowy, wywołanie ma postać `$.goMap.createMarker()`, a nie `$('#map').goMap.createMarker()`. W jednym wywołaniu funkcji `createMarker()` można dodać tylko jeden znacznik, aby zatem dodać ich kilka, trzeba będzie kilkakrotnie wywołać tę funkcję. Dobrym rozwiązaniem jest zdefiniowanie poszczególnych znaczników w tablicy i dodanie ich przy użyciu funkcji jQuery `$.each()`. Oto przykład:

```

var markers = [
{
    latitude : 45.53940,
    longitude : -122.59025,
    title : 'Znacznik nr 1'
},
{
    latitude : 45.53821,
    longitude : -122.59796,
    title : 'Znacznik nr 2'
},
{
    latitude : 45.53936,
    longitude : -122.58159,
}
];

```

```
        title : 'Znacznik nr 3'  
    }  
];  
$('#map').goMap(  
{  
    latitude : 45,  
    longitude : -122,  
    zoom : 15  
}); // koniec funkcji goMap  
$.each(markers,function(i,marker) {  
    $.goMap.createMarker(marker);  
}); // koniec funkcji each
```

Może się także zdarzyć, że będziemy chcieli usunąć jakiś znacznik wyświetlony na mapie (na przykład po kliknięciu przycisku *Ukryj znaczniki*). Wtyczka *GoMap* udostępnia w tym celu funkcję `clearMarkers()`. Jest ona wywoływaną podobnie jak funkcja `createMarkers()`, czyli bez stosowania selektora jQuery:

```
$.goMap.clearMarkers();
```

Załóżmy, że na naszej stronie jest umieszczony przycisk o identyfikatorze `removeMarkers`. Poniżej został przedstawiony kod pokazujący, jak można skojarzyć z nim procedurę obsługi, która w odpowiedzi na kliknięcie przycisku usunie wszystkie znaczniki wyświetcone na mapie:

```
$('#removeMarkers').click(function() {  
    $.goMap.clearMarkers();  
}); // koniec funkcji click
```

Jeśli chcemy mieć możliwość usunięcia konkretnego znacznika, najpierw musimy określić jego identyfikator, a później usunąć go, wywołując funkcję `removeMarker()`. Przedstawiony poniżej skrypt tworzy nową mapę (wiersze od 1. do 13.), a następnie dodaje do elementu strony o identyfikatorze `remove` procedurę obsługi zdarzeń `click`, która usuwa wybrany znacznik.

```
1  $('#map').goMap({  
2      markers : [  
3          {  
4              latitude : 45.53940,  
5              longitude : -122.59025,  
6              title : 'Znacznik nr 1',  
7              id : 'marker1'  
8          }  
9      ],  
10     zoom : 15,  
11     maptype : 'ROADMAP',  
12     scaleControl : true  
13 }); // koniec funkcji goMap  
14 $('#remove').click(function() {  
15     $.goMap.removeMarker('marker1');  
16 });
```

Najistotniejszą zmianą związaną z tworzeniem znacznika wprowadzoną w tym przykładzie jest właściwość zapisana w wierszu 7. — `id : 'marker1'`. Określa ona identyfikator znacznika. Kiedy znacznik będzie miał określony identyfikator, można się do niego odwołać i usunąć przy użyciu funkcji `removeMarker()` (wiersz 15.). Wtyczka *GoMap* udostępnia także funkcję `showHideMarker()`, która pozwala na naprzemienne ukrywanie i wyświetlanie znacznika. Przykładowo wiersze od 14. do 16. poprzedniego przykładu można by zmienić w następujący sposób, by każde kliknięcie elementu strony powodowało na przemian ukrycie i ponowne wyświetlenie znacznika:

```
$('#remove').click(function() {
    $.goMap.showHideMarker('marker1');
});
```

Dodawanie okienek informacyjnych do znaczników

Do każdego znacznika można także dodać dymek (nazywany okienkiem informacyjnym). W tym celu do literala obiektowego każdego znacznika należy dodać kolejną właściwość o nazwie `html`, zawierającą literał obiektowy z dwiema właściwościami: `content` oraz `popup`. Oto przykład:

```
$('#map').goMap({
    markers : [
        {
            address : '2200 NE 71st Ave Portland, OR, USA',
            title : 'Znacznik nr 1',
            html : {
                content : '<p>Fajne miejsce</p>',
                popup : true
            }
        }
    ],
    zoom : 15,
}); // koniec funkcji goMap
```

We właściwości `content` znajduje się kod HTML, który chcemy wyświetlić w okienku informacyjnym — można w nim umieścić zwyczajne znaczniki HTML oraz teksty. Z kolei właściwość `popup` określa, czy okienko to będzie widoczne bezpośrednio po wyświetleniu mapy (w takim przypadku właściwości tej należy przypisać wartość `true`, jak na powyższym przykładzie), czy też użytkownik będzie musiał kliknąć znacznik, aby je wyświetlić (w takim przypadku właściwości należy przypisać wartość `false`).

W okienkach informacyjnych można wyświetlić dowolny kod HTML — tabele, obrazki, listy. Można też w nich pokazać całkiem sporo tekstu — ich wymiary są automatycznie dostosowywane do zawartości.

Wskazówka: Do określania postaci zawartości okienek informacyjnych przy użyciu CSS można skorzystać z selektorów elementów potomnych. Jeśli na przykład w znaczniku `<div>` użytym do wyświetlania mapy podaliśmy identyfikator `map` (patrz krok 3. na stronie 407), postać znaczników `<p>` umieszczanych wewnętrz okienek informacyjnych można by określić za pomocą selektora `#map p`.

Przykład zastosowania wtyczki GoMap

W tym przykładzie wykonasz czynności konieczne do wyświetlenia na swojej stronie mapy Google przy użyciu wtyczki GoMap. Dodatkowo napiszesz kod, który pozwoli użytkownikom poprosić o wyświetlenie instrukcji dojazdu we wskazane miejsce.

Uwaga: Informacje na temat pobierania przykładów dotyczących do tej książki można znaleźć na stronie 43.

1. W edytorze tekstów otwórz plik *map.html* umieszczony w katalogu *R12*.

Zanim zaczniesz dodawać do strony kod JavaScript, zmodyfikujesz nieco jej kod HTML — dodasz pusty znacznik *<div>*.

2. Odszukaj znacznik *<h1>* umieszczony w treści strony (*<h1 class="shadow-
Line">Mapy Google</h1>*). Kliknij pusty wiersz umieszczony bezpośrednio
poniżej nagłówka i wpisz w nim:

```
<div id="map"></div>
```

Właśnie utworzyłeś pojemnik na mapę — to w nim wtyczka GoMap doda mapę Google Maps.

Kolejną czynnością będzie dodanie reguły stylu, która określi wysokość i szerokość mapy.

3. Na początku pliku, w sekcji nagłówka strony, umieszczony jest wewnętrzny arkusz stylów; dodaj do niego poniższą regułę (umieść ją tuż poniżej otwierającego znacznika *<style>*):

```
#map {  
    height:400px;  
    width:760px;  
}
```

Aktualny obszar mapy to 400 pikseli wysokości i 760 pikseli szerokości. Kolejnym zadaniem będzie dodanie kodu JavaScript.

4. Kliknij pusty wiersz bezpośrednio powyżej zamkającego znacznika *</head>* i wpisz:

```
<script  
src="http://maps.google.com/maps/api/js?sensor=false"></script>  
<script src="../_js/jquery-1.6.3.min.js"></script>  
<script src="../_js/jquery.gomap-1.3.2.min.js"></script>
```

Pierwszy znacznik *<script>* wczytuje z serwera Google.com kod JavaScript za-wierający wszystko, co jest niezbędne do wyświetlenia mapy. Drugi znacznik *<script>* wczytuje bibliotekę jQuery, a trzeci — wtyczkę GoMap. Teraz jesteś już gotów do wyświetlenia mapy.

5. Poniżej wierszy dodanych w poprzednim kroku dodaj kolejny znacznik *<script>* i umieść w nim wywołanie funkcji *\$(document).ready()*:

```
<script>  
$(document).ready(function() {  
  
}); // koniec funkcji ready  
</script>
```

Teraz wystarczy, że odwołasz się do znacznika *<div>* utworzonego w kroku 2. i dodasz wywołanie funkcji *goMap()*.

6. Wewnątrz funkcji *\$(document).ready()* wpisz następujący fragment kodu:

```
$('#map').goMap({  
    latitude : 45.53940,  
    longitude : -122.59025  
}); // koniec funkcji goMap
```

Powyższy kod pokazuje mapę wewnątrz znacznika *<div>* dodanego w kroku 2. i wyświetla w jej środku punkt o podanych współrzędnych (nic nie stoi na przeszkozie, byś zmienił te współrzędne i wyświetlił na mapie zupełnie inne miejsce świata).

7. Zapisz plik i wyświetl go w przeglądarce.

Jeśli Twój komputer jest podłączony do internetu (czyli strona może się odwołać do witryny Google.com), zobaczysz na stronie mapę o wymiarach 760×400 pikseli.

Zrób zbliżenie prezentowanego fragmentu mapy.

8. Zmodyfikuj skrypt, dodając przecinek za właściwością `longitude` i dopisując kolejną właściwość — `zoom` (wyróżnioną poniżej pogrubioną czcionką):

```
$('#map').goMap({  
    latitude : 45.53940,  
    longitude : -122.59025,  
    zoom : 16  
}); // koniec funkcji goMap
```

W powyższy sposób można całkowicie oddalić mapę (używając wartości 0) bądź też wyświetlić ją w taki sposób, że będą widoczne szczegółowe informacje o ulicach (używając wartości 17). Kolejnym krokiem będzie dodanie skali, dzięki czemu użytkownik będzie w stanie oszacować odległości na mapie.

9. Ponownie zmodyfikuj skrypt. Tym razem dodaj przecinek za właściwością `zoom` i dodaj wyróżniony fragment poniższego kodu:

```
$('#map').goMap({  
    latitude : 45.53940,  
    longitude : -122.59025,  
    zoom : 16,  
    scaleControl : true  
}); // koniec funkcji goMap
```

Ten nowy wiersz sprawi, że w lewym, dolnym rogu mapy pojawi się znacznik skali. Kolejną czynnością, jaką wykonasz, będzie zmiana typu mapy (zazwyczaj wtyczka GoMap wyświetla mapę hybrydową, prezentującą zdjęcia satelitarne z nałożonymi na nie ulicami i ich nazwami) na zdjęcia satelitarne.

10. Wprowadź następną, ostatnią już zmianę w kodzie funkcji `goMap()` — dodaj kolejną właściwość:

```
$('#map').goMap({  
    latitude : 45.53940,  
    longitude : -122.59025,  
    zoom : 16,  
    scaleControl : true,  
    maptype : 'SATELLITE'  
}); // koniec funkcji goMap
```

Podstawowa mapa jest już gotowa. Teraz, aby zaznaczyć konkretny punkt na tej mapie, dodasz do niego znacznik.

11. Za wywołaniem funkcji `goMap()` dodaj poniższy fragment kodu:

```
$.goMap.createMarker({  
    latitude : 45.53743,  
    longitude : -122.59473,  
    title : 'hole1'  
}); // koniec funkcji createMarker
```

Powyższy kod ustawia znacznik dokładnie w tym samym miejscu, które jest wyświetlone pośrodku mapy. Oczywiście, można podać inne wartości szerokości i długości geograficznych, dodając tym samym znacznik w innym miejscu mapy.

Ostatnią czynnością będzie dodanie do znacznika okienka informacyjnego.

12. Zmodyfikuj kod funkcji dodanej w poprzednim kroku tak, by wyświetlała okienko informacyjne (zmiany zostały zaznaczone pogrubioną czcionką):

```
$.goMap.createMarker({
    latitude : 45.53743,
    longitude : -122.59473,
    title : 'hole1',
    html : {
        content : '<h2>Dołek nr 1</h2><p>Par 4, 346 jardów</p>',
        popup : true
    }
}); //koniec funkcji createMarker
```

Ten nowy fragment kodu dodaje do znacznika okienko informacyjne. Zapisz zmiany wprowadzone w pliku, a następnie wyświetl go w przeglądarce. Gotową wersję tego przykładu znajdziesz w pliku *complete_map.html* umieszczonym w katalogu *R12*.

V

CZĘŚĆ

Rozwiązywanie problemów, wskazówki i sztuczki

- Rozdział 13. „Wykorzystywanie wszystkich możliwości jQuery”
- Rozdział 14. „Zaawansowane techniki języka JavaScript”
- Rozdział 15. „Diagnozowanie i rozwiązywanie problemów”

Wykorzystywanie wszystkich możliwości jQuery

Biblioteka jQuery w ogromnym stopniu ułatwia pisanie programów w języku JavaScript i stanowi narzędzie, za pomocą którego projektanci stron mogą szybko i łatwo wzbogacać swoje strony wyszukanymi możliwościami interakcji. Jednak stosowanie jej 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', '#F03');
```

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 znaczco szybsze, zatem w większości przypadków zapewne będziesz korzystał właśnie z niego (jak czyni większość programistów).

Uwaga: W innej popularnej bibliotece JavaScript — Prototype (www.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 znaczącego spowolnienia działania programu. Założmy 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 149 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 powstawania niewygodnego kodu. 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 162), wewnętrz 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 wewnętrz 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 150) pozwala na zastąpienie tekstu umieszczonego wewnątrz elementu, a funkcja `.html()` (patrz strona 150) na zmianę umieszczonego wewnątrz elementu kodu HTML. Gdybyśmy na przykład chcieli umieścić komunikat o błędzie w znaczniku `` 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 151) lub przed nim (`before()`, patrz strona 151).

Dodawanie i modyfikacja treści umożliwiają wyświetlanie nowych komunikatów o błędach, prezentowanie etykiet ekranowych (patrz strona 340), tworzenie wyróżnionych cytatów (patrz strona 163), 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 138), a to z kolei wiąże się z koniecznością wykonania wielu, niezauważalnych czynności. Dlatego też częste modyfikowanie zawartości strony może doprowadzić do znaczącego spadku wydajności jej działania.

W tym przypadku nie liczy się wielkość dodawanych treści — znacznie większy wpływ na wydajność strony ma sama liczba wprowadzanych modyfikacji. Założymy 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 3. do znacznika `<div>` — nagłówek, a w wierszu 4. — 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.

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.

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 432). Techniki opisane w tym punkcie rozdziału pozwolą przyspieszyć wykonywane operacje selekcji elementów strony i poprawić wydajność naszych programów pisanych w JavaScriptie.

- **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 odwoływać 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. Założymy 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. Rozwiążanie 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 434, 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 podległego!

Uwaga: Na stronie <http://jsperf.com/sawmac-selector-test> można znaleźć test pozwalający sprawdzić wydajność 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` 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 podległych, takie jak `$('.sidebar .nav a')`, bądź też skorzystać z funkcji `.find()` (opisanej w poprzednim punkcie): `($('#main').find('.sidebar').find('.nav a'))`.

Uwaga: Jeśli interesują Cię inne sposoby poprawiania wydajności programów korzystających z biblioteki jQuery, warto obejrzeć prezentację na <http://addyosmani.com/jqprovenperformance/> oraz <http://jqfundamentals.com/#chapter-9>. Wiele porad i informacji dotyczących stosowania jQuery można znaleźć na stronie <http://tutorialzine.com/2011/06/15-powerful-jquery-tips-and-tricks-for-developers/>.

Korzystanie z dokumentacji jQuery

Na witrynie jQuery umieszczono bardzo szczegółową dokumentację wszystkich funkcji biblioteki, jest ona dostępna na stronie <http://docs.jquery.com/> (patrz rysunek 13.1). 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.

Na rysunku 13.1 pokazane zostały główne kategorie API. Wystarczy kliknąć jeden z tych odnośników, by wyświetlić stronę zawierającą listę wszystkich funkcji związanych z danym aspektem działania jQuery. W poniższej liście przedstawiono i opisano każdą z tych kategorii.

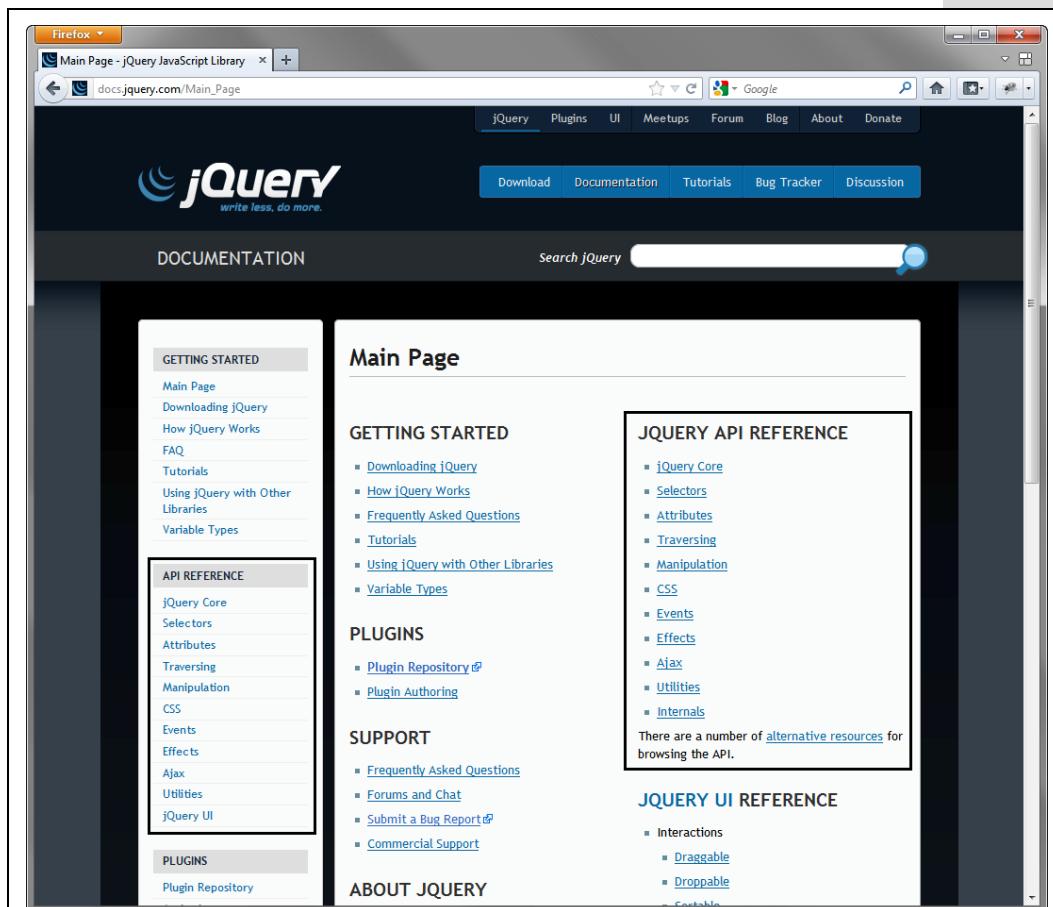
- Kategoria *Core* (podstawowe możliwości; <http://api.jquery.com/category/core/>). Znajdziesz tu informacje o zaledwie kilku funkcjach; większość z nich to funkcje zaawansowane, których prawdopodobnie nigdy nie będziesz musiał używać. Do kategorii tej należy także główna funkcja biblioteki: `jQuery()`. Szczegółowe informacje na jej temat można znaleźć na stronie <http://api.jquery.com/jQuery/>.

Uwaga: Podczas lektury tej książki nauczyłeś się używać funkcji `$()` i to głównie z niej korzystałeś do pobierania elementów stron — na przykład `$(‘p’)`. Dokładnie ta sama funkcja jest dostępna także pod nazwą `jQuery()`, a to jedyne jej zastępca nazwa. Obu tych funkcji można używać zamienne, a zatem w dołączonych do książki przykładach możesz się spotkać na przykład z wywołaniem:

```
jQuery('p').css('color', '#F03');
```

które jest odpowiednikiem wywołania w postaci:

```
$(‘p’).css('color', '#F03');
```



Rysunek 13.1. Główna strona dokumentacji jQuery zawiera odnośniki do wielu przydatnych informacji; te, które pozwolą Ci poznać wszystkie funkcje udostępniane przez bibliotekę, znajdują się w sekcji API Reference (wyróżnionej na rysunku ramką) oraz w lewej kolumnie. Jak możesz się przekonać, lewa kolumna będzie widoczna także po wyświetleniu innych stron dokumentacji

- 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 zajrzesz do tej sekcji dokumentacji, możesz ich poznać jeszcze więcej.
- 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 ``), wykonać na nim jakieś operacje (dodać klasę, stopniowo wyświetlić i tym podobne), a następnie odszukać wewnątrz niego jakiś inny element (przykładowo znacznik `` umieszczony we wcześniej pobranym znaczniku ``) 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/manipulation/>). Za każdym razem, gdy zechcemy dodać coś do strony bądź z niej coś usunąć, konieczne jest przeprowadzenie pewnych zmian w drzewie DOM (modelu obiektów dokumentu, patrz strona 138). 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 149), 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 na 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 155.
- 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 447.
- 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 *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 4. części książki.

- 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 (patrz strona 72), wykonanie pewnych operacji na każdym elemencie tablicy lub właściwości obiektu (funkcja `$.each()` opisana na stronie 160) 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 odrębnych efektów ani nie są nam w stanie pomóc w aktualizowaniu treści strony), kiedy jednak zdobędziesz więcej wiedzy i doświadczenia, warto ponownie zająć się na tej stronie dokumentacją.

Co ciekawe, kiedy wyświetlimy stronę jednej z powyższych kategorii (takich jak *Ajax* lub *Selectors*), na liście pokazanej w lewej kolumnie strony pojawią się dodatkowe kategorie funkcji, które nie zostały wymienione na głównej stronie dokumentacji. Oto one.

- 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 *Deferred objects* (obiekty odroczone, <http://api.jquery.com/category/deferred-object/>). Nie musisz szukać dalej, wystarczy, że zajrzesz 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 *Dimensions* (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 kategorii *CSS*.
- 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 4. części książki).

- 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 które znajdują się informacje o tym, co robi i jak działa. Na rysunku 13.2 przedstawiony 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 podkategoriei można kliknąć, by przejść na stronę z listą wszystkich należących do niej funkcji.

The screenshot shows the jQuery API documentation for the `.css()` method. At the top, there's a navigation bar with "Nazwa funkcji" pointing to ".css()", and "Kategoria i podkategorie" pointing to the category list. Below the header, the main content area has two sections: "Contents:" and "Categories:". The "Contents:" section lists two entries: `.css(propertyName)` (marked with a circled '1') and `.css(propertyName , value)` (marked with a circled '2'). The "Categories:" section shows `css` | `Manipulation` > `Style Properties`. A curved arrow points from the circled '1' to the detailed description of the first entry. The detailed description for `.css(propertyName)` includes a "Returns: String" label, a "Description" paragraph, and a "version added: 1.0" note. Below this, there's a larger text block explaining the purpose of the `.css()` method. To the right of the screenshot, there's a descriptive caption:

Rysunek 13.2. 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. Klikając niewielką, niebieską strzałkę w dół (zaznaczoną w kółku), można przeskoczyć do miejsca strony, w którym dany sposób wywołania został opisany

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 13.2 widać, że funkcji `css()` można używać na dwa różne sposoby (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 — umieszczone z prawej strony szarego paska, widocznego mniej więcej pośrodku rysunku). Informacje zamieszczone na stronie pozwalają się zorientować, że do funkcji należy przekazać jeden argument, a wynikiem wywołania jest łańcuch znaków. Założymy 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').

Drugi sposób użycia funkcji `css()` (oznaczony cyfrą 2) został przedstawiony jako `css(propertyName, value)`. A zatem zakładając 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 'width', a drugim — liczba określająca zamierzoną szerokość elementu:

```
$('#tooltip').css('width', 300);
```

Dokumentacja zawiera także informacje o dwóch innych sposobach stosowania funkcji `css()` w celu ustawiania wartości właściwości CSS. Oto one.

- `.css(propertyName, function(index,value))` — w tym przypadku istnieje możliwość dynamicznego wyznaczenia wartości, która zostanie przypisana właściwości CSS. Rozwiążanie 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(map)` — ten sposób wywołania został opisany na stronie 155; oznacza on, że w wywołaniu funkcji jest przekazywany literal obiektowy, dzięki czemu, za jednym zamachem, można ustawić wartości wielu właściwości CSS.

Ważne jest, by uzmysolić 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 założmy, 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 kliką 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 162, 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 (wyróżnionej pogrubioną czcionką):

```
$('#gallery').click(function() {  
    $(this).find('img');  
}); // koniec funkcji click
```

Wywołanie `$(this).find('img')` tworzy nową kolekcję pobranych elementów; wyrażenie `$(this)` odwołuje się w najpierw do znacznika `<div>`, a następnie wywołanie `.find('img')` odnajduje wszystkie znaczniki `` 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 poznaleś 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 podanymi na stronie 200, 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 *R07*).

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 13.3. Zawiera on znacznik *<div>* o identyfikatorze *gallery*, a wewnątrz niego cztery obrazki umieszczone w odnośnikach.

Poruszanie się po DOM



```
<div id="gallery">
  <a> <img>
  <a> <img>
  <a> <img>
  <a> <img>
```

Rysunek 13.3. 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 nim w jakiś sposób powiązane (wyświetlić ramkę wokół elementu *<div>* lub zmodyfikować znacznik ** i tym podobne). Właśnie w takich przypadkach mogą pomóc funkcje służące do poruszania się po DOM

Zgodnie z informacjami podanymi na stronie 138, do przedstawienia powiązań pomiędzy elementami stron WWW można wykorzystać relacje rodzinne. Przykładowo znacznik *<div>* przedstawiony na rysunku 13.3 jest rodzicem znaczników *<a>*, a kolejne one są rodzicami umieszczonych wewnątrz znaczników **. Jednocześnie znaczniki *<a>* są dziećmi znacznika *<div>* oraz rodzeństwem w stosunku do pozostałych znaczników *<a>*. I analogicznie, każdy znacznik ** jest dzieckiem znacznika *<a>*, wewnątrz którego jest umieszczony; a ponieważ w znacznikach *<a>* nie ma innych znaczników, żaden ze znaczników ** nie ma rodzeństwa.

Poniżej przedstawionych zostało kilka funkcji służących do poruszania się po DOM, dostępnych w bibliotece jQuery.

- Funkcja `.find()` odnajduje konkretny element wewnętrz 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 `` umieszczone wewnętrz 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 wewnętrz innych) aktualnie pobranego elementu. Jeśli zatem w przypadku zilustrowanym na rysunku 13.3 aktualnie pobranym elementem będzie `<div>`, funkcji `.find()` możemy użyć, by pobrać znaczniki `<a>` lub ``.

Uwaga: Zajrzyj do poprzedniego podrozdziału, 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. Założmy, że na stronie znajduje się znacznik `<div>`, a wewnętrz niego grupa kolejnych znaczników `<div>`. Kliknięcie głównego znacznika `<div>` powinno spowodować wyświetlenie pozostałych, które są początkowo ukryte, i dodanie do nich czerwonego obramowania. Założmy, 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 wewnętrz znacznika głównego znajdowały się kolejne znaczniki `<div>`. W takim przypadku użycie powyższego kodu doprowadziłoby do zmiany wyglądu wszystkich znaczników `<div>` umieszczonych wewnętrz 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 wewnętrz pobranego znacznika, funkcja `.parent()` podróży w górę DOM i odnajduje znaczniki przodków. Taka możliwość może się przydać, gdy-

byśmy na przykład dodali procedurę obsługi zdarzeń do znaczników `<a>` z rysunku 13.3, lecz chcemy 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() {
    // dodanie obramowania do odnośnika
    $(this).css('outline', '2px solid red');
    // dodanie koloru tła do znacznika div
    $(this).parent().css('backgroundColor', 'white');
  },
  function() {
    // 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 183). Aby przekonać się, jak w praktyce działa ten fragment kodu, wystarczy wyświetlić w przeglądarce plik `parent.html` umieszczony w przykładach do książki, w katalogu `R13`.

- 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 13.3 każdy obrazek jest umieszczony wewnątrz znacznika `<a>`; innymi słowy, ten znacznik `<a>` jest rodzicem obrazka. W jaki sposób możemy pobrać znacznik `<div>`, 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    $(this).css('outline', '2px red solid');
3    $(this).closest('div').css('backgroundColor', 'white');
4  }); // koniec funkcji click
```

Użyte wierszach 2. i 3. wyrażenie `$(this)` odwołuje się do znacznika ``. Wywołanie `.closest('div')` oznacza natomiast, że chcemy znaleźć najbliższego przodka będącego znacznikiem `<div>`. Najbliższym — bezpośrednim — przodkiem obrazka jest znacznik `<a>`, jednak nie jest to `<div>`, dlatego jQuery go pominie i sprawdzi kolejnego przodka, i tak dalej, aż do momentu odszukania znacznika `<div>`.

- Funkcja `.siblings()`. Funkcja ta może się przydać, kiedy chcemy odszukać element znajdujący się na tym samym poziomie struktury kodu HTML, co element aktualnie pobrany. Założmy, że cały czas posługujemy się przykładowym kodem przedstawionym na rysunku 13.3. Tym razem chcemy, by w momencie kliknięcia jednego z odnośników wszystkie pozostałe zostały nieznacznie wygąszone, 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 `<div>`. 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 `<div>`.

Także funkcja `.siblings()` umożliwia przekazanie jednego argumentu — selektora — w ten sposób pozwala na ograniczenie liczby pobieranych elementów. Przykładowo założmy, że wewnątrz znacznika `<div>` przedstawionego na rysunku 13.3. przed grupą odnośników znajduje się nagłówek oraz jeden akapit tekstu. Ponieważ zarówno ten nagłówek, jak i akapit umieszczone są wewnątrz znacznika `<div>` 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 `R13`.

- 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 191. 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 klikniętym pytaniem (innymi słowy — następnego znacznika należącego do rodzeństwa klikniętego elementu). Funkcja `.next()`, podobnie jak `.siblings()`, pozwala na podanie opcjonalnego selektora ograniczającego zwarcane 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/>.

KLINIKĄ ZAAWANSOWANEGO UŻYTKOWNIKA

Przerywanie poruszania się po DOM przy użyciu funkcji .end()

Aby móc zrobić jak najwięcej, pisząc jak najmniej kodu, jQuery pozwala na tworzenie sekwencji wywołań. Technika ta została opisana na stronie 149, 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. Założymy 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 założmy, 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() {
    $(this).fadeTo(250,1);
    // rozjaśniamy znacznik <div>
    $(this).find('h2').css('color', '#F30');
    $(this).find('p').css('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 próbowały nadać tej sekwencji poniższą postać, pojawiłyby się problemy:

```
$( 'div' ).click(function() {
    $(this).fadeTo(250,1)
        .find('h2').css('color', '#F30')
        .find('p').css('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 rozpoczęć poszukiwanie znaczników `<p>`:

```
$( 'div' ).click(function() {
    $(this).fadeTo(250,1)
        .find('h2').css('color', '#F30').end()
        .find('p').css('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>`.

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 150 w rozdziale 4. Niżej zostały pokrótko 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 zamkającym 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 HTML przed aktualnie pobranym elementem.

- Funkcja `.after()` działa podobnie do funkcji `.before()`, z tą różnicą, że nowy kod HTML jest dodawany za aktualnie pobranym elementem (za jego znacznikiem zamkającym).

To, której funkcji użyjemy, zależy przede wszystkim od tego, co chcemy osiągnąć. Jak już wspominaliśmy na stronie 134, 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
9     ↪+ '</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 160). W wierszu 4. aktualnie pobrany obrazek jest zapisywany w zmiennej (to bardzo dobre rozwiązanie, o czym już wspominaliśmy na stronie 422). 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 R13.

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 wewnętrz 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 wewnętrz podanego kodu HTML. Założmy 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, wewnętrz których są umieszczone aktualnie pobrane elementy. Przykładowo założymy, 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. Założmy, 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/>.

Zaawansowana obsługa zdarzeń

W rozdziale 5. poznaleś wygodne funkcje jQuery służące do określania procedur obsługi zdarzeń. Gdybyś na przykład chciał, by kliknięcie nagłówka `h1` powodowało wyświetlenie okienka informacyjnego, mógłbyś w tym celu zastosować funkcję `.click()`:

```
$(‘h1’).click(function() {  
    alert(‘ałà!’);  
}); //koniec funkcji click
```

Biblioteka jQuery udostępnia funkcje służące do odpowiadania na różne zdarzenia, takie jak `.submit()` obsługująca przesyłanie formularza czy też `.mouseout()`, która pozwala coś zrobić, gdy użytkownik usunie wskaźnik myszy z obszaru elementu. Wszystkie te funkcje są jednak skróconym sposobem wywołania funkcji `.bind()`, opisanej na stronie 188.

Do funkcji `.bind()` można przekazać kilka argumentów: typ zdarzenia (`‘click’`, `‘mouseover’` i tak dalej), oraz funkcję, którą należy wykonać w celu jego obsługi. Przedstawiony powyżej fragment kodu można napisać na nowo w następujący sposób:

```
$(‘h1’).bind(‘click’, function() {  
    alert(‘ałà!’);  
}); //koniec funkcji bind
```

Jednak zarówno funkcja `.bind()`, jak i pozostałe uproszczone sposoby określania procedur obsługi zdarzeń, takie jak `.click()` lub `.hover()`, mają jedno poważne ograniczenie: mogą operować wyłącznie na kodzie HTML, który w momencie wywołania stanowi zawartość strony. Wystarcza to w przykładach prezentowanych w tej książce — kod HTML strony jest wczytywany, przeglądarka określa stosowane w nim procedury obsługi zdarzeń, a użytkownik może prowadzić interakcje ze stroną. Kiedy jednak zdobędziesz doświadczenie, zaczniesz tworzyć aplikacje, które będą często aktualizować zawartość strony (za pomocą takich funkcji jQuery jak `.append()`, `.before()` oraz innych funkcji opisanych w tym rozdziale). Niestety, procedury obsługi zdarzeń nie będą operować na kodzie HTML dodanym do strony po momencie ich przypisania, a to może powodować problemy.

Załóżmy na przykład, że tworzymy grę działającą na stronie WWW: jej celem jest usunięcie wszystkich chwastów z ogrodu. Gracz musi kliknąć chwast, by go usunąć z ekranu. Oczywiście, chwasty, jak to mają w zwyczaju, szybko wypełniają cały ekran. Innymi słowy, program cały czas dodaje kolejne chwasty, a użytkownik musi je kliknąć, aż do chwili, kiedy wszystkie znikną. W naszej grze chwasty są reprezentowane przez znaczniki `<div>`, wewnętrznych których jest umieszczony obrazek chwastu. Program bezustannie dodaje na stronę kolejne znaczniki `<div>`, natomiast gracz stara się wszystkie usunąć. Oznacza to, że w każdym z tych znaczników musimy określić

procedurę obsługi zdarzeń `click`, tak by mogły odpowiedzieć na kliknięcie. Można by to zrobić, dodając do programu następujący fragment kodu:

```
$('.weed').click(function() {  
    $(this).remove();  
}); // koniec funkcji click
```

Jednak problem w tym rozwiążaniu polega na tym, że będzie ono operować wyłączenie na tych elementach, które istniały w treści strony w momencie jego wykonywania. Jeśli programowo dodamy do strony kolejne znaczniki `<div class="weed">`, procedura obsługi zdarzeń `click` nie zostanie w nich określona.

Kod, który operuje wyłącznie na istniejącej zawartości strony, stanowi także spory problem w aplikacjach korzystających z technologii AJAX, takich jak opisane w 4. części książki. Technologia ta pozwala aktualizować zawartość stron przy wykorzystaniu informacji pobieranych z serwera WWW. Przykładowo aplikacja Gmail może wyświetlać nowe wiadomości poczty elektronicznej, bezustannie pobierając je z serwera i aktualizując informacje prezentowane w przeglądarce. W tym przypadku po uruchomieniu przeglądarki i rozpoczęciu korzystania z aplikacji lista otrzymanych wiadomości będzie się zmieniać. Wszystkie procedury obsługi zdarzeń określone w momencie wczytywania strony nie będą działały w nowej treści dodawanej później z serwera.

Oczywiście, można ponownie przypisywać procedury obsługi zdarzeń w momencie aktualizacji strony, jednak taka metoda jest wolna i nieefektywna. Na szczęście, jQuery udostępnia jeszcze jedną funkcję związaną z obsługą zdarzeń, której można używać właśnie w takich sytuacjach — jest nią funkcja `.delegate()`. Funkcja ta jest aktywna nawet po dodaniu do zawartości strony nowych elementów, co oznacza to, że wystarczy ją wywołać w programie tylko raz, a będzie odpowiadała nawet na zdarzenia kierowane do nowych elementów strony, dodanych do niej po wywołaniu. Poniżej przedstawiona została podstawowa składnia tej funkcji:

```
$('#container').delegate('selector', 'event', function() {  
    // tutaj dodajemy kod obsługujący zdarzenia  
}); // koniec funkcji delegate
```

Jak widać, postać wywołania tej funkcji jest dosyć dziwna, warto ją więc dokładniej przeanalizować.

- Zaczynamy od pobrania elementu nadziednego — pojemnika. To nieco myjące rozwiążanie, gdyż w przypadku funkcji `.bind()` odwołujemy się do elementu, w którym chcemy obsługiwać zdarzenia. W tym przypadku odwołujemy się natomiast do elementu *zawierającego* ten element strony, w którym chcemy określić procedurę obsługi zdarzeń. Gdybyśmy chcieli dodać procedurę obsługi zdarzeń `click` do wszystkich elementów listy umieszczonych wewnętrz znacznika `<div>` o identyfikatorze `sidebar`, musielibyśmy zacząć od pobrania tego elementu `<div>`:

```
$('#sidebar')
```

Gdybyśmy natomiast chcieli określić procedurę obsługi zdarzeń `mouseover` dla wszystkich znaczników `<a>` na stronie, musielibyśmy się odwołać do samego znacznika `<body>`:

```
$('body')
```

- Następnie musimy dodać wywołanie funkcji `.delegate()` i przekazać do niego trzy argumenty: selektor określający, do jakich elementów strony ma być dodawana procedura obsługi zdarzeń, nazwę zdarzenia oraz funkcję obsługującą te zdarzenia. Pełny kod wywołania funkcji `.delegate()` w celu przypisania procedury obsługi zdarzeń `click` wszystkim elementom listy umieszczonym w znaczniku `<div>` o identyfikatorze `sidebar` wyglądałby następująco:

```
$('#sidebar').delegate('li', 'click', function() {  
    // tu wykonujemy jakieś operacje  
}); // koniec funkcji delegate
```

Analogicznie, aby dodać procedurę obsługi zdarzeń `mouseover` do wszystkich znaczników `<a>` na stronie, należałoby użyć następującego wywołania funkcji `.delegate()`:

```
$( 'body' ).delegate( 'a', 'mouseover', function() {  
    // tu wykonujemy jakieś operacje  
}); // koniec funkcji delegate
```

Aby przetestować działanie funkcji `.delegate()`, wystarczy otworzyć w przeglądarce plik `delegate.html` dostępny w przykładach, w katalogu `R13`.

Uwaga: Biblioteka jQuery udostępnia także funkcję o nazwie `.live()`, która działa podobnie do `.delegate()` pod tym względem, że także pozwala na określanie procedury obsługi zdarzeń w elementach strony dodawanych po jej wywołaniu. Jednak działa znacznie wolniej od `.delegate()`, a jej stosowanie nie jest zalecane.

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 477, zawiera wartościowe wskazówki dla początkujących, jednak z powodzeniem możesz napisać wiele programów bez korzystania z informacji zamieszczonych w pozostałych podrozdziałach. Jeśli jednak chcesz rozwinać 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. Założymy 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 założymy, 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 91) 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.');
}
```

Uwaga: W naszym przykładzie przedstawiony powyżej fragment kodu można umieścić w procedurze obsługi zdarzeń `submit` (patrz strona 277), dzięki czemu możemy go użyć do sprawdzania poprawności hasła w momencie przesyłania formularza.

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 118), 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 o 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 wewnętrz wyrażenia warunkowego zarówno odpowiedź udzielona przez użytkownika, jak i prawidłowa odpowiedź na pytanie są przekształcone 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 `toLowerCase()`, 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 do funkcji opisanych na stronie 114, które zwracają jakieś wartości).

Przeszukiwanie łańcuchów znaków: zastosowanie `indexOf()`

Język JavaScript udostępnia kilka technik przeszukiwania łańcuchów znaków i odnajdywania wewnętrz 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 umieszczoneo wiele informacji; na przykład w przeglądarce Internet Explorer 7 działającej w systemie Windows XP właściwość ta przyjmuje wartość Mozilla/4.0 (compatible; MSIE 7.0; Windows NT 5.1). A zatem, gdybyśmy chcieli sprawdzić, czy przeglądarką użytkownika jest Internet Explorer 7, wystarczy spróbować odnaleźć we właściwości `userAgent` łańcuch znaków „MSIE7”.

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 zwrocona 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:

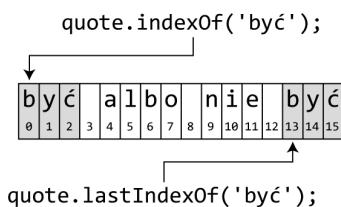
```
var quote = 'być albo nie być';
var searchPosition = quote.indexOf('być'); //zwraca 0
```

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 ē) — liczba 2 (więcej informacji na ten temat można znaleźć na stronie 75; w podobny sposób są indeksowane także tablice).

Metoda `indexOf()` rozpoczyna poszukiwanie 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 14
```

Wyniki zwracane przez obie metody zostały zilustrowane na rysunku 14.1. 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 14.1. 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 założymy, 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 14.2). 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.

`quote.slice(7);`

ht t t p : / / w w w . s a w m a c . c o m
0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20

Rysunek 14.2. 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 łańcucha

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(początek, koniec);
```

Argument `początek` 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 14.3, 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.

```
var quote='być albo nie być';
        quote.slice(0,3);
```

b y Ą Ą a l b o n i e b y Ą Ą
0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15

quote.slice(4,8);

quote.slice(-7,-4);

Rysunek 14.3. Metoda `slice()` pobiera fragment łańcucha znaków. Początkowy łańcuch nie jest przy tym w żaden sposób modyfikowany. Przykładowo wywołanie `quote.slice(0,5)` 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 określona 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 14.3).

Wskazówka: Gdybyśmy na przykład chcieli pobrać wszystkie znaki łańcucha, zaczynając od siódmego znaku przed końcem aż do samego końca, w wywołaniu metody `slice()` należałoby pominąć drugi argument:

```
quote.slice(-7);
```

Odnajdywanie wzorów w łańcuchach

Czasami może pojawić się konieczność przejrzenia łańcucha w poszukiwaniu nie konkretnej wartości, lecz określonego wzorca znaków. Założymy 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 `243///kkkmnnn`), 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 mylące. Poniżej przedstawione zostało jedno z często używanych wyrażeń regularnych:

```
/^[-\w. ]+@[a-zA-Z0-9][a-zA-Z0-9]+\.\+[a-zA-Z]{2,4}$/
```

Nie ma w nim nic, co mogłoby wyglądać na „regularne”, no chyba że jesteśmy superbymi z Omicrona 9. Do tworzenia wyrażeń regularnych używane są znaki i ich sekwencje, takie jak *, +, ? oraz \w, które są następnie tłumaczone przez interpretera JavaScriptu do postaci umożliwiającej dopasowywanie ich do prawdziwych znaków zapisywanych w łańcuchach — liter, cyfr i tym podobnych.

Uwaga: Profesjonalisci 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 dalszej części rozdziału, od strony 461), jednak najprostszą z nich jest metoda `search()`. Działa podobnie do metody `indexOf()`, lecz zamiast odnajdywać jeden łańcuch znaków wewnętrz innego, poszukuje wzorca (czyli wyrażenia regularnego). Założymy, że chcemy odnaleźć łańcuch znaków „być” w łańcuchu „być albo nie być”. Dowiedziałeś się, jak to zrobić na stronie 447, a poniżej pokazujemy, 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 447), metoda `indexOf()` działa w dokładnie taki sam sposób. Skoro obie metody działają tak samo, możesz się zastanawiać, po co w ogóle zwracać sobie głowę wyrażeniami regularnymi? Otóż, wyrażenia regularne mają tę zaletę, że pozwalają odnajdywać wzorce, czyli są w stanie realizować znacznie bardziej złożone i subtelniesze 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 regularne 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 14.1 zawarto listę znaków najczęściej używanych w wyrażeniach regularnych.

Tabela 14.1. Kilka 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 przeciwnieństwo \w.
\d	dowolnej cyfry 0 – 9.
\D	dowolnego znaku z wyjątkiem cyfr. Stanowi przeciwnieństwo \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).
[^]	dowolnego znaku z wyjątkiem podanych w nawiasach. Przykładowo wyrażenie [^aęęęioóAĄĘĘIÓÓU] 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 460).
\	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 w postaci \..

Uwaga: Jeśli wszystkie rozważania o wyrażeniach „regularnych” powodują jedynie ból głowy, zapewne uciecze Cię wiadomość, że w książce (na stronie 456) 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 ułatwiają ich poznanie i przyswojenie sobie tej wiedzy. Założymy, 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 zajrzesz do tabeli 14.1, 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, najbliższym sposobem dopasowania pięciu kolejnych cyfr zapisanych jedna za drugą będzie użycie wyrażenia w postaci: \d\d\d\d\d. (Na stronie 454 został przedstawiony sposób zapisania tego samego wyrażenia w krótszej postaci).

3. Dopasowanie tylko pięciu liczb podanych jedna za drugą.

Wyrażenie regularne jest jak rakieta precyjnie 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 jaajaaja, 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 457.

4. Zastosowanie wyrażenia w kodzie JavaScript.

Założymy, że już zapisałś 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 14.2. Symbole te umieszcza się bezpośrednio za sprawdzanym znakiem.

Tabela 14.2. Znaki służące do wielokrotnego dopasowywania tego samego znaku lub wzorca

Znak	Odpowiada
?	braków wystąpień lub dokładnie jednemu wystąpieniu poprzedzającego elementu wzorca. 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.
*	braków 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.
{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 interpretera JavaScriptu, że ma się ona powtórzyć pięć razy. Wyrażenie \d{100} będzie odpowiadać 100 cyfr zapisanym jedna obok drugiej.

Przeanalizujmy kolejny przykład. Założymy, 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 461). Innymi słowy, chodzi nam o odnalezienie dowolnego łańcucha znaków pasującego do podstawowego wzorca nazwy pliku z rozszerzeniem GIF, na przykład `logo.gif`, `banner.gif` bądź `ad.gif`.

1. Określenie wspólnej postaci 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`.

2. Odszukanie 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 14.1, w wyrażenях 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. Odszukanie dowolnej liczby znaków umieszczonych przed rozszerzeniem `.gif`.

By utworzyć wyrażenie odpowiadające dowolnej liczbie dowolnych znaków, należy użyć kombinacji symboli `*`; zostaną one zrozumiane tak: „znajdź jeden znak `(.)` występujący zero lub dowolną ilość razy `(*)`”. To wyrażenie regularne zostanie dopasowane do całej zawartości każdego łańcucha znaków. Gdybyśmy jednak użyli go w wyrażeniu regularnym, takim jak `.*\.gif`, okazałoby się, że zostało ono dopasowane do nieco większego ciągu znaków niż planowana przez nas nazwa pliku. Gdyby na przykład sprawdzany łańcuch znaków miał postać "Plik nosi nazwę logo.gif", to wyrażenie `.*\.gif` zostało dopasowane do całego łańcucha, a nie tylko do nazwy pliku `logo.gif`. Dlatego też w wyrażeniu musimy użyć znaku `\S`, odpowiadającego każdemu znakowi z wyjątkiem odstępów, a wtedy wyrażenie `\S\.\gif` zostanie dopasowane do nazwy `logo.gif`.

4. Ignorowanie wielkości 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/I
```

Koniecznie należy zwrócić uwagę, że argument `i` jest umieszczony poza prawym ukośnikiem wyznaczającym koniec wyrażenia regularnego.

5. Zastosowanie wyrażenia w skrypcie.

```
var testString = 'Plik nosi nazwę logo.gif'; //testowany łańcuch znaków
var gifRegex = /\S*\.\gif/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 461).

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 14.2.

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 14.1) 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 na stronie 464).

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 `R14`. (Więcej informacji na temat pobierania przykładów znajdziesz na stronie 43).

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 www.regexlib.com, a konkretnie na stronie <http://regexlib.com/Search.aspx?k=polish>.

Powyzsze wyrażenie regularne wygląda na skomplikowane, jeśli jednak rozdzielimy je na fragmenty (i będziemy mogli skorzystać z dobrych wyjaśnień, takich jak zamieszczone poniżej), będzie można zrozumieć zasadę jego działania.

- \() odpowiada otwierającemu znakowi nawiasu. Ponieważ nawiasy są używane w wyrażeniach regularnych do tworzenia podwzorców grup, zatem nawias otwierający ma w nich specjalne znaczenie. By poinstruować interpreter JavaScriptu, że chodzi o faktyczny znak nawiasu otwierającego, musimy go poprzedzić odwrotnym ukośnikiem (przypomina to nieco poprzedzanie cudzysłówów i apostrofów opisane na stronie 58).
- Znak ? oznacza, że nawias otwierający jest opcjonalny, dzięki czemu numery telefonów, w których numer kierunkowy nie jest zapisany w nawiasie, na przykład 22-555-2121, także zostaną dopasowane do wyrażenia.
- Fragment (\d{2}) stanowi podwzorzec odpowiadający dwóm cyfrom.
- Fragment \)? 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})` jest kolejnym podwzorcem pasującym do sekwencji trzech cyfr.
- Fragment `[- .]` odpowiada znakowi odstępu, łącznikowi lub kropce.
- Fragment `(\d{4})` jest ostatnim podwzorcem odpowiadającym sekwencji czterech cyfr.

Uwaga: Podwzorce to fragmenty wzorca umieszczone wewnątrz nawiasów, takie jak `(\d{3})` przedstawiony w powyższym przykładzie. Stają się one bardzo użyteczne w przypadku stosowania metody `replace()`, opisanej w ramce na stronie 464.

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.comm`). Poniższe wyrażenie regularne sprawdza, czy łańcuch znaków zawiera adres poczty elektronicznej zapisany w prawidłowej postaci:

`[-\w.]+@[A-z0-9][-A-z0-9]+\.\{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-z0-9]` odpowiada jednej literze lub cyfrze.
- Fragment `[-A-z0-9]+\.` 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 uznaje 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!#$%&!*+\/=?^`{|}~@(?:[a-zA-Z][a-zA-Z-]*(?:[a-zA-Z][a-zA-Z-]*))?)?+[.](?:[a-zA-Z][a-zA-Z-]+)$/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 473 dowiesz się, jak skonwertować każdy z tych zapisów na jeden, standaryzowany 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})
```

- Nawiąsy — () — łączą dwa podane wewnętrz 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.
- Nawiązy — () — 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.

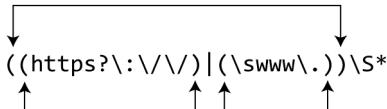
Adresy stron WWW

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 14.4 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.

1



2

3

Rysunek 14.4. 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 początek zewnętrznej grupy (na rysunku 14.4 została ona oznaczona numerem 1).
- (oznacza początek wewnętrznej grupy (na rysunku 14.4 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 wyrażenях regularnych, aby został potraktowany dosłownie, trzeba go poprzedzić odwrotnym ukośnikiem.
-) oznacza koniec pierwszej grupy wewnętrznej (na rysunku 14.4 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 14.4).
- (oznacza początek drugiej grupy wewnętrznej (na rysunku 14.4 została ona oznaczona numerem 3).

- \b odpowiada początkowi słowa.
- Fragment www\ . odpowiada sekwencji znaków www..
-) oznacza koniec drugiej grupy wewnętrznej (na rysunku 14.4 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 14.4 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 ^, a jego końcu znak \$, a ze środka wyrażenia usunąć znak \b. W efekcie uzyskamy wyrażenie w postaci `^(https?:\/\/|www\.)\S*$`.

Dopasowywanie wzorców

Metoda `search()` opisana na stronie 451 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 założymy, 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:

```
// pobranie zawartości wielowierszowego pola tekstowego
var text='moja witryna to www.missingmanuals.com';
// utworzenie wyrażenia regularnego
var urlRegex = /(https?:\/\/|www\.)\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 tekstuowego, 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 464). 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 rzeczą biorąc, `url[0]` zawiera łańcuch znaków "www.missingmanuals.com" (więcej informacji na temat łańcuchów znaków można znaleźć na stronie 72).

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
}
```

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. 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 455):

```
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 ze strony 461 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 = /(https?:\/\/|www\.)\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 75); a zatem, w naszym przykładzie `url[0]` zawiera pierwszy odnaleziony adres URL, a `url[1]` — drugi.

Zastępowanie tekstu

Wyrażeń regularnych można także używać do zastępowania fragmentów łańcuchów znaków. Założmy, że mamy łańcuch znaków zawierający datę zapisaną w postaci 28/10/2011. Chcemy jednak, by na naszej stronie daty były prezentowane w formacie 28.10.2011. Taką zmianę możemy wykonać właśnie przy użyciu metody `replace()`. Oto składnia jej wywołania:

```
lancuch.replace(wyrażenieRegularne, '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='28/10/2008'; // łańcuch znaków
2 var replaceRegex = /\//g // wyrażenie regularne
3 var date = date.replace(replaceRegex, '.'); // zastępujemy /znakiem .
4 alert(date); // 28.10.2008
```

W wierszu 1. tworzona jest zmienna zawierająca datę zapisaną jako '28/10/2008'. W rzeczywistym programie łańcuch ten mógłby zostać podany przez użytkownika w polu formularza. W wierszu 2. tworzony 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 '28.10/2008'. 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 — '28.10.2008' — w okienku informacyjnym.

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 `match()`. Oprócz tego, za jej pomocą można także zapamiętywać podwzorce podane w wyrażeniu regularnym i używać ich podczas zastępowania tekstu. Zgodnie z wyjaśnieniami podanymi w notatce na stronie 458, 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 463 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 wcześniej, na przykład w postaci: 28 10 2008 lub 28-10-2008? Zamiast używać długiego kodu JavaScript sprawdzającego i następującego 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 459. 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 zmiennika. W powyższym przykładzie łańcuch zmiennika 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 zmiennika 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”.

Testowanie wyrażeń regularnych

Przykładowe wyrażenia regularne zostały zamieszczone w pliku `example_regex.txt`, dołączonym do przykładów z tego rozdziału. 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 14.5). 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*.

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 do 6. Język JavaScript udostępnia wiele sposobów posługiwania się liczbami.

Rysunek 14.5. 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

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 71) 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 wiele problemów, gdyż interpreter JavaScriptu zazwyczaj konwertuje łańcuchy na liczby, jeśli zauważa, 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 (3×4) i dzięki temu uzyskamy wynik 12.

Gdybyśmy jednak zastosowali operator dodawania (+), interpreter JavaScriptu nie dokonałby 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 65). 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 następnej stronie).

- 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'.

Uwaga: Najprawdopodobniej zastanawiasz się, co robi liczba 10 w wywoaniu funkcji `parseInt()`. Otóż, język JavaScript może obsługiwać liczby ósemkowe (które, w odróżnieniu do systemu dziesiątkowego używającego cyfr 0 – 9, korzysta tylko z ósmu cyfr: 0 – 7). Umieszczając liczbę 10 w wywoaniu funkcji `parseInt()`, informujemy interpretera JavaScriptu, by wszelkie dane wejściowe potraktował jako liczby w systemie dziesiątkowym. Dzięki temu JavaScript poprawnie zinterpretuje liczbę podaną jako '08' — zostanie ona potraktowana jako liczba dziesiętna. Przykładowo w poniższym fragmencie kodu zmienna `age` przyjmie wartość 0:

```
var age = '08 lat';
age = parseInt(age);
```

Natomiast w poniższym przykładzie zmienna `age` przyjmie wartość 8:

```
var age = '08 lat';
age = parseInt(age,10);
```

Innymi słowy, podczas korzystania z funkcji `parseInt()` zawsze należy podawać 10 jako drugi argument jej wywołania.

- 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 liczba
if (isNaN(x)) {
    //ten fragment nie zostanie wykonany, gdyż x JEST liczbą
} else {
    //ten fragment zostanie wykonany, gdyż x jest liczbą
}
```

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 założymy, że używamy języka JavaScript do dynamicznego wyznaczenia wysokości elementu `<div>` na stronie, w zależności od wysokości okna przeglądarki. Innymi słowy, wysokość elementu `<div>` 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 piksele, musimy taki wynik zaokrąglić do najbliższej liczby całkowitej (na przykład 300).

Liczبę 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.

```
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()` zaokrąga przekazaną wartość w górę (na przykład wywołanie `Math.ceil(4.0001)` zwróci 5), natomiast metoda `Math.floor()` zaokrąga w dół (na przykład wywołanie `Math.floor(4.9999)` zwróci 4). Nazwy tych metod podchodzą od angielskich słów *ceiling* — sufit, a *floor* — podłoga. Łatwo więc sobie wyobrazić, że pierwsza z nich — `Math.ceil()` — zaokrąga 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 10,00 oraz 8.9 zamiast 8,90).

Na szczęście, dostępna jest metoda o nazwie `toFixed()`, pozwalająca konwertować liczby nałańcuchy 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 włańcucha 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 pomocąłańcuch znaków, zostanie zgłoszony błąd:

```
var cost='10'; // lańcuch znaków
var printCost=cost.toFixed(2) + ' zł'; // błąd
```

Aby ominąć ten problem, wystarczy skonwertowaćłańcuch znaków na liczbę przy użyciu jednego ze sposobów opisanych na stronie 465, na przykład:

```
var cost='10'; // lańcuch 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. Założymy, że dysponujemy tablicą pytań quizowych (takich jak w przykładowym programie przedstawionym na stronie 118). 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 do 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 zaokrąglą 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 75, 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 użytkowych fragmentów kodu, nadających się do wielokrotnego stosowania (patrz strona 110). 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 110) 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 określić 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 14.3). Aby pobrać aktualny rok, można użyć następującego fragmentu kodu:

```
var now = new Date();
var year = now.getFullYear();
```

Tabela 14.3. Metody służące do pobierania informacji z obiektu Date

Metoda	Zwraca
getFullYear()	rok; na przykład 2008.
getMonth()	miesiąc wyrażony jako liczba z zakresu od 0 do 11: 0 odpowiada styczniovi, 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 liczbę 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 473).

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 75). 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 słoniem w czasie dnia lub z księżycem w nocy.

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 środa 1 lutego 2012 roku jest reprezentowana jako liczba 131328083200000.

Nie, to nie żart. Z punktu widzenia interpretera JavaScriptu 1 stycznia 1970 roku to początek czasu. Ta data (nazywana także „epoką Unixa”) 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.

Z każdym razem, gdy używamy jakiejś metody obiektu Date, takiej jak `getFullYear()`, interpreter JavaScriptu wykonuje obliczenia matematyczne (bazujące na liczbie sekund, jaką 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 od 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(2013,0,1);
// 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 zauważyc inny sposób tworzenia obiektu daty: `new Date(2013,0,1)`. Więcej informacji na jego temat można znaleźć na stronie 476.

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 lub 18:23:42. W dobie zegarków elektronicznych sporo osób może być jednak przyzwyczajonych do wyrażania czasu w formie 12-godzinnej, gdzie zapis 10:34 p.m. odpowiada godzinie 22:34. Takie osoby zapewne ucieszą się na wieść, że stosunkowo łatwo można zmodyfikować powyższy skrypt i dostosować go do prezentowania czasu w zapisie 12-godzinnym.

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 po południu (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  if (hour < 12) {
4      meridiem = 'am';
5  } else {
6      meridiem = 'pm';
7  }
8  hour = hour % 12;
9  if (hour==0) {
10     hour = 12;
11 }
12 hour = hour + ' ' + meridiem;
```

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 połnocy odpowiadają 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 pozostała 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 wyświetcone 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 466, 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: 7:03:02 p.m. lub 4:09:02 a.m bądź nawet całkowicie pomija sekundy i zwraca czas w postaci 7:23 p.m.:

```
function printTime(secs) {
    var sep = ':'; //znak separatorka
    var hours,minutes,seconds,time;
    var now = new Date();
    hours = now.getHours();
    if (hours < 12) {
        meridiem = 'am';
    } else {
        meridiem = '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 + ' ' + meridiem;
}
```

Kod tej funkcji można znaleźć przykładach do książki, w pliku *printTime.js* umieszczonym w katalogu *R14*. 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 *printTime.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 *printTime()*. Jeśli chcemy wyświetlać także sekundy, możemy wywołać funkcję w następujący sposób: *printTime(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?” (patrz ramka na stronie 473).

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 2012 roku, trzeba użyć następującego wywołania:

```
var ny2012 = new Date(2012,0,1,0,0,0);
```

Taki kod zostanie zinterpretowany w następujący sposób: „utwórz nowy obiekt `Date` reprezentujący 1 stycznia 2012 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 2012 roku, możemy użyć wywołania w postaci:

```
var ny2012 = new Date(2012,0,1);
```

Uwaga: Koniecznie trzeba pamiętać, że w JavaScriptie styczniovi odpowiada wartość 0, lutemu — 1 i tak dalej; informacje na ten temat można znaleźć na stronie 471.

Tworzenie daty z zadaną liczbą dni w przód

Zgodnie z informacjami podanymi w ramce na stronie 473, interpreter JavaScriptu traktuje daty jako liczbę milisekund, która upłynęła od 1 stycznia 1970 roku. Kolejnym sposobem utworzenia obiektu `Date` jest przekazanie w wywołaniu metody `Date()` liczby milisekund:

```
new Date(milisekundy);
```

A zatem, datę 1 stycznia 2012 roku można także uzyskać w następujący sposób:

```
var ny2012 = new Date(1325404800000);
```

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.

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 uplynęły od 1 stycznia 1970 roku do dziś. W wierszu 3. obliczamy, 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.

Łączenie różnych elementów

Wiesz już, że język JavaScript umożliwia wykonywanie wielu zadań. Niektóre z nich to: validacja formularzy, dodawanie efektu podmiany rysunków, tworzenie galerii zdjęć lub wz bogacanie interfejsu użytkownika (na przykład paneli z zakładkami i efektu accordion). 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 40 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 znajął się kod JavaScript biblioteki jQuery. Także wtyczki są udostępniane jako zewnętrzne pliki, dlatego trzeba je dołączyć do strony, aby móc ich używać. Dołączanie zewnętrznych plików JavaScript jest bardzo proste:

```
<script type="text/javascript" src="js/ui.tabs.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 291. 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 293, można użyć w dowolnym formularzu, dlatego warto umieścić ją w odrębnym pliku. To samo dotyczy każdego kodu używanego w wielu miejscach. Na stronie 286 dowiedziałeś się, jak za pomocą kodu JavaScript aktywować pierwsze pole formularza. Tę technikę możesz zastosować w każdym formularzu. Podobnie rzeczą się ma z metodą opisaną w ramce na stronie 284. Rozwiążanie to zapobiega wielokrotnemu przesyłaniu 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().
```

Zauważ, że ponieważ kod oparto na bibliotece jQuery, należy umieścić go w funkcji `$(document).ready()` (wiersze 1. i 11.). Każdy plik zewnętrzny korzystający z tej biblioteki należy rozpocząć od kodu podanego w wierszu 1., a zakończyć znakami z wiersza 11. z tego fragmentu.

Uwaga: Biblioteka jQuery potrafi obsłużyć wiele funkcji `$(document).ready()`. Dlatego możesz utworzyć kilka zewnętrznych plików JavaScript do wykonywania różnych operacji na stronie i umieścić tę funkcję w każdym z nich, jak również użyć jej w znaczniku `<script>` na danej stronie. JQuery poprawnie wykona taki kod.

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 umieszczisz pole wyszukiwania, a poniżej odrębny formularz do składania zamówień, kod z wiersza 3. aktywuje kontrolkę do wyszukiwania, a nie pierwsze pole tekstowe formularza zamówień. 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. Założmy, 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 219), a także użyć języka JavaScript do wyświetlania odnośników do stron zewnętrznych w nowym oknie (patrz strona 252). 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 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. Założmy, 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 155):

```
$(‘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 używany 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):

```
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).

Operator trójargumentowy

Często wykonywaną operacją programistyczną jest przypisywanie wartości do zmiennej na podstawie określonego warunku. Założmy, 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';
}
```

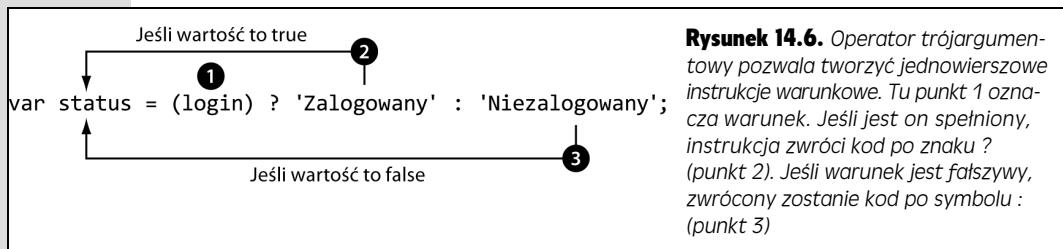
Tu prosta instrukcja warunkowa (patrz strona 91) ustawia wartość zmiennej `status` w zależności od tego, czy użytkownik jest zalogowany, czy nie. JavaScript udostępnia skrótny zapis tej często używanej konstrukcji. Jest to tak zwany *operator trójargumentowy*. Umożliwia on tworzenie prostych instrukcji 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. Rysunek 14.6 ilustruje jego działanie.



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 będą często przeglądał.

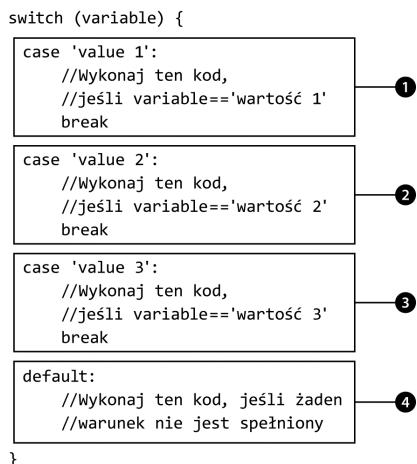
Instrukcja Switch

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.

Założmy, 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 dziesięciu wierszach kodu. Jeśli chcesz wielokrotnie sprawdzić wartość danej zmiennej, bardziej eleganckie (i czytelne) będzie użycie instrukcji `switch`. Jej podstawową strukturę przedstawia rysunek 14.7.



Rysunek 14.7. 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 14.7 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 dwukropki. 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;. Jeśli ją pominiesz, interpreter przejdzie do sprawdzania następnych warunków.

Brak instrukcji break; jest problemem, zwłaszcza jeśli w instrukcji switch użyjesz końcowego słowa default (numer 4 na rysunku 14.7). Skrypt wykona polecenia zapisane po tym słowie, jeśli żaden z poprzednich warunków nie jest spełniony (jest to odpowiednik ostatniej klauzuli else w instrukcjach warunkowych). Jeśli pominiesz instrukcję break; w warunku, który ma wartość true, interpreter uruchomi także kod po słowie default.

Oto jak za pomocą instrukcji switch można przekształcić kod instrukcji if-else ze strony 483:

```
switch (favoriteColor) {  
    case 'niebieski':  
        message = 'Niebieski to zimny kolor.';  
        break;  
    case 'czarny':  
        message = 'Czarny 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 zwięzły i czytelny.

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 'czarny':  
        message = 'Czarny 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.

Tworzenie kodu JavaScript o krótkim czasie wczytywania

Kiedy zaczynasz umieszczać skrypty w zewnętrznych plikach JavaScript, odwiedzający odczuja 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*) nigdy nie są umieszczane w pamięci podręcznej. Dlatego jeśli użytkownik otwiera stronę za pomocą protokołu <https://> (na przykład przez wpisanie adresu <https://www.sawmac.com>), pobrane dokumenty — w tym zewnętrzne pliki JavaScript — trzeba wczytywać za każdym razem, kiedy będą potrzebne.

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 w tym rozdziale opisano narzędzie używane w witrynie Yahoo!, 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://developer.yahoo.com/yui/compressor>. W poprzednim wydaniu książki zamieściliśmy instrukcję jego stosowania, która mogła się spodobać jedynie prawdziwemu komputerowemu hackerowi; wymagała pobrania pliku JAR (napisanego w Java) oraz korzystania z przerząjącego wiersza polecen systemu operacyjnego. Na szczęście, jakiś przedsiębiorczy i usłużny programista utworzył internetową wersję tego narzędzia, która pozwala na wklejenie zminimalizowanego kodu lub nawet przesłanie pliku JavaScript z lokalnego komputera.

1. Uruchom przeglądarkę i wyświetl w niej stronę <http://www.refresh-sf.com/yui/>.

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.

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 nieoczekiwanej 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.

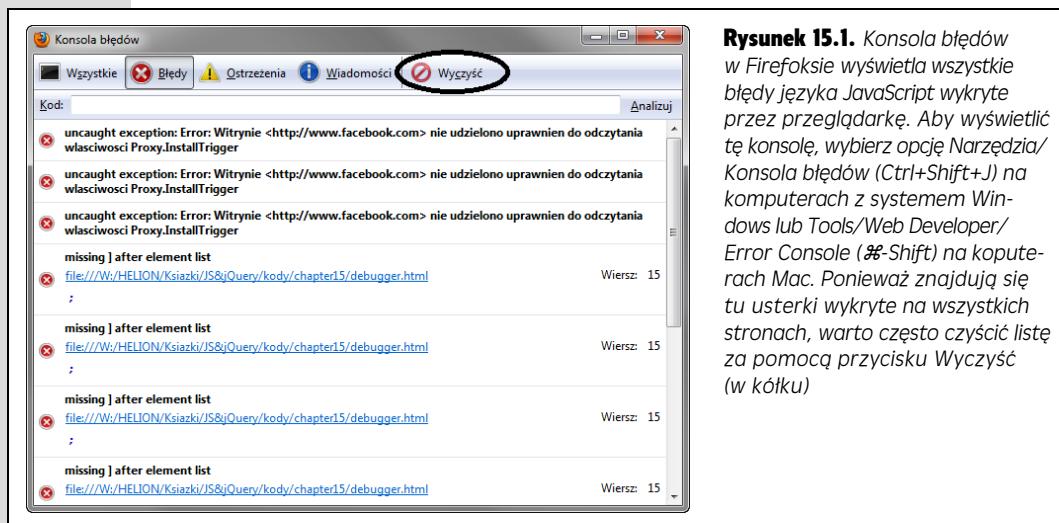
Najczęstsze błędy w kodzie JavaScript

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, cudzysłówów 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 zamkającego znaku specjalnego. Na przykład instrukcja `alert('witaj');` jest nieprawidłowa, ponieważ brakuje w niej końcowego nawiasu. Poprawny zapis to `alert('witaj');`.

Pominięcie zamkającego nawiasu to błąd składniowy (patrz ramka na stronie 491). 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 48) pojawia się komunikat o błędzie typu „Missing) after argument list” [czyli brak) po liście argumentów]. Internet Explorer 9 (patrz strona 50) wyświetla informację w stylu „Oczekiwano znaku)”. Konsola błędów przeglądarki Chrome pokazuje komunikat „Uncaught SyntaxError: Unexpected token ;”, a konsola Safari (patrz strona 51) udostępnia najmniej przydatny komunikat typu „SyntaxError: Parse Error”. Jak dowiedziałeś się na stronie 51, Firefox zwykle wyświetla najbardziej zrozumiałe komunikaty o błędach, dlatego warto rozpocząć analizę problemów właśnie od tej przeglądarki (patrz rysunek 15.1).



Rysunek 15.1. Konsola błędów w Firefoksie wyświetla wszystkie błędy języka JavaScript wykryte przez przeglądarkę. Aby wyświetlić tę konsolę, wybierz opcję Narzędzia/Konsola błędów (Ctrl+Shift+J) na komputerach z systemem Windows lub Tools/Web Developer/Error Console (⌘-Shift) na komputerach Mac. Ponieważ znajdują się tu usterki wykryte na wszystkich stronach, warto często czyścić listę za pomocą przycisku Wyczyszczyć (w kółku)

Błąd składniowy w instrukcji `alert('witaj');` jest dobrze widoczny. Jednak jeśli w kodzie znajdują się zagnieżdżone nawiasy, łatwo jest pominąć zamkajający znak tego typu, natomiast trudno szybko dostrzec taką pomyłkę, na przykład:

```
if ((x>0) && (y<10) {  
    // Różne operacje.  
}
```

W tym fragmencie brakuje ostatniego zamkającego nawiasu w instrukcji warunkowej (po wyrażeniu `(y<10)`). Pierwszy wiersz powinien wyglądać następująco: `if ((x>0) && (y<10)) {`. Firefox udostępnia najbardziej zrozumiałą opis tego problemu:

„Missing) after condition” (czyli brak) po warunku). W tabeli 15.1 znajdziesz listę komunikatów o błędach składniowych, wyświetlanych przez konsolę błędów przeglądarki Firefox.

Tabela 15.1. Konsola błędów przeglądarki Firefox (patrz strona 48) udostępnia najbardziej zrozumiałą opis błędów składniowych. Jeśli skrypt nie działa, uruchom go w Firefoksie i wyświetl konsolę błędów. Oto kilka najczęstszych komunikatów o błędach składniowych i ich opis

Komunikat o błędzie	Wyjaśnienie
Unterminated string literal	Brak otwierającego lub zamk傢acego cudzysłowu: <code>var name = Janek';</code> Ten błąd pojawia się też przy braku dopasowania ograniczników: <code>var name = 'Janek";</code>
Missing) after argument list	Brak zamk傢acego nawiasu w wywoaniu funkcji lub metody: <code>alert('witaj' ;</code>
Missing) after condition	Brak zamk傢acego nawiasu w instrukcji warunkowej: <code>if (x==0</code>
Missing (before condition	Brak otwierającego nawiasu w instrukcji warunkowej: <code>if x==0)</code>
Missing } in compound statement	Brak zamk傢acego nawiasu klamrowego w instrukcji warunkowej: <code>if (score == 0) { alert('Koniec gry'); // Brak znaku } w tym wierszu.</code>
Missing } after property list	Brak zamk傢acego nawiasu klamrowego w obiekcie JavaScript: <code>var x = { fName: 'Robert', lName: 'Kowalski' // Brak znaku } w tym wierszu.</code>
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 cudzysłówów: <code>var message='Mike'u, tu wystąpił błąd.';</code>
Missing variable name	Wynika z próby użycia zarezerwowanego słowa języka JavaScript (patrz strona 61) jako nazwy zmiennej: <code>var if="Błąd składniowy.";</code>

Błąd składniowy wystąpi także wtedy, jeśli zapomnisz podać drugi cudzysłów lub apostrof. Na przykład 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”. Przeglądarka Safari ponownie udostępnii nieprzydatne informacje: „SyntaxError: Parse Error”.

Także nawiasy klamrowe występują w parach. Są one potrzebne w instrukcjach warunkowych (patrz strona 91), pętlach (patrz strona 104), przy tworzeniu literałów obiektowych języka JavaScript (patrz strona 157) i w obiektach JSON (patrz strona 386):

```
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. Założymy, że chcesz dodać następujący fragment:

```
if ((name=='robert') && (score==0)) {  
    alert('Przegrałeś, ale przynajmniej masz piękne imię');
```

Najpierw wpisz zewnętrzne elementy, aby utworzyć szkielet instrukcji warunkowej:

```
if ( ) {  
}
```

Na tym etapie prawie nie ma kodu, dlatego łatwo jest zauważyc, czy wpisano wszystkie znaki specjalne. Następnie dodawaj krok po kroku dalszy kod do czasu utworzenia całego programu. W ten sam sposób warto tworzyć złożone literaly obiektowe języka JavaScript, używane na przykład do ustawiania opcji wtyczki Validation (patrz strona 291) lub tworzenia obiektów JSON (patrz strona 386). Zacznię 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.

WIEDZA W PIŁUŁCE

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 cudzysłówów. 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. Założymy, 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). Na stronie 493 dowiedziałeś się, że język JavaScript uwzględnia wielkość znaków, dla tego 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 182).

- ◆ **Błędy logiczne.** Czasem skrypt działa, ale w nieoczekiwany sposób. W kodzie może znajdować się instrukcja `if-else` (patrz strona 91), 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 63). 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++) {
    // 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++`), 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 107).

Błędy logiczne są zwykle najtrudniejsze do wykrycia. Jednak dzięki technikom diagnostycznym, opisanych na stronie 496, rozwiążesz niemal każdy problem, jaki napotkasz.

Cudzysłowy i apostrofy

Początkujący programiści często mają problemy z cudzysłowami i apostrofami. Symbole te służą do tworzeniałańcuchów liter i innych znaków (patrz strona 57). Takich ciągów można używać jako komunikatów na stronach lub zmiennych w programach. JavaScript, podobnie jak inne języki programowania, umożliwia tworzeniełańcuchów znaków za pomocą cudzysłowów i apostrofów. Poniższa instrukcja:

```
var name="Janek";
```

oznacza to samo co następuña:

```
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 58, 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 włańcuchach znaków. Bardzo łatwo jest popełnić następujący błąd:

```
var message='Mike'u, tu kryje się błąd.';
```

Zwróc 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 błędą. 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 łączaniu apostrofów i cudzysłowów. Możesz otoczyć cudzysłowami łańcuch znaków z apostrofami lub na odwrotnie. 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 58, a tu znajdziesz krótkie przypomnienie. Aby utworzyć sekwencję ucieczki, poprzedź dany znak specjalnym 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 61 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 Firefox, Safari oraz Opera 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 i zastąpienie go łańcuchem znaków. Z kolei przeglądarki Chrome oraz Internet Explorer 9 wygenerują błąd, lecz nie wyświetla żadnego okienka informacyjnego na jego temat.

Pojedynczy znak równości w instrukcjach warunkowych

Instrukcje warunkowe (patrz strona 91) 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 jest utworzyć instrukcję, w której warunek jest zawsze spełniony:

```
if (score=0) {  
    alert('Koniec gry');  
}
```

Ten kod ma sprawdzać wartość zmiennej `score`. Jeśli wynosi ona 0, powinno pojawić się okno dialogowe z wiadomością „Koniec gry”. 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=0` przypisze wartość 0 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 0.

Aby uniknąć tego błędu, należy zawsze używać dwóch znaków równości przy sprawdzaniu, czy dwie wartości są takie same:

```
if (score==0) {  
    alert('Koniec gry');  
}
```

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, `getElementsByName()` 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 zobaczyisz 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 40 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 na stronie 41, 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).

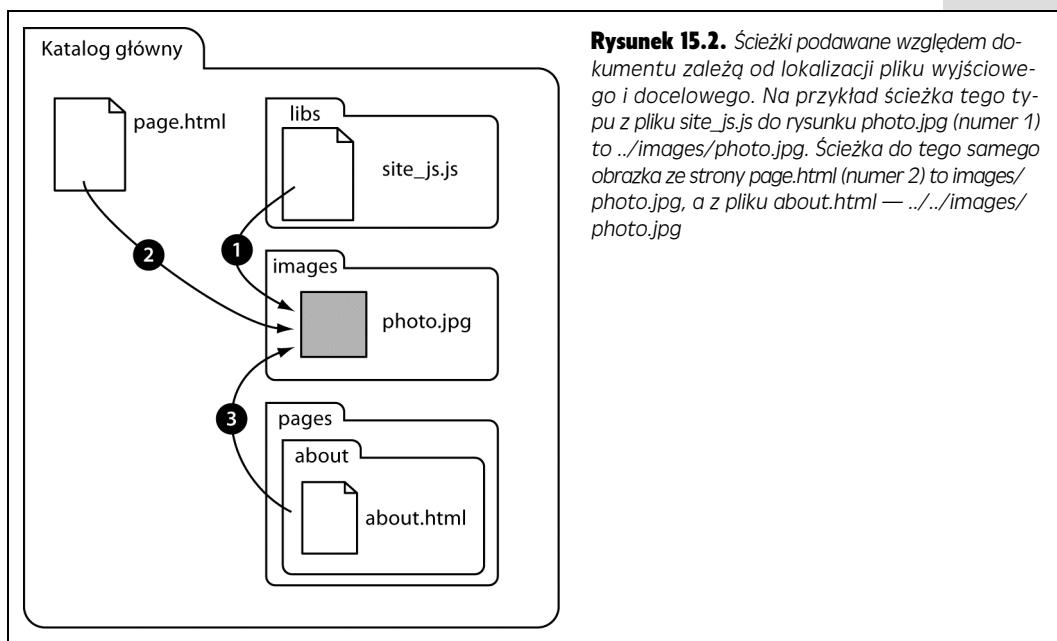
Więcej informacji o ścieżkach znajdziesz na stronie 41. Jeśli używasz zewnętrznych plików JavaScript i odkryjesz, że skrypt nie działa, dokładnie sprawdź, czy wpisałeś poprawne ścieżki.

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 132).

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 15.2 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



Rysunek 15.2. Ścieżki podawane względem dokumentu zależą od lokalizacji pliku 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`

(`site_js.js` w katalogu `libs`) i rysunku (`photo.jpg` w katalogu `images`). Założymy, że w pliku `site_js.js` znajduje się ścieżka do pliku `photo.jpg`, potrzebna na przykład do wstępnego pobrania obrazka (patrz strona 221) lub dynamicznego wyświetlenia go na stronie.

W pliku `site_js.js` ścieżka do rysunku `photo.jpg` podana względem dokumentu to `../images/photo.jpg` (numer 1 na rysunku 15.2). 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 15.2) 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 15.2).

Istnieje kilka sposobów na rozwiązywanie tego problemu. Przede wszystkim możliwe jest, że nigdy nie natrafisz na taką sytuację, ponieważ nie będziesz umieszczał w plikach JavaScript ścieżek do innych dokumentów. Jednak jeśli chcesz stosować tę technikę, powinieneś używać ścieżek podawanych względem katalogu głównego (patrz strona 41), 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 (patrz strona 40) i zdefiniować zmienną przechowującą ścieżkę do odpowiedniego pliku podaną względem dokumentu (danej strony).

Możesz też zastosować podejście użyte w pokazie slajdów ze strony 234. 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 zajrzesz 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 116. 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 182 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.  
});
```

Diagnozowanie przy użyciu dodatku Firebug

Dodatek Firebug to jedno z najlepszych narzędzi dla projektantów stron WWW. Jest bezpłatny, łatwy w instalacji i użyciu oraz pomaga poprawić kod HTML, CSS i JavaScript. Firebug to dodatek do Firefoksa, który obejmuje zestaw przydatnych narzędzi diagnostycznych. Narzędzie to pozwala zbadać kod HTML, CSS i — co najważniejsze w tej książce — JavaScript.

WIEDZA W PIŁUŁCE

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. Dwie najważniejsze techniki zapobiegania usterkom to:

- ◆ **Tworzenie skryptów w krótkich fragmentach.** Jak już prawdopodobnie zauważłeś, programy JavaScript bywają mało czytelne z uwagi na znaki {}, , ', instrukcje if, else, funkcje i tak dalej. Nie próbuj tworzyć 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.

Założmy, że chcesz obok pola na komentarz wyświetlić liczbę wpisanych w nim liter. Rozwiązań 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.

Möżesz 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. Polecenie alert() lub funkcja console.log() dodatku Firebug (jej opis znajdziesz na następnej stronie) pomagają zobaczyć efekt wystąpienia zdarzenia keydown. Jeśli pierwszy fragment działa, można przejść do etapu 2., przetestować kod i tak dalej.

Wraz z nabyciem 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 7, 8 oraz 9, Firefoksie 5 i 6, najnowszych wersjach przeglądarek Chrome oraz Safari oraz w innych aplikacjach, których mogą używać osoby odwiedzające daną witrynę.

Instalowanie i włączanie dodatku Firebug

Ten dodatek znajdziesz na stronie www.getfirebug.com bądź też na witrynie Mozilla Add-Ons. Aby go zainstalować, wykonaj następujące operacje:

1. Odwiedź stronę <http://addons.mozilla.org/firefox/addon/firebug> przy użyciu Firefoksa i kliknij przycisk **Zainstalu.**

Aby ustrzec użytkowników przed instalowaniem złośliwego i niebezpiecznego oprogramowania, Firefox blokuje pierwszą próbę zainstalowania dodatku. Na ekranie zostanie wyświetcone okno *Software Installation*, ostrzegające przed niebezpieczeństwem, jakie niesie złośliwe oprogramowanie (och, doprawdy?). Nie obawiaj się jednak — Firebug jest całkowicie bezpieczny.

2. Ponownie kliknij przycisk **Zainstalu.**

W efekcie Firefox zainstaluje rozszerzenie, jednak nie będzie ono działało aż do momentu ponownego uruchomienia przeglądarki. Na ekranie zostanie wyświetcone niewielkie okienko dialogowe wyjaśniające, w czym rzecz.

3. Kliknij przycisk **Uruchom ponownie teraz.**

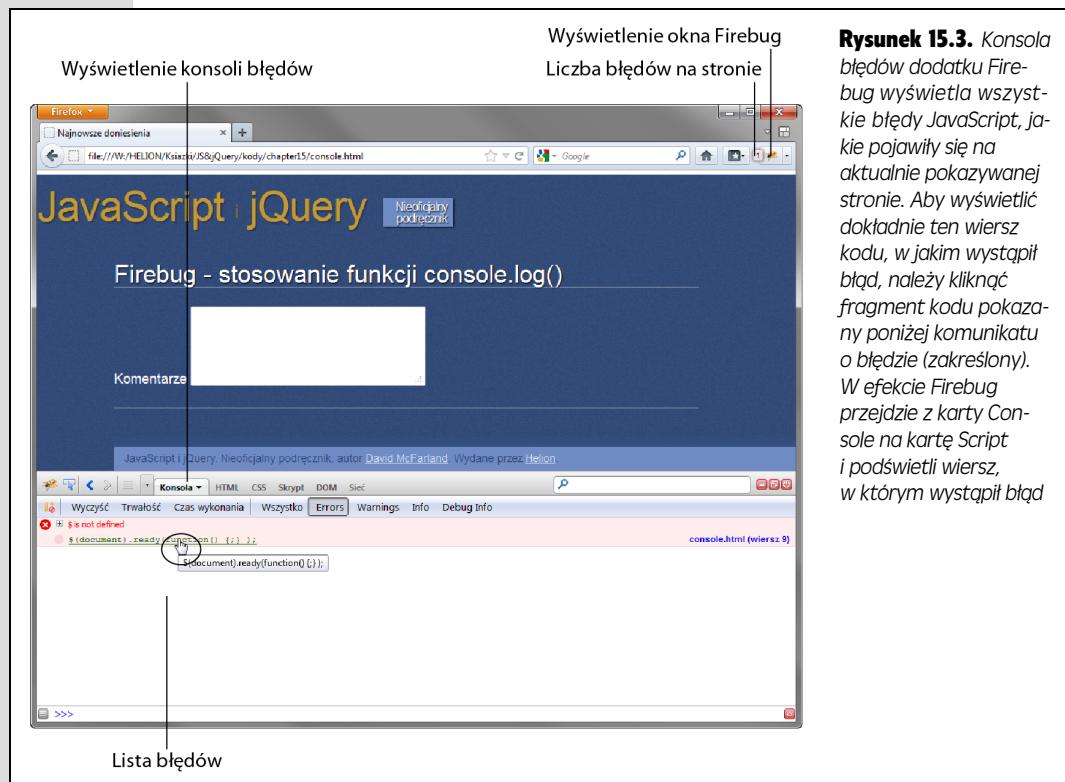
No i proszę — Firebug został zainstalowany i jest już gotowy do użycia. Najpierw jednak trzeba go otworzyć.

4. Wybierz opcję *Narzędzia/Dla twórców witryn/Firebug* lub naciśnij klawisz F12; na komputerach Mac wybierz opcję *Tools/Web Developer/Firebug/Enable Firebug*.

Teraz możesz już zacząć korzystać z dodatku Firebug przy diagnozowaniu i poprawianiu swoich skryptów.

Przeglądanie błędów za pomocą dodatku Firebug

Firebug udostępnia łatwiejszy i bardziej przydatny sposób przeglądania błędów niż wbudowana konsola błędów przeglądarki Firefox. Kiedy korzystasz z Firebuga i wczytasz stronę z błędami w kodzie JavaScript, w prawym górnym rogu przy przycisku dodatku pojawi się liczba znalezionych usterek (patrz rysunek 15.3). Kliknij przycisk z ikoną dodatku Firebug, aby otworzyć jego konsolę z listą wszystkich błędów w skryptach JavaScript.



Błędy widoczne w konsoli dodatku są takie same jak w konsoli błędów przeglądarki Firefox (patrz rysunek 15.1), ale Firebug wyświetla tylko problemy wykryte na aktualnie otwartej stronie (konsola błędów przeglądarki zawiera listę wszystkich usterek z każdej odwiedzonej strony). Ponieważ Firebug zapewnia bardzo łatwy dostęp do informacji o błędach, prawdopodobnie zrezygnujesz z konsoli błędów Firefoksa na rzecz tego dodatku.

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 43). 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 uruchomisz pętlę 20 razy, będziesz musiał zamknąć naprawdę wiele okienek.

Firebug udostępnia lepszy sposób obserwacji programu. Konsola dodatku Firebug 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 45), 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, zgodnie z informacjami zamieszczonymi w rozdziale 1.

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);
```

Mögesz też poprzedzić wartość zmiennej informacją:

```
console.log('Nazwa użytkownika: %s', name);
```

Oznacza to, że najpierw należy przekazać do funkcji `log()` łańcuch znaków, następnie przecinek i nazwę zmiennej, której wartość chcesz wyświetlić. Sekwencja `%s` oznacza: „Wstaw zamiast mnie wartość zmiennej”. Skrypt zastępuje więc ciąg `%s` wartością zmiennej `name`, dlatego w konsoli pojawi się komunikat typu „Nazwa użytkownika: Robert”.

Mögna w ten sposób wyświetlić więcej niż jedną zmienną. Wystarczy użyć dla każdej z nich sekwencji `%s`. Założmy, że chcesz wyświetlić wartości dwóch zmiennych, `name` i `score`, wraz z niestandardową wiadomością. Należy to zrobić w następujący sposób:

```
console.log('Wynik gracza %s to %s', name, score);
```

Przy użyciu sekwencji %s można wyświetlać także wartości liczbowe, jednak Firebug udostępnia dwa dodatkowe ciągi. %d reprezentuje liczby całkowite (na przykład 1, 2 i 10), a %f — liczby zmiennoprzecinkowe (na przykład 1.22 i 2.4444). Użycie jednej z tych sekwencji spowoduje wyświetlenie liczb w innym kolorze, co ułatwia odróżnienie ich od reszty tekstu.

Możesz zmodyfikować wcześniejszy wiersz, aby wyświetlić liczbę zapisaną w zmiennej score:

```
console.log('Wynik gracza %s to %d', name, 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(). Przeglądarki, które nie obsługują metody console.log() (takie jak wcześniejsze wersje Internet Explorera), po napotkaniu wywołania log() wygenerują błąd.

Przykład — korzystanie z konsoli dodatku Firebug

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 43.

Najpierw zainstaluj dodatek Firebug w Firefoksie, stosując się do instrukcji ze strony 497. Pamiętaj, że Firebug działa tylko w przeglądarce Firefox.

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 zamknięty 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 161. Sprawia ona, że przeglądarka wczytuje cały kod strony przed uruchomieniem programu JavaScript. Najpierw użyj funkcji log() Firebuga do wyświetlenia komunikatu o uruchomieniu funkcji .ready().

3. Dodaj do skryptu kod wyróżniony pogrubieniem:

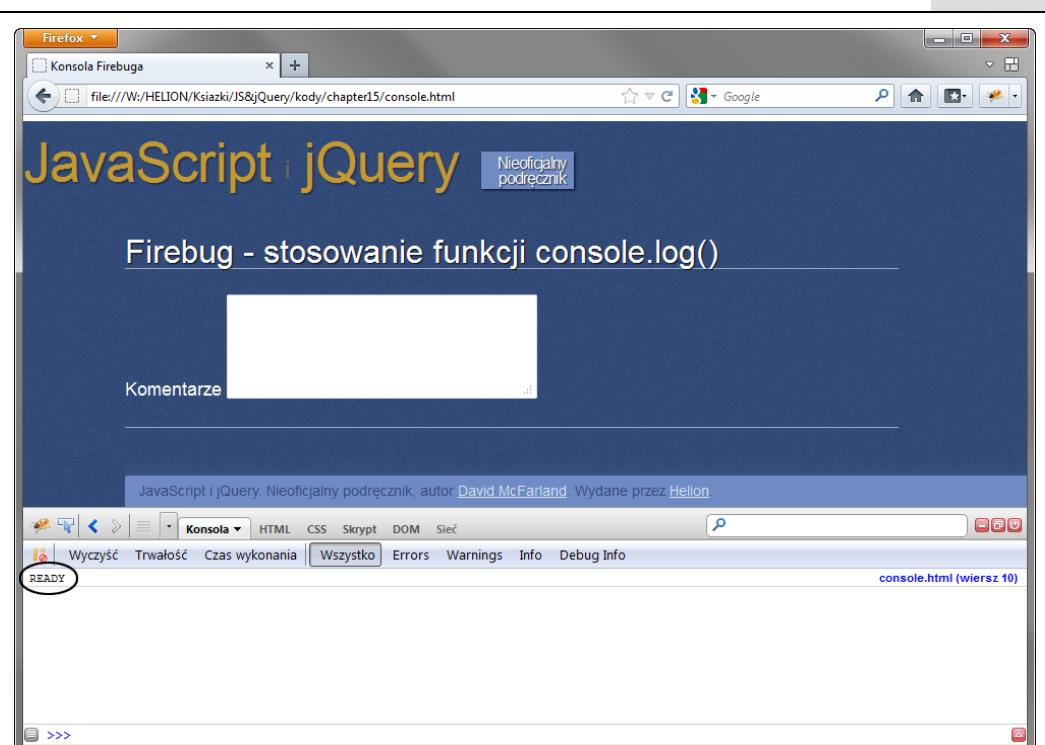
```
<script>
$(document).ready(function() {
    console.log('READY');
}); // koniec funkcji ready
</script>
```

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()`), Firebug zapisze tekst „READY” 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 zadeemonstrować działanie funkcji `log()`. Do rozwijanej strony dodasz później wiele wywołań tej funkcji, aby dobrze ją poznać.

4. Zapisz plik i otwórz go w Firefoksie. Jeśli okno Firebug nie jest widoczne, wyświetl je, klikając przycisk umieszczony na pasku narzędzi przeglądarki.

Aby wyświetlić konsolę Firebuga, możesz też wybrać opcję *Narzędzia/Dla twórców witryn/Firebug/Otwórz*.

W konsoli powinno pojawić się słowo *READY* (wyróżnione kółkiem na rysunku 15.4). Rozwijany skrypt ma wyświetlać liczby znaków w polu formularza. Wartość ta ma się zmieniać po każdym wprowadzonym znaku. Aby uzyskać ten efekt, należy dodać do pola tekstuowego zdarzenie `keyup` (patrz strona 174). Na każdym etapie rozwijania skryptu dodasz funkcję `console.log()`, aby kontrolować, co dzieje się w programie.



Rysunek 15.4. Jeśli z lewej strony ikony Firebuga na pasku narzędzi przeglądarki zostanie wyświetlona jakaś liczba, oznacza to, że na stronie pojawił się błąd (może zwyczajna literówka). Aby go wyświetlić w konsoli Firebuga, należy kliknąć ikonę dodatku

5. Po wierszu dodanym w kroku 3. wpisz poniższy kod:

```
$('#comments').keyup(function() {  
    console.log('Zdarzenie: keyup');  
}); // koniec funkcji keyup
```

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 zdarzeń znajdziesz na stronie 174). Wywołana tu funkcja console.log() wyświetla w konsoli Firebug 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 Firefoksie i wpisz kilka znaków w polu tekstowym. W konsoli Firebuga powinieneś zobaczyć kilka wierszy z tekstem „Zdarzenie: keyup” (po jednym dla każdego wpisanego znaku). 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 wpisany w kroku 5.:

```
1 $('#comments').keyup(function() {  
2     console.log('Zdarzenie: keyup');  
3     var text = $(this).val();  
4     console.log('Treść komentarza: %s', 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 275). 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 Firefoksie 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: %s', text);  
5     var chars = text.length;  
6     console.log('Liczba znaków: %d', chars);  
7 }); // koniec funkcji keyup
```

Wiersz 5. sprawdza liczbę znaków zapisanych w zmiennej text (właściwość length omówiono na stronie 446) 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.). Ponieważ zmienna `chars` przechowuje liczbę, użyto sekwencji `%d`, aby wyświetlić liczbę całkowitą. 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: %s', text);
8     var chars = text.length;
9     console.log('Liczba znaków: %d', chars);
10    $('#count').text("Liczba znaków: " + chars);
11  });
12}); // koniec funkcji keyup
13}); // koniec funkcji ready
14</script>
```

10. Zapisz plik i wyświetl go w Firefoksie.

Otwórz Firebug. Strona i konsola powinny wyglądać jak te z rysunku 15.5. Gotową wersję rozwiązania zawiera plik `complete_console.html` w katalogu `R15` 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.

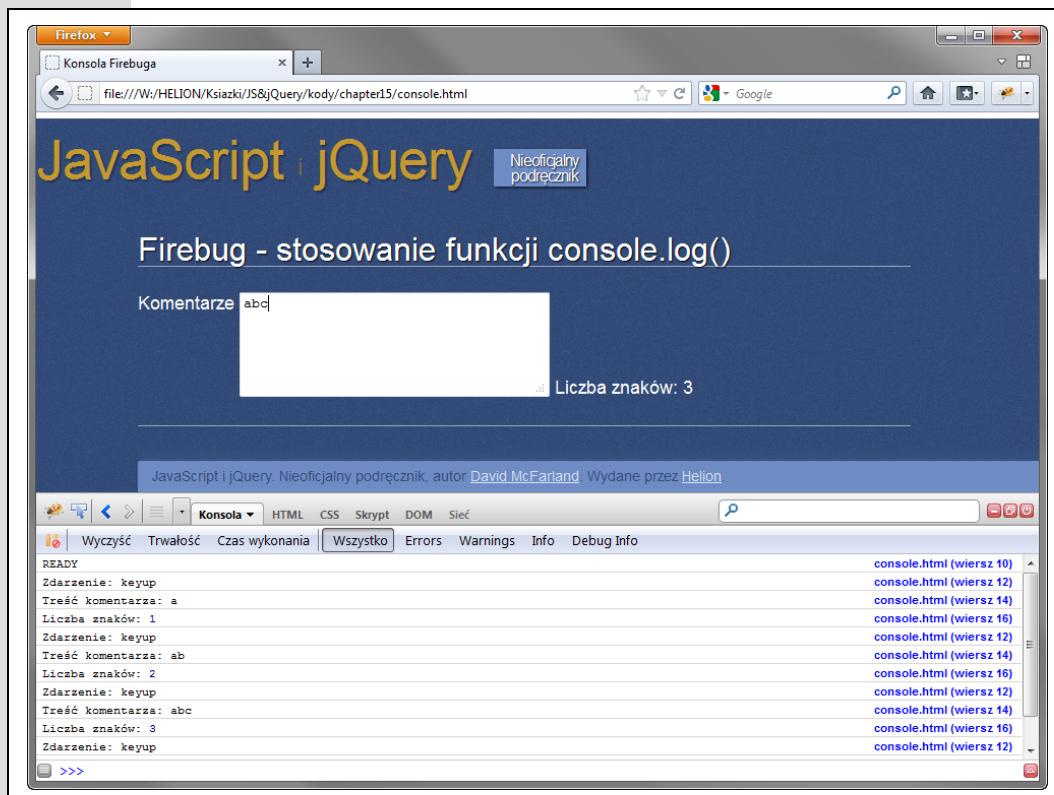
Diagnozowanie zaawansowane

Konsola fragment to doskonałe narzędzie do wyświetlania komunikatów z informacjami o funkcjonowaniu programu. Jednak czasem skrypt działa tak szybko, że trudno jest zauważyc, jakie operacje zachodzą na poszczególnych etapach. Trzeba wtedy spowolnić program. Na szczęście Firebug udostępnia rozbudowany debugger skryptów JavaScript. Narzędzie to pozwala przejść po programie wiersz po wierszu i zobaczyć, co dzieje się na każdym etapie skryptu.

Uwaga: Debuggery JavaScriptu o podobnych możliwościach dostępne są także w innych przeglądarkach, takich jak Chrome, Opera, Safari oraz Internet Explorer 9.

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 Firebuga, które pozwalają uruchomić program wiersz po wierszu. W ten sposób możesz dokładnie prześledzić, jak działają poszczególne instrukcje. Proces korzystania z debuggera przebiega następująco:



Rysunek 15.5. Konsola dodatku Firebug 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()`. Obie funkcje działają także w konsolach przeglądarek Safari oraz Chrome, nie działają natomiast w przeglądarkach Opera oraz Internet Explorer 9

1. Otwórz stronę w Firefoksie.

Najpierw należy zainstalować i włączyć dodatek Firebug, co opisano na stronie 477.

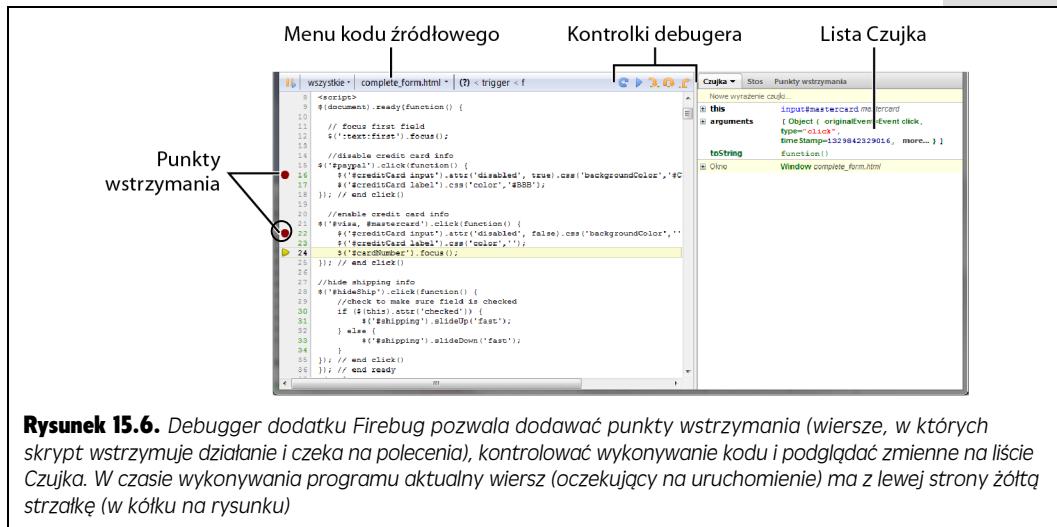
2. Otwórz dodatek Firebug.

Kliknij ikonę Firebuga (przycisk jest umieszczony na pasku narzędzi przeglądarki, z jego prawej strony). Możesz też wcisnąć klawisz F12 (tylko na komputerach z systemem Windows) lub wybrać opcję *Narzędzia/Dla twórców witryny/Firebug/Otwórz* (w systemie Windows), lub *Tools/Web Developer/Firebug/Enable Firebug* (na komputerach Mac).

Wskazówka: Jeśli nie chcesz, aby strona pojawiała się w wąskim obszarze bezpośrednio nad Firebugiem, wybierz opcję *Narzędzia/Dla twórców witryny/Firebug/Firebug UI Location/Odlączony* (w systemie Windows) lub *Tools/Web Developer/Firebug/"Open Firebug in New Window"* (na komputerach Mac).

3. Kliknij zakładkę **Skrypt** (patrz rysunek 15.6).

Zakładka **Skrypt** zawiera kod źródłowy diagnozowanego pliku. Jeśli testujesz skrypt ze strony, zobaczysz cały jej kod, także HTML. W przypadku zewnętrznych plików JavaScript widoczny będzie tylko jego kod JavaScript.



Rysunek 15.6. Debugger dodatku Firebug pozwala dodawać punkty wstrzymania (wiersze, w których skrypt wstrzymuje działanie i czeka na polecenia), kontrolować wykonywanie kodu i podglądać zmienne na liście Czujka. W czasie wykonywania programu aktualny wiersz (oczekujący na uruchomienie) ma z lewej strony żółtą strzałkę (w kółku na rysunku)

4. Wybierz z menu kodu źródłowego (patrz rysunek 15.6) 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. Pojawia się czerwone kółko reprezentujące 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.

Ponieważ aby otworzyć Firebuga i dodać punkty wstrzymania, trzeba najpierw wyświetlić stronę w Firefoksie, diagnozowany kod JavaScript mógł już zostać 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 rozpoczęć 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 dodatku Firebug do przejścia przez program krok po kroku.

Firebug udostępnia cztery kontrolki (patrz rysunek 15.6), 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 *Czujka* (patrz rysunek 15.6).

Celem przechodzenia przez program krok po kroku jest śledzenie, co dzieje się w każdym wierszu skryptu. Lista *Czujka* 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 507 dowiesz się, jak korzystać z listy *Czujka*.

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 508 znajduje się przykład ilustrujący naprawianie skryptu).

10. Przetestuj stronę w Firefoksie. 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 15.6):

- **Kontynuuj.** 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.

- **Przejdź do kolejnej linii aktywnej funkcji.** Ta przydatna opcja uruchamia aktualny wiersz kodu, a następnie zatrzymuje działanie skryptu. Jej nazwa wynika z tego, że jeśli bieżący wiersz zawiera wywołanie funkcji, debugger nie wkracza w jej kod, ale przechodzi przez funkcję i zatrzymuje się w następnym wierszu. Warto korzystać z tej opcji, jeśli wiadomo, że dana funkcja działa bezbłędnie. Na przykład jeżeli skrypt wywołuje funkcję biblioteki jQuery, warto przejść przez

nią. 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 wcześniej opcji *Przejdz na początek funkcji*.

- **Przejdz na początek 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, jeśli 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 korzystać z tej opcji. Dobra jest też używać opcji *Przejdz do kolejnej linii aktywnej funkcji* zamiast *Przejdz na początek funkcji* 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 *Przejdz na początek funkcji*, wkrocysz w złożony świat jQuery. Jeśli tak się stanie, zauważysz to, ponieważ zakładka 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 *Wykonuj aż do powrotu*.

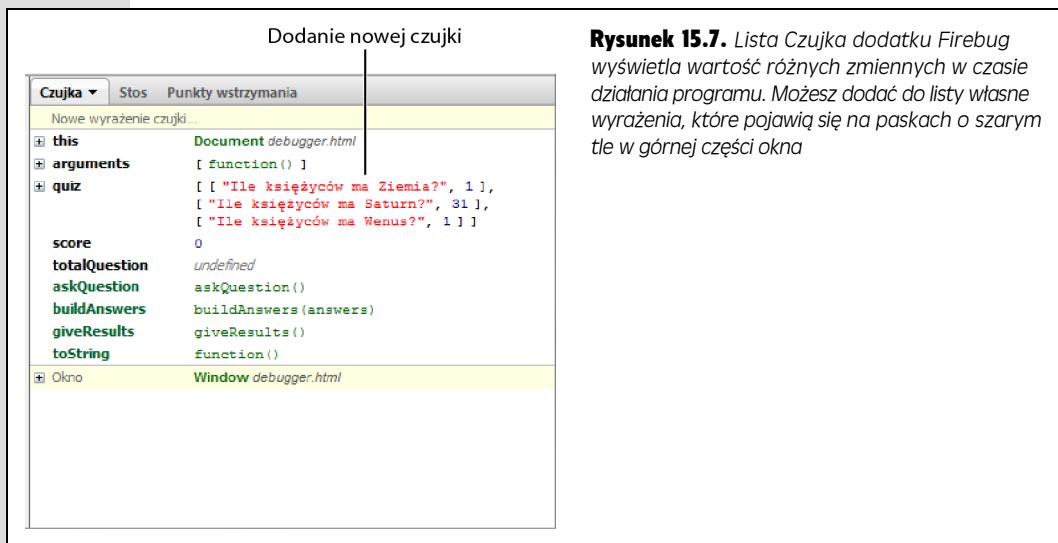
- **Wykonuj aż do powrotu.** Ten przycisk powoduje wyjście debuggera z wywołania funkcji. Zwykle jest używany po kliknięciu przycisku *Przejdz na początek funkcji*. Wybranie tej opcji powoduje wykonanie kodu funkcji bez zatrzymywania się w każdym jej wierszu. Kiedy klikniesz ten przycisk, debugger wróci 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 *Czujka* (patrz rysunek 15.7). Wyświetla ona zmienne i funkcje dostępne w kontekście wykonywanego wiersza kodu. Oznacza to, że jeśli umieścisz punkt wstrzymania w funkcji, zobaczyysz listę wszystkich zdefiniowanych w niej zmiennych. Jeżeli dodasz taki punkt w głównym ciele skryptu, pojawią się funkcje zdefiniowane w tym obszarze. Ponadto na liście *Czujka* 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 107), możesz ją dodać, a następnie obserwować jej zmiany przy każdym uruchomieniu danej pętli.

Listę *Czujka* możesz traktować jak okno z ciągle wywoływanym poleceniem `console.log()`. Lista ta wyświetla wartość zmiennej lub wyrażenia w momencie wykonywania danego wiersza kodu.



Rysunek 15.7. Lista Czujka dodatku Firebug 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

Lista *Czujka* 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 przycisku *Przejdź na początek funkcji* lub *Przejdź do kolejnej linii aktywnej funkcji* zauważysz, że zmienna przyjmuje nieoczekiwanyą wartość, prawdopodobnie znalazłeś wiersz, w którym pojawia się błąd.

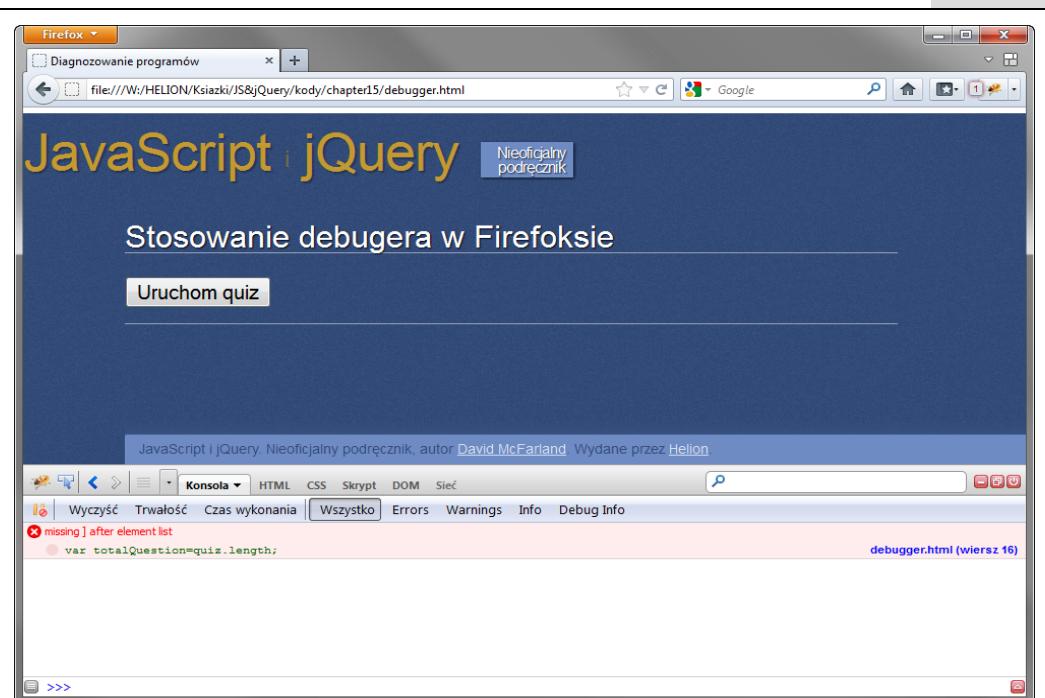
Przykład diagnozowania

W tym przykładzie użyjesz Firebuga 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 *R15*.

Uwaga: Informacje o pobieraniu przykładowych plików znajdziesz na stronie 43.

Aby przejść przez ten przykład, musisz uruchomić przeglądarkę Firefox oraz zainstalować i włączyć dodatek Firebuga (patrz instrukcje na stronie 497).

1. Uruchom przeglądarkę Firefox i otwórz plik *debugger.html* z katalogu *R15*. Przy ikonie Firebuga w górnym, prawym rogu okna przeglądarki pojawi się liczba 1, co oznacza, że na stronie znaleziono jeden błąd (patrz rysunek 15.8). Aby dowiedzieć się, na czym polega problem, musisz otworzyć konsolę Firebuga.
2. Kliknij ikonę Firebuga widoczną w prawym, górnym rogu okna przeglądarki i przejdź na kartę *Konsola*.



Rysunek 15.8. Konsola Firebug to pierwszy przystanek przy wykrywaniu błędów składniowych i czasu wykonania, które uniemożliwiają wykonanie skryptu

Powinieneś zobaczyć komunikat o błędzie widoczny na rysunku 15.8 — „missing] after element list”. Nawiąsy kwadratowe służą do tworzenia tabel (patrz strona 74), dlatego wygląda na to, że brakuje zamk傢cego znaku tego typu. Jest to błąd składniowy (patrz ramka na stronie 491), ponieważ reprezentuje „usterkę gramatyczną” w kodzie (jest to odpowiednik braku kropki na końcu zdania). Po prawej stronie komunikatu o błędzie Firebug informuje, że problem wystąpił w wierszu 15.

3. Uruchom edytor tekstu i otwórz plik *debugger.html*. Znajdź wiersz 15. (znajduje się w nim tylko znak ;). Dodaj zamk傢cy nawias klamrowy po symbolu ;, aby wiersz wyglądał następuj膮co:

];

Nowy nawias klamrowy zamkija zagniezdzoną tablicę (patrz strona 120), która obejmuje wszystkie pytania i odpowiedzi z quizu.

4. Zapisz plik. Wróć do przegladarki Firefox i odśwież stronę.

Następny błąd! Tym razem konsola informuje, że „\$ is not defined”, i wskazuje wiersz 10. z funkcją \$(document).ready() biblioteki jQuery. Kiedy Firefox 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-1.6.3.min.js"></script>
```

Standardowym problemem przy korzystaniu z zewnętrznych plików jest błędnie podana ścieżka do skryptu. Tu plik *jquery-1.6.3.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-1.6.3.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-1.6.3.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 Firefox i odśwież stronę.

Brak błędów! Wygląda na to, że strona została naprawiona. Czy aby na pewno?

7. Kliknij przycisk *Uruchom 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 491). 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.

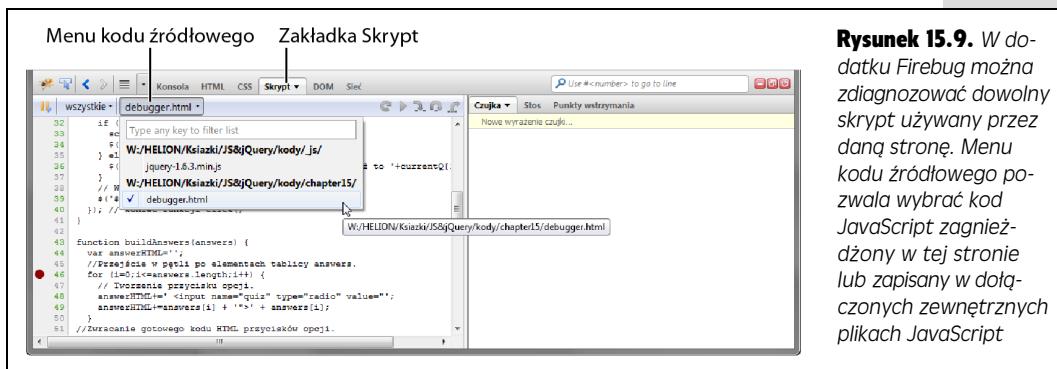
Czy znalazłeś funkcję? Choć w kodzie nie ma metody `askQuestions()`, powinieneś zauważyć funkcję `askQuestion()` (bez litery „s”).

8. Wróć do edytora tekstu i usuń ostatnią literę „s” z wywołania `askQuestions()` w wierszu 70. (w końcowej części skryptu). Zapisz plik, odśwież go w Firefoksie i ponownie kliknij przycisk *Uruchom 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 dodatku Firebug jest pusta, dlatego technicznie nie ma błędu w kodzie JavaScript. Problem musiał wystąpić w logice programu. Aby odkryć przyczynę błędu, trzeba użyć debuggera dodatku Firebug.

9. Kliknij zakładkę *Skrypt* dodatku Firebug i wybierz plik *debugger.html* z menu kodu źródłowego, widocznego bezpośrednio nad wspomnianą zakładką (patrz rysunek 15.9).

Zakładka *Skrypt* 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.



Ponieważ problemem jest przycisk opcji z napisem „undefined”, warto rozpoczęć 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. Na zakładce *Skrypt* dodatku Firebug prześwietn stronę do wiersza 46. Kliknij margines po lewej stronie liczby 46, aby wstawić punkt wstrzymania (w kółku na rysunku 15.9).

Po lewej stronie wiersza 46 pojawi się czerwone kółko. Reprezentuje ono punkt wstrzymania, czyli miejsce w kodzie, w którym interpreter przerwie działanie skryptu. 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 programu, co przypomina korzystanie z funkcji `console.log()` (patrz strona 499). Musisz tylko poinformować dodatek Firebug, jakie zmienne chcesz obserwować.

11. Kliknij pasek *Nowe wyrażenie czujki* po prawej stronie okna dodatku Firebug, wpisz literę **i, a następnie wciśnij klawisz *Enter*.**

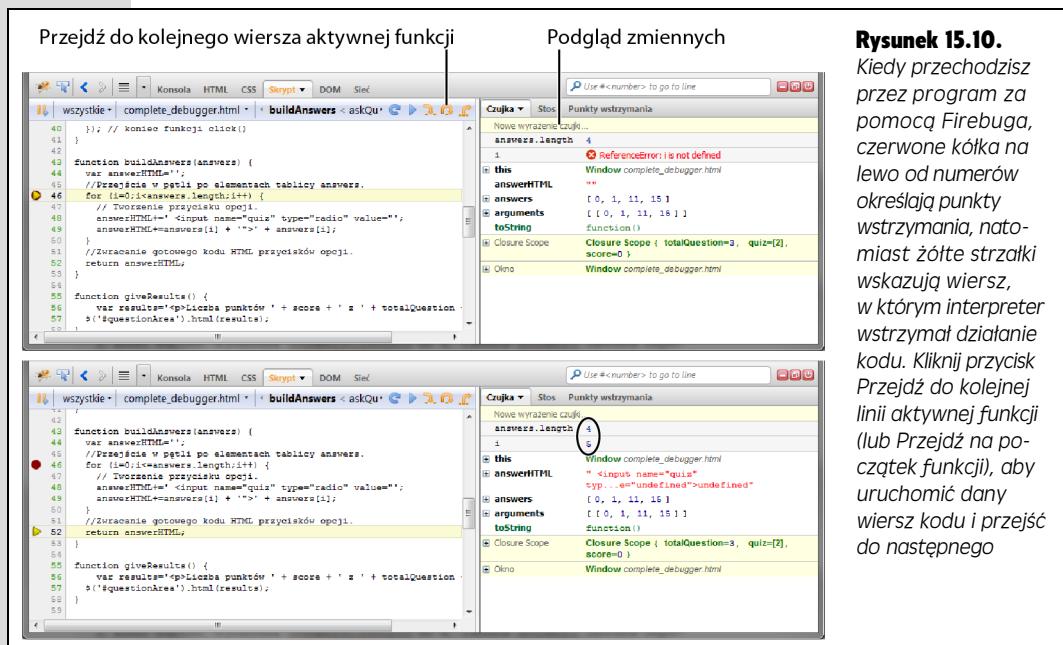
Dodałeś w ten sposób zmienną `i` do listy *Czujka*. Jest to zmienna licznika pętli `for`, określająca liczbę uruchomień tej pętli (więcej informacji o pętlach znajdziesz na stronie 107). W czasie działania skryptu będziesz mógł śledzić zmiany tej wartości. Następnie dodaj nową obserwowaną zmienną.

12. Ponownie kliknij pasek *Nowe wyrażenie czujki*, wpisz *wyrażenie answers.length* i wciśnij klawisz *Enter*.

Nie przejmuj się wartością, którą Firebug 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 Odśwież w Firefoksie lub wciśnij kombinację *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 góra część rysunku 15.10). Zauważ, że na zakładce *Czujka* 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 15.10.**

Kiedy przechodzisz przez program za pomocą Firebuga, czerwone kółka na lewo od numerów określają punkty wstrzymania, natomiast żółte strzałki wskazują wiersz, w którym interpreter wstrzymał działanie kodu. Kliknij przycisk *Przejdź do kolejnej linii aktywnej funkcji* (lub *Przejdź na początek funkcji*), aby uruchomić dany wiersz kodu i przejść do następnego

Z kolei wartość wyrażenia *answers.length* to 4. Tablica *answers* zawiera odpowiedzi przekazane do funkcji (elementy tej tablicy są widoczne w dolnej części listy *Czujka*). 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 *Przejdź do kolejnej linii aktywnej funkcji* (patrz rysunek 15.10).

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 *Przejdź do kolejnej linii aktywnej funkcji*, dopóki zmienna *i* na liście *Czujka* 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 nieoczekiwany momencie. Pamiętaj, że w pętli *for* środkowa instrukcja to warunek, który musi być spełniony, aby skrypt uruchomił kod w pętli (patrz strona 107). 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`” pojawi się dle tego, 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 Firefoksie.

Możesz kliknąć czerwone kółko w oknie skryptu dodatku Firebug, 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. Firebug znacznie ułatwia zbadanie „wnętrza” skryptu i ustalenie przyczyn problemów.

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

- **Encyklopedia języka JavaScript w witrynie Mozilla Developer Center** (<https://developer.mozilla.org/en/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.
- **Krótki przegląd języka JavaScript w witrynie DevGuru** (www.devguru.com/technologies/javascript) to jednostronicowa lista słów kluczowych i metod tego języka. Kliknij wybrane pojęcie, a pojawi się strona z opisem danego słowa kluczowego lub określonej metody.

- **Witryna Google Doctype** (<http://code.google.com/doctype>) zawiera omówienie technologii JavaScript, DOM i CSS, a także informuje, które funkcje są obsługiwane przez poszczególne przeglądarki. Jest to swoista encyklopedia dla projektantów stron WWW.
- **Dokumentacja języka JavaScript na witrynie MSDN** ([http://msdn.microsoft.com/en-us/library/ye4tbz0\(v=VS.94\).aspx](http://msdn.microsoft.com/en-us/library/ye4tbz0(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 Flanagan (wydawnictwo O'Reilly) to najbardziej wyczerpująca drukowana encyklopedia języka JavaScript. Jest to treściwa i obszerna pozycja, jednak 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).

Artykuły i prezentacje

- A (Re)-Introduction to JavaScript autorstwa Simona Willisona (<http://www.slideshare.net/simon/a-reintroduction-to-javascript>). Ta ciekawa prezentacja to przegląd języka JavaScript w formie slajdów.

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.

¹ Wydanie polskie: *JavaScript. Podręcznik programisty*, RM, 2002 — przyp. tłum.

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 działaniu tego języka i programowaniu w nim, jednak książka zawiera niewiele przykładów, które można przenieść bezpośrednio na własne strony.

jQuery

Niniejsza książka jest poświęcona głównie bibliotece jQuery, jednak warto dużo lepiej poznać to rozbudowane, przyspieszające pracę i ciekawe narzędzie.

Artykuły i prezentacje

- *jQuery Essentials* to prezentacja autorstwa Marca Grabanskiego (<http://www.slideshare.net/1Marc/jquery-essentials>). Stanowi zabawny, szczegółowy i godny polecenia pokaz slajdów obejmujący podstawowe zagadnienia związane z jQuery.
- *jQuery Tutorials for Designers* (www.webdesignerwall.com/tutorials/jquery-tutorials-for-designers) to artykuł ze znanego bloga poświęconego projektowaniu stron WWW, Web-Designer Wall. W tekście tym opisano 10 ciekawych, a przy tym prostych technik opartych na bibliotece jQuery.
- *jQuery Cheatsheet* (<http://woorkup.com/2011/05/12/jquery-visual-cheat-sheet-1-6/>) to plik w formacie PDF, który na jednej przeznaczonej do druku stronie zawiera wszystkie funkcje biblioteki jQuery.

Witryny

- **JQuery.com** to „dom” doskonałej biblioteki języka JavaScript, jaką jest jQuery. Znajdziesz tu grupy dyskusyjne, dokumentację, wtyczki i inne materiały do pobrania.
- **Dokumentacja jQuery** jest dostępna w formie rozwijanej przez użytkowników Wiki (<http://docs.jquery.com>). Każdy może dodać lub zmodyfikować opis dowolnej z wielu funkcji jQuery, jednak za większość dokumentacji odpowiada mała grupa osób. Jest to najbardziej wyczerpujące źródło informacji na temat jQuery.
- **Witryna jQuery for Designers** (<http://jqueryfordesigners.com>) zawiera samouczki w formie nagrani wideo i tekstu. Dzięki nim dowiesz się, jak przygotować ciekawe efekty wizualne, utworzyć przydatne interfejsy i ogólnie usprawnić witrynę za pomocą biblioteki jQuery.

² Wydanie polskie: *Head First JavaScript. Edycja polska*, Helion, Gliwice 2009 — przyp. tłum.

- **Witryna Learning jQuery** (<http://www.learningjquery.com>) udostępnia artykuły autorstwa czołowych programistów pracujących nad jQuery.
- **Script Junkie** (<http://msdn.microsoft.com/en-us/scriptjunkie>) to witryna stworzona przez firmę Microsoft, przeznaczona dla projektantów i twórców stron WWW. Zawiera ogólne artykuły na temat języków HTML, CSS oraz JavaScript (z dużym naciskiem na jQuery).

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.
- *jQuery: Novice to Ninja³* autorstwa Earla' Casteldina i Craiga Sharkie (wydawnictwo Sitepoint) zawiera wiele porad dotyczących stosowania jQuery oraz wiele praktycznych przykładów prezentujących sposoby rozwiązywania problemów powszechnie spotykanych podczas tworzenia interfejsów użytkownika stron WWW.

AJAX

AJAX łączy mechanizmy przeglądarek, język JavaScript i skrypty działające na serwerze. Oznacza to potrójną zabawę (i jednocześnie trzy razy większe problemy). Na szczęście dostępnych jest wiele materiałów do nauki tej technologii.

Witryny

- **Ajaxian** (www.ajaxian.com) to doskonałe źródło najnowszych informacji na temat AJAX-a, platform związanych z JavaScriptem i przydatnych serwisów internetowych. Jest to witryna skierowana do profesjonalnych programistów języka JavaScript, jednak zawiera też wiele nowinek, ciekawostek i adresy stron, w których w kreatywny sposób zastosowano AJAX.

³ Wydanie polskie: *jQuery. Od nowicjusza do wojownika ninja*, Helion, Gliwice 2012 — przyp. tłum.

Książki

- *Head Rush Ajax*⁴ autorstwa Bretta McLaughlina (wydawnictwo O'Reilly) to prawdopodobnie najlepsze wprowadzenie do AJAX-a dla osób, które dopiero uczą się języka JavaScript i tworzenia skryptów uruchamianych po stronie serwera. Swobodna forma i układ graficzny ułatwiają zrozumienie podstawowych zagadnień oraz technik związanych z AJAX-em.

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

- *Show Love to the Object Literal* (www.wait-till-i.com/2006/02/16/show-love-to-the-object-literal) autorstwa Chrisa Heilmana to krótki artykuł w blogu z omówieniem właściwego korzystania z literałów obiektowych języka JavaScript.
- *Do Object Oriented JavaScript* (<http://msdn.microsoft.com/en-us/scriptjunkie/ff698282.aspx>) to krótka prezentacja wideo autorstwa Chrisa Pelsa, zawierająca wprowadzenie do zaawansowanych zagadnień programowania w języku JavaScript. Stanowi doskonałe miejsce, by zacząć naukę programowania obiektowego w tym języku.
- *Sorting a JavaScript array using array.sort()* (www.javascriptkit.com/javatutors/arraysort.shtml) zawiera przydatne informacje na temat sortowania zawartości tablic. Znajdziesz tu też szybki sposób na losowe rozmieszczenie elementów tablicy (przypomina to tasowanie talii kart).

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.
- **Unobtrusive JavaScript** (www.onlinetools.org/articles/unobtrusivejavascript/index.html) to miniatura witryna Christiana Heilmanna, poświęcona zagadnieniu „nieinwazyjnego” kodu JavaScript. Podejście to polega na zapewnianiu dostępności do witryny wszystkim użytkownikom, nawet tym korzystającym z przeglądarek bez obsługi języka JavaScript.

⁴ Wydanie polskie: *Head Rush Ajax*, Helion, Gliwice 2007 – przyp. tłum.

- **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.
- **Centrum programistów JavaScript w witrynie Yahoo!** (<http://developer.yahoo.com/javascript>) to jedno z najbogatszych w sieci WWW źródeł wiedzy na temat języka JavaScript. Wiele informacji dotyczy biblioteki języka JavaScript rozwijanej przez Yahoo! (YUI), a także serwisów oferowanych przez tę firmę (na przykład Yahoo! Maps).
- **Dział poświęcony językowi JavaScript na witrynie Mozilla Developers Network** (<https://developer.mozilla.org/en/javascript>) zawiera bardzo dużo informacji na temat JavaScriptu, w tym jego encyklopedię, o której wspomniałem na początku tego dodatku, jak również szczegółowy przewodnik (<https://developer.mozilla.org/en/JavaScript/Guide>) opisujący różne istniejące wersje języka oraz przykłady pojęć związanych z tym językiem.

Książki

- *JavaScript Patterns*⁵ (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 książce *JavaScript: The Good Parts*⁶ Douglas Crockford (wydawnictwo O'Reilly) opisuje najbardziej przydatne elementy języka JavaScript i metody unikania błędów. Douglas wie, o czym pisze, ponieważ jest starszym architektem do spraw języka JavaScript w firmie Yahoo!. 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 podroziale znajdziesz listę kilku przydatnych źródeł.

⁵ Wydanie polskie: *JavaScript. Wzorce*, Helion, Gliwice 2012 — przyp. tłum.

⁶ Wydanie polskie: *JavaScript — mocne strony*, Helion, Gliwice 2009 — przyp. tłum.

Witryny

- **Kompletny przewodnik po języku CSS w witrynie WestCiv** (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.
- **Przegląd języka CSS w witrynie SitePoint** (<http://reference.sitepoint.com/css>) to następne internetowe źródło wiedzy na temat języka CSS. Witryna ta jest łatwa w użyciu i udostępnia wyszukiwarkę.
- **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*⁷ wydanie 2. 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*⁸ 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 całą za jednym razem.
- *Stunning CSS3* autorstwa Zoe Mickely Gillenwater (wydawnictwo New Riders) to dobrze zaprojektowana książka, która opisuje wiele aspektów nowego standardu CSS3 i przedstawia złożone projekty tworzone za jego pomocą; zawiera także opisy wielu technik pozwalających na efektywne korzystanie z CSS3.

⁷ Wydanie polskie: *CSS. Nieoficjalny podręcznik*, Helion, Gliwice 2007 — przyp. tłum.

⁸ Wydanie polskie: *CSS. Kaskadowe arkusze stylów. Przewodnik encyklopedyczny. Wydanie III*, Helion, Gliwice 2008 — przyp. tłum.



Skorowidz

A

adres URL, 41
AJAX, Asynchronous JavaScript and XML, 357, 394
proste operacje, 358
aktualizowanie zawartości strony, 377
aktywowanie pola, 282, 286, 479
animacje, 205
animowany pasek, 211
API, Application Programming Interface, 394, 426
API key, 395
apostrofy, 491
argument
callback, 373
data, 373
url, 373
arkusz stylów
anythingslider.css, 327
gallery.css, 232
atrybut
bgColor, 160
checked, 276
class, 137, 138, 140
data-tooltip, 347
href, 41, 250
rel, 237
src, 41, 159, 220
target, 252
atrybuty
HTML, 159
znaczników, 138
automatyczna konwersja typów, 66
automatyczne uzupełnianie, 406
automatyzacja tworzenia kodu, 136

B

bezwzględne pozycjonowanie, 201
biblioteka jQuery, 18, 128
biblioteki, 46
biblioteki JavaScript, 128
blok deklaracji, 23
blokowanie
odnośnika, 252
przesyłania danych, 284
błąd składniowy, 489
błędy, 48, 380, 487
czasu wykonania, 491
logiczne, 491
składniowe, 491
typograficzne, 214

C

CDN, content distribution network, 129
CSS, Cascading Style Sheets, 21, 144
cudzysłowy, 491
czas trwania animacji, 214
czas wczytywania witryny, 484

D

dane
kanału Flickr, 398
kanału w formacie JSON, 401
uwierzytelniające, 381
zdjęć, 398
daty, 289
daty i godziny, 471
dzień tygodnia, 472
format godzin, 474
miesiąc, 471
obiekt Date, 471
wyświetlanie czasu, 475

debugger Firebug, 505
deklaracja, 23
diagnozowanie, 508
diagnozowanie zaawansowane, 503
długość łańcucha, 446
dodatek Firebug, 153, , 496–511
dodawanie
 efektu rollover, 223
 elementów do tablicy, 76
 etykietek ekranowych, 340
 kod CSS, 342
 kod HTML, 340
 kod JavaScript, 343
 formatu JSON, 396
 identyfikatorów, 396
 jQuery do strony, 132
 kalendarzy, 289
 kanału Flickr, 395
 komunikatów o błędach, 296
 map Google Maps, 404
 plików zewnętrznych, 46
 reguł walidacji, 294, 296
 skryptu, 40
 sliderów, 325
 tekstu, 45
 treści, 149, 368, 423
 zdjęć, 400
dokumentacja jQuery, 426, 430
dołączanie zewnętrznych plików, 477
DOM, Document Object Model, 137, 361
dopasowywanie wzorców, 461
dostęp do obiektu JSON
 notacja tablicowa, 388
 składnia z kropką, 388
dynamiczne
 dodawanie znaczników, 413
 modyfikowanie stron, 333
 rozmiieszczanie elementów, 350
działanie funkcji, 113

E

efekt
 FancyBox, 238
 powiększania, 210
 rollover, 219–224
 stopniowego wyświetlania, 228
efekty
 jQuery, 198
 wizualne, 197, 199
element
 data.items, 402
 media, 400
 pojemnika, 315
 strony, 136
 wyzwalający, 341, 343

elementy
 formularza, 273
 nawigacyjne mapy, 410
 prezentacji, 241
 etykiety ekranowe, 340, 344

F

FAQ, Frequently Asked Questions, 191
FIFO, First In, First Out, 78
filtr
 checked, 274
 contains(), 147
 even, 146
 first, 147
 has(), 147
 hidden(), 147
 last, 147
 odd, 146
 not(), 147
 selected, 275
 visible, 148
filtry jQuery, 146
Firebug, 153, 496–511
 console.log(), 499
 instalowanie, 497
 przeglądanie błędów, 498
Flickr, 395
Flickr API, 395
format
 JSON, 377, 386
 JSONP, 386
 XML, 377, 381
 ZIP, 485
formatowanie
 danych, 373
 komunikatów, 311
formularz logowania, 202
formularze, 271
formularze inteligentne, 281
funkcja
 \$(), 422
 \$(document).ready(), 133
 \$.each(), 401
 \$.getJSON(), 394–398, 401
 .after(), 151, 438
 .append(), 151, 438
 .before(), 151, 438
 .bind(), 441
 .children(), 434
 .click(), 441
 .clone(), 167
 .closest(), 435
 .delegate(), 442
 .each(), 160

.empty(), 440
.error(), 380
.find(), 425, 434
.html(), 150, 438
.next(), 436
.parent(), 434
.prepend(), 151, 438
.prev(), 436
.remove(), 152, 439
.replaceWith(), 439
.siblings(), 435, 436
.text(), 150, 423, 438
.unwrap(), 440
.wrap(), 439
.wrapInner(), 440
addClass(), 154, 159, 250
animate(), 205–212
append(), 424
attr(), 159, 221, 226
bind(), 188, 189, 191
buildAnswers(), 511
clearMarkers(), 414
click(), 233, 323
clone(), 153
console.log(), 499
createMarker(), 413
css(), 155, 156, 431, 479
each(), 161, 225, 390
errorResponse(), 380
fadeIn(), 200, 232
fadeOut(), 160, 200
fadeTo(), 200
fadeToggle(), 200
fancybox(), 237, 261
focus(), 282
get(), 372, 374
getJSON(), 387
goMap(), 407
height(), 334
hide(), 143, 198
hover(), 184, 223, 227
html(), 377
innerWidth(), 335
jQuery(), 422
load(), 365–372
not(), 254
offset(), 337
openExt(), 254
outerHeight(), 336
outerWidth(), 335
parseFloat(), 467
parseInt(), 466
position(), 337
post(), 372, 374
prepend(), 232, 384
preventDefault(), 187, 251, 370
print(), 113
printTime(), 476
printToday(), 112
processContacts(), 388
processData(), 384
processResponse(), 378
prompt(), 72
ready(), 182
removeAttr(), 159, 160
removeClass(), 154
removeMarker(), 414
scrollTop(), 340
serialize(), 376
show(), 198
showHideMarker(), 414
slideDown(), 202
slideToggle(), 202
slideUp(), 202
stop(), 216
submit(), 277
text(), 377
toggle(), 198
toggleClass(), 155, 204
unbind(), 187
val(), 273, 275
validate(), 294, 306, 308
width(), 334
zrotna, callback function, 197, 209
funkcje, 111
 argumenty, 114
 przekazywanie danych, 113
 zwracanie wartości, 115
funkcje
 anonimowe, 160, 176
 do obsługi odpowiedzi, 363
 do manipulacji kodem HTML, 438
 do poruszania się po DOM, 433
 systemu operacyjnego, 29
 wbudowane, 56
wykonawcze zdarzeń, 174
wywoływanie zwołanie, 373

G

galeria fotografii, 228
generowanie
 liczby losowej, 470
 zdarzenia click, 322
Google Maps, 404
gra
 Pac-Man, 60
 pasjans, 90
 Pong, 336
usuwanie chwastów, 441

H

HTML, Hypertext Markup Language, 19

I

identyfikator

- button, 181
- changeStyle, 155
- disable, 187
- headlines, 368
- newslinks, 368
- popUp, 149
- signup, 277
- username, 282

indeksowanie tablic, 75

informacje o stylach, 236

informacje zwrócone przez serwer, 376

instrukcja

- break, 483
- else if, 95
- if, 94
- if-else, 122, 484
- new Date(), 471
- switch, 483
- Switch, 482

instrukcje, 55

instrukcje warunkowe, 89, 91, 99

interaktywne efekty graficzne, 219

interaktywny pokaz slajdów, 330

interfejs programowania aplikacji, 394

Internet Explorer, 44

interpreter JavaScript, 38

J

JavaScript, 15

jednostka em, 22

język

- CSS, 21
- HTML5, 18, 289, 342
- JavaScript, 15, 361
- PHP, 16, 367

języki

- kompilowane, 39
- serwerowe, 367
- skryptowe, 39
- używane po stronie serwera, 361

jQuery, 17, 128

jQuery UI, 135, 325

JSON, JavaScript Object Notation, 386

JSONP, JSON with padding, 393

K

kanał Flickr, 395

kanaly, 395

kanaly RSS, 395

karty, 314, 316, 318

kaskadowe arkusze stylów, 144

kategoria

- Ajax, 428
- Attributes, 427
- Core, 426
- CSS, 428
- Data, 429
- Deffered objects, 429
- Dimenstions, 429
- Effects, 428
- Events, 428
- Forms, 429
- Internals, 430
- Manipulation, 428
- Offset, 430
- Selectors, 427
- Traversing, 428
- Utilities, 429

klasa, 154

- active, 321
- externalLink, 154
- focus, 479
- pq, 164
- pullquote, 167
- tooltip, 345

klauzula else, 94

klikanie, 28

kliknięcie obrazka, 328

kod JavaScript formularza, 310

kolejki FIFO, 78

kolejność

- dodawania plików, 42
- wykonywania operacji, 65
- kolor tła, 159
- komentarz jednowierszowy, 85
- komentarze, 85
- komentarze wielowierszowe, 85
- komponenty interfejsu użytkownika, 325
- kompresja plików, 484
- komunikacja przeglądarki z serwerem, 360, 362
- komunikaty o błędach, 293, 300, 302, 308, 489

konfigurowanie

- serwera sieciowego, 362
- strony z galerią, 235

konsola błędów

- Chrome, 51
- Firebug, 500, 504, 509
- Firefox, 48, 488

Internet Explorer 9, 50
JavaScript, 48, 49
Safari, 51
kontrolki typu mapy, 410

L

liczba dopasowań, 463
liczby, 464
formatowanie, 468
generowanie, 469
Math.random(), 470
wyszukiwanie, 467
zaokrąglanie, 468
lista
Czujka, 507
wypunktowana, 317
zagnieżdżona, 265
literały obiektowe, 158, 246, 375
logowanie, 381, 383
losowe pobieranie, 470

Ł

łańcuchy wywołań funkcji, 149
łańcuchy znaków, 445
indexOf(), 448
match(), 461
pobieranie fragmentów, 449
prompt(), 465
przeszukiwanie, 447
replace(), 463
slice(), 450
wyszukiwanie wzorca, 450
z zapytaniem, 373, 375
zamiana na liczbę, 465

łączenie
liczb i łańcuchów znaków, 66
łańcuchów znaków, 65
warunków, 97
wywołań w sekwencję, 422

M

mapa
hybrydowa, 409
interaktywna, 405
menu, 29
animowane, 263
nawigacyjne, 263

metoda
bind(), 188
blur(), 258
close(), 257

document.getElementById(), 139
document.getElementsByTagName(),
139
encodeURIComponent(), 375
focus(), 258
GET, 363, 374
getDay(), 472
indexOf(), 447
match(), 461, 462, 464
Math.random(), 470
moveBy(), 258
moveTo(), 258
open(), 255, 259, 363
parseInt(), 157
POST, 363, 374
prompt(), 465
replace(), 251, 463
resizeBy(), 258
resizeTo(), 258
scrollBy(), 258
scrollTo(), 258
search(), 461
slice(), 449
metody, 83
obiektu Date, 471
przesyłania danych, 374
miniaturki zdjęć, 403
model obiektów dokumentu, 137
modyfikowanie
działania slidera, 332
opcji FancyBox, 240
stron WWW, 134
właściwości CSS, 155
wyglądu slidera, 329

N

nakładka, overlay, 333
NaN, not a number, 66
narzędzia, 24
narzędzie W3C Validator, 69
nawiasy klamrowe, 95, 111
nawigacja, 263
nazwa zastępcza funkcji, 422
nazwy zmiennych, 60, 116
negowanie warunków, 98

O

obiekt
Date, 471, 473
errorPlacement, 309
jQuery, 140
rules, 306
window, 84

obiekt
 XMLHttpRequest, 360, 362
 zdarzenia, 185

obiekty, 82
 jQuery, 231
 JSON, 389
 zagnieżdżone, 391

obramowanie zakładek, 318

obsługa
 błędów, 380
 kanalów Flickr, 397
 odpowiedzi, 363
 quizów, 123
 zaawansowana zdarzeń, 441
 zdarzeń, 174
 ządań ajaksowych, 365

odnośnik z tekstem, 250

odnośniki, 249

odwołanie zwrotne JSONP, 397

okienka informacyjne, 415

określanie lokalizacji, 407

opcja
 changeSpeed, 239
 cyclic, 240
 easingIn, 239
 equalTo, 300
 max, 300
 maxlength, 299
 min, 299
 minlength, 299
 overlayColor, 238
 overlayOpacity, 238, 246
 padding, 239
 range, 300
 rangelength, 299
 titlePosition, 239
 transitionIn, 239

opcje
 formularza, 284
 kanalów Flickr, 397
 wtyczki FancyBox, 238
 wtyczki GoMap, 409

operacje na miniaturkach, 432

operator
 !=, 92
 !==, 92
 *, 68
 /, 68
 +, 66
 ++, 68
 +=, 68
 <, 92
 <=, 92
 -=, 68
 ==, 92
 ==>, 92
 >=, 92
 I, 97
 LUB, 98
 modulo, 474
 NIE, 98
 przypisania, 62
 trójargumentowy, 481
 typeof, 84

operatorzy
 logiczne, 97
 porównywania, 92

optymalizacja selektorów, 425

otwieranie
 odnośników, 253
 okien, 259
 strony na stronie, 262

P

 pakiet WAMPP, 362

panel
 kart, 314, 320
 zakładek, 324

parametry funkcji, 113, 116

pary nazwa – wartość, 373, 387

pasek nawigacyjny, 268

pętla
 do-while, 109
 for, 107
 while, 104

pętle
 automatyczne, 148
 nieskończone, 105

plik
 animate.html, 213
 anythingslider.css, 326
 compete_quiz.html, 123
 complete_animate.html, 216
 complete_complex_tabs.html, 324
 complete_debugger.html, 508, 513
 complete_do-while.html, 110
 complete_events_intro.html, 181
 complete_fancybox.html, 247
 complete_gallery.html, 229, 233
 complete_in-page-links.html, 263
 complete_load.html, 372
 complete_map.html, 418
 complete_menu.html, 266, 270
 complete_slider.html, 329
 complete_tabs.html, 324
 complete_tooltip.html, 353
 complete_validation.html, 312
 conditional.html, 109

console.html, 500
contacts.php, 387
events_intro.html, 177
example_regex.txt, 464
fadeIn.html, 46
fancybox.html, 244
fancybox.png, 240
faq.html, 192
find.html, 433
flickr.html, 400
flickr_json.txt, 396
form.css, 312
form.html, 286
gallery.html, 230
in-page-links.html, 262
jQuery, 131
jquery.easing.1.3.js, 244
jquery.fancybox-1.3.4.css, 236
jquery.fancybox-1.3.4.js, 235
jquery.js, 133, 293
kompletny_login.html, 386
load.html, 369
login.html, 382
login.php, 383
map.html, 416
menu.html, 268
open_external.js, 254
print_date.html, 112
printTime.js, 476
pull-quote.html, 165
quiz.html, 119
rollover.html, 224
selectors.html, 141, 142
signup.html, 203
slider.html, 327
tabs.html, 320
time.html, 476
todays_news.html, 368
tooltip.html, 344
validation.html, 304
XML, 381

pliki
zewnętrzne, 477
zewnętrzne JavaScript, 40

pobieranie
czasu, 472
elementów, 432
elementów formularzy, 273
elementów klasy, 143
elementów strony, 175
informacji, 70
informacji z witryn, 393
losowe, 470
miesiąca, 471
odnośników, 249
odpowiedzi, 364

podwzorce, 464
pojemnik, 84, 316
pojemnik kart, 315, 318
pokaz slajdów, 244
polecenie
 alert(), 56, 71, 119
 document.write(), 56, 106
 isNaN(), 56
 parseInt(), 103
 pop(), 78, 79
 printToday(), 112
 prompt(), 70
 push(), 77, 78
 shift(), 78, 79
 unshift(), 78

położenie
 elementu, 337, 338
 etykiety, 348

poruszanie się po DOM, 433, 437
pq, pull quote, 164
prezentacja, 240
program
 Aptana Studio, 25
 BBEdit, 25
 CoffeeCup, 25
 CoffeeCup Free HTML Editor, 24
 Dreamweaver, 25
 Eclipse, 25
 EditPlus, 25
 Expression Web Designer, 25
 HTML-Kit, 24
 JSMIn, 485
 MAMP, 362
 Notepad++, 24
 Packer, 485
 textMate, 25
 TextWrangler, 24
 YUI Compressor, 485

programy
 komputerowe, 38
 kompresujące, 485

projekt jQuery UI, 325
przechowywanie obrazków w pliku, 240
przechwytywanie kliknąć, 378
przeciąganie i upuszczanie, 90
przeglądarka internetowa, 360
przekazywanie
 funkcji do zdarzenia, 175
 wyrażenia regularnego, 462
 zdarzeń, 188

przekształcanie
 tablicy PHP, 387
 listy, 268

przesyłanie
 formularza, 278
plików z użyciem AJAX-a, 406

przetwarzanie danych, 376
przewijanie strony, 339
przypisanie polu klasy focus, 479
przypisywanie zdarzenia, 175
pseudoelement first, 323
punkty wstrzymania, 503
pusty znacznik <div>, 368

R

ramka wewnętrzwerszowa, iframe, 259
reakcje na zdarzenia, 186
referencje do okien, 257
reguły walidacji, 293, 295
reguły zaawansowane, 298
rodzaje błędów, 491
rozmiar
dokumentu, 334
okna, 334
zdjęcia, 400

S

selektor, 23
#gallery, 230, 233
#navigation, 267
#newsItem, 372
selektry
atrybutów, 145, 253
CSS, 140
do formularzy, 274
dziesięciu, 144
elementów, 141
elementów potomnych, 143
elementów sąsiadujących, 144
identyfikatorów, 141, 425
klas, 142
proste CSS, 141
zaawansowane, 143
serwer aplikacji, 361
ASP.NET, 361
PHP, 361
serwer bazodanowy, 361
MySQL, 361
SQL Server, 361
serwer CDN, 130
serwer sieciowy, 361
Apache, 361
IIS Microsoftu, 361
serwis Flickr, 397
sieć dystrybucji treści, 129
silnik renderujący, 38
skala mapy, 409
skróty klawiaturowe, 29

skrypty, 38
po stronie klienta, 37
po stronie serwera, 37, 367
slider AnythingSlider, 326
slidery, 313, 325
słowa zarezerwowane, 61, 492
słowo kluczowe, 61

case, 483
Boolean, 84
function, 111
if, 111
number, 84
return, 115
String, 84
this, 162
var, 63, 111, 346
sprawy CSS, 240
sprawdzanie danych, 90
SSL, Secure Socket Layer, 485
stan przycisków, 276
stopniowe wz bogacanie, 369
strona logowania, 382
strony zewnętrzne, 252
struktura
funkcji, 113
galerii, 229
kodu panelu kart, 319
styl #fancybox-close, 242
symbol {}, 386

Ś

ścieżka
bezwzględna, 41
do obrazka, 228
do zewnętrznego pliku, 494
względna, 41
zapisana względem strony, 494
śledzenie działania skryptu, 499

T

tablica preloadImages, 222
tablica, 72
dodawanie elementów, 77
usuwanie elementów, 79
właściwość length, 76
zapisywanie danych, 79
tablice
wielowymiarowe, 120
zagnieżdżone, 120
technologia AJAX, 394
tempo animacji, easing, 207, 215
testowanie wyrażeń regularnych, 464

t tworzenie

- adresu URL, 395
 - animowanego menu, 263
 - daty, 476
 - egzemplarza, 84
 - flag, 95
 - galerii zdjęć, 234, 237
 - instancji obiektu, 84
 - kolejek, 78
 - komunikatów, 69
 - liczb losowych, 469
 - list właściwości, 158
 - literałów obiektowych, 298
 - nowych okien, 255
 - obiektów JSON, 386
 - obiektu, 84
 - pasków interaktywnych, 219
 - slidera, 326, 327
 - tablic, 74
 - wyrażeń regularnych, 450, 451
 - wyróżnianych cytatów, 163
 - zmiennych, 59, 346
- typ obiektu, 84
- typy danych, 56
- liczby, 57
 - łańcuchy znaków, 57
 - wartości logiczne, 58

U

ukrywanie

- etykiety, 344
 - odpowiedzi, 195
 - pól, 289
 - rysunków, 228, 232
 - znacznika, 414
- unikanie błędów, 497
- ustawianie wartości elementu formularza, 275

usuwanie

- atributów HTML, 159
- elementów, 152
- elementów z tablicy, 79
- zdarzeń, 187
- znaczników, 414

używanie

- elementów tablicy, 75
- funkcji, 431
- instrukcji warunkowych, 101
- komentarzy, 86
- pętli do-while, 109
- selektorów, 425
- typów danych, 63
- zmiennych, 62

W

- W3C Validator, 69
- W3C, World Wide Web Consortium, 138
- walidacja, 295
- formularzy, 90, 291
 - prosta, 303
 - stron, 21
 - zaawansowana, 297, 305
 - zdalna, 301
- validator, 21, 69, 253
- wartość, 23
- elementu formularza, 275
 - null, 462
 - właściwości CSS, 156
 - zmiennej, 67
- wczytywanie rysunków, 221
- węzły, node, 138
- wielkość liter, 446
- wielkość znaków, 493
- właściwości, 23, 83
- CSS, 155, 157
 - obiektu zdarzenia, 186
 - okna przeglądarki, 256
- właściwość
- float, 168
 - height, 256
 - items, 399
 - left, 256
 - location, 257
 - markers, 411
 - menubar, 257
 - scrollbars, 256
 - status, 256
 - toolbar, 257
 - top, 256
 - visibility, 147
 - width, 256
- współrzędne
- elementu, 338
 - elementu wyzwalającego, 349
- etykiety, 350, 352
- geograficzne, 408
- obrazka, 337
- wstępne pobieranie, preload, 221
- wtyczka
- AnythingSlider, 326, 328, 330
 - Datapicker, 289
 - Datepicker, 135
 - DD Mega Menu, 270
 - easing, 208
 - FancyBox, 234, 240, 244, 260
 - Form, 406
 - gmap3, 405
 - GMAP3, 409

- wtyczka
 GoMap, 404, 411, 414
 jqDock, 270
 jQuery Tools Tooltip, 354
 jQuery UI Tooltip, 354
 lightbox, 236
 Navigation, 266
 Taconite, 406
 Tweet!, 406
 qTip2, 354
 Validation, 293, 301, 309
- wtyczki do tworzenia etykiet, 354
- wyłączanie pól, 283, 286
- wyrażenia regularne, 227, 450
 adresy e-mail, 458
 adresy stron WWW, 460
 daty, 459
 kod pocztowy, 457
 numer telefonu, 457
 podgrupy, 456
 rozszerszenie GIF, 454
 testowanie, 464
 wielokrotne dopasowywanie, 454
 wybrane symbole, 452
 zastępowanie łańcuchów, 463
 zastosowanie w kodzie, 453, 455
- wyrażenie `$(this)`, 162, 321, 347, 423
- wyróżnianie wierszy, 177
- wyróżniany cytat, pull quote, 163
- wysyłanie żądania, 364
- wyśrodkowanie mapy, 407
- wyświetlanie
 czasu, 473, 475
 kodu HTML, 153
 map Google Maps, 404
 miniaturek zdjęć, 400
 odnośników, 260
 etykiety, 343
 formularza logowania, 251
 mapy, 415
- wywołanie
 wtyczki lightBox, 236
 funkcji, 112
- Z**
- zagłędzanie
 instrukcji, 100
 literałów obiektowych, 389
- zakładka aktywna, 317
- zakładki, 315, 317
- zamiana łańcucha znaków na liczbę, 465
- zapisywanie danych, 79
- zarządzanie zdarzeniami, 188
- zasięg zmiennych, 496
- zastępowanie tekstu, 463
- zawartość kart, 318
- zaznaczanie miejsc na mapie, 411
- zdarzenia, 169
- zdarzenia specyficzne, 181
- zdarzenie
 blur, 173, 279
 change, 173, 280
 click, 170, 230, 280
 dblclick, 171
 focus, 173, 278
 hover, 212
 hover(), 183
 keydown, 174
 keypress, 174
 keyup, 174
 kliknięcia, 170
 load, 172, 182
 mousedown, 170, 171
 mousemove, 172
 mouseout, 171, 212, 222
 mouseover, 171
 mouseup, 170
 reset, 173
 resize, 172
 scroll, 172
 submit, 173, 277
 toggle(), 184
 unload, 172
- zestaw AMP, 362
- zewnętrzne pliki JavaScript, 477
- zmiana właściwości
 CSS, 157
 src, 220, 227
- zmienianie koloru, 479
- zmienna
 \$this, 321
 counter, 107
 evt, 186
 extLink, 163
 imgPath, 232
 message, 116
 oldSrc, 223
 preloadImage, 226
 prodID, 373
 sessID, 373
- zmienne, 59
- globalne, 117
- lokalne, 117
- znacznik
 , 20, 237
 body, 20, 44
 br, 105
 content, 381
 div, 144, 212

<form>, 271
<h1>, 21, 102
<head>, 20
<html>, 20
, 159, 233
<input>, 271
<label>, 273
, 264
<p>, 20, 144
<script>, 38, 70, 405
<scripts>, 132
<select>, 271
, 164, 167, 340
, 20, 144
<textarea>, 271
<tr>, 146
, 143, 264

znaczniki HTML, 20
znak
 \$, 423
 cudzysłówu, 58
 karetki, 63
 kropki, 149
 plusa, 145
 przecinka, 227
 równości, 62, 396, 493
 średnika, 56, 158
 tabulacji, 63
 ukośnika, 85
 zapytania, 373, 395
znaki specjalne, 176

ż

żądanie
 GET, 386
 POST, 386
 XMLHTTP, 397



534

SKOROWIDZ

Kolofon

Tekst tej książki został złożony w programie Adobe InDesign CS4 przez Newgen North America. Przy jej produkcji pomagała Rebecca Demarest.

Okładka książki bazuje na projekcie całej serii, opracowanym przez Davida Freedmana i zmodyfikowanym przez Mike'a Kohnake'a, Karen Montgomery i firmę Fitch (www.fitch.com). Za projekt tylnej okładki, ilustrację psa oraz dobór kolorów odpowiada firma Fitch.

Wewnętrzny układ książki został zaprojektowany przez Davida Futata, na podstawie układu serii stworzonego przez Phila Simpsona. Tekst książki został wydrukowany czcionką Adobe Minion, nagłówki — czcionką Adobe Formata Condensed, a fragmenty kodu — TheSansMonoCondensed firmy LucasFont. Ilustracje umieszczone w tekście przygotował Robert Romano przy użyciu programów Adobe Photoshop oraz Illustrator CS5.5.

PROGRAM PARTNERSKI

GRUPY WYDAWNICZEJ HELION



1. ZAREJESTRUJ SIĘ
2. PREZENTUJ KSIĄŻKI
3. ZBIERAJ PROWIZJĘ

Zmień swoją stronę WWW
w działający bankomat!

Dowiedz się więcej i dołącz już dzisiaj!

<http://program-partnerski.helion.pl>

GRUPA WYDAWNICZA

Helion SA