

Marcin Lis

Dynamiczny HTML

101 PRAKTYCZNYCH SKRYPTÓW

Tchnij życie w swoją stronę!

"Pływające" napisy oraz wyskakujące i wysuwane podpowiedzi.
Zautomatyzowana zamiana obrazów.
Pasek nawigacyjny w CSS.
Menu z efektem przenikania.

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

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

Autor oraz Wydawnictwo HELION 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 Wydawnictwo HELION nie ponoszą również żadnej odpowiedzialności za ewentualne szkody wynikłe z wykorzystania informacji zawartych w książce.

Redaktor prowadzący: Ewelina Burska

Projekt okładki: Mateusz Obarek, Maciej Pokoński

Materiały graficzne na okładce zostały wykorzystane za zgodą iStockPhoto Inc.

Listingi oraz kody źródłowe przykładów prezentowanych w książce można pobrać ze strony <ftp://ftp.helion.pl/przyklady/dhtml1.zip>

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

Drogi Czytelniku!

Jeżeli chcesz ocenić tę książkę, zajrzyj pod adres

http://helion.pl/user/opinie?dhtml1_ebook

Możesz tam wpisać swoje uwagi, spostrzeżenia, recenzję.

ISBN: 978-83-246-7798-6

Copyright © Helion 2010

Printed in Poland.

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

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

Spis treści

Wstęp	7
Rozdział 1. Okna i treść	11
Skrypt 1. [C][E][F][O][S]	Otwarcie nowego okna o zadanych parametrach 11
Skrypt 2. [C][E][F][O][S]	Zamykanie okna przeglądarki 18
Skrypt 3. [C][E7][F][O][S]	Interfejs do przewijania treści strony (przez kliknięcia) ... 21
Skrypt 4. [C][E7][F][O][S]	Automatyczne przewijanie treści strony 24
Skrypt 5. [C][E7][F][O][S]	Podświetlanie komórki tabeli lub innego elementu witryny (CSS) 26
Skrypt 6. [C][E][F][O][S]	Podświetlanie komórki tabeli lub innego elementu witryny (JavaScript) 27
Skrypt 7. [C][E][F][O][S]	Rozpoznanie typu przeglądarki 29
Skrypt 8. [C][E][F][O][S]	Strona zależna od typu przeglądarki 32
Skrypt 9. [C][E][F][O][S]	Modalne okno dialogowe 33
Skrypt 10. [C][E][F][O][S]	Strona tytułowa (splash screen) 38
Skrypt 11. [C][E][F][O][S]	Treść wyświetlana w zagnieżdżonym oknie (wybór tematu z listy) 39
Skrypt 12. [C][E][F][O][S]	Treść wyświetlana w zagnieżdżonym oknie (temat wybierany kliknięciem) 41
Skrypt 13. [C][E][F][O][S]	Treść przedstawiana w symulowanych kartach 43
Skrypt 14. [C][E][F][O][S]	Logowanie użytkowników 48
Skrypt 15. [C][E][F][O][S]	Dostęp do ukrytej treści 50
Skrypt 16. [C][E][F][O][S]	Hasło zabezpieczające witrynę 53
Skrypt 17. [C][E][F][O][S]	Zliczanie liczby odwiedzin 56
Skrypt 18. [C][E][F][O][S]	Uniemożliwienie zaznaczenia fragmentu strony 59
Skrypt 19. [E][F][O]	Dodanie strony do zakładek 60
Rozdział 2. Data i czas	65
Skrypt 20. [C][F][E][O][S]	Zegar cyfrowy 65
Skrypt 21. [C][F][E][O][S]	Stoper, czyli odmierzanie czasu 68
Skrypt 22. [C][E][F][O][S]	Odliczanie zadanego czasu 70
Skrypt 23. [C][F][E][O][S]	Odliczanie czasu do zadanej daty 74
Skrypt 24. [C][E][F][O][S]	Obliczanie liczby dni między podanymi datami 75
Skrypt 25. [C][E][F][O][S]	Kalendarz 78
Skrypt 26. [C][E][F][O][S]	Kalendarz w dowolnym miejscu strony (drag & drop) 83
Skrypt 27. [C][E][F][O][S]	Kalendarz typu pop-up pozwalający na wskazanie daty 84

Skrypt 28. [C][E][F][O][S]	Treść zależna od godziny (pory dnia)	89
Skrypt 29. [C][E][F][O][S]	Treść zależna od daty	91
Skrypt 30. [C][E][F][O][S]	Treść zmieniająca się o określonej godzinie (w określonych godzinach)	92
Rozdział 3. Style CSS		95
Skrypt 31. [C][E][F][O][S]	Dynamiczna zmiana stylu strony	95
Skrypt 32. [C][E][F][O][S]	Styl zależny od przeglądarki	98
Skrypt 33. [C][E][F][O][S]	Powiększanie i zmniejszanie tekstu	99
Skrypt 34. [C][E][F][O][S]	Przypisanie wybranemu elementowi stylu wprowadzonego przez użytkownika	102
Skrypt 35. [C][E][F][O][S]	Kompozycja stylu z wybranych elementów	104
Rozdział 4. Obsługa formularzy		107
Skrypt 36. [C][E][F][O][S]	Walidacja formularzy	107
Skrypt 37. [C][E][F][O][S]	Weryfikacja z uwzględnieniem formatu danych i wyróżnianiem błędnych pól	111
Skrypt 38. [C][E][F][O][S]	Wyszukiwanie frazy w tekście strony	114
Skrypt 39. [C][E][F][O][S]	Automatyczne podświetlanie wszystkich wystąpień poszukiwanego ciągu znaków	118
Skrypt 40. [C][E][F][O][S]	Automatyczne przenoszenie kursora między elementami formularza	120
Skrypt 41. [C][E][F][S]	Blokada wpisywania w formularzu wybranych znaków (blokada klawiszy)	121
Skrypt 42. [C][E][F][O][S]	Dynamiczna weryfikacja danych w trakcie ich wprowadzania (według określonego wzorca)	124
Skrypt 43. [C][E][F][O][S]	Pole tekstowe automatycznie zmieniające swoją wielkość	126
Skrypt 44. [C][E][F][O][S]	Ograniczenie liczby znaków wpisywanych do rozszerzonego pola tekstowego	128
Skrypt 45. [C][E][F][O][S]	Wyróżnianie aktywnego elementu formularza (wityny)	129
Rozdział 5. Rozmaitości		133
Skrypt 46. [C][E][F][O][S]	Potwierdzanie operacji przez użytkownika	133
Skrypt 47. [C][E][F][O][S]	Okno potwierdzania z odliczaniem	135
Skrypt 48. [C][E][F][O][S]	Modyfikacja paska tytułowego	138
Skrypt 49. [C][E][F][O][S]	Modyfikacja paska stanu	139
Skrypt 50. [C][E][F][O][S]	Pasek postępu (zdarzeniowy)	140
Skrypt 51. [C][E][F][O][S]	Pasek postępu (czasowy)	143
Skrypt 52. [C][E][F][O][S]	Weryfikacja adresu e-mail	146
Rozdział 6. Pływające napisy		149
Skrypt 53. [C][E][F][O][S]	Napis pływający w polu tekstowym	149
Skrypt 54. [C][E][F][O][S]	Napis pływający w dowolnym miejscu wityny	153
Skrypt 55. [C][E][F][O][S]	Napis płynnie zmieniający kolor	157
Skrypt 56. [C][E][F][O][S]	Kolor płynący po napisie	160
Skrypt 57. [C][E][F][O][S]	Napis na sinusoidzie	164
Skrypt 58. [C][E][F][O][S]	Litery pojawiające się pojedynczo (symulacja pisania na klawiaturze)	168

Rozdział 7. Boksy reklamowe, podpowiedzi itp. 171

Skrypt 59. [C][E][F][O][S]	Wyskakująca podpowiedź	171
Skrypt 60. [C][E][F][O][S]	Wysuwana podpowiedź (opis)	174
Skrypt 61. [C][E][F][O][S]	Pojawiająca się podpowiedź	178
Skrypt 62. [C][E][F][O][S]	Warstwa (opis, okno) przesuwana za pomocą myszy	180
Skrypt 63. [C][E][F][O][S]	Boks ze zmienną treścią (reklamy, wiadomości itp.)	184
Skrypt 64. [C][E][F][O][S]	Boks z efektem skrołowania	186
Skrypt 65. [C][E][F][O][S]	Boks z efektem przejścia	190

Rozdział 8. Odnośniki 193

Skrypt 66. [C][E][F][O][S]	Odnośnik z potwierdzeniem	193
Skrypt 67. [C][E][F][O][S]	Element strony jako odnośnik (symulacja odnośników)	195
Skrypt 68. [C][E][F][O][S]	Wybór odnośnika z listy rozwijanej (manualny)	196
Skrypt 69. [C][E][F][O][S]	Wybór odnośnika z listy rozwijanej (automatyczny)	198
Skrypt 70. [C][E][F][O][S]	Odnośnik z dodatkowym opisem	199
Skrypt 71. [C][E][F][O][S]	Zdecyduj, gdzie otwierać odnośniki	201

Rozdział 9. Obrazy 203

Skrypt 72. [C][E][F][O][S]	Zmiana obrazu po najechaniu myszą	203
Skrypt 73. [C][E][F][O][S]	Zautomatyzowana zamiana obrazów	205
Skrypt 74. [C][E][F][O][S]	Przesuwanie obrazu po stronie	207
Skrypt 75. [C][E][F][O][S]	Zmiana rozmiarów obrazu z podaniem nowych wartości	208
Skrypt 76. [C][E][F][O]	Skalowanie obrazu za pomocą myszy	210
Skrypt 77. [C][E][F][O][S]	Pokaz slajdów	212
Skrypt 78. [C][E][F][O][S]	Obraz wyświetlany na nowej warstwie przykrywającej zawartość strony	216
Skrypt 79. [C][E][F]	Obraz wyświetlany w nowym oknie	218
Skrypt 80. [C][E][F][O][S]	Lupa (powiększanie fragmentów obrazu)	220
Skrypt 81. [C][E7][F][O][S]	Logo stale widoczne w wybranym miejscu strony	223
Skrypt 82. [C][E][F][O][S]	Obraz płynący po stronie	226
Skrypt 83. [C][E][F][O][S]	Ładowanie obrazów z paskiem postępu (I)	230
Skrypt 84. [C][E][F][O][S]	Ładowanie obrazów z paskiem postępu (II)	234
Skrypt 85. [C][E][F][O][S]	Galeria obrazów z podpisami	237
Skrypt 86. [C][E][F][O][S]	Wyszukiwanie obrazów po opisie	240

Rozdział 10. Menu 245

Skrypt 87. [C][F][E][O][S]	Pasek nawigacyjny w CSS	245
Skrypt 88. [C][F][E8][O][S]	Klasyczne menu poziome z użyciem CSS	247
Skrypt 89. [C][E][F][O][S]	Menu poziome z użyciem JavaScriptu	251
Skrypt 90. [C][E][F][O][S]	Wysuwane menu poziome	256
Skrypt 91. [C][E][F][O][S]	Menu z efektem przenikania	258
Skrypt 92. [C][E][F][O][S]	Menu wysuwane z boku	260
Skrypt 93. [C][E][F][O][S]	Przełączane menu z niezależnymi pozycjami	263
Skrypt 94. [C][E][F][O][S]	Menu przełączane wykluczające	266
Skrypt 95. [C][E][F][O][S]	Przełączane menu z animacją	267
Skrypt 96. [C][E][F][O][S]	Przesuwany boks menu (menu ustawiane przez użytkownika)	270
Skrypt 97. [C][E][F][S]	Menu kontekstowe	272

Rozdział 11. Powiązane opcje i menu hierarchiczne	275
Skrypt 98. [C][E][F][O][S] Boczne drzewo menu	275
Skrypt 99. [C][E8][F][O][S] Menu rozwijane z podpozycjami	278
Skrypt 100. [C][E][F][O][S] Powiązane listy rozwijane	282
Skrypt 101. [C][E][F][O][S] Powiązane opcje wyboru	288
Skorowidz	291

Wstęp

O książce

Obecnie posiadanie własnej witryny WWW należy do dobrego tonu, a tworzyć strony może praktycznie każdy. Nie potrzeba do tego nawet znajomości języka HTML, gdyż bardzo popularne stały się systemy zarządzania treścią (jak np. Joomla!), istnieją jednak edytory, w których witryny buduje się w trybie graficznym. Już dawno minęły czasy, kiedy wystarczała „zwykła”, statyczna strona. Obecnie królują witryny interaktywne, z elementami animacji, wykorzystujące najróżniejsze efekty do przyciągnięcia użytkowników.

Stosowane są w tym celu różne technologie, takie jak DHTML, Java, JavaScript, Kaskadowe arkusze stylów (CSS), ActiveX, PHP, ASP itp. W niniejszej książce zostało przedstawionych 101 praktycznych skryptów, które można wykorzystać na stronie WWW; są one oparte na DHTML-u i JavaScriptcie. Skrypty te pozwolą na uatrakcyjnienie każdej, zarówno prywatnej, jak i firmowej witryny.

Książka przeznaczona jest zarówno dla użytkowników początkujących, jak i średnio zaawansowanych, znających już przynajmniej podstawy HTML-a, JavaScriptu czy DHTML-a. Korzystanie z niej nie sprawi problemu początkującym, gdyż każdy skrypt jest przedstawiony w postaci gotowej do uruchomienia na stronie WWW. Aby osiągnąć zamierzony efekt, wystarczy zatem wkleić kod do swojej witryny, nie przejmując się technicznymi aspektami jego działania (dostępne są także wszystkie zawarte w książce listingi).

Natomiast użytkownicy średnio zaawansowani znajdą tu pomysły i przykłady realizacji pewnych efektów oraz sposoby wykorzystania różnych technologii, które mogą być inspiracją do tworzenia własnych skryptów. Nie jest to oczywiście teoretyczny podręcznik dokładnie omawiający każdą konstrukcję programistyczną. To książka dla praktyków, prezentująca szereg ciekawych przykładów i rozwiązań technicznych.

Kwestie techniczne

Przy każdym skrypcie zaznaczono, w jakich przeglądarkach będzie działał. Co prawda obecne przeglądarki to już dopracowane produkty, a przykłady zostały przygotowane w taki sposób, aby były poprawnie interpretowane w każdej z nich, zdarzają się jednak drobne wyjątki. Dlatego każdy skrypt został oznaczony tak, aby od razu można było stwierdzić, w których przeglądarkach będzie poprawnie interpretowany. W związku z tym oznaczenia pojawiające się przy przykładach są następujące (w kolejności alfabetycznej):

[C] — Chrome,

[E] — Internet Explorer,

[F] — Firefox,

[O] — Opera,

[S] — Safari.

Uwzględnione zostały tylko te przeglądarki i tylko te wersje, które mają widoczny udział w rynku, czyli po prostu są popularne. Testowane były wersje: Chrome 3, Internet Explorer 6, 7 i 8, Firefox 2 i 3, Opera 9 i 10, Safari 4.

Jeżeli przy oznaczeniu przeglądarki występuje numer, np. [E7], oznacza to, że do prawidłowego działania skryptu niezbędna jest co najmniej taka wersja produktu. Czyli dla oznaczenia [E7] jest to Internet Explorer w wersji co najmniej 7.

Wszystkie prezentowane kody (chyba że w opisie zaznaczono inaczej) zostały zapisane tak, aby były zgodne ze ścisłymi wersjami standardów HTML 4.01 (w większości przypadków również z HTML 5) oraz XHTML 1.0. Aby jednak nie marnować miejsca na stronach, stosowane są wyłącznie prolog oraz nagłówki dokumentu charakterystyczne dla HTML 4.01 Strict. Uzyskanie kodu zgodnego z pozostałymi standardami sprowadzać się więc będzie do wprowadzenia modyfikacji w prologu i, ewentualnie, w sekcji nagłówkowej.

Z reguły strony WWW dzieli się na osobne pliki zawierające kod HTML, CSS i JavaScript. W książce zastosowane zostało jednak inne podejście. Cała treść umieszczana jest w pojedynczym pliku, tak by można było swobodnie analizować zależności w kodzie bez przełączania się pomiędzy różnymi plikami. Oczywiście nic nie stoi na przeszkodzie, aby dzielić prezentowane dane na części funkcjonalne i zapisywać osobno.

Listingi i kody źródłowe

Wszystkie listingi oraz kody źródłowe przykładów prezentowanych w książce można pobrać ze strony <ftp://ftp.helion.pl/przyklady/dhtml1.zip>.

Pliki tekstowe z listingami zawierają fragmenty kodu w takiej postaci, w jakiej zostały zaprezentowane na listingach w książce (czasem są to tylko niezbędne fragmenty). Oprócz tego w osobnych katalogach znajdują się pełne wersje kodów (X)HTML, składające się na gotowe do uruchomienia przykłady. Każdy przykład jest więc prezentowany zarówno w postaci fragmentu omawianego w książce (o ile omawiany jest tylko fragment, a nie pełny kod), jak i w pełnej wersji gotowej do użycia w przeglądarce.

Rozdział 1.

Okna i treść

Rozdział pierwszy zawiera skrypty związane z obsługą okien oraz prezentacją danych na stronach WWW. Znajduje się w nim kilkanaście przykładów realizujących takie efekty, jak modalne okna dialogowe, rozpoznawanie typu przeglądarki czy system logowania użytkowników. Przedstawione zostaną także: programowa symulacja kart, sposoby na dynamiczne wyróżnianie elementów witryny, np. komórek tabeli, czy automatyczne przewijanie zawartej na stronie treści.

Skrypt 1.

[C][E][F][O][S]

Otwarcie nowego okna o zadanych parametrach

Witryny niekiedy otwierają nowe okna przeglądarki, np. w odpowiedzi na kliknięcie przycisku czy odnośnika. W ten sposób często prezentowane są regulaminy czy też tematy pomocy dotyczące danego serwisu. Do otwarcia nowego okna służy metoda `open`, którego wywołanie ma postać:

```
open(URL, nazwa [, właściwości[, zamiana]])
```

Nowe okno będzie miało nazwę *nazwa* i zawierało dokument wskazywany przez *URL*. Wygląd okna można określić za pomocą opcjonalnego argumentu *właściwości*, który może przyjmować parametry przedstawione w tabeli 1.1. Poszczególne parametry należy oddzielać od siebie znakami przecinka. Opcjonalny argument *zamiana* ustawiony na `true` określa, że otwierany dokument ma się pojawić w historii otwieranych witryn jako nowy wpis, a ustawiony na `false`, że ma zamienić aktualny wpis. Metoda `open` zwraca obiekt wskazujący nowe okno, pozwalający na wykonywanie na nim innych operacji (wywoływanie metod, zmiana właściwości itp.). Należy jednak pamiętać, że otwieranie nowych okien może być zablokowane w ustawieniach przeglądarki bądź skonfigurowane tak, że zamiast okna otwierana jest nowa karta, a także o tym, że nie każda przeglądarka respektuje wszystkie podane w tabeli właściwości 1.1. Część właściwości może być również uwzględniana tylko przy obecności innych (można to doskonale przetestować za pomocą omawianego skryptu).

Tabela 1.1. Właściwości określające wygląd okna otwieranego za pomocą metody `open`¹

Właściwość	Znaczenie	Dopuszczalne wartości	Przykład
left	Współrzędna <i>x</i> lewego górnego rogu okna.	liczby całkowite	left=100
top	Współrzędna <i>y</i> lewego górnego rogu okna.	liczby całkowite	top=10
height	Wysokość obszaru okna zawierającego treść strony, włącznie z wysokością poziomego paska przewijania.	liczby całkowite, minimalna wartość: 100	height=200
width	Szerokość obszaru okna zawierającego treść strony, włącznie z szerokością pionowego paska przewijania.	liczby całkowite, minimalna wartość: 100	width=200
menubar	Określa, czy ma być widoczny pasek menu.	yes, no	menubar=yes
toolbar	Określa, czy ma być widoczny pasek narzędziowy (nawigacyjny).	yes, no	toolbar=yes
location	Określa, czy ma być widoczny pasek adresu.	yes, no	location=yes
directories	Określa, czy ma być widoczny pasek ustawień osobistych (Personal Toolbar). Jego zawartość zależy od konkretnej przeglądarki.	yes, no	directories=yes
status	Określa, czy ma być widoczny pasek stanu (statusu).	yes, no	status=yes
resizable	Określa, czy wymiary okna mogą być zmieniane.	yes, no	resizable=yes
scrollbars	Określa, czy w przypadku, kiedy dokument nie mieści się w oknie, mają być wyświetlane paski przewijania.	yes, no	scrollbars=yes

Część (X)HTML skryptu została przedstawiona na listingu 1.1.

Listing 1.1. Elementy interfejsu pozwalające określić parametry okna

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN"
"http://www.w3.org/TR/html4/strict.dtd">
<html lang="pl">
  <head>
    <meta http-equiv="Content-Type" content="text/html; charset=utf-8">
    <title>Otwieranie nowego okna przeglądarki</title>
  </head>
  <body>
    <div id="divParametry">
      <label for="tfAdres">Adres</label>
      <input type="text" id="tfAdres" />
      <label for="tfWysokosc">Szerokość</label>
      <input type="text" id="tfSzerokosc" />
      <label for="tfWysokosc">Wysokość</label>
      <input type="text" id="tfWysokosc" />
    </div>
  </body>
</html>
```

¹ W tabeli nie uwzględniono właściwości charakterystycznych wyłącznie dla konkretnych modeli przeglądarek.

```
<label for="tfNazwa">Nazwa</label>
<input type="text" id="tfNazwa" />

<label for="chbPasekStanu">Pasek stanu</label>
<input type="checkbox" id="chbPasekStanu" />
<label for="chbPasekAdresu">Pasek adresu</label>
<input type="checkbox" id="chbPasekAdresu" />
<label for="chbPasekMenu">Pasek menu</label>
<input type="checkbox" id="chbPasekMenu" />
<label for="chbPasekNarzedziowy">Pasek narzędziowy</label>
<input type="checkbox" id="chbPasekNarzedziowy" />
<label for="chbPasekUstawienOsobistych">
  Pasek ustawień osobistych</label>
<input type="checkbox" id="chbPasekUstawienOsobistych" />
<label for="chbPaskiPrzewijania">Paski przewijania</label>
<input type="checkbox" id="chbPaskiPrzewijania" />
<label for="chbZmianaRozmiarow">Zmiana rozmiarów</label>
<input type="checkbox" id="chbZmianaRozmiarow" />

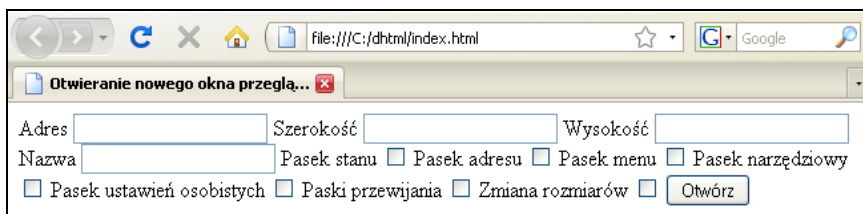
<input type="button" value="Otwórz" id="btnOtworz"
  onclick="btnOtworzClick();" />
</div>
</body>
</html>
```

Na stronie została umieszczona warstwa (`divParametry`), a w niej cztery pola tekstowe, sześć pól wyboru typu `checkbox` oraz przycisk. Wszystkie te elementy zostały zdefiniowane za pomocą znaczników `input` z atrybutem `type` ustawionym na odpowiednią wartość: `text` — dla pól tekstowych, `checkbox` — dla pól wyboru, `button` — dla przycisku. Każdy element ma też przypisaną etykietę opisującą jego funkcję, a także jednoznacznie identyfikujący go identyfikator (czyli wartość atrybutu `id`). Etykiety są generowane za pomocą znaczników `<label>`. Powiązanie między danym elementem a etykietą jest realizowane za pomocą atrybutu `for` znacznika `<input>`. Wartość tego atrybutu wskazuje identyfikator powiązanego elementu (wartość atrybutu `id` znacznika `<input>`). Przyciskowi została przypisana procedura obsługi zdarzenia `click` (wartość atrybutu `onclick`)² w postaci funkcji JavaScript o nazwie `btnOtworzClick`.

Tak przygotowana strona po wczytaniu do przeglądarki przyjmie postać zaprezentowaną na rysunku 1.1. Elementy zostaną wyświetlone jeden za drugim w tylu wierszach, na ile pozwala rozmiar okna przeglądarki. Układ może być zmieniony za pomocą stylów CSS. Wygląd można dopasować do własnych potrzeb. Jedną z propozycji ostylowania³ została przedstawiona na listingu 1.2.

² Często popularnie mówi się o obsłudze zdarzenia `onclick` czy procedurze obsługi zdarzenia `onclick`. W książce będzie stosowana bardziej ścisła terminologia (choć ze ścisłe formalnego punktu widzenia też nie w 100 proc. opisująca technikę zdarzeń). Zdarzenia to bowiem `click`, `mousedown`, `mouseover` itd., a atrybuty znaczników oraz właściwości obiektów w postaci `onclick`, `onmousedown` itd. to procedury obsługi zdarzeń.

³ „Ostylowanie” to neologizm oznaczający, jak łatwo się domyślić, przypisanie elementom witryny stylów CSS, a tym samym zmianę sposobu ich prezentacji.



Rysunek 1.1. Elementy pozwalające na określenie właściwości nowego okna

Listing 1.2. Style CSS formatujące układ strony z listingu 1.1

```
<style type="text/css">
  #divParametry{
    border:1px solid black;
    background-color:#F0F0F0;
    width:250px;
    padding:5px;
    float:left;
  }
  label{
    float:left;
    clear:left;
    margin:2px;
  }
  input{
    float:right;
  }
  #btnOtworz{
    margin-top:30px;
    float:right;
  }
</style>
```

Po umieszczeniu takiej treści znacznika `<style>` w sekcji `<head>` i ponownym wczytaniu kodu do przeglądarki uzyskamy widok zaprezentowany na rysunku 1.2. Uwaga! W przypadku Internet Explorera poprawna interpretacja kodu następuje dopiero w wersji 8. Dla wersji wcześniejszych niezbędne byłoby przygotowanie innego zestawu stylów.

Nowe okno będzie otwierane po kliknięciu przycisku *Otwórz*. Procedurą obsługi zdarzenia `click` tego przycisku jest funkcja `btnOtworzClick`, której zadaniem jest odczytanie wprowadzonych danych i odpowiednie wywołanie metody `open`. W kodzie strony, w sekcji `<head>`, należy więc umieścić skrypt przedstawiony na listingu 1.3.

Listing 1.3. Skrypt odczytujący dane i otwierający nowe okno

```
<script type="text/javascript">
  var defAdres = "http://helion.pl/ksiazki/jscpk.htm";
  var defNazwa = "nowe_okno";
  var defSzerokość = 800;
  var defWysokość = 600;
  function btnOtworzClick()
```

Rysunek 1.2.

*Kod strony
po nadaniu stylów*

The screenshot shows a web browser window with the address bar displaying 'file:///C:/jdhtml/index.html'. The main content area contains a form titled 'Otwieranie nowego okna przeglą...'. The form has four text input fields labeled 'Adres', 'Szerokość', 'Wysokość', and 'Nazwa'. Below these are seven checkboxes: 'Pasek stanu', 'Pasek adresu', 'Pasek menu', 'Pasek narzędziowy', 'Pasek ustawień osobistych', 'Paski przewijania', and 'Zmiana rozmiarów'. An 'Otwórz' button is located at the bottom right of the form.

```
{
    var tfAdres = document.getElementById("tfAdres");
    var tfSzerokosc = document.getElementById("tfSzerokosc");
    var tfWysokosc = document.getElementById("tfWysokosc");
    var tfNazwa = document.getElementById("tfNazwa");
    var chbPasekStanu = document.getElementById("chbPasekStanu");
    var chbPasekAdresu = document.getElementById("chbPasekAdresu");
    var chbPasekMenu = document.getElementById("chbPasekMenu");
    var chbPasekNarzedziowy = document.getElementById("chbPasekNarzedziowy");
    var chbPasekUstawienOsobistych =
        document.getElementById("chbPasekUstawienOsobistych");
    var chbPaskiPrzewijania =
        document.getElementById("chbPaskiPrzewijania");
    var chbZmianaRozmiarow = document.getElementById("chbZmianaRozmiarow");

    if(tfAdres && tfAdres.value) adres = tfAdres.value;
    else adres = defAdres;
    if(tfSzerokosc && tfSzerokosc.value &&
        !isNaN(szerokość = parseInt(tfSzerokosc.value)) && szerokość > 0 );
    else szerokość = defSzerokość;
    if(tfWysokosc && tfWysokosc.value &&
        !isNaN(wysokość = parseInt(tfWysokosc.value)) && wysokość > 0);
    else wysokość = defWysokość;
    if(tfNazwa && tfNazwa.value) nazwa = tfNazwa.value;
    else nazwa = defNazwa;
    if(chbPasekStanu && chbPasekStanu.checked) pasekStanu = "yes";
    else pasekStanu = "no";
    if(chbPasekAdresu && chbPasekAdresu.checked) pasekAdresu = "yes";
    else pasekAdresu = "no";
    if(chbPasekMenu && chbPasekMenu.checked) pasekMenu = "yes";
    else pasekMenu = "no";
    if(chbPasekNarzedziowy && chbPasekNarzedziowy.checked)
        pasekNarzędziowy = "yes";
    else pasekNarzędziowy = "no";
}
```

```

if(chbPasekUstawienOsobistych && chbPasekUstawienOsobistych.checked)
    pasekUstawienOsobistych = "yes";
else pasekUstawienOsobistych = "no";
if(chbPaskiPrzewijania && chbPaskiPrzewijania.checked)
    paskiPrzewijania = "yes";
else paskiPrzewijania = "no";
if(chbZmianaRozmiarow && chbZmianaRozmiarow.checked)
    zmianaRozmiarow = "yes";
else zmianaRozmiarow = "no";

str = "width=" + szerokość + ",height=" + wysokość;
str += ".status=" + pasekStanu + ".location=" + pasekAdresu;
str += ".menubar=" + pasekAdresu + ".toolbar=" + pasekNarzędziowy;
str += ".directories=" + pasekUstawienOsobistych;
str += ".scrollbars=" + paskiPrzewijania + ".resizable=" + zmianaRozmiarow;
window.open(adres, nazwa, str);
}
</script>

```

Zaprezentowany kod jest dosyć długi ze względu na konieczność odczytania wszystkich danych i choć wstępnego sprawdzenia ich poprawności. Konstrukcja skryptu jest jednak bardzo prosta. Na początku znajdują się definicje zmiennych określających wartości parametrów domyślnych (zmienne zaczynające się od ciągu `def`). Wartości domyślne zostaną użyte, gdy użytkownik nie poda żadnych danych lub też w kodzie strony nie będzie odpowiednich elementów interfejsu. Zostało przyjęte, że domyślnym adresem będzie <http://helion.pl/ksiazki/jscpk.htm>, domyślną nazwą okna — `nowe_okno`, domyślną szerokością okna — 800 pikseli, a domyślną wysokością — 600 pikseli. Oczywiście te wartości można dowolnie zmieniać.

Treść funkcji `btnOtworzClick` (wywoływanej po kliknięciu przycisku *Otwórz*) zaczyna się od deklaracji zmiennych pomocniczych i przypisania im odwołań do elementów interfejsu: pól tekstowych i pól wyboru. Odwołania są pobierane przez standardową metodę `getElementById` obiektu `document`, której w postaci argumentu przekazywane są identyfikatory elementów (wartości atrybutu `id`).

Kolejny blok to seria instrukcji warunkowych, w których ustalane są wartości zmiennych odzwierciedlających wartości zapisane w elementach interfejsu. Dla pól tekstowych używana jest konstrukcja badająca istnienie pola oraz wartości w tym polu. Przykładowo dla pola adresowego (`tfAdres`) jest to:

```

if(tfAdres && tfAdres.value) adres = tfAdres.value;
else adres = defAdres;

```

Należy ją rozumieć następująco: jeżeli istnieje pole `adres` (`tfAdres`) i (`&&`) jest w nim wpisana jakaś wartość (`tfAdres.value`), wartość ta staje się wartością zmiennej `adres` (`adres = tfAdres.value`); w przeciwnym przypadku wartością zmiennej `adres` staje się wartość domyślna (`adres = defAdres`), czyli ta zapisana w zmiennej `defAdres`.

Wykorzystywane są tu specyficzne właściwości JavaScriptu. Po pierwsze, wyrażenie `tfAdres` w instrukcji warunkowej zostanie potraktowane jako `false`, gdy wartość zmiennej `tfAdres` będzie równa `null` bądź `undefined`, czyli wtedy gdy metoda `getElementById` nie odnajdzie na stronie elementu o nazwie `tfAdres`. To właśnie pozwala wykryć brak

elementu na witrynie. Z kolei wyrażenie `tfAdres.value` zostanie potraktowane jako `false`, jeżeli w polu tekstowym `tfAdres` nie będzie żadnych danych (będzie zawierało pusty ciąg znaków). To pozwala wykryć brak danych.

W przypadku pól dotyczących wysokości (`tfWysokosc`) i szerokości (`tfSzerokosc`) instrukcje wyglądają nieco inaczej. Użyte zostały dodatkowe wyrażenia warunkowe, sprawdzane jest bowiem to, czy wprowadzone dane dadzą się przetworzyć na wartość całkowitą i czy wartość ta jest większa od 0. Instrukcja pobierająca szerokość ma postać:

```
if(tfSzerokosc && tfSzerokosc.value &&
    !isNaN(szerokosc = parseInt(tfSzerokosc.value)) && szerokosc > 0 );
else szerokosc = defSzerokosc;
```

Wyrażenie `szerokosc > 0` jest oczywiście prawdziwe, gdy wartość zmiennej `szerokosc` jest większa od 0. Ta zmienna nie była jednak nigdzie wcześniej definiowana, a powstaje w wyrażeniu:

```
isNaN(szerokosc = parseInt(tfSzerokosc.value))
```

Najpierw wartość zapisana we właściwości `value` obiektu `tfSzerokosc` (czyli wpisana do pola tekstowego `tfSzerokosc`) jest poddawana działaniu metody `parseInt` zamieniającej ciąg znaków na wartość liczbową. Następnie wynik jest przypisywany zmiennej `szerokosc` oraz poddawany działaniu funkcji `isNaN` badającej, czy przekazany jej argument ma wartość specjalną `NaN`. Ponieważ w przypadku gdy ciąg nie może być przetworzony na wartość całkowitą, metoda `parseInt` zwraca właśnie `NaN`, to podana konstrukcja umożliwia wykrycie błędnej wartości wpisanej do pola tekstowego (wykonywany jest wtedy blok `else`).

Nieco inną konstrukcję mają instrukcje warunkowe badające stan pól wyboru. Przykładowo dla pola `chbPasekStanu` instrukcja ma postać:

```
if(chbPasekStanu && chbPasekStanu.checked) pasekStanu = "yes";
else pasekStanu = "no";
```

Pierwsza część wyrażenia warunkowego (`chbPasekStanu`) bada, czy istnieje obiekt związany z polem (czyli czy to pole faktycznie znajduje się na stronie). Druga część (`chbPasekStanu.checked`) sprawdza, czy pole zostało zaznaczone (czyli czy właściwość `checked` obiektu `chbPasekStanu` ma wartość `true`). Jeżeli pole istnieje i jest zaznaczone, zmiennej `pasekStanu` przypisywany jest ciąg `yes`, a w przeciwnym przypadku — ciąg `no`. W ten sam sposób badany jest stan pozostałych pól wyboru.

Na końcu funkcji tworzony jest ciąg znaków opisujących wszystkie parametry okna. Zapisywany jest w zmiennej pomocniczej `str`. Wartości poszczególnych parametrów pobierane są oczywiście z utworzonych wyżej zmiennych (`szerokosc`, `wysokosc`, `pasekStanu` itd.). Przykładowa postać tego ciągu jest następująca:

```
width=320,height=200,status=no,location=no,menubar=no,toolbar=no,directories=no,
scrollbars=yes,resizable=yes
```

Zmienne `adres`, `nazwa` i `str` są następnie używane w instrukcji wywołującej metodę `open`, a tym samym otwierającej nowe okno przeglądarki:

```
window.open(adres, nazwa, str);
```

Warto użyć tak przygotowanego skryptu do sprawdzenia, jak różne przeglądarki interpretują różne zestawy właściwości dotyczących nowo otwieranych okien.

Skrypt 2. [C][E][F][O][S]

Zamykanie okna przeglądarki

Pierwszy skrypt otwierał nowe okno przeglądarki o zadanych parametrach. Skoro można otworzyć okno, to z pewnością można też je zamknąć. Służy do tego metoda `close` obiektu `window`. Warto napisać korzystającą z tej metody dodatkową funkcję, która posłuży do zamykania wskazanego okna oraz pozwoli na potwierdzanie przez użytkownika chęci wykonania takiej operacji⁴. Treść tak działającej funkcji została przedstawiona na listingu 1.4.

Listing 1.4. Funkcja zamykająca okno przeglądarki

```
function zamknijOkno(wndObj, strPotwierdź)
{
    if(wndObj && !wndObj.closed){
        if (strPotwierdź && !confirm(strPotwierdź))
            return false;
        wndObj.close();
        return wndObj.closed;
    }
    return false;
}
```

Funkcja przyjmuje dwa argumenty: `wndObj` oraz `strPotwierdź`. Pierwszy wskazuje obiekt okna, które ma być zamknięte, a drugi — komunikat, który ma być wyświetlony w oknie z potwierdzeniem chęci wykonania operacji. W pierwszej instrukcji warunkowej (`if(wndObj&&!wndObj.closed)`) sprawdzane jest to, czy istnieje obiekt okna⁵ oraz czy właściwość `closed` ma wartość `true`. Dalsze operacje mają bowiem sens tylko wtedy, gdy odpowiedź na oba pytania brzmi „tak” (okno istnieje i jest otwarte). Wewnętrzna instrukcja `if` ma wyrażenie warunkowe złożone z dwóch członów. Pierwszy (`strPotwierdź`) bada, czy został przekazany parametr `strPotwierdź` i czy jest różny od `false`, `null` i pustego ciągu znaków. Drugi (`!confirm(strPotwierdź)`) to wywołanie metody `confirm` i stwierdzenie, jaki zwróciła wynik.

Metoda `confirm` wyświetla okno dialogowe zawierające komunikat przekazany jej w postaci argumentu i zwraca wartość `true`, jeśli użytkownik kliknął przycisk *OK* (*Yes*,

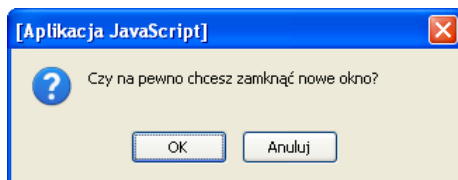
⁴ Oczywiście osobna funkcja zamykająca okna nie jest konieczna i można bezpośrednio wywoływać metodę `close`. Jednak dzięki takiej funkcji łatwo będzie można decydować, czy wyświetlać potwierdzenia zamykania okien, oraz sprawdzać, czy zamknięcie fatycznie wystąpiło. Kod będzie też bardziej jednolity.

⁵ Dokładniej rzecz ujmując, jest tu sprawdzane tylko to, czy parametr `wndObj` jest różny od `unknown`, `false`, wartości `0` bądź pustego ciągu znaków. Nie jest badane, czy jego typem jest `object`.

Tak), oraz *false*, jeśli kliknięty został przycisk *Anuluj* (*Cancel*, *No*, *Nie*). Przykładowy wygląd takiego okna został przedstawiony na rysunku 1.3. Wygląd ten może być różny w zależności od zastosowanej przeglądarki.

Rysunek 1.3.

Okno dialogowe pozwalające na potwierdzenie chęci wykonania operacji



Cała instrukcja warunkowa:

```
if (strPotwierdź && !confirm(strPotwierdź))  
    return false;
```

ma zatem następujące znaczenie — jeżeli został przekazany ciąg `strPotwierdź`, wyświetli okno potwierdzające z zawartym w tym ciągu komunikatem; jeśli kliknięty został przycisk *Anuluj* (*Nie*), wykonaj instrukcję `return false`, a tym samym zakończ działanie funkcji. Jeżeli nie został przekazany ciąg `strPotwierdź` (bądź ma wartość `false` lub `null`), wykonaj dalsze instrukcje funkcji.

Okno wskazywane przez `wndObj` jest zamykane przez wywołanie metody `close`:

```
wndObj.close();
```

Następnie zwracana jest wartość właściwości `closed` obiektu `wndObj`. Ta właściwość jest równa `true`, jeśli okno faktycznie zostało zamknięte, oraz `false`, jeśli z jakichś powodów okno nie zostało zamknięte (np. było to główne okno przeglądarki, co zostanie wyjaśnione w dalszej części opisu). Ostatecznie funkcja będzie zwracała wartość `true`, jeśli okno wskazywane przez `objWnd` zostało zamknięte, bądź `false`, jeśli nie zostało zamknięte.

Do przetestowania działania funkcji `zamknijOkno` posłuży strona, której kod został przedstawiony na listingu 1.5.

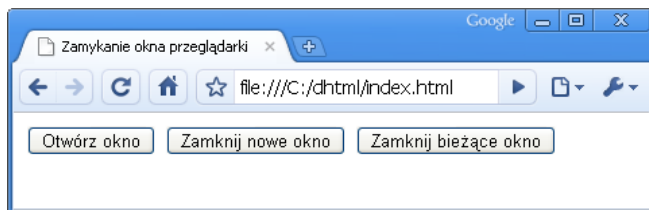
Listing 1.5. Strona testująca funkcję `zamknijOkno`

```
<body>  
  <div id="divParametry">  
    <input type="button" value="Otwórz okno"  
      onclick="btnOtwórzClick();" />  
    <input type="button" value="Zamknij nowe okno"  
      onclick="btnZamknijClick();" />  
    <input type="button" value="Zamknij bieżące okno"  
      onclick="btnZamknijBieżąceClick();" />  
  </div>  
</body>
```

Ta witryna zawiera trzy przyciski. Jej wygląd został zaprezentowany na rysunku 1.4. Każdemu z przycisków została przypisana procedura obsługi zdarzenia `click`. W pierwszym przypadku jest to funkcja `btnOtwórzClick`, w drugim — funkcja `btnZamknijClick`, a w trzecim — funkcja `btnZamknijBieżąceClick()`. Kliknięcie przycisku *Otwórz okno* będzie powodowało otwarcie nowego okna, kliknięcie przycisku *Zamknij nowe okno* będzie powodowało zamknięcie tego okna, natomiast kliknięcie przycisku *Zamknij bieżące okno* spowoduje podjęcie próby zamknięcia okna bieżącego (czyli głównego). Dzięki tej ostatniej opcji będzie można się przekonać, które przeglądarki pozwalają na taką operację.

Rysunek 1.4.

*Przyciski sterujące
otwieraniem
i zamykaniem okien*



Skrypt zawierający procedury obsługi zdarzeń wykonujące opisane czynności został przedstawiony na listingu 1.6.

Listing 1.6. Procedury obsługi zdarzeń dla przycisków

```
<script type="text/javascript">
  var okno = null;
  function zamknijOkno(wndObj, strPotwierdź)
  {
    // tutaj kod funkcji
  }
  function btnZamknijClick()
  {
    if(zamknijOkno(okno, "Czy na pewno chcesz zamknąć nowe okno?"))
      okno = null;
  }
  function btnZamknijBieżąceClick()
  {
    zamknijOkno(window, false);
  }
  function btnOtwórzClick()
  {
    if(okno == null || okno.closed){
      okno = window.open("about:blank", "", "width=100,height=100");
    }
  }
</script>
```

Na początku kodu znajduje się globalna zmienna `okno` przechowująca odwołanie do obiektu otwieranego okna. Jej pierwotną wartością jest `null`. Zadaniem funkcji `btnOtwórzClick`, wywoływanej po kliknięciu przycisku *Otwórz okno*, jest utworzenie nowego okna. Powinno być ono jednak otwarte tylko wtedy, gdy żadne inne nie zostało wcześniej otwarte, czyli gdy zmienna `okno` ma wartość `null` lub też otwarte wcześniej okno jest w stanie zamkniętym (`okno.closed` równe `true`). Dlatego też wywołanie

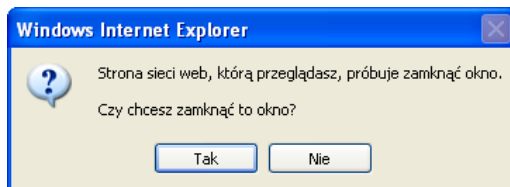
metody `open` (opisanej przy omawianiu poprzedniego skryptu) znajduje się w instrukcji warunkowej. Wynik działania metody `open`, czyli odniesienie do nowo otwartego okna, jest przypisywany zmiennej `okno`.

Funkcja `btnZamknijClick` jest wywoływana w odpowiedzi na kliknięcie przycisku *Zamknij nowe okno*. Oczywiście jej zadaniem jest zamknięcie nowego okna, wywoływana jest więc funkcja `zamknijOkno` i przekazywane są jej dwa parametry. Pierwszy wskazuje okno do zamknięcia (zmienna `okno`), drugi — tekst, który ma się pojawić w oknie z potwierdzeniem chęci wykonania tej operacji. Jeśli procedura zamykania zakończy się sukcesem (funkcja `zamknijOkno` zwróci wartość `true`), zmiennej `okno` zostanie przypisana wartość `null` wskazująca, że okno zostało zamknięte, a zatem procedura otwarcia będzie mogła być ponowiona.

Funkcja `btnZamknijBieząceClick` jest wywoływana po kliknięciu przycisku *Zamknij bieżące okno*. Wywołuje ona funkcję `zamknijOkno`, przekazując jej jako pierwszy argument wartość `window`, a jako drugi — wartość `false`. To oznacza, że funkcja `zamknijOkno` podejmie próbę zamknięcia bieżącego, czyli głównego okna (bieżącej karty) przeglądarki bez wyświetlania prośby o potwierdzenie. To pozwala przetestować, jak zachowują się poszczególne przeglądarki w takiej sytuacji. Łatwo się przekonać, że większość produktów uniemożliwia taką operację. Nie stanie się zatem nic (ostrzeżenie może się pojawić w konsoli JavaScriptu — tak działa np. Firefox) lub też zostanie wyświetlone systemowe okno z ostrzeżeniem. Takie okno, widoczne na rysunku 1.5, wyświetla np. Internet Explorer. Jedynie Opera faktycznie zamyka bieżące okno (kartę) bez żadnego potwierdzenia.

Rysunek 1.5.

*Okno ostrzegające,
że witryna próbuje
zamknąć główne okno*



Skrypt 3. [C][E7][F][O][S] Interfejs do przewijania treści strony (przez kliknięcia)

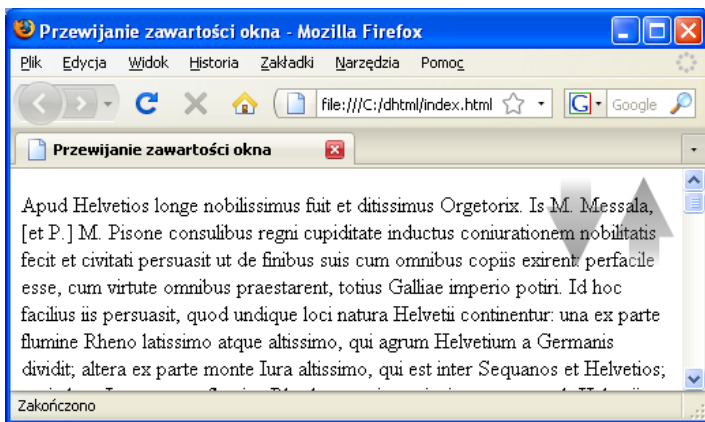
Jeśli zawartość witryny nie mieści się w oknie, może być przesuwana za pomocą pasków przewijania. Czasem jednak przewijanie powinno być zrealizowane w sposób programowy. Na szczęście dodanie do strony interfejsu pozwalającego na wykonanie takiej czynności nie jest trudne. Jak będzie wyglądał interfejs, zależy oczywiście od konkretnych potrzeb. Trzeba tylko wiedzieć, jak przewijać stronę. Służy do tego metoda `scrollBy`. Schemat wywołania jest następujący:

```
scrollBy(px, py)
```

Przesunięcie zawartości okna następuje o px pikseli w poziomie i py pikseli w pionie (tylko wtedy, gdy bieżąca zawartość nie mieści się w oknie).

Przyjmijmy więc, że w prawym górnym rogu okna przeglądarki chcemy umieścić dwie półprzezroczyste (tak by nie zasłaniały zawartości witryny) strzałki pozwalające na przewijanie treści w górę i w dół. Przesunięcie o stałą, z góry zdefiniowaną liczbę pikseli będzie następowało po kliknięciu na jedną lub drugą. Pozycja strzałek będzie oczywiście musiała być stała i niezmienna. Strona będzie więc wyglądała tak, jak zaprezentowano to na rysunku 1.6. Kod takiej strony przyjmie postać przedstawioną na listingu 1.7.

Rysunek 1.6.
Strona z interfejsem pozwalającym na przewijanie



Listing 1.7. *Strona z interfejsem pozwalającym na przewijanie treści*

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN"
"http://www.w3.org/TR/html4/strict.dtd">
<html lang="pl">
  <head>
    <meta http-equiv="Content-Type" content="text/html; charset=utf-8">
    <title>Przewijanie zawartości okna</title>
  </head>
  <body>
    <div id="divScroll">
      
      
    </div>
    <div id="divDane">
      // tutaj treść strony
    </div>
  </body>
</html>
```

W sekcji `<body>` znajdują się dwie warstwy: `divScroll` i `divDane`. Druga służy do umieszczenia treści — najlepiej większej ilości tekstu, tak by można było swobodnie testować omawiany efekt. Na pierwszej warstwie znajdują się z kolei dwa obrazy zdefiniowane

za pomocą znaczników ``. Obrazy są wczytywane z plików o nazwach *dol.png* oraz *gora.png* i powinny zawierać strzałki (skierowane oczywiście w dół i w górę). Te pliki można przygotować w dowolnym programie graficznym. Oba znaczniki `` mają zdefiniowany atrybut `class`, dzięki czemu będzie można przypisać im jednolity styl. Każdy z nich ma też swoją procedurę obsługi zdarzenia `click`. Dla obrazu *dol.png* jest to funkcja `imgDownClick`, a dla obrazu *gora.png* — funkcja `imgUpClick`.

Aby warstwa ze strzałkami znalazła się w odpowiedniej pozycji, niezbędne jest nadanie stylów. Zostały przedstawione na listingu 1.8.

Listing 1.8. *Style CSS dla interfejsu przewijania strony*

```
<style type="text/css">
#divScroll{
    position:fixed;
    top:5px;
    right:5px;
    opacity:0.4;
    filter:alpha(opacity=40)
}
.scrollimgs{
    cursor:pointer;
}
</style>
```

Pierwszy selektor dotyczy warstwy `divScroll`. Nadawane jest jej pozycjonowanie ustalone (`fixed`), a także położenie w odległości 5 pikseli od prawej krawędzi okna i 5 pikseli od górnej krawędzi okna. Dzięki cechom `opacity` oraz `filter:alpha` (charakterystycznej dla przeglądarki Internet Explorer) warstwa, a więc i jej zawartość, będą częściowo przezroczyste. Drugi z selektorów dotyczy elementów z atrybutem `class` o wartości `scrollimgs`, czyli obrazów strzałek. Przypisywany jest im kursor — wskaźnik (`pointer`).

Ostatnim elementem przykładu jest skrypt zawierający procedury obsługi kliknięć dla obrazów strzałek. Kod został przedstawiony na listingu 1.9.

Listing 1.9. *Skrypt przewijający treść witryny*

```
<script type="text/javascript">
    step = 100;
    function imgUpClick()
    {
        window.scrollBy(0, -step);
    }
    function imgDownClick()
    {
        window.scrollBy(0, step);
    }
</script>
```

Zmienna `step` określa liczbę pikseli, o którą zostanie przesunięta treść strony (w górę lub w dół) po pojedynczym kliknięciu. Funkcja `imgUpClick` jest wywoływana po kliknięciu strzałki w górę, wywołuje więc metodę `scrollBy`, przekazując jej jako drugi argument wartość `-step` (czyli `-100`). Funkcja `imgDownClick` jest wywoływana po kliknięciu strzałki w dół, wywołuje więc metodę `scrollBy`, przekazując jej jako drugi argument wartość `step` (czyli `100`). To pozwala na stopniowe przewijanie treści witryny. Pierwszy argument metody `scrollBy` jest stale równy `0`, ponieważ treść nie ma być przesuwana w poziomie.

Skrypt 4. [C][E7][F][O][S]

Automatyczne przewijanie treści strony

Skrypt 3. pozwalał na przewijanie treści strony za pomocą kliknięć. Ta czynność może być jednak zautomatyzowana. Przy identycznym interfejsie możemy spowodować, aby ruch następował już po najechaniu kursorem myszy na jedną ze strzałek. Będzie to wymagało zmiany kodu warstwy sterującej `divScroll`, a także napisania nowego skryptu. Kod warstwy przyjmie postać widoczną na listingu 1.10.

Listing 1.10. Treść warstwy ze strzałkami do przewijania treści

```
<div id="divScroll">
  
  
</div>
```

Wszystkie elementy pozostały takie same jak w skrypcie 3., natomiast zamiast zdarzenia `click` obsługiwane są zdarzenia `mouseover` i `mouseout`. Tym samym po najechaniu kursorem myszy na strzałkę z obrazka `dol.png` zostanie wywołana funkcja `startScroll` i zostanie jej przekazany parametr `down` (ciąg znaków `down`), natomiast po najechaniu kursorem myszy na strzałkę z obrazka `gora.png` również zostanie wywołana funkcja `startScroll`, ale zostanie jej przekazany parametr `up` (ciąg znaków `up`). Gdy kursor myszy znajdzie się poza jednym bądź drugim obrazem, zostanie z kolei wywołana funkcja `stopScroll`.

Treść skryptu zawierającego obie wspomniane funkcje jest widoczna na listingu 1.11.

Listing 1.11. Skrypt przesuwający treść witryny

```
<script type="text/javascript">
  var step = 2;
  var speed = 10;
```



```
var direction = 'down';
var tid = null;
function scroll()
{
    if(direction == 'up')
        scrollBy(0, -step);
    else
        scrollBy(0, step);
}
function startScroll(dir)
{
    direction = dir;
    tid = setInterval('scroll();', speed);
}
function stopScroll()
{
    clearInterval(tid);
}
</script>
```

Na początku znajdują się definicje zmiennych sterujących. Są to:

- ◆ `step` — określa liczbę pikseli, o jaką będzie przesuwana treść strony w każdej fazie ruchu; im wyższa wartość, tym większy skok, a tym samym szybszy, ale i mniej płynny przesuw;
- ◆ `speed` — określa czas między kolejnymi przesunięciami tekstu, a tym samym szybkość animacji;
- ◆ `direction` — określa kierunek ruchu; `up` — w górę, `down` — w dół;
- ◆ `tid` — przechowuje identyfikator timera niezbędnego do animacji przesuwu; początkowa wartość to `null`.

Funkcja `scroll` odpowiada za przesuwanie treści witryny. Jedno wywołanie tej funkcji to pojedyncze przesunięcie treści o liczbę pikseli zapisaną w zmiennej `step`. Przesunięcie jest wykonywane za pomocą metody `scrollBy`. O kierunku przesunięcia decyduje wartość globalnej zmiennej `direction`. Gdy `direction` zawiera ciąg znaków `up`, przesunięcie następuje w górę, a więc wykonywana jest instrukcja `scrollBy(0, -step);`, a w każdym innym przypadku — instrukcja `scrollBy(0, step);`.

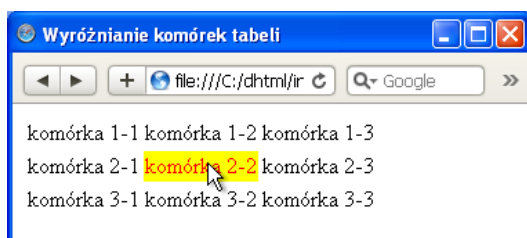
Funkcja `startScroll` przyjmuje jeden argument określający kierunek przesuwu treści strony i przypisuje go do zmiennej globalnej `direction`. Następnie wywołuje metodę `setInterval`, która powoduje cykliczne wywoływanie funkcji `scroll`. Czas (w milisekundach) między kolejnymi wywołaniami jest określany przez wartość zmiennej `speed`. Rezultat wywołania `setInterval`, czyli identyfikator timera, jest przypisywany zmiennej `tid`. Dzięki temu w funkcji `stopScroll` timer może być zatrzymany poprzez wywołanie funkcji `clearInterval`.

Skrypt 5. [C][E7][F][O][S]

Podświetlanie komórki tabeli lub innego elementu witryny (CSS)

Czasami chcemy wyróżnić element czy też fragment witryny, nad którym aktualnie znajduje się kursor myszy. Taki efekt przydaje się np. w przypadku tabel. Można wtedy wyróżnić wskazywaną w danej chwili komórkę, tak jak zostało to zaprezentowane na rysunku 1.7. Najprościej zrealizować to za pomocą zwykłych stylów CSS. Przygotujmy zatem tabelę, której komórki będą wyróżniane za pomocą żółtego tła i czerwonych liter. Tak działający kod został zaprezentowany na listingu 1.12.

Rysunek 1.7.
Wyróżniona komórka tabeli



Listing 1.12. *Wyróżnianie wskazanych komórek tabeli*

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN"
"http://www.w3.org/TR/html4/strict.dtd">
<html lang="pl">
<head>
  <meta http-equiv="Content-Type" content="text/html; charset=utf-8">
  <title>Wyróżnianie komórek tabeli</title>
  <style type="text/css">
    td:hover{
      background-color:yellow;
      color:red;
    }
    td{
      cursor:default;
    }
  </style>
</head>
<body>
  <table>
  <tr>
    <td>komórka 1-1</td><td>komórka 1-2</td><td>komórka 1-3</td>
  </tr>
  <tr>
    <td>komórka 2-1</td><td>komórka 2-2</td><td>komórka 2-3</td>
  </tr>
  <tr>
    <td>komórka 3-1</td><td>komórka 3-2</td><td>komórka 3-3</td>
  </tr>
```

```
</table>
</body>
</html>
```

W sekcji `<body>` znajduje się zwyczajna tabela zdefiniowana za pomocą znaczników `<table>`, `<tr>` i `<td>`. Nie ma ona żadnych specjalnych właściwości. Komórki zdefiniowane za pomocą znacznika `<td>` będą jednak wyróżniane za pomocą stylów CSS. Zmieniony zostanie też kształt kursora znajdującego się nad komórkami — będzie to strzałka (widać to na rysunku 1.7). Te efekty są realizowane za pomocą reguł CSS umieszczonych w znaczniku `<style>`.

Kształt znajdującego się nad komórkami kursora jest określany przez regułę z selektorem `td`. Definicja:

```
cursor:default;
```

określa ten kształt na domyślny, czyli strzałkę (bez tej definicji nad komórkami zawierającymi tekst pojawiałby się kursor tekstowy o kształcie wielkiej litery I). Wyróżnienie komórek zapewnia reguła z selektorem `td:hover`. Użyta w nim pseudoklasa `hover` powoduje, że umieszczone w tym selektorze definicje będą obowiązywały tylko wtedy, gdy nad daną komórką zostanie umieszczony kursor. Wtedy kolor tła zmieni się na żółty (`background-color:yellow;`), a kolor tekstu na czerwony (`color:red;`).

Skrypt 6. [C][E][F][O][S] Podświetlanie komórki tabeli lub innego elementu witryny (JavaScript)

Poprzedni skrypt pozwalał wyróżnić komórkę tabeli (a także dowolny inny element witryny) wyłącznie za pomocą stylów CSS. Ta metoda, choć bardzo prosta, nie zawsze się sprawdzi. Niezbędne może być bowiem wykonanie bardziej skomplikowanych akcji niż tylko prosta zmiana stylu. Skrypt 5. nie działał również w przeglądarce Internet Explorer 6, bowiem ta wersja nie obsługuje prawidłowo pseudoklasy `hover`. Nic jednak nie stoi na przeszkodzie, aby zmieniać style za pomocą JavaScriptu. Wystarczy wykorzystać zdarzenia `mouseover` (wykonywane, gdy kursor znajdzie się nad elementem) i `mouseout` (wykonywane, gdy kursor opuści obszar elementu). Tabela ze skryptu 5. musiałaby więc zostać zmieniona tak, jak przedstawiono to na listingu 1.13.

Listing 1.13. Tabela z obsługą zdarzeń `onmouseover` i `onmouseout`

```
<body>
  <table>
    <tr>
      <td onmouseover="tdmover(this);" onmouseout="tdmout(this);">komórka 1-1</td>
      <td onmouseover="tdmover(this);" onmouseout="tdmout(this);">komórka 1-2</td>
```

```
<td onmouseover="tdmover(this);" onmouseout="tdmout(this);">komórka 1-3</td>
</tr>
<!-- dalsze wiersze tabeli -->
</table>
```

Ogólna konstrukcja tabeli pozostała taka sama jak w przypadku skryptu 5., każda komórka zyskała jednak atrybuty `onmouseover` i `onmouseout` definiujące procedurę obsługi zdarzeń. Dla zdarzenia `mouseover` jest to funkcja `tdmover`, a dla zdarzenia `mouseout` — funkcja `tdmout`. Należy zwrócić uwagę na sposób wywoływania tych funkcji. Każda otrzymuje argument określający komórkę tabeli, która jest inicjatorem zdarzenia. W tym przypadku słowo `this` to synonim komórki, która zapoczątkowała zdarzenia, czyli tej, nad którą znalazł się kursor myszy, lub tej, nad której kursor został usunięty.

Skrypt z funkcjami `tdmover` i `tdmout` zmieniającymi style wskazanych elementów znajduje się w listingu 1.14.

Listing 1.14. *Skrypt zmieniający style komórek tabeli*

```
<script type="text/javascript">
  function tdmover(obj)
  {
    if(obj){
      obj.style.backgroundColor = "yellow";
      obj.style.color = "red";
    }
  }
  function tdmout(obj)
  {
    if(obj){
      obj.style.backgroundColor = "white";
      obj.style.color = "black";
    }
  }
</script>
```

Ogólna konstrukcja obu funkcji jest taka sama. Otrzymują w postaci argumentu obiekt, którego styl ma zostać zmodyfikowany (w tym przykładzie jest to zawsze komórka tabeli, ale mógłby to być też dowolny inny element witryny). Za pomocą instrukcji `if` badane jest, czy argument `obj` faktycznie zawiera jakąś wartość (nie jest to `null`, `undefined` czy pusty ciąg znaków). Jeśli tak jest, następuje odwołanie do właściwości `style` obiektu wskazywanego przez `obj`. Ponieważ obiekt ten odzwierciedla styl elementu strony WWW, za jego pomocą można zmodyfikować styl. Zmieniane są więc cechy `backgroundColor` oraz `color`. Należy przy tym pamiętać, że o ile w CSS cechy dwuczłonowe (wieloczłonowe) zapisywane są z kreską oddzielającą, np. `background-color`, to odpowiadające im właściwości w obiekcie `style` pisaną są razem, a poszczególne członki (z wyjątkiem pierwszego) rozpoczynają się wielkimi literami, np. `backgroundColor`.

Skrypt 7.

[C][E][F][O][S]

Rozpoznanie typu przeglądarki

Aby rozpoznać typ przeglądarki używanej przez użytkownika, najlepiej odwołać się do właściwości `userAgent` obiektu `navigator`. Znajdziemy w niej wszystkie niezbędne informacje nawet wtedy, gdy identyfikacja produktu została zmieniona w jego opcjach konfiguracyjnych (część przeglądarek udostępnia taką możliwość)⁶. Oto kilka przykładowych opisów⁷:

```
Mozilla/5.0 (Windows; U; Windows NT 6.0; pl; rv:1.9.0.3) Gecko/2008092417  
Firefox/3.0.3  
Mozilla/4.0 (compatible; MSIE 6.0; Windows NT 5.1; SV1; .NET CLR 3.5.21022; FDM)  
Opera/9.51 (X11; Linux i686; U; Fedora; en)
```

W pierwszej wydają się nieczytelne, ale bliższa analiza pozwoli się zorientować, że:

- ♦ w pierwszym przypadku mamy do czynienia z przeglądarką Firefox w wersji 3.0.3 pracującą pod kontrolą systemu Windows Vista (NT 6.0);
- ♦ w drugim przypadku mamy do czynienia z przeglądarką Internet Explorer w wersji 6.0 (MSIE 6.0) pracującą pod kontrolą systemu Windows XP (NT 5.1);
- ♦ w trzecim przypadku mamy do czynienia z przeglądarką Opera w wersji 9.51 pracującą pod kontrolą systemu Linux (dystrybucja Fedora).

To pozwala napisać skrypt rozpoznający ze sporym prawdopodobieństwem typ przeglądarki. Wystarczy sprawdzać, czy w ciągu znajdującym się we właściwości `userAgent` znajdują się charakterystyczne ciągi znaków. Takie zadanie realizuje funkcja przedstawiona na listingu 1.15.

Listing 1.15. Funkcja pobierająca kod przeglądarki

```
function getBrowserType()  
{  
    var agent = navigator.userAgent.toLowerCase();  
  
    if(agent.indexOf('firefox') != -1){  
        return "ff";  
    }  
    else if(agent.indexOf('opera') != -1){  
        return "op";  
    }  
    else if(agent.indexOf('chrome') != -1){  
        return "ch";  
    }  
}
```

⁶ Niestety, w nielicznych przypadkach spotkamy się także z sytuacją, kiedy przeglądarka nie udostępnia prawidłowych informacji we właściwości `userAgent` (może się tam znajdować np. pusty ciąg znaków).

⁷ Pod adresem <http://www.user-agents.org/> znajduje się internetowa baza danych, w której można znaleźć wiele innych przykładów.

```
else if(agent.indexOf('safari') != -1){
    return "sa";
}
else if(agent.indexOf('msie') != -1){
    return "ie";
}
else{
    return "un";
}
}
```

Najpierw funkcja odczytuje wartość zapisaną we właściwości agent obiektu navigator i poddaje ją działaniu metody toLowerCase. W ten sposób wszystkie litery w ciągu zostaną zmienione na małe, co ułatwi dalszą analizę. Przekształcony ciąg jest zapisywany w zmiennej pomocniczej agent. Złożona instrukcja warunkowa if...else if bada następnie, czy w zmiennej agent występuje jeden z charakterystycznych ciągów znaków pozwalający na ustalenie typu przeglądarki. Do ustalenia, czy dany ciąg występuje, używana jest metoda indexOf. Efektem działania funkcji jest kod określający przeglądarkę. Przyjęto następujące oznaczenia:

- ♦ ch — Chrome,
- ♦ ff — Firefox,
- ♦ ie — Internet Explorer,
- ♦ op — Opera,
- ♦ sa — Safari,
- ♦ un — nieznana przeglądarka.

Taka funkcja może być użyta np. do zainicjowania globalnej zmiennej określającej typ przeglądarki. W zależności od stanu tej zmiennej może wtedy wykonywać różne fragmenty kodu dla różnych typów przeglądarek. Sposób użycia funkcji do wyświetlenia nazwy przeglądarki używanej przez użytkownika witryny został przedstawiony na listingu 1.16.

Listing 1.16. *Strona wyświetlająca typ przeglądarki*

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN"
"http://www.w3.org/TR/html4/strict.dtd">
<html lang="pl">
<head>
  <meta http-equiv="Content-Type" content="text/html; charset=utf-8">
  <title>Rozpoznanie typu przeglądarki</title>
  <script type="text/javascript">
    var browserCode = getBrowserType();
    function getBrowserType()
    {
      //treść funkcji
    }
    function printBrowserType()
```

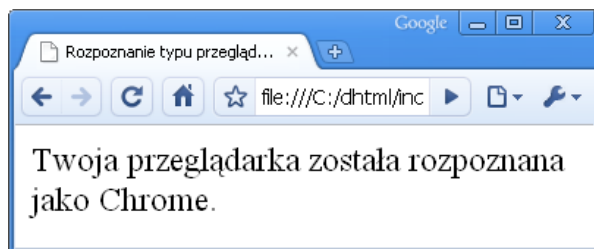
```
{
  var typ = "nieznana";
  switch(browserCode){
    case 'ch' : typ = "Chrome";break;
    case 'ie' : typ = "Internet Explorer";break;
    case 'ff' : typ = "Firefox";break;
    case 'op' : typ = "Opera";break;
    case 'sa' : typ = "Safari";break;
  }
  document.write(typ);
}
</script>
</head>
<body>
  <div>
    Twoja przeglądarka została rozpoznana jako
    <script type="text/javascript">printBrowserType();</script>.
  </div>
</body>
</html>
```

W sekcji <body> została umieszczona warstwa ze zwykłym tekstem, w który został wpleciony fragment kodu JavaScript. Jest to wywołanie funkcji `printBrowserType`. Jej zadaniem jest wyświetlenie nazwy rozpoznanej przeglądarki. Definicja funkcji znajduje się w sekcji <head> w znaczniku <script> razem z opisaną wyżej funkcją `getBrowserType` i pozostałą częścią skryptu.

Na początku skryptu została zdefiniowana zmienna `browserCode` i został jej przypisany wynik wykonania funkcji `getBrowserType`. Ponieważ instrukcja inicjująca zmienną zostanie wykonana automatycznie po załadowaniu strony do przeglądarki, w całym dalszym skrypcie będzie dostępny kod rozpoznanej przeglądarki.

Funkcja `printBrowserType` bada po prostu stan zmiennej globalnej `browserCode`. W tym celu jest używana instrukcja wyboru `switch`. Nazwa przeglądarki (rozpoznana na podstawie kodu) jest przypisywana zmiennej pomocniczej `type`, która jest następnie używana jako argument metody `write` obiektu `document`. Dzięki temu pojawia się na ekranie w miejscu wywołania funkcji `printBrowserType`. Przykładowy efekt działania skryptu został przedstawiony na rysunku 1.8.

Rysunek 1.8.
*Rozpoznanie typu
przeglądarki*



Skrypt 8. [C][E][F][O][S]

Strona zależna od typu przeglądarki

Jeśli umiemy rozpoznać przeglądarkę, możemy przygotować skrypt, który będzie wczytywał różne wersje witryny w zależności od tego, z jakiego produktu korzysta użytkownik odwiedzający stronę. Sposób rozpoznania typu przeglądarki został przedstawiony w skrypcie 7. Służyła do tego celu funkcja `getBrowserType`. Z powodzeniem można jej też użyć w tym skrypcie. Trzeba będzie przygotować pliki z wersjami witryny dla każdej z rozpoznanych przeglądarek (lub grup przeglądarek). Sposób działania będzie następujący:

- ♦ Strona główna zostanie zapisana w pliku *index.html*.
- ♦ Na stronie głównej będzie umieszczony skrypt rozpoznający przeglądarkę.
- ♦ Po rozpoznaniu będzie wczytywany plik HTML odpowiedni dla danej przeglądarki.
- ♦ Jeśli rozpoznanie nie uda się, zostanie wyświetlona lista odnośników umożliwiających użytkownikowi samodzielny wybór wersji witryny.

Tak działający kod został przedstawiony na listingu 1.17.

Listing 1.17. *Różne witryny dla różnych przeglądarek*

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN"
"http://www.w3.org/TR/html4/strict.dtd">
<html lang="pl">
<head>
  <meta http-equiv="Content-Type" content="text/html; charset=utf-8">
  <title>Strona dla wybranej przeglądarki</title>
  <script type="text/javascript">
    redirect();
    function getBrowserType()
    {
      //kod funkcji
    }
    function redirect()
    {
      switch(getBrowserType()){
        case 'ch' : location.href="index_ch.html";break;
        case 'ie' : location.href="index_ie.html";break;
        case 'ff' : location.href="index_ff.html";break;
        case 'op' : location.href="index_op.html";break;
        case 'sa' : location.href="index_sa.html";break;
      }
    }
  </script>
</head>
<body>
  <div>
    <a href="index_ch.html">Strona dla Chrome</a>
    <a href="index_ff.html">Strona dla Firefoksa</a>
```



```
<a href="index_ie.html">Strona dla Internet Explorera</a>
<a href="index_op.html">Strona dla Opery</a>
<a href="index_sa.html">Strona dla Safari</a>
</div>
</body>
</html>
```

W sekcji `<body>` została umieszczona warstwa wraz z listą odnośników do różnych wersji strony. Zostanie wyświetlona, gdy nie uda się rozpoznać przeglądarki lub też z innych powodów skrypt nie zadziała. Przyjęto, że poszczególne wersje witryny znajdują się w plikach o nazwach zgodnych ze schematem `index_kod.html`. Użyte zostały kody przeglądarek podane w opisie skryptu 1.7.

Skrypt umieszczony w sekcji `<head>` korzysta z funkcji `getBrowserType` przedstawionej w skrypcie 7. Za dokonanie przekierowania, czyli wczytanie strony odpowiedniej dla rozpoznanej przeglądarki, odpowiada funkcja `redirect`, która jest wywoływana na samym początku kodu. W jej wnętrzu znajduje się instrukcja wyboru `switch` badająca wartość zwróconą przez wywołanie `getBrowserType()`. W kolejnych blokach `case` wczytywane są pliki odpowiadające poszczególnym przeglądarkom. Wczytanie pliku HTML jest osiągane dzięki przypisaniu jego nazwy właściwości `href` obiektu `location`, schematycznie:

```
location.href = "nazwa_pliku";
```

Skrypt 9.

[C][E][F][O][S]

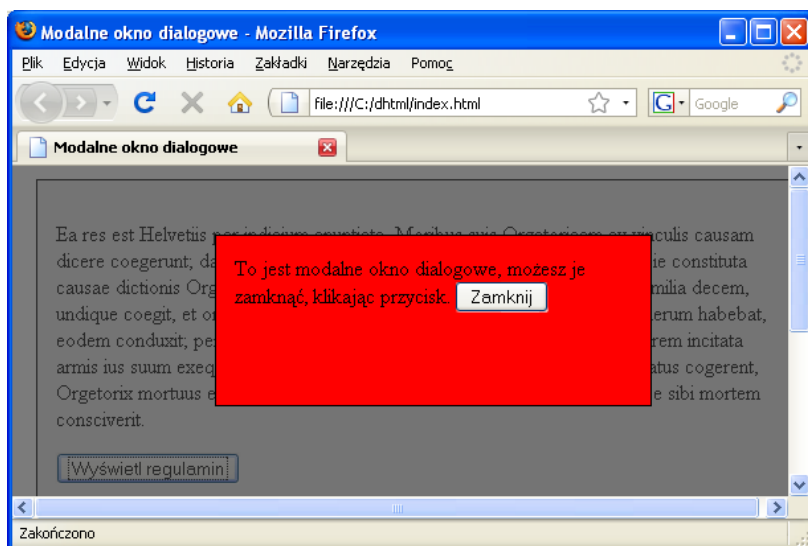
Modalne okno dialogowe

Często chcemy wyświetlić na stronie modalne okno dialogowe, czyli takie, które na czas swojej obecności na ekranie blokuje pozostałe elementy witryny. Może to być okno z ważnym komunikatem, potwierdzeniem wykonania pewnej operacji, regulaminem, który trzeba zaakceptować itp. Niegdyś do takich celów używane były tylko standardowe okna potwierdzeń i komunikatów (czy też nowo otwierane okna przeglądarki) generowane za pomocą standardowych funkcji `alert` czy `prompt`. Obecnie bardzo często stosowane są w tym celu zwykłe warstwy, których wystrój można jednak przygotować tak, aby przypominały zwykłe okno systemowe. Kwestia wystroju nie jest tu najważniejsza, każdy musi go opracować we własnym zakresie, tak by tworzone z warstwy okno pasowało do pozostałych części witryny. Jak jednak spowodować modalność warstwy? Wystarczy odpowiednio manipulować sposobem wyświetlania i cechą `z-index`.

Skrypt będzie zatem działał w taki sposób, że kliknięcie znajdującego się na stronie przycisku spowoduje wyszarzenie i zablokowanie całej zawartości oraz wyświetlenie okna (warstwy) z komunikatem (rysunek 1.9)⁸. Obok komunikatu będzie dostępny

⁸ To rozwiązanie nie będzie działało w pełni poprawnie w przypadku przeglądarki Internet Explorer w wersji wcześniejszej niż 7. i stron zawierających elementy formularzy (pola tekstowe, pola wyboru) ze względu na nieprawidłowy sposób wyświetlania takich elementów w tych produktach.

Rysunek 1.9.
*Modalna warstwa
 przesłaniająca
 główną treść
 witryny*



przycisk pozwalający na zamknięcie okna i przywrócenie poprzedniego stanu witryny. Kod sekcji `<body>` takiej witryny przyjmie postać widoczną na listingu 1.18.

Listing 1.18. Treść sekcji `<body>` skryptu 9.

```
<body>
  <div id="transparentDiv">
  </div>
  <div id="dialogDiv">
    To jest modalne okno dialogowe, możesz je zamknąć, klikając przycisk.
    <input type="button" value="Zamknij" id="btnClose"
      onclick="closeModal();" />
  </div>
  <div id="mainDiv">
  <p>Treść akapitu</p>
  <input type="button" value="Wyświetl regulamin" class="myButton"
    id="btn1" onclick="showModal(300, 100);" />
  <p>Treść akapitu</p>
  </div>
</body>
```

Główna część strony została umieszczona w warstwie o identyfikatorze `mainDiv`, która zawiera dwa akapity tekstowe oraz przycisk. Kliknięcie przycisku będzie wywoływało okno dialogowe (czyli modalną warstwę). Za wykonanie tej czynności będzie odpowiedzialna funkcja `showModal`. Podane w jej wywołaniu parametry liczbowe określają szerokość i wysokość okna.

Do uzyskania opisywanego efektu niezbędne były dwie dodatkowe warstwy. Pierwsza ma identyfikator `transparentDiv` i jej zadaniem jest przykrycie strony, tak by cała istniejąca zawartość stała się nieaktywna. Ta warstwa może być całkowicie lub częściowo przezroczysta. Będą o tym decydowały style CSS. Druga z warstw (o identyfikatorze

dialogDiv) to właśnie nasze okno dialogowe. Zostanie wyświetlona nad warstwą przykrywającą treść witryny (transparentDiv). Dzięki temu zawarte na niej elementy będą widoczne i aktywne. Na warstwie dialogDiv znajduje się przycisk *Zamknij*, któremu została przypisana procedura obsługi zdarzenia click w postaci funkcji closeModal. Ta funkcja będzie zamykała okno dialogowe, a więc i przywracała domyślny stan witryny.

Aby warstwy spełniały swoje zadanie, trzeba im przypisać style CSS. Odpowiednie reguły CSS zostały przedstawione na listingu 1.19.

Listing 1.19. *Style CSS dla modalnych warstw*

```
<style type="text/css">
#mainDiv
{
    background-color: #EFEFEF;
    border: 1px solid #000000;
    margin: 10px;
    padding: 14px;
    width: 95%;
}
#transparentDiv
{
    filter:alpha(opacity=70);
    opacity:0.7;
    background-color:#404040;
    z-index:1000;
    position:absolute;
    top:0px;
    left:0px;
    display:none;
}
#dialogDiv
{
    background-color: red;
    border: 1px solid #000000;
    margin: 10px;
    padding: 14px;
    position:absolute;
    z-index:2000;
    display:none;
}
</style>
```

Reguła dotycząca głównej warstwy jest tylko przykładowa. Zastosowane w niej definicje określają jedynie podstawowe parametry wyświetlania: kolor tła, obramowanie, marginesy, szerokość. Te parametry można zmieniać dowolnie. Dużo ważniejsze są dwie kolejne reguły. Ta z selektorem #transparentDiv określa zachowanie warstwy przykrywającej całą treść witryny (tak by zrealizować funkcję modalności). Dlatego ważne jest jej położenie (lewy górny róg ma współrzędne $x=0$ i $y=0$), sposób pozycjonowania (bezwzględny — w nowych przeglądarkach można wręcz zastosować pozycjonowanie ustalone (fixed), które jednak nie zadziała poprawnie w Internet Explorerze 6) oraz właściwość z-index. Wartość z-index musi być większa niż wartości z-index wszystkich innych (niemodalnych) elementów witryny. Została również

ustalona przezroczystość warstwy na 70 proc. Użyte zostały definicje zgodne ze standardem CSS (`opacity:0.7`), a także stosowana w przeglądarkach Internet Explorer (`filter:alpha(opacity=70)`).

Reguła z selektorem `#dialogDiv` dotyczy bezpośrednio warstwy tworzącej okno dialogowe. Ta warstwa będzie pozycjonowana względem warstwy `transparentDiv` (`position:absolute`), ale musi być wyświetlona nad nią. Dlatego też cecha `z-index` ma wartość 2000 (oczywiście w tym przypadku wartość 1001 byłaby równie dobra). Ponieważ obie omawiane warstwy mają być wyświetlane tylko na żądanie, zostały wstępnie ukryte przez zastosowanie definicji `display:none`.

Skrypt wyświetlający i zamykający tak przygotowane okno dialogowe został przedstawiony na listingu 1.20.

Listing 1.20. *Skrypt sterujący wyświetlaniem i chowaniem warstw*

```
<script type="text/javascript">
var dialogWidth = 300;
var dialogHeight = 200;
function showModal(dW, dH)
{
    var dialogDiv = document.getElementById("dialogDiv");
    var transparentDiv = document.getElementById("transparentDiv");

    if(!dialogDiv) return;
    if(!transparentDiv) return;
    if(!dW) var dW = dialogWidth;
    if(!dH) var dH = dialogHeight;

    dialogDiv.style.width = dW + "px";
    dialogDiv.style.height = dH + "px";

    var clientWidth = parseInt(document.documentElement.clientWidth);
    var clientHeight = parseInt(document.documentElement.clientHeight);
    var scrollHeight = parseInt(document.documentElement.scrollHeight);
    var scrollWidth = parseInt(document.documentElement.scrollWidth);

    transparentDiv.style.width = Math.max(scrollWidth, clientWidth) + "px";
    transparentDiv.style.height = Math.max(scrollHeight, clientHeight) + "px";

    var left = Math.floor((clientWidth - dW) / 2);
    var top = Math.floor((clientHeight - dH) / 2);

    dialogDiv.style.top = top + "px";
    dialogDiv.style.left = left + "px";

    dialogDiv.style.display = "block";
    transparentDiv.style.display = "block";
}
function closeModal()
{
    var dialogDiv = document.getElementById("dialogDiv");
    var transparentDiv = document.getElementById("transparentDiv");

    if(!dialogDiv) return;
```

```
if(!transparentDiv) return;

dialogDiv.style.display = "none";
transparentDiv.style.display = "none";
}
</script>
```

Na początku skryptu znajdują się deklaracje dwóch zmiennych określających domyślną szerokość i wysokość okna dialogowego. Te wartości zostaną użyte, jeśli funkcja `showModal` zostanie wywołana bez argumentów. Kod funkcji rozpoczyna się od pobrania za pomocą metody `getElementById` odwołań do warstw `dialogDiv` i `transparentDiv` oraz zapisania ich nazw w zmiennych o nazwach zgodnych z nazwami warstw. Następnie badane jest, czy zmienne zostały utworzone, a ich wartości są różne od `null`, czyli czy warstwy istnieją. W przypadku gdyby warstw nie było, wykonywanie funkcji jest przerywane za pomocą instrukcji `return`. Badane jest również to, czy zostały przekazane argumenty `dW` (szerokość warstwy) i `dH` (wysokość warstwy). Jeśli nie, tworzone są zmienne `dW` oraz `dH` i są im przypisywane zdefiniowane wcześniej w kodzie wartości domyślne.

W kolejnym kroku modyfikowane są właściwości `width` i `height` obiektu `style` warstwy `dialogDiv`, a tym samym ustalana jest wysokość i szerokość okna dialogowego:

```
dialogDiv.style.width = dW + "px";
dialogDiv.style.height = dH + "px";
```

Niezbędne jest także ustalenie rozmiarów półprzezroczystej warstwy przykrywającej główną część dokumentu. To jednak wymaga wykonania kilku dodatkowych operacji, sposób określania rozmiarów zależy bowiem od tego, czy zawartość dokumentu mieści się w oknie przeglądarki, czy też nie. Jeżeli rozmiary dokumentu są większe niż rozmiary okna przeglądarki (obecne są paski przewijania), trzeba przykryć cały obszar dokumentu. Wartości powinny być więc pobierane z właściwości `scrollWidth` i `scrollHeight`. Jeśli natomiast rozmiary dokumentu są mniejsze, trzeba przykryć cały obszar okna (rozmiar dokumentu będzie wtedy mniejszy niż rozmiar okna). Wartości powinny być więc pobierane z właściwości `clientWidth` i `clientHeight`.

Najprostszym rozwiązaniem jest więc pobranie wszystkich wymienionych właściwości, a następnie użycie metody `max` obiektu `Math` do wybrania większej z nich. Dlatego właśnie kod określający rozmiary warstwy `transparentDiv` ma postać:

```
transparentDiv.style.width = Math.max(scrollWidth, clientWidth) + "px";
transparentDiv.style.height = Math.max(scrollHeight, clientHeight) + "px";
```

Zostało przyjęte, że warstwa z oknem dialogowym (`dialogDiv`) znajdzie się w środku okna przeglądarki. A zatem pozycja okna jest określana według wzorów:

```
left = (szerokość przeglądarki - szerokość okna dialogowego) / 2
top = (wysokość przeglądarki - wysokość okna dialogowego) / 2
```

Po wykonaniu wszystkich opisanych wyliczeń oraz określeniu rozmiarów i położeń obie warstwy są wyświetlane przez zmianę właściwości `display` na `block`. Otrzymują więc blokowy sposób wyświetlania.

Zadanie funkcji `closeModal` jest o wiele prostsze. Ma po prostu zamknąć okno dialogowe, czyli spowodować, aby nie były wyświetlane warstwy `transparentDiv` i `dialogDiv`. Pobiera więc najpierw odwołania do tych warstw, a następnie zmienia wartość właściwości `display` na `none`.

Skrypt 10. [C][E][F][O][S] Strona tytułowa (splash screen)

Technikę modalnego okna dialogowego przedstawioną w skrypcie 9. można wykorzystać do utworzenia strony powitalnej (tytułowej, z ang. splash screen). W ten sposób przed przedstawieniem głównej treści witryny można zaprezentować np. bardzo ważną informację, która nie powinna być przeoczona przez użytkownika. Oczywiście tego typu efektu nie można nadużywać. Struktura witryny wyświetlającej stronę tytułową została przedstawiona na listingu 1.21.

Listing 1.21. Strona z ekranem powitalnym

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN"
"http://www.w3.org/TR/html4/strict.dtd">
<html lang="pl">
  <head>
    <meta http-equiv="Content-Type" content="text/html; charset=utf-8">
    <title>Ekran powitalny</title>
    <style type="text/css">
      /* definicje stylów */
    </style>
    <script type="text/javascript">
      var timeout = 10000;

      /* tutaj treść z listingu 1.20 */

      function showSplash()
      {
        showModal(600, 600);
        setTimeout("closeModal();", timeout);
      }
    </script>
  </head>
  <body onload="showSplash();">
    <div id="transparentDiv">
    </div>
    <div id="dialogDiv">
      <!-- tutaj treść ekranu powitalnego -->
      <input type="button" value="Zamknij" id="btnClose"
        onclick="closeModal();" />
    </div>
    <div id="mainDiv">
      <!-- tutaj główna część witryny -->
    </div>
  </body>
</html>
```

Ogólna struktura kodu jest taka sama jak w przykładzie 9. Wprowadzone modyfikacje sprowadzają się do automatycznego wyświetlenia modalnego okna o dowolnej zawartości. Okno jest wyświetlane za pomocą funkcji `showSplash` wywoływanej w zdarzeniu `load` znacznika `<body>`. Wewnątrz funkcji jest wywoływana funkcja `showModal`, wyświetlająca okno, oraz metoda `setTimeout`, uruchamiająca odliczanie czasu, po którym okno zostanie zamknięte. Wywołanie:

```
setTimeout("closeModal();", timeout);
```

oznacza, że po czasie wyznaczonym wartością zmiennej `timeout` zostanie wywołana funkcja `closeModal` zamykająca modalne okno. Zmienna `timeout` dla wygody została zdefiniowana na początku skryptu. Wyznaczany przez nią czas należy podawać w milisekundach.

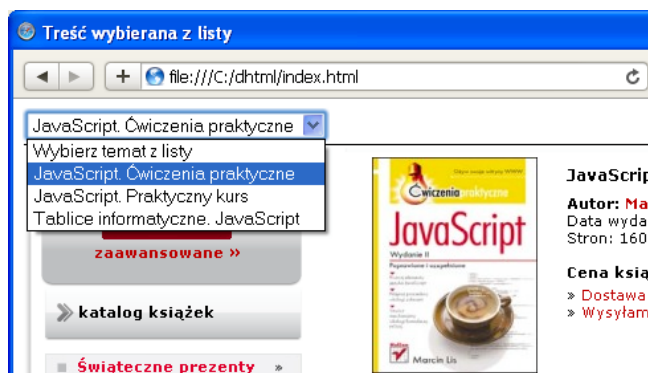
Ta wersja skryptu pozwala również użytkownikowi witryny na samodzielne zamknięcie okna przed upływem zadanego czasu. W warstwie `dialogDiv` został bowiem pozostawiony przycisk *Zamknij*, którego kliknięcie również wywołuje funkcję `closeModal`. Co prawda może to spowodować dwukrotne wywołanie tej funkcji (raz za pomocą przycisku, a drugi raz za pomocą timera), jednak nie stanowi to problemu i nie ma negatywnego wpływu na działanie skryptu.

Skrypt 11. [C][E][F][O][S] Treść wyświetlana w zagnieżdżonym oknie (wybór tematu z listy)

Skrypt 11. umożliwia wyświetlanie na stronie rozmaitych niezależnych treści wybieranych z listy rozwijanej. Mogą to być zarówno podstrony naszego własnego serwisu, jak i inne witryny WWW. Jest to możliwe dzięki temu, że dane są wyświetlane w tzw. pływającej (zagnieżdżonej) ramce, czyli w niezależnym oknie zagnieżdżonym w witrynie. Interfejs strony będzie wyglądał tak, jak zaprezentowano to na rysunku 1.10, i będzie generowany przez kod z listingu 1.22.

Rysunek 1.10.

Menu pozwalające na wczytanie wybranej witryny do ramki



Listing 1.22. Treść sekcji `<body>` skryptu 11.

```
<body>
  <div id="menuDiv">
    <select size="1" id="opcje" onchange="opcjeChange(this);">
      <option value="">Wybierz temat z listy</option>
      <option value="http://helion.pl/ksiazki/cwjas2.htm"
        >JavaScript. Ćwiczenia praktyczne</option>
      <option value="http://helion.pl/ksiazki/jscpk.htm"
        >JavaScript. Praktyczny kurs</option>
      <option value="http://helion.pl/ksiazki/tijs.htm"
        >Tablice informatyczne. JavaScript</option>
    </select>
  </div>
  <iframe width="100%" height="600" id="iframe">
  </iframe>
</body>
```

W sekcji `<body>` znajduje się warstwa o identyfikatorze `menuDiv`, a w niej lista zdefiniowana za pomocą znacznika `<select>`. Jest to lista jednokrotnego wyboru, o czym świadczy atrybut `size` o wartości 1. Zdarzeniu `change`, które powstaje, gdy zmieni się aktywna pozycja listy, przypisano procedurę obsługi w postaci funkcji `opcjeChange`. Funkcji przekazywany jest argument wskazujący obiekt bieżącej listy (`this`). Poszczególne opcje zostały zdefiniowane za pomocą znaczników `<option>`. Każdemu znacznikowi przypisano atrybut `value` wskazujący adres strony WWW, która ma zostać wczytana po wybraniu danej pozycji.

Za listą znajduje się znacznik `<iframe>` definiujący zagnieżdżoną ramkę. Nadano jej identyfikator `iframe`, a także szerokość równą 100 proc. obiektu nadrzędnego (w tym przypadku jest to szerokość dokumentu) oraz wysokość równą 600 pikseli. Aby po wybraniu z listy opcje danej pozycji przypisany jej adres został wczytany do ramki, do strony należy dodać kod JavaScript zaprezentowany na listingu 1.23.

Listing 1.23. Kod wczytujący dane do zagnieżdżonej ramki

```
<script type="text/javascript">
  function opcjeChange(lista)
  {
    if(!lista) return;
    var iframe = document.getElementById('iframe');
    var url = lista[lista.selectedIndex].value;
    if(url){
      iframe.src = url;
    }
  }
</script>
```

Funkcja `opcjeChange` jest wywoływana w odpowiedzi na zmianę stanu listy opcji i otrzymuje argument (`lista`) wskazujący tę listę. Najpierw za pomocą instrukcji warunkowej `if` sprawdzane jest, czy argument faktycznie został przekazany. Jeśli nie został, wykonywanie kodu jest kończone przez wywołanie instrukcji `return`. Jeśli argument został przekazany, za pomocą metody `getElementById` obiektu `document` pobierane

jest odwołanie do obiektu ramki zagnieżdżonej `iframe`. Odwołanie jest zapisywane w zmiennej o takiej samej nazwie jak identyfikator ramki.

Adres URL, który ma być wczytany do ramki, pobierany jest przez odwołanie się do właściwości `value` aktywnego elementu listy. Indeks tego elementu znajduje się we właściwości `selectedIndex`, dlatego też cała instrukcja wykonująca to zadanie ma postać:

```
var url = lista[lista.selectedIndex].value;
```

Po jej wykonaniu zmienna `url` będzie zawierała odczytany adres. Adres ten jest przypisywany właściwości `src` obiektu `iframe`, co powoduje wczytanie wskazywanej przezeń strony do ramki. Dzięki instrukcji warunkowej `if(url)` przypisanie odbywa się tylko wtedy, gdy adres udało się pobrać i nie jest on pustym ciągiem znaków.

Do kodu można też dodać styl zmieniający wygląd zagnieżdżonej ramki. Może to być np. następująca reguła CSS:

```
#iframe{
  border:none;
  border-top:1px solid black;
  margin-top:2px;
  padding-top:2px;
}
```

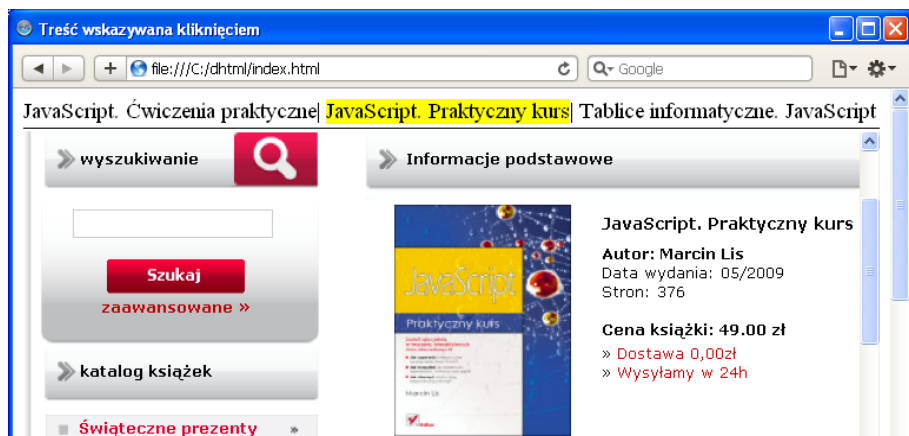
Usuwa ona ramkę okalającą (`border:none;`) z wyjątkiem jej górnej części, która jest określana jako ciągła czarna linia o grubości 1 piksela (`border-top:1px solid black;`)⁹. Dodaje także 2-pikselowy górny margines zewnętrzny (`margin-top:2px;`) i wewnętrzny (`padding-top:2px;`).

Skrypt 12. [C][E][F][O][S] Treść wyświetlana w zagnieżdżonym oknie (temat wybierany kliknięciem)

Skrypt 12., podobnie jak 11., pozwala na wczytywanie do okna zagnieżdżonego w stronie danych z dowolnych adresów WWW. Tym razem jednak, tak jak jest to widoczne na rysunku 1.11, spis tematów będzie stale widoczny w górnej części witryny, przy czym napis aktualnie wskazywany przez kursor myszy będzie podświetlany. Wczytanie danych będzie następowało po kliknięciu danego tematu.

W związku z takim podejściem konstrukcja skryptu będzie musiała być nieco inna niż w przykładzie 11., choć ogólna zasada wczytywania danych do zagnieżdżonej ramki pozostanie oczywiście niezmienna. Sekcja `<body>` przyjmie postać widoczną na listingu 1.24.

⁹ Zmiany te nie są właściwie respektowane przez przeglądarki z rodziny Internet Explorer, gdyż ramka zagnieżdżona ma w tych produktach własne obramowanie niezależne od tego definiowanego za pomocą stylów.



Rysunek 1.11. Lista tematów jest wyświetlana w górnej części witryny

Listing 1.24. Treść sekcji `<body>` skryptu 12.

```
<body>
  <div id="menuDiv">
    <span onclick="go('http://helion.pl/ksiazki/cwjas2.htm')">JavaScript. Ćwiczenia praktyczne</span> |
    <span onclick="go('http://helion.pl/ksiazki/jscpk.htm')">JavaScript. Praktyczny kurs</span> |
    <span onclick="go('http://helion.pl/ksiazki/tijs.htm')">Tablice informatyczne. JavaScript</span>
  </div>
  <iframe width="100%" height="600" id="iframe">
</iframe>
</body>
```

Ponieważ wszystkie wyświetlane tematy mają — w pewnym sensie — pełnić rolę odnośników, muszą być odpowiednio wyróżnione w kodzie. Dlatego też zostały zastosowane znaczniki ``. Dzięki temu wszystkie napisy będą ułożone w poziomie, ale każdy będzie mógł być traktowany jako osobny element. Znacznikom została przypisana klasa `mylink`, co pozwoli zastosować do nich jednolite style CSS, a także procedura obsługi zdarzenia `click`. To oznacza, że po kliknięciu tematu ujętego w znacznik `` zostanie wywołana funkcja o nazwie `go`, argumentem funkcji będzie natomiast przypisany do danego tematu adres strony WWW (URL). Za spisem tematów została umieszczona zagnieżdżona ramka zdefiniowana za pomocą znacznika `<iframe>`.

Treść skryptu JavaScript zawierającego kod funkcji `go` została przedstawiona na listingu 1.25.

Listing 1.25. Kod funkcji wczytującej treść do zagnieżdżonej ramki

```
<script type="text/javascript">
  function go(url)
  {
    if(!url) return;
    var iframe = document.getElementById('iframe');
```

```
        iframe.src = url;
    }
</script>
```

Konstrukcja funkcji jest bardzo prosta. Otrzymuje ona w postaci argumentu adres strony, która ma być wczytana do ramki. Za pomocą instrukcji warunkowej sprawdza więc, czy argument faktycznie został przekazany, a następnie pobiera odwołanie do ramki i zapisuje je w zmiennej `iframe`. W ostatnim kroku zawartość zmiennej `url` (czyli adres strony) jest przypisywana właściwości `src` obiektu `iframe`, co powoduje, że żądana treść pojawia się w oknie.

Do skryptu warto dodać jeszcze style CSS. Styl dotyczący ramki może pozostać taki sam jak w przypadku przykładu 11. Przydatne będą też następujące reguły:

```
.mylink{
    cursor:pointer;
}
.mylink:hover{
    background-color:yellow;
}
```

Pierwsza powoduje, że kursor znajdujący się nad elementami klasy `mylink` (czyli — w tym przykładzie — wyróżnieniami zdefiniowanymi za pomocą znaczników ``) będzie miał kształt wskaźnika. Druga powoduje, że po najechaniu kursorem myszy na elementy klasy `mylink` jego kolor tła (`background-color`) zostanie zmieniony na żółty (`yellow`)¹⁰.

Skrypt 13. [C][E][F][O][S] Treść przedstawiana w symulowanych kartach

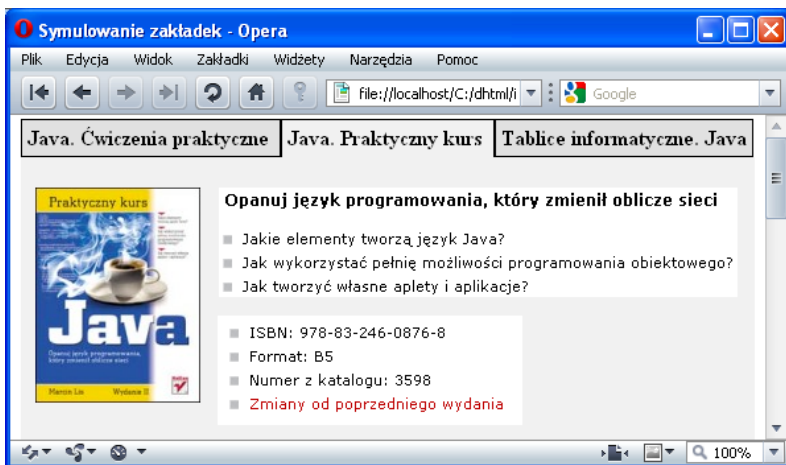
Czasem na stronie trzeba przedstawić wiele informacji z różnych zakresów tematycznych. Przykładowo mogą to być różne części specyfikacji produktu w sklepie internetowym czy informacje dotyczące książek z wybranego zakresu tematycznego. Umieszczenie wszystkich danych w jednym miejscu może być nieczytelne i zajmie wiele miejsca. Lepszym rozwiązaniem jest umożliwienie użytkownikowi witryny wyboru tego, co chce zobaczyć. Stosuje się wtedy rodzaj zakładek czy też kart, np. takich, jakie przedstawiono na rysunku 1.12. Kliknięcie nagłówka karty powoduje wyświetlenie przypisanej jej treści. To zadanie można zrealizować w stosunkowo prosty sposób, stosując umieszczone jedna na drugiej warstwy z danymi i używając JavaScriptu do przełączania się między nimi. Jeśli do tego zastosujemy style CSS i, być może, elementy grafiki, można osiągnąć atrakcyjny efekt wizualny. Wystrój takich kart to jednak kwestia

¹⁰ Ten efekt nie będzie widoczny w przeglądarce Internet Explorer w wersji 6., ze względu na brak poprawnej obsługi w tym produkcie pseudoklasy `hover`.

indywidualnych upodobań oraz dopasowania do interfejsu danej witryny. W przykładzie zostanie więc zastosowany stosunkowo prosty efekt (widoczny na rysunku 1.12). Najważniejszy jest pomysł na konstrukcję takiej witryny i przełączenie kart. Kod sekcji `<body>` przyjmie postać zaprezentowaną na listingu 1.26.

Rysunek 1.12.

Strona prezentująca dane w symulowanych kartach



Listing 1.26. Warstwy użyte do symulacji kart

```
<body>
  <div id="menuDiv">
    <span onclick="show(1);" class="tabh" id="tab1"
      style="background-color:#F0F0F0;border-bottom:1px solid #F0F0F0;">
      Java. Ćwiczenia praktyczne
    </span><span onclick="show(2);" class="tabh" id="tab2">
      Java. Praktyczny kurs
    </span><span onclick="show(3);" class="tabh" id="tab3">
      Tablice informatyczne. Java
    </span>
  </div>
  <div id="dataContainer">
    <div id="data1" class="tabs"
      style="display:block;">
      Treść pierwszej zakładki
    </div>
    <div id="data2" class="tabs" style="display:none">
      Treść drugiej zakładki
    </div>
    <div id="data3" class="tabs" style="display:none">
      Treść trzeciej zakładki
    </div>
  </div>
</body>
```

Na początku została umieszczona warstwa o identyfikatorze `menuDiv`, która zawiera nagłówki kart. Każdy nagłówek składa się ze zwykłego znacznika ``, któremu została przypisana procedura zdarzenia `click` w postaci funkcji `show`, a także klasa `tabh`

oraz unikatowy identyfikator (tab1, tab2 i tab3). Funkcja show w każdym wywołaniu otrzymuje numer karty, która została kliknięta, co pozwoli na przełączenie kart (czyli włączenie klikniętej, a wyłączenie pozostałych).

Przyjęto, że pierwsza karta będzie domyślnie włączona, dlatego też pierwszy nagłówek (tab1) otrzymał osobny styl:

```
style="background-color:#F0F0F0;border-bottom:1px solid #F0F0F0;"
```

Kolor tła został zmieniony na kolor tła karty (#F0F0F0), ten sam kolor otrzymała również dolna część obramowania. Dzięki temu będzie się wydawało, że nagłówki zlewa się z kartą (tak jak w przypadku drugiej karty na rysunku 1.12).

Za warstwą menuDiv znajduje się kolejna, o identyfikatorze dataContainer. To w niej zostały umieszczone „karty”, czyli warstwy z treścią. Każda ma przypisany identyfikator (data1, data2, data3), a także klasę tabs. Ponieważ przyjęliśmy, że po wczytaniu strony ma być widoczna pierwsza karta (data1), została jej przypisana definicja CSS display:block;, dzięki czemu będzie wyświetlana, natomiast pozostałe otrzymały definicję display:none;, co spowoduje, że nie będą wyświetlane.

Aby wszystkie warstwy zostały prawidłowo ułożone, niezbędne jest odpowiednie zdefiniowanie stylów CSS. Oczywiście istnieje w tym pewna dowolność i można je dostosować do własnych potrzeb. Niezbędne jest jednak uwzględnienie kilku czynników. Style mogłyby wyglądać tak jak na listingu 1.27.

Listing 1.27. Style CSS do symulacji kart

```
<style type="text/css">
  #dataContainer{
    position:relative;
    width:100proc;
    height:800px;
    background-color:#F0F0F0;
    margin-top:5px;
  }
  .tabh{
    cursor:pointer;
    border-top:1px solid black;
    border-bottom:1px solid black;
    border-left:1px solid black;
    border-right:1px solid black;
    padding:4px;
    background-color:#E0E0E0;
    font-weight:bold;
  }
  .tabs{
    margin-top:10px;
    position:absolute;
    left:0px;
    top:0px;
  }
</style>
```

Pierwsza reguła dotyczy warstwy-kontenera dla kart (`containerDiv`). O ile jej wysokość, szerokość, kolor czy margines górny można dobrać wedle własnych potrzeb, o tyle należy zwrócić uwagę na sposób pozycjonowania. Musi on współgrać ze sposobem pozycjonowania warstw-kart, czyli tych, które opisuje klasa `tabs`. Ponieważ karty będą pozycjonowane bezwzględnie, tak by móc nałożyć je jedną nad drugą, kontener nie może być pozycjonowany statycznie. W tym przypadku zastosowano pozycjonowanie względne (`position:relative;`).

Druga reguła (z selektorem `.tabh`) dotyczy wyróżnień liniowych tworzących nagłówki tabeli (znaczniki należące do klasy `tabh`). Tu należy zwrócić uwagę na definicję `cursor:pointer;` dzięki której kursor myszy znajdujący się nad nagłówkiem przyjmie postać wskaźnika. To udogodnienie, dzięki któremu użytkownik witryny od razu będzie wiedział, że nagłówek można kliknąć. Zostały także zdefiniowane osobne obramowania dla każdej strony wyróżnienia. Obramowaniem będziemy manipulowali za pomocą funkcji JavaScript.

Trzecia reguła dotyczy warstw-kart. Jak już zostało wspomniane, są one pozycjonowane bezwzględnie (`position:absolute;`), dzięki czemu wszystkie można ustawić w tej samej pozycji. Warstwą, względem której są pozycjonowane, jest `containerDiv` (dzieje się tak dlatego, że warstwa ta ma pozycjonowanie różne od statycznego). Pozycja jest określana przez właściwości `top` i `left`, a więc wyznaczane jest położenie lewego górnego rogu.

Aby przykład zaczął działać zgodnie z opisem, niezbędne jest jeszcze dołączenie kodu JavaScript zaprezentowanego na listingu 1.28.

Listing 1.28. *Kod JavaScript obsługujący karty*

```
<script type="text/javascript">
function show(num)
{
    var data1 = document.getElementById("data1");
    var data2 = document.getElementById("data2");
    var data3 = document.getElementById("data3");

    var tab1 = document.getElementById("tab1");
    var tab2 = document.getElementById("tab2");
    var tab3 = document.getElementById("tab3");

    if(!data1 || !data2 || !data3 || !tab1 ||
       !tab2 || !tab3) return;

    data1.style.display = "none";
    data2.style.display = "none";
    data3.style.display = "none";

    tab1.style.borderBottom = "1px solid black";
    tab2.style.borderBottom = "1px solid black";
    tab3.style.borderBottom = "1px solid black";

    tab1.style.backgroundColor = "#E0E0E0";
    tab2.style.backgroundColor = "#E0E0E0";
```

```
tab3.style.backgroundColor = "#E0E0E0";

if(num < 1 || num > 3) num = 1;

switch(num){
  case 1 : data1.style.display = "block";
          tab1.style.borderBottom = "1px solid #F0F0F0";
          tab1.style.backgroundColor = "#F0F0F0";
          break;
  case 2 : data2.style.display = "block";
          tab2.style.borderBottom = "1px solid #F0F0F0";
          tab2.style.backgroundColor = "#F0F0F0";
          break;
  case 3 : data3.style.display = "block";
          tab3.style.borderBottom = "1px solid #F0F0F0";
          tab3.style.backgroundColor = "#F0F0F0";
          break;
}
}
</script>
```

Na początku funkcji `show` za pomocą metody `getElementById` obiektu `document` pobierane są odwołania do poszczególnych elementów witryny: trzech warstw z danymi (zmienna `dataX`) i trzech nagłówków (zmienna `tabX`). Za pomocą instrukcji warunkowej `if` jest też sprawdzane to, czy pobranie wszystkich odwołań zakończyło się sukcesem. Jeśli nie, wykonywanie kodu jest kończone za pomocą instrukcji `return`. Sprawdzane jest również to, czy argument `num` określający numer klikniętej karty mieści się w prawidłowym zakresie 1 – 3. Jeśli jego wartość jest nieprawidłowa, zostanie zmieniona na 1, co oznacza, że strona zachowa się tak, jakby została kliknięta pierwsza karta.

Oprócz opisanych czynności następuje również „wyzerowanie” właściwości kart i nagłówków — przypisywane są im wartości domyślne. Właściwość `display` warstw `data1` – `data3` otrzymuje wartość `none`, co oznacza, że na krótką chwilę przestaną być wyświetlane (na tyle krótką, że nie będzie to możliwe do zauważenia¹¹). Standardowe wartości otrzymuje także dolna ramka (`borderBottom`) oraz kolor tła nagłówków (`backgroundColor`).

Po wykonaniu opisanych czynności w instrukcji warunkowej `switch` badany jest stan parametru `num` i w zależności od tego, jaką wartość zawiera, włączane jest wyświetlanie odpowiedniej karty. Włączenie wyświetlania odbywa się przez przypisanie wartości `block` właściwości `display` obiektu `style` (co oznacza blokowy sposób wyświetlania). Zmieniany jest także kolor tła i dolnej ramki wybranego nagłówka (na zgodny z kolorem tła karty — `#F0F0F0`), co powoduje, że tło nagłówka zlewa się z kolorem tła karty (oczywiście ten efekt nie jest obligatoryjny i można z niego zrezygnować; wymaga też dokładnego dopasowywania warstw).

¹¹ Alternatywnym rozwiązaniem jest modyfikacja właściwości `z-index`, a tym samym kolejności wyświetlania warstw, bądź też znaczne rozbudowanie kodu i zastosowanie serii instrukcji warunkowych oraz rozpatrywanie wszystkich możliwych sytuacji.

Skrypt 14.

Logowanie użytkowników

[C][E][F][O][S]

Za pomocą JavaScriptu niestety nie jesteśmy w stanie zabezpieczyć naszej witryny tak, aby w pełni skutecznie zabronić dostępu osobom niepowołanym. Cała treść skryptu zabezpieczającego będzie przecież widoczna w kodzie źródłowym strony. Nie należy jednak zabezpieczeń tego typu całkowicie lekceważyć. Zdecydowana większość użytkowników internetu nie jest w stanie sobie z nimi poradzić. Jeśli więc nie tworzymy banku internetowego, takie proste zabezpieczenie, oparte jedynie na języku skryptowym działającym po stronie przeglądarki, może się okazać całkowicie wystarczające. Zdecydowanie lepiej takie autoryzacje przeprowadzać jednak za pomocą skryptów serwera, np. PHP¹².

Jeśli jednak decydujemy się na JavaScript, na stronie głównej trzeba umieścić formularz logowania. Może on przyjąć postać zaprezentowaną na listingu 1.29.

Listing 1.29. Kod formularza logowania

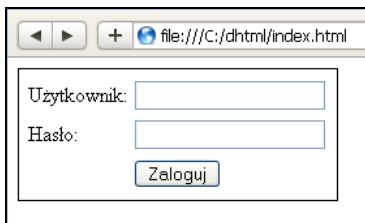
```
<body>
  <div id="loginForm">
    <table border="0" style="border:1px outset black;padding:4px;">
      <tr>
        <td>Użytkownik:</td>
        <td>
          <input type="text" id="user" />
        </td>
      </tr><tr>
        <td>Hasło:</td>
        <td>
          <input type="password" id="pass" />
        </td>
      </tr><tr>
        <td colspan="2" style="text-align:center">
          <input type="button" value="Zaloguj"
            onclick="btnZalogujClick();" />
        </td>
      </tr>
    </table>
  </div>
</body>
```

Formularz składa się z dwóch pól tekstowych umożliwiających wprowadzenie nazwy użytkownika i hasła oraz przycisku wywołującego procedurę weryfikacji danych. Pola tekstowe są generowane za pomocą znaczników `<input>` z atrybutem `type` ustawionym na `text`, a przycisk za pomocą takiego samego znacznika z atrybutem `type` ustawionym na `button`. Przyciskowi została przypisana procedura obsługi zdarzenia `click`

¹² Odpowiednie przykłady można znaleźć w książkach *PHP. 101 praktycznych skryptów* (<http://helion.pl/ksiazki/php102.htm>) i *PHP5. Praktyczny kurs* (<http://helion.pl/ksiazki/php5pk.htm>).

w postaci funkcji `btnZalogujClick`. Pola otrzymały swoje identyfikatory, tak by można się było do nich odwoływać w skrypcie.

Do formatowania formularza została użyta zwykła tabela HTML. Co prawda nie jest to obecnie polecany sposób (zaleca się używanie stylów CSS), ale za to bardzo wygodny i prosty do wykonania. Po wczytaniu zaprezentowanego kodu do przeglądarki zobaczymy widok przedstawiony na rysunku 1.13.

Rysunek 1.13.*Formularz logowania*

Weryfikacją wprowadzonych do formularza danych zajmie się skrypt przedstawiony na listingu 1.30.

Listing 1.30. *Kod formularza logowania*

```
<script type="text/javascript">
  var users = new Array("user1", "user2", "user3");
  var passwords = new Array("pass1", "pass2", "pass3");

  function btnZalogujClick()
  {
    var user = document.getElementById("user").value;
    var pass = document.getElementById("pass").value;
    for(var i = 0; i < users.length; i++){
      if((user == users[i]) && (pass == passwords[i])){
        document.location.href = "strona_glowna.html";
        return;
      }
    }
    alert('Niepoprawne dane!');
  }
</script>
```

Nazwy użytkowników i odpowiadające im hasła zostały zapisane w tablicach `users` i `passwords` w taki sposób, że nazwie w komórce `user[0]` odpowiada hasło w komórce `passwords[0]`, nazwie w komórce `user[1]` — hasło w komórce `passwords[1]` itd.¹³ Weryfikacją danych zajmuje się funkcja `btnZalogujClick` wywoływana po kliknięciu przycisku *Zaloguj*. Funkcja najpierw pobiera wartości zapisane w polach `user` oraz `pass` i zapisuje w zmiennych o nazwach zgodnych z nazwami pól. Następnie w pętli `for` porównuje odczytane dane z kolejnymi wartościami tablic `users` i `passwords`.

¹³ Alternatywnym sposobem byłoby utworzenie pojedynczej tablicy, w której znalazłyby się obiekty z polami zawierającymi nazwę użytkownika i hasło.

Jeżeli w którymś przypadku zostanie stwierdzona zgodność, wczytywana jest strona zapisana w pliku *strona_glowna.html*. Wczytanie odbywa się przez przypisanie nazwy pliku właściwości href obiektu location. W takiej sytuacji jest też przerywane wykonywanie kodu (instrukcja return). Jeżeli pętla for zakończy się bez wykonania kodu występującego w instrukcji if, oznaczać to będzie, że wprowadzone dane są niepoprawne. W takim przypadku za pomocą metody alert wyświetlane jest okno dialogowe z odpowiednią informacją.

Skrypt 15. [C][E][F][O][S]

Dostęp do ukrytej treści

Na witrynie można umieścić fragment, który będzie wyświetlany tylko na wyraźne życzenie użytkownika. Możliwość wyświetlenia takich ukrytych danych można dodatkowo uzależnić od konieczności podania prawidłowego kodu dostępu. Oczywiście nie będzie to profesjonalne zabezpieczenie, gdyż zarówno kod, jak i ukryta treść będą widoczne w kodzie witryny. W ten sposób można jednak np. schować fragment recenzji, który zdradza kluczowe elementy akcji książki bądź filmu, wynik meczu i inne tego typu dane. Na stronie trzeba będzie umieścić elementy interfejsu pozwalające na włączanie i wyłączanie ukrytych danych. Przykładowa witryna będzie więc miała postać przedstawioną na rysunku 1.14. Jest ona generowana przez kod HTML znajdujący się na listingu 1.31.

Rysunek 1.14.
Witryna z interfejsem
do wyświetlania
i ukrywania
części danych



Listing 1.31. Kod sekcji <body> przykładu 15.

```
<body>
  <div id="codeDiv">
    Jeśli chcesz zobaczyć ukrytą część witryny, musisz podać kod:<br />
    <input type="text" id="tfKod" />
    <input type="button" value="Pokaż" onclick="btnPokażClick();"
      id="btnPokaż"/>
```

```
<input type="button" value="Ukryj" onclick="btnUkryjClick();"
      id="btnUkryj" disabled="disabled" />
</div>
<div>
  To jest początek witryny.
</div>
<div id="hiddenDiv" style="display:none;color:blue;">
  Ten tekst znajduje się na schowanej warstwie.
</div>
<div>
  Tutaj znajduje się dalsza część witryny.
</div>
</body>
```

Na początku została umieszczona warstwa o identyfikatorze `codeDiv`. Znajdują się na niej: pole tekstowe umożliwiające wprowadzenie przez użytkownika kodu, a także dwa przyciski. Wszystkie te elementy są generowane za pomocą znaczników `<input>` z odpowiednio ustawionym parametrem `type`. Każdy z nich ma również przypisany atrybut `id` o wartości wskazującej identyfikator danego elementu. Przyciskowi *Pokaż* została przypisana procedura obsługi zdarzenia `click` w postaci funkcji `btnPokażClick`, natomiast procedurą obsługi tego zdarzenia dla przycisku *Ukryj* jest funkcja `btnUkryjClick`. Przycisk *Ukryj* ma również przypisany atrybut `disabled`, co spowoduje, że początkowo będzie wyłączony (nieaktywny).

Za warstwą `codeDiv` znajdują się kolejne, definiujące zawartość witryny. Wśród nich znajduje się warstwa o identyfikatorze `hiddenDiv`. To na niej należy umieścić zawartość, która będzie prezentowana na żądanie. Warstwie została przypisana reguła CSS `display:none`, co spowoduje, że po wczytaniu strony nie będzie widoczna na ekranie. Użyta została również reguła `color:blue`, wyróżniająca tekst warstwy na niebiesko.

Aby osiągnąć efekt widoczny na rysunku 1.14, niezbędne jest dodanie do kodu reguł CSS. Zostały zaprezentowane na listingu 1.32.

Listing 1.32. Style CSS dla skryptu 15.

```
<style type="text/css">
  div{
    margin:2px;
    padding:2px;
    text-align:justify;
  }
  #codeDiv{
    right:2px;
    top:2px;
    border:1px solid black;
    float:right;
    width:200px;
    margin-left:10px;
  }
</style>
```

Wszystkie warstwy otrzymały 2-pikselowy margines wewnętrzny i zewnętrzny, a także wyrównywanie tekstu do prawej i lewej (`text-align:justify;`). Warstwa `codeDiv`, ze względu na jej umiejscowienie, wymagała jednak dodatkowych definicji. Przede wszystkim otrzymała prawy przepływ (`float:right;`), dzięki czemu została dosunięta do prawego brzegu ekranu. Została też odsunięta o 2 piksele od prawej (`right`) i górnej (`top`) krawędzi. Ograniczona została także jej szerokość (`width:200px;`) oraz zmniejszony został lewy margines (`margin-left:10px;`).

Funkcje wywoływane po kliknięciu przycisków *Pokaż* i *Ukryj* zostały przedstawione na listingu 1.33.

Listing 1.33. *Kod funkcji wywoływanych po kliknięciu przycisków Pokaż i Ukryj*

```
<script type="text/javascript">
  function btnPokażClick()
  {
    var tfKod = document.getElementById("tfKod");
    var hiddenDiv = document.getElementById("hiddenDiv");
    var btnPokaz = document.getElementById("btnPokaz");
    var btnUkryj = document.getElementById("btnUkryj");
    if(!tfKod || !hiddenDiv || !btnUkryj || !btnPokaz) return;

    if(tfKod.value == "01234"){
      hiddenDiv.style.display = "block";
      btnUkryj.disabled = false;
      btnPokaz.disabled = true;
    }
    else
      alert("Wprowadzony kod jest niepoprawny!");
  }
  function btnUkryjClick()
  {
    var hiddenDiv = document.getElementById("hiddenDiv");
    var btnPokaz = document.getElementById("btnPokaz");
    var btnUkryj = document.getElementById("btnUkryj");
    if(!hiddenDiv || !btnUkryj || !btnPokaz) return;

    hiddenDiv.style.display = "none";
    btnUkryj.disabled = true;
    btnPokaz.disabled = false;
  }
</script>
```

Funkcja `btnPokażClick` jest wywoływana po kliknięciu przycisku *Pokaż*. Jej zadaniem jest weryfikacja kodu wprowadzonego do pola tekstowego i wyświetlenie ukrytej warstwy bądź komunikatu o niepoprawnych danych. Najpierw są pobierane i zapisywane w zmiennych pomocniczych odwołania do wszystkich niezbędnych elementów witryny: pola tekstowego `tfKod`, warstwy `hiddenDiv`, przycisku `btnPokaz` i przycisku `btnUkryj`. W przypadku gdyby któregoś z elementów nie było, wykonywanie kodu jest przerywane przez wywołanie instrukcji `return`.

Jeżeli jednak elementy interfejsu strony są obecne, pobierana jest wartość zapisana w polu tekstowym `tfKod` (`tfKod.value`) i porównywana z ciągiem znaków `01234` (może być zmieniony na dowolny inny). Jeżeli zostanie stwierdzona zgodność (czyli kod

jest poprawny), następuje odsłonięcie warstwy `hiddenDiv`¹⁴. Ta czynność jest wykonywana przez przypisanie wartości `block` właściwości `display` (czyli ustalenie blokowego sposobu wyświetlania warstwy). Dodatkowo jest wyłączany przycisk Pokaż, a włączany przycisk Ukryj. Włączenie przycisku jest wykonywane przez przypisanie wartości `true` właściwości `disabled`, a wyłączenie — przez przypisanie tej właściwości wartości `false`.

Jeżeli ciąg wprowadzony do pola `tfKod` jest różny od `01234`, za pomocą instrukcji `alert` wyświetlany jest komunikat o błędnych danych. Oczywiście ukryta warstwa nie pojawia się wtedy na ekranie.

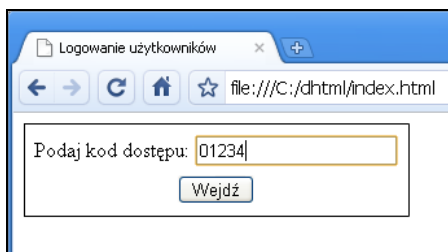
Funkcja `btnUkryjClick` jest wywoływana po kliknięciu przycisku *Ukryj*. Początek kodu jest podobny do funkcji `btnPokażClick` — pobierane są odwołania do niezbędnych elementów. Różnica jest taka, że pomijane jest przy tym pole `tfKod`, które nie jest tu do niczego potrzebne. Funkcja ukrywa warstwę `hiddenDiv` poprzez przypisanie wartości `none` właściwości `style` (tym samym warstwa znika z ekranu), a także wyłącza przycisk *Ukryj* i włącza przycisk *Pokaż*. Te czynności wykonywane są na takiej samej zasadzie jak w przypadku funkcji `btnPokażClick`.

Skrypt 16. [C][E][F][O][S] Hasło zabezpieczające witrynę

Skrypt 14. umożliwiał wyposażenie witryny w prosty system logowania. Nie stanowił on problemu dla nieco bardziej świadomych użytkowników sieci, gdyż wszelkie dane można było znaleźć w kodzie źródłowym strony. Jeśli jednak naprawdę chcemy utrudnić życie osobom niepowołanym do odwiedzin, a nie chcemy stosować skryptów działających po stronie serwera (jak np. PHP), możemy zastosować podejście z pozoru jeszcze prostsze do złamania. Będzie to pojedynczy kod zabezpieczający treść. Po wejściu na stronę użytkownik zobaczy więc formularz widoczny na rysunku 1.15. Złamanie tego kodu wcale jednak nie będzie proste. Nie pojawi się on bowiem w żadnym miejscu kodu źródłowego strony. Zanim omówimy dokładniej zasadę działania skryptu, przyjrzyjmy się najpierw sekcji `<body>` generującej formularz. Została przedstawiona na listingu 1.34.

Rysunek 1.15.

Formularz pozwalający na wprowadzenie kodu dostępowego



¹⁴ Oczywiście z weryfikacji można zrezygnować i po prostu wyświetlać warstwę po kliknięciu przycisku *Pokaż*, a ukrywać ją po kliknięciu przycisku *Ukryj*.

Listing 1.34. Kod formularza pozwalającego na wprowadzenie kodu dostępu

```
<body>
  <div id="loginForm">
    <table border="0" style="border:1px outset black;padding:4px;">
      <tr>
        <td>Podaj kod dostępu:</td>
        <td>
          <input type="text" id="tfKod" />
        </td>
      </tr><tr>
        <td colspan="2" style="text-align:center">
          <input type="button" value="Wejdź"
            onclick="btnWejdźClick();" />
        </td>
      </tr>
    </table>
  </div>
</body>
```

Formularz logowania został umieszczony na warstwie o identyfikatorze `loginDiv` i składa się z jednego pola tekstowego o identyfikatorze `tfKod` oraz jednego przycisku. Oba elementy zostały utworzone za pomocą znacznika `<input>`. Przyciskowi została przypisana procedura obsługi zdarzenia `click` w postaci funkcji `btnWejdźClick`. Będzie wywoływana po kliknięciu przycisku.

Jak spowodować, aby po podaniu prawidłowego hasła użytkownik został przekierowany na właściwą stronę, po podaniu błędnego zobaczył komunikat o błędzie, ale by hasło nie było widoczny w treści witryny? Pomysł jest bardzo prosty — hasło (kod autoryzacji) powinno być jednocześnie nazwą pliku HTML z główną częścią witryny. Aby jednak utrudnić proste ataki typu brute-force, dodatkowo warto poddać hasło działaniu funkcji skrótu, np. SHA1. Wtedy nazwa pliku będzie ciągiem, wydawałoby się, przypadkowych znaków, np.:

```
11904a4e8b77f6242e2d288705023adad00a9310
```

Ten ciąg odpowiada hasłu w postaci `01234`. W JavaScriptcie nie ma wbudowanej funkcji SHA1, ale też nie musimy jej tworzyć własnoręcznie. Dostępnych jest wiele darmowych implementacji. W sekcji `<head>` trzeba więc będzie umieścić odwołanie do pliku z tą funkcją, np.:

```
<script type="text/javascript" src="webtoolkit.sha1.js"></script>
```

W przykładzie zostanie użyty kod udostępniany przez serwis *webtoolkit.info*, ale można też zastosować dowolny inny.

Najprostsza wersja funkcji wykonywanej po kliknięciu przycisku wyglądałaby tak jak na listingu 1.35.

Listing 1.35. Podstawowa wersja funkcji `btnWejdźClick`

```
<script type="text/javascript">
  function btnWejdźClick()
  {
    var kod = document.getElementById("tfKod").value;
```

```
if(!kod) return;
kod = SHA1(kod);
document.location.href = kod + '.html';
}
</script>
```

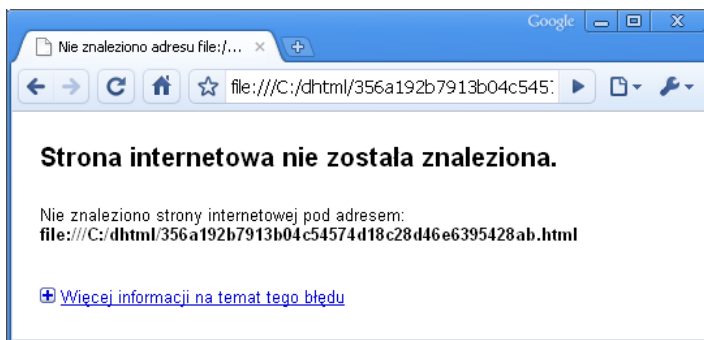
Wartość wprowadzona do pola jest pobierana i zapisywana w zmiennej kod. Zmienna ta jest następnie poddawana działaniu funkcji skrótu SHA1. Do uzyskanego w ten sposób ciągu znaków dodawane jest rozszerzenie .html i tak przygotowana nazwa pliku jest przypisywana właściwości href obiektu location. Tym samym następuje próba wczytania witryny o podanej nazwie, np. dla hasła 01234 będzie to nazwa:

11904a4e8b77f6242e2d288705023adad00a9310.html

Oczywiście taki plik musi istnieć na serwerze (aby uzyskać nazwę pliku dla innego hasła, wystarczy wykonać instrukcję `alert(SHA1("hasło"))`).

Jeśli kod był prawidłowy, zostanie wczytana strona zawarta we wskazanym pliku. Niestety, jeśli kod był nieprawidłowy, użytkownik zobaczy systemowy komunikat przeglądarki, np. taki jak zaprezentowano na rysunku 1.16. Nie jest to eleganckie rozwiązanie. Aby tego uniknąć, można użyć bardziej złożonej wersji kodu, która została przedstawiona na listingu 1.36.

Rysunek 1.16.
*Komunikat o braku
właściwej strony*



Listing 1.36. *Zaawansowana weryfikacja kodu dostępu*

```
<script type="text/javascript">
function btnWejdźClick()
{
    var kod = document.getElementById("tfKod").value;
    if(!kod) return;
    kod = SHA1(kod);

    var ajaxObj = window.XMLHttpRequest ? new XMLHttpRequest() :
        new ActiveXObject("Microsoft.XMLHTTP");
    if(ajaxObj){
        ajaxObj.open("HEAD", kod + ".html", false);
        ajaxObj.send(null);
        if(ajaxObj.status == 200){
            document.location.href = kod + '.html';
        }
    }
}
```

```
        else{  
            alert('Nieprawidłowy kod!');  
        }  
    }  
}  
</script>
```

Początek funkcji jest taki sam jak w przykładzie z listingu 1.35. Jednak po otrzymaniu wyniku działania funkcji skrótu SHA1 wykonywane jest sprawdzenie, czy istnieje na serwerze plik w postaci *kod.html*. Jeśli istnieje, jest wczytywany (na takiej samej zasadzie jak w poprzedniej wersji funkcji). Jeżeli nie istnieje, za pomocą metody `alert` jest wyświetlany stosowny komunikat.

Kluczową sprawą jest oczywiście sposób badania, czy plik o wskazanej nazwie jest na serwerze. Została w tym celu użyta technika AJAX-a. Niestety, w tej książce nie ma miejsca na jej dokładne omówienie. Osoby zainteresowane tą techniką znajdą szczegółowe opisy w takich publikacjach, jak *AJAX. Ćwiczenia* (<http://helion.pl/ksiazki/cajax.htm>) czy *AJAX i PHP. Praktyczny kurs* (<http://helion.pl/ksiazki/ajphpk.htm>). Pisząc jednak w skrócie: na początku jest tworzony i zapisywany w zmiennej `ajaxObj` obiekt typu `XMLHttpRequest`. Pozwala on na wysyłanie do serwera żądań za pomocą protokołu HTTP. Za pomocą metody `open` żądanie jest przygotowywane, a za pomocą metody `send` — wysyłane. Pierwszy argument metody `open` określa typ żądania, drugi — adres URL (w tym przypadku nazwę pliku, którego istnienie chcemy sprawdzić), a trzeci — sposób transmisji (w tym przypadku chodzi o transmisję synchroniczną). Po zakończeniu wykonywania metody `send` we właściwości `status` obiektu `ajaxObj` będzie się znajdowała wartość określająca odpowiedź serwera. Gdy plik istnieje¹⁵, będzie to wartość 200, stąd instrukcja warunkowa:

```
if(ajaxObj.status == 200){
```

Uwaga! Tej wersji skryptu nie można wczytywać z dysku, bowiem technika AJAX-a działa tylko przy korzystaniu z protokołu HTTP. Pliki skryptu należy więc umieścić na serwerze (może być to serwer lokalny) i w odwołaniu podawać pełny adres URL, np. <http://mojadomena.com/index.html> czy <http://localhost/index.html>.

Skrypt 17.

[C][E][F][O][S]

Zliczanie liczby odwiedzin

Użycie cookies pozwala zorientować się, ile razy użytkownik odwiedzał naszą witrynę. Nie jest to technika, na której można w 100 proc. polegać (cookie można przecież skasować, zmienić bądź też wyłączyć ich przyjmowanie w przeglądarce), ale do wielu celów w zupełności wystarczająca. Przy pierwszych odwiedzinach należy zapisać cookie o wybranej nazwie i wartości 1, a przy każdych kolejnych zwiększać tę wartość o 1. Przyjmijemy przy tym, że okresem zliczania będzie jeden miesiąc (oczywi-

¹⁵ Dokładniej rzecz ujmując: gdy możliwe jest odwołanie (pobranie) zasobu wskazywanego przez URL.

ście czas ten będzie można dowolnie wydłużyć lub skrócić). Takie zadanie realizuje skrypt przedstawiony na listingu 1.37¹⁶.

Listing 1.37. Skrypt zliczający liczbę odwiedzin

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN"
"http://www.w3.org/TR/html4/strict.dtd">
<html lang="pl">
<head>
  <meta http-equiv="Content-Type" content="text/html; charset=utf-8">
  <title>Zliczanie liczby odwiedzin</title>
  <script type="text/javascript">
    function getCookie(name)
    {
      var c = document.cookie;
      var point = c.indexOf(name);
      if (point != -1){
        endAt = c.indexOf(":", point + name.length);
        if (endAt == -1) endAt = c.length;
        var temp = c.substring(point + 1 + name.length, endAt);
        return temp;
      }
      return false;
    }
    function setCookie(name, value)
    {
      data = new Date();
      data.setMonth(data.getMonth() + 1);
      var cookie = name + "=" + value + ";";
      cookie += "expires=" + data.toGMTString();
      document.cookie = cookie;
    }
    function check()
    {
      var value, str;
      if((value = getCookie("vcount")) != false){
        if(value == 1)
          str = "W ciągu miesiąca na tej stronie byłeś 1 raz.";
        else
          str = "W ciągu miesiąca na tej stronie byłeś " + value + " razy.";
        value++;
        setCookie("vcount", value);
      }
      else{
        str = "Na tej stronie nie było Cię w ciągu ostatniego miesiąca.";
        setCookie("vcount", 1);
      }
      var div1 = document.getElementById("div1");
      if(div1) div1.innerHTML = str;
    }
  </script>
</head>
<body onload="check();">
  <div id="div1">
```

¹⁶ W przypadku przeglądarki Chrome 3 pliki z przykładu należy umieścić na serwerze WWW. Inaczej (przy wczytywaniu z dysku lokalnego) efekt może nie być widoczny.

```
</div>
</body>
</html>
```

W sekcji `<body>` została umieszczona warstwa o identyfikatorze `div1`. To na niej pojawi się informacja o liczbie odwiedzin z ostatniego miesiąca. Zdarzeniu `load` sekcji `<body>` została przypisana funkcja `check`, która dzięki temu jest wykonywana po załadowaniu strony i wykonuje (ze pomocą funkcji dodatkowych) wszystkie niezbędne czynności związane z odczytem i zapisem cookie.

Najpierw sprawdza, czy na komputerze użytkownika istnieje cookie o nazwie `vcount`. Odpowiada za to instrukcja `if` ze złożonym warunkiem:

```
if((value = getCookie("vcount")) != false){
```

Jest w nim używana pomocnicza funkcja `getCookie` pobierająca wartość cookie o zadanej nazwie (jeżeli cookie nie istnieje, zwracana jest wartość `false`).

Jeśli cookie istnieje, odczytana wartość jest używana w komunikacie przypisywanym pomocniczej zmiennej `str`. Następnie do wartości dodawane jest 1, a cookie ponownie zapisywane. Do zapisu jest używana funkcja `setCookie`. W przeciwnym wypadku (nie ma cookie) na komputerze użytkownika jest zapisywane nowe cookie o nazwie `vcount` i wartości 1. Zmienna `str` otrzymuje też stosowny komunikat. Na końcu kodu wartość zmiennej `str` jest przypisywana właściwości `innerHTML` obiektu warstwy `div1`, dzięki czemu tekst pojawia się na ekranie.

Funkcja `setCookie` tworzy obiekt z bieżącą datą (wywołanie `new Date()`), a następnie przesuwą ją o jeden miesiąc do przodu (oczywiście można to zmienić i ustalić inny czas ważności). W tym celu używane są metody `getMonth` (pobranie aktualnego miesiąca) oraz `setMonth` (ustawienie miesiąca). Następnie z argumentów przekazanych funkcji (`name` i `value`) tworzony jest ciąg tworzący nazwę i wartość cookie, a do niego dodawane jest określenie ważności utworzone przez wywołanie metody `toGMTString`. Cookie jest zapisywane przez przypisanie utworzonego ciągu właściwości `cookie` obiektu `document`.

Funkcja `getCookie` ma nieco bardziej złożone zadanie. Musi wyodrębnić z ciągu wszystkich cookie zapisanych we właściwości `cookie` obiektu `document` wartość tego, którego nazwa została przekazana w postaci argumentu `name`. Najpierw za pomocą metody `indexOf` indeks wystąpienia nazwy jest pobierany i zapisywany w zmiennej `point`. Dalsze czynności są wykonywane, jeżeli wartość ta jest różna od `-1` (czyli dane cookie istnieje). W przeciwnym wypadku (`point` równe `-1`) zwracana jest wartość `false` i funkcja kończy działanie.

Jeżeli nazwa cookie została znaleziona, oznacza to, że jego wartość znajduje się między aktualnym indeksem plus jeden (za znakiem równości, przykładowy ciąg ma postać `vcount=1;`) a wystąpieniem znaku średnika lub też końcem ciągu. Odszukiwany jest zatem znak `;`, a indeks jego wystąpienia lub indeks końca ciągu (jeśli średnik nie został znaleziony) jest zapisywany w zmiennej pomocniczej `endAt`. Na zakończenie za pomocą metody `substring` wartość cookie jest wyodrębniana i zwracana jako wynik działania funkcji.

Skrypt 18. [C][E][F][O][S]

Uniemożliwienie zaznaczenia fragmentu strony

Jeżeli chcemy zablokować możliwość zaznaczania fragmentów witryny, wystarczy, że dodamy do niej prosty skrypt. Przyjmijmy, że na witrynie została umieszczona warstwa oraz akapit, takie jak zaprezentowano na listingu 1.38, i chcemy móc selektywnie wyłączać możliwość zaznaczania treści zawartej w tych elementach. Wystarczy wtedy, jeśli właściwości `onload` sekcji `<body>` przypiszemy wywołanie funkcji blokującej elementy, a jako argumenty prześlemy ich identyfikatory (wartości atrybutów `id`). To, czy element będzie blokowany, czy też nie, będzie oczywiście zależało od tego, czy jego identyfikator pojawi się w wywołaniu funkcji `zablokujElementy`.

Listing 1.38. Sekcja `<body>` strony pozwalającej zablokować niektóre elementy

```
<body onload="zablokujElementy('div1','p1');">
  <div id="div1">
    Tekst warstwy
  </div>
  <p id="p1">
    Tekst akapitu
  </p>
</body>
```

Część skryptowa strony oprócz funkcji `zablokujElementy` będzie zawierała dodatkowo funkcję `blokujZaznaczenie`. Odpowiedni kod został przedstawiony na listingu 1.39. Należy go umieścić w sekcji `<head>` witryny.

Listing 1.39. Funkcje blokujące zaznaczanie elementów witryny

```
<script type="text/javascript">
  function blokujZaznaczenie(el)
  {
    if(!el) return;
    el.onselectstart=function(){return false;};
    el.onmousedown=function(){return false;};
    el.style.cursor = "default";
  }
  function zablokujElementy()
  {
    for(var i = 0; i < arguments.length; i++){
      var obj = document.getElementById(arguments[i]);
      if(obj) blokujZaznaczenie(obj);
    }
  }
</script>
```

Funkcja `zablokujElementy` przyjmuje dowolną liczbę argumentów, którymi powinny być ciągi znaków identyfikujące elementy, w których ma być zablokowane zaznaczanie treści. Lista argumentów będzie dostępna wewnątrz funkcji we właściwości `arguments`. Dostęp do pojedynczego argumentu (o zadanym indeksie) uzyskuje się dzięki odwołaniu `arguments[indeks]`. Całkowita liczba przekazanych argumentów znajduje się we właściwości `length` obiektu `arguments`.

Dlatego też do odczytu wszystkich argumentów została użyta zwykła pętla `for`. W jej wnętrzu za pomocą metody `getElementById` pobierane są odwołania do elementów o identyfikatorach odczytanych z tablicy `arguments` oraz jeśli dany element istnieje, wywoływana jest funkcja `blokujZaznaczanie`, która wykonuje właściwą blokadę.

Uzyskanie blokady nie jest trudne. Wystarczy, jeśli procedurą zdarzenia `selectstart` (dla przeglądarek z rodziny Internet Explorer) lub `mousedown` (dla pozostałych przeglądarek) była funkcja zwracająca wartość `false`. W zaprezentowanym kodzie rodzaj przeglądarki nie jest rozpoznawany i modyfikowane są procedury obsługi obu zdarzeń. Przypisywane są im funkcje anonimowe, których jedyną zawartością jest instrukcja `return false;`. Dodatkowo jest też zmieniany rodzaj kursora przypisanego danemu elementowi (na domyślny — `default`).

Skrypt 19.

[E][F][O]

Dodanie strony do zakładek

Każda popularna przeglądarka pozwala na dodanie adresu witryny do listy zakładek. Wymaga to wciśnięcia wybranej kombinacji klawiszy lub wybrania odpowiedniej pozycji z menu. Niektóre witryny pozwalają jednak na utworzenie zakładki po kliknięciu jakiegoś elementu interfejsu, np. odnośnika czy przycisku. Wtedy wywoływany jest odpowiedni skrypt. Niestety, różne przeglądarki wymagają różnych metod tworzenia zakładek. Najlepsza sytuacja jest w Internet Explorerze — istnieje odpowiednia metoda wykonująca to zadanie (i to nawet w tak leciwych i niespotykanych już wersjach jak 4.). Poradzimy sobie też ze współczesnymi wersjami Firefoksa. Niestety, w przypadku Opery występuje problem. O ile w wersjach do 9.20 można było sprytnie utworzyć „wirtualny” odnośnik i wymusić jego programowe kliknięcie, o tyle obecnie spotykane wersje nie pozwalają na zastosowanie tej metody (co najmniej do wersji 10.1). W związku z tym trzeba umieszczać na stronie zwykły odnośnik z atrybutem `rel` o wartości `sidebar` (ten sposób zadziałałby też w przypadku Firefoksa).

To wszystko powoduje, że potrzebne będą dwie wersje strony. Przykład będzie więc działał następująco:

- ♦ Na stronie pojawi się lista witryn, które będzie można dodawać do zakładek.
- ♦ W przeglądarkach Internet Explorer i Firefox znajdują się zwykle, tekstowe elementy listy; kliknięcie elementu będzie powodowało wywołanie odpowiedniej funkcji JavaScript.

- ◆ W przeglądarce Opera elementami listy będą odpowiednio spreparowane odnośniki; kliknięcie odnośnika spowoduje dodanie wybranego adresu do zakładek.

W związku z tym kod sekcji `<body>` przyjmie postać widoczną na listingu 1.40.

Listing 1.40. Kod sekcji `<body>` przykładu 19.

```
<body>
<div>
  Aby dodać adres do zakładek, kliknij wybraną pozycję.
  <ul>
    <li onclick="dodajDoZakładek('http://helion.pl/',
      'Wydawnictwo Helion');">
      <script type="text/javascript">
        generujLink('http://helion.pl/', 'Wydawnictwo Helion');
      </script>
    </li>
    <li onclick="dodajDoZakładek('http://onepress.pl/',
      'Wydawnictwo OnePress');">
      <script type="text/javascript">
        generujLink('http://onepress.pl/', 'Wydawnictwo OnePress');
      </script>
    </li>
  </ul>
</div>
</body>
```

Lista jest tworzona za pomocą standardowego znacznika ``, a jej pozycje za pomocą znaczników ``. Każdy znacznik `` ma przypisaną procedurę obsługi zdarzenia `click` w postaci funkcji `dodajDoZakładek`. Funkcja otrzymuje dwa argumenty. Pierwszy wskazuje adres, który ma być dodany do zakładki, a drugi — nazwę zakładki. Zawartość znacznika `` nie jest tworzona w kodzie HTML, ale generowana przez funkcję `generujLink`. Dzięki temu zawartość będzie mogła być różna w zależności od tego, w której przeglądarce zostanie otworzona witryna¹⁷. Funkcja `generujLink` przyjmuje takie same argumenty jak funkcja `dodajDoZakładek`.

Kod obu wspomnianych funkcji został zaprezentowany na listingu 1.41.

Listing 1.41. Dodawanie wybranego adresu do zakładek

```
<script type="text/javascript">
  function dodajDoZakładek(url, tytuł)
  {
    if(window.sidebar)
      window.sidebar.addPanel(tytuł, url, "");
    else if(window.opera && window.print){
      //nic nie rób
    }
  }
</script>
```

¹⁷ Alternatywnym rozwiązaniem jest generowanie przez skrypt całej zawartości listy (włącznie ze znacznikami ``). Wtedy w przypadku umieszczania odnośników w generowanych pozycjach listy można by uniknąć obsługi zdarzenia `click`.

```

else if(document.all)
    window.external.AddFavorite(url, tytuł);
else{
    alert('Niestety, ta przeglądarka nie pozwala na taką operację.');
```

Funkcja `generujLink` otrzymuje argumenty wskazujące adres odnośnika (`url`) oraz jego nazwę (`tytuł`). Dla Opery musi wygenerować faktyczny odnośnik, a dla pozostałych przeglądarek — sam tytuł. Z Operą mamy do czynienia, jeśli złożony warunek `window.opera && window.print` jest prawdziwy. Wtedy też powstaje odnośnik o schematycznej postaci:

```
<a href="url" rel="sidebar">tytuł</a>
```

Odnośnik jest umieszczany na stronie w miejscu wywołania skryptu dzięki metodzie `document.write`¹⁸. Kliknięcie takiego odnośnika spowoduje pojawienie się okna dodawania zakładki. W przypadku wszystkich innych przeglądarek w miejscu wywołania funkcji pojawi się tylko tytuł odnośnika. A zatem pojedynczy element listy (wzroku) w przeglądarce Opera będzie miał postać:

```
<li onclick="dodajDoZakładek('url', 'tytuł');"><a href="url"
rel="sidebar">tytuł</a></li>
```

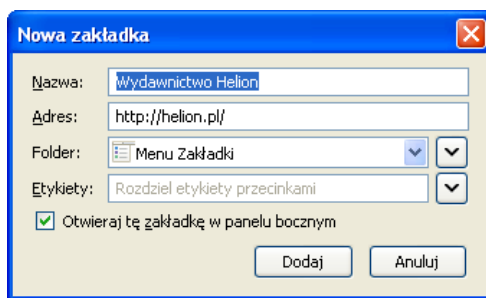
a w innych przeglądarkach:

```
<li onclick="dodajDoZakładek('url', 'tytuł');">tytuł</li>
```

Konstrukcja funkcji `dodajDoZakładek` jest prosta. Przyjmuje ona takie same argumenty jak `generujLink`, czyli `url` i `tytuł`. Jeżeli jest obecny obiekt `window.sidebar`, co jest charakterystyczne dla przeglądarki Firefox, jest wywoływana metoda `addPanel` tego obiektu i jest przekazywany tytuł i adres zakładki. Pojawi się wtedy okno pozwalające na dodanie zakładki widocznej na rysunku 1.17. Należy zwrócić uwagę, że taka zakładka będzie standardowo otwierana na panelu bocznym. Aby tego uniknąć, należy usunąć zaznaczenie widocznego w oknie pola wyboru. To niestety niedogodność programowanego dodawania zakładek w Firefoksie.

¹⁸ Nie jest to najbardziej eleganckie rozwiązanie, należy bowiem raczej unikać stosowania metody `document.write`, a zamiast niej bezpośrednio manipulować węzłami dokumentu. Instrukcja ta została jednak użyta ze względu na jej prostotę i mniejszą komplikację kodu.

Rysunek 1.17.
*Dodawanie zakładek
w Firefoksie*



Druga część instrukcji warunkowej (`else if(window.opera && window.print){}`) może budzić pewne zdziwienie, gdyż nie ma wewnątrz niej żadnego kodu (miejsce to jest oznaczone komentarzem). Przy zaproponowanej strukturze kodu jest to potrzebne, aby nie nastąpiła żadna reakcja w przeglądarce Opera (zakładka jest przecież dodawana poprzez kliknięcie w odnośnik).

Kolejna część instrukcji warunkowej bada, czy istnieje właściwość `all` obiektu `document`, co jest charakterystyczne dla przeglądarek z rodziny Internet Explorer (uwaga: ta właściwość jest też w przeglądarkach Opera, ale zostały one wykluczone z przetwarzania w poprzedniej części instrukcji warunkowej). Jeżeli istnieje, jest wykonywana metoda `AddFavorite` obiektu `external`, co powoduje wyświetlenie okna dodawania zakładek.

Ostatni człon instrukcji warunkowej jest wykonywany dla wszystkich innych przeglądarek i powoduje wyświetlenie okna dialogowego z informacją, że dany produkt nie umożliwia programowego dodawania zakładek.

Rozdział 2.

Data i czas

Rozdział drugi zawiera skrypty związane z datą i czasem. To wszelkiego rodzaju zegary, stopery i systemy odliczania czasu, ale również różne typy kalendarzy, w tym kalendarz typu pop-up pozwalający na szybkie wskazanie dowolnej daty. Zaprezentowane zostaną też przykłady uzależniania treści witryny od daty i czasu odwiedzin oraz sposoby korzystania z interwałów. Charakterystyczną cechą takich skryptów jest oczywiście korzystanie z obiektu typu `Date` oraz udostępnianych przez niego metod pozwalających uzyskać informacje konieczne do prawidłowej pracy kodu.

Skrypt 20. Zegar cyfrowy

[C][F][E][O][S]

Wszelkiego rodzaju zegary podające na stronie WWW aktualny czas są wykorzystywane bardzo często. Ich wykonanie zwykle nie jest też skomplikowane. Do wyświetlenia czasu wykorzystuje się obiekt `Date` i udostępniane przez niego metody. Zegar cyfrowy może być umieszczony na prawie dowolnym elemencie witryny: warstwie, akapicie, polu tekstowym czy nawet na przycisku. Do generowania ciągu znaków opisujących bieżący czas w formacie GG:MM:SS wystarczy funkcja JavaScript zaprezentowana na listingu 2.1.

Listing 2.1. *Funkcja generująca bieżący czas*

```
<script type="text/javascript">
function wyswietlCzas(nazwa, innerHTML)
{
    var e1 = document.getElementById(nazwa);
    if(!e1) return;

    var data = new Date();
    var godziny = data.getHours();
    var minuty = data.getMinutes();
    var sekundy = data.getSeconds();
```

```

var czas = godziny;
czas += ((minuty < 10) ? ":0" : ":") + minuty;
czas += ((sekundy < 10) ? ":0" : ":") + sekundy;

if(innerHTML)
    el.innerHTML = czas;
else
    el.value = czas;

setTimeout("wyswietlCzas('" + nazwa + "','" + innerHTML + ")", 1000);
}
</script>

```

Funkcja `wyswietlCzas` ma za zadanie pobrać aktualny czas i wyświetlić go na elemencie witryny o identyfikatorze wskazywanym przez pierwszy argument (`nazwa`). Drugi argument określa, czy jest to element typu warstwa, akapit i czas należy przypisać właściwości `innerHTML` (argument `innerHTML` równy `true`), czy też jest to element formularza (pole tekstowe, przycisk) i czas należy przypisać właściwości `value` (argument `innerHTML` równy `false`).

Najpierw za pomocą metody `getElementById` jest pobierane i przypisywane zmiennej `el` odwołanie do elementu strony wskazywanego przez argument `nazwa`. Jeśli takiego elementu nie ma (czyli `nazwa` jest równy `null`), działanie funkcji jest przerywane za pomocą instrukcji `return`.

Następnie tworzony jest nowy obiekt typu `Date` (zawierający bieżącą datę oraz czas) i zapisywany w zmiennej `data`. Do uzyskania liczby godzin, minut i sekund stosowane są metody `getHours`, `getMinutes` i `getSeconds`. Dane te są przypisywane zmiennym `godzina`, `minuta` i `sekunda`. Ponieważ dane dotyczące minut i sekund powinny mieć format dwucyfrowy, gdy któraś z nich ma wartość mniejszą od 9, na początku dodawany jest znak 0. Używany jest w tym celu operator warunkowy. Dane dotyczące godzin nie są poddawane tej operacji. Wszystkie dane są łączone w zmiennej `czas`, której wartość jest wyświetlana na ekranie.

Sposób wyświetlania wartości zmiennej `czas` zależy od stanu argumentu `innerHTML`. Gdy jest on równy `true`, zawartość zmiennej jest przypisywana właściwości `innerHTML` obiektu `el` (co jest charakterystyczne dla warstw, akapitów itp.), gdy jest równy `false`, zawartość zmiennej jest przypisywana właściwości `value` (co jest charakterystyczne dla pól tekstowych, przycisków itp.).

Funkcja `wyswietlCzas` jest wywoływana cyklicznie co sekundę dzięki instrukcji:

```
setTimeout("wyswietlCzas('" + nazwa + "','" + innerHTML + ")", 1000);
```

Jest to wywołanie metody `setTimeout`, której pierwszym argumentem jest ciąg znaków określający funkcję do wywołania, a drugim — wartość określająca, po jakim czasie to wywołanie ma nastąpić. Czas określany jest w milisekundach. Ponieważ w każdym kolejnym wywołaniu funkcja `wyswietlCzas` musi zawierać przekazane jej argumenty, pierwszy parametr metody `setTimeout` jest konstruowany za pomocą operatora `+`.

Tak przygotowanej funkcji możemy użyć na dowolnej witrynie. Na listingu 2.2 przedstawiony został kod sekcji `<body>` zawierającej warstwę o identyfikatorze `div1`. Zdarzeniu `load` sekcji została przypisana procedura obsługi w postaci funkcji `wyswietlCzas`. A więc po uruchomieniu strony zostanie wykonana instrukcja:

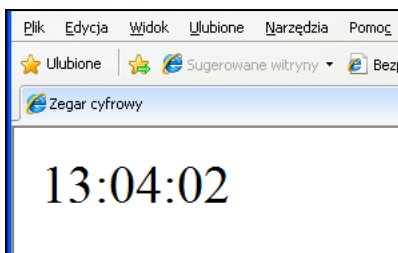
```
wyswietlCzas('div1', true);
```

powodująca rozpoczęcie wyświetlania czasu. Dane pojawią się na warstwie, tak jak zaprezentowano to na rysunku 2.1.

Listing 2.2. *Przykładowa sekcja `<body>` dla strony wyświetlającej bieżący czas*

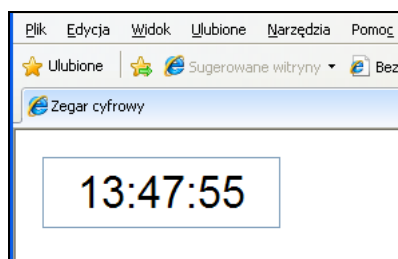
```
<body onload="wyswietlCzas('div1', true);">
  <div id="div1">
  </div>
</body>
```

Rysunek 2.1.
*Czas wyświetlany
na warstwie*



Gdyby czas miał być wyświetlony w polu tekstowym, np. takim, które widoczne jest na rysunku 2.2, sekcja `<body>` mogłaby mieć postać przedstawioną na listingu 2.3. W takim przypadku wartością drugiego argumentu funkcji `wyswietlCzas` musi być `false`. Możliwość modyfikacji zawartości pola tekstowego została wyłączona przez użycie atrybutu `readonly` znacznika `<input>`.

Rysunek 2.2.
*Zegar w polu
tekstowym*



Listing 2.3. *Strona z polem tekstowym, w którym pojawi się aktualny czas*

```
<body onload="wyswietlCzas('tf1', false);">
  <div id="div1">
    <input type="text" readonly="readonly" id="tf1" size="6"
      style="text-align:center;" />
  </div>
</body>
```

Skrypt 21. [C][F][E][O][S]

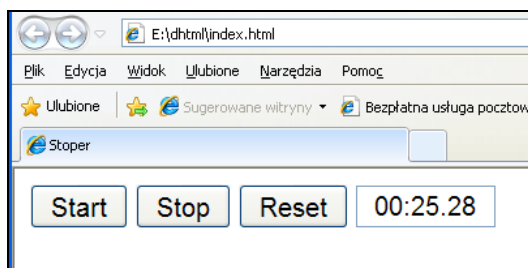
Stoper, czyli odmierzanie czasu

Stronę WWW można w prosty sposób wyposażyć w funkcję stopera, który będzie pozwalał na odmierzanie czasu z dokładnością np. do setnych części sekundy. Interfejs takiego stopera może być różny. Przyjmijmy, że na witrynie mają pojawić się trzy przyciski realizujące funkcje startu, stopu i zerowania (resetu) oraz pole tekstowe służące do wyświetlania danych. Kod HTML sekcji `<body>` przyjmie zatem postać widoczną na listingu 2.4, natomiast strona będzie wyglądała tak, jak zaprezentowano to na rysunku 2.3.

Listing 2.4. *Kod interfejsu stopera*

```
<body>
  <div id="timerDiv">
    <input type="button" id="btnStart" value="Start"
      onclick="btnStartClick();" />
    <input type="button" id="btnStop" value="Stop"
      onclick="btnStopClick();" />
    <input type="button" id="btnReset" value="Reset"
      onclick="btnResetClick();" />
    <input type="text" readonly="readonly" id="tf1" size="6"
      style="text-align:center;" />
  </div>
</body>
```

Rysunek 2.3.
*Interfejs sterujący
stoperem*



Przyciski zostały wygenerowane za pomocą znaczników `<input>` z atrybutem `type` ustawionym na `button`. Każdy z nich otrzymał własną procedurę obsługi zdarzenia `click`. Są to funkcje: `btnStartClick`, `btnStopClick` i `btnResetClick`. Pole tekstowe jest generowane za pomocą znacznika `<input>` z atrybutem `type` ustawionym na `text`. Został w nim również użyty atrybut `readonly` powodujący, że dane z pola będą mogły być tylko odczytywane. Pole otrzymało identyfikator `tf1`, dzięki czemu będzie się można do niego odwoływać w kodzie skryptu. Skrypt został przedstawiony na listingu 2.5.

Listing 2.5. *Skrypt obsługujący stoper*

```
<script type="text/javascript">
  var licznik = 0;
  var timerId = null;
```

```
function btnStartClick()
{
    if(timerId) return;
    wyswietl();
    timerId = setInterval("licz();wyswietl();", 10);
}
function btnStopClick()
{
    if(!timerId) return;
    clearInterval(timerId);
    timerId = null;
}
function btnResetClick()
{
    licznik = 0;
    wyswietl();
}
function wyswietl()
{
    var csec = licznik % 100;
    var sekundy = Math.floor(licznik / 100) % 60;
    var minuty = Math.floor(licznik / (60 * 100));

    csec = csec < 10 ? "0" + csec : csec;
    sekundy = sekundy < 10 ? "0" + sekundy : sekundy;
    minuty = minuty < 10 ? "0" + minuty : minuty;

    var tf1 = document.getElementById('tf1');
    if(tf1) tf1.value = minuty + ":" + sekundy + "." + csec;
}
function licz()
{
    licznik++;
}
</script>
```

Na początku kodu znajdują się dwie zmienne globalne — `licznik` oraz `timerId`. Pierwsza przechowuje licznik czasu, natomiast druga — identyfikator timera. Początkowym stanem licznika jest 0, a identyfikatora — `null`. Zwiększaniem stanu licznika zajmuje się funkcja `licz`, która będzie wywoływana cyklicznie po starcie stopera. Start następuje po kliknięciu przycisku Start, czyli wywołaniu funkcji `btnStartClick`.

Funkcja `btnStartClick` bada najpierw, czy `timerId` jest różne od `null` (instrukcja `if(timerId)`). Jeśli tak jest, oznacza to, że timer działa, nie należy go więc ponownie uruchamiać — jest zatem wykonywana instrukcja `return`. Jeżeli jednak `timerId` jest równe `null`, następuje uruchomienie zegara:

```
timerId = setInterval("licz();wyswietl();", 10);
```

Powyższa instrukcja powoduje cykliczne wywoływanie funkcji `licz` i `wyswietl` co 10 milisekund, czyli co jedną setną sekundy.

Uwaga! Interwał 10 milisekund jest dosyć bezpieczny i w większości systemów taki licznik będzie działał poprawnie. Należy jednak pamiętać, że nie można zagwarantować, iż kolejne wywołania zainicjowane przez `setInterval` będą wykonywane dokładnie po zadanym czasie. Zależy to od bieżącego obciążenia systemu. Można rozważyć zwiększenie interwału do 100 milisekund.

Funkcja `wyswietl` musi przeliczyć aktualne wskazanie licznika (zapisane w zmiennej `licznik`) na liczbę centysekund (setnych części sekundy), sekund i minut. Liczbę centysekund otrzymujemy, wykonując zwyczajne dzielenie modulo:

```
licznik % 100
```

liczbę sekund dzięki dzieleniu:

```
(licznik / 100) % 60
```

a liczbę minut dzięki dzieleniu:

```
licznik / (60 * 100)
```

Obliczone wartości są zaokrąglane w dół dzięki metodzie `Math.floor` i zapisywane w zmiennych pomocniczych `csec`, `sekundy` i `minuty`. W przypadku stwierdzenia, że uzyskane liczby są mniejsze od 10, do danych dodawane jest początkowe 0, tak by wyświetlany ciąg miał stałą długość. Na zakończenie wartości zmiennych są łączone w jeden ciąg, który jest przypisywany właściwości `value` pola tekstowego. Dzięki temu pojawiają się na ekranie.

Funkcja `btnStopClick` jest wywoływana po kliknięciu przycisku `Stop`. Jej zadaniem jest zatrzymanie odliczania. Jest to wykonywane tylko wtedy, gdy zegar działa, czyli zmienna `timerId` ma wartość różną od `null`. Odliczanie jest zatrzymywane dzięki wywołaniu metody `clearInterval`. Po zastopowaniu timera zmiennej `timerId` jest przypisywana wartość `null`. To sygnał, że funkcja odliczania nie jest aktywna.

Jedynym zadaniem funkcji `btnResetClick` jest wyzerowanie licznika, czyli przypisanie zmiennej `licznik` wartości 0.

Skrypt 22. [C][E][F][O][S]

Odliczanie zadanego czasu

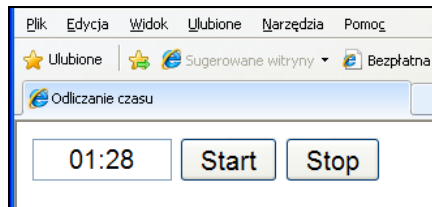
W skrypcie 21. pokazano, jak umieścić na stronie WWW stoper określający to, ile czasu upłynęło od danego zdarzenia (kliknięcia). Przydatny może być również skrypt wykonujący zadanie odwrotne, czyli odliczający zadany czas i wykonujący po jego upływie pewną akcję. Jaką? To zależy od konkretnych potrzeb — nie ma tu żadnych ograniczeń. W przykładzie wyświetlimy po prostu informacyjne okno dialogowe. Na listingu 2.6 znajduje się treść sekcji `<body>`. To oznacza, że interfejs strony będzie wyglądał tak, jak zaprezentowano to na rysunku 2.4.

Listing 2.6. *Interfejs strony odliczającej zadany czas*

```

<body>
  <div id="timerDiv">
    <input type="text" id="tf1" size="6" style="text-align:center;" />
    <input type="button" id="btnStart" value="Start"
      onclick="btnStartClick();" />
    <input type="button" id="btnStop" value="Stop"
      onclick="btnStopClick();" />
  </div>
</body>

```

Rysunek 2.4.
*Odliczanie
zadanego czasu*

Na witrynie widoczne są dwa przyciski oraz pole tekstowe. Przycisk *Start*, z procedurą obsługi zdarzenia `click` w postaci funkcji `btnStartClick`, będzie powodował rozpoczęcie odliczania, a przycisk *Stop*, z procedurą obsługi zdarzenia `click` w postaci funkcji `btnStopClick`, będzie powodował przerwanie odliczania. Pole tekstowe będzie pełniło dwie funkcje. Po pierwsze, umożliwi wprowadzenie liczby minut i sekund, które mają być odliczone, po drugie, będzie w nim prezentowany czas pozostały do końca odliczania. Obsługę odliczania zapewnią funkcje JavaScript przedstawione na listingu 2.7.

Listing 2.7. *Funkcje obsługujące odliczanie*

```

<script type="text/javascript">
  var licznik = 0;
  var timerId = null;
  function btnStartClick()
  {
    if(timerId) return;
    var tf1 = document.getElementById('tf1');
    if(!tf1) return;

    var arr = tf1.value.split(":", 2);
    var minuty = parseInt(arr[0]);
    var sekundy = parseInt(arr[1]);

    if(isNaN(minuty) || isNaN(sekundy) || sekundy < 0 ||
       sekundy > 59 || minuty < 0 ){
      alert("Prawidłowym formatem jest MM:SS, np. 1:28.");
      return;
    }

    licznik = minuty * 60 + sekundy;

    tf1.disabled = true;
    if(licznik == 0) licz();
  }

```

```

        else
            timerId = setInterval("licz();wyswietl();", 1000);
        }
    }
    function btnStopClick()
    {
        if(!timerId) return;
        clearInterval(timerId);
        timerId = null;

        var tf1 = document.getElementById('tf1');
        if(!tf1) return;
        tf1.disabled = false;
    }
    function wyswietl()
    {
        var sekundy = licznik % 60;
        var minuty = Math.floor(licznik / 60);

        sekundy = sekundy < 10 ? "0" + sekundy : sekundy;
        minuty = minuty < 10 ? "0" + minuty : minuty;

        var tf1 = document.getElementById('tf1');
        if(tf1) tf1.value = minuty + ":" + sekundy;
    }
    function licz()
    {
        if(--licznik <= 0){
            licznik = 0;
            wyswietl();
            clearInterval(timerId);
            timerId = null;
            alert("Upłynął zadany czas!");
            var tf1 = document.getElementById('tf1');
            if(!tf1) return;
            tf1.disabled = false;
        }
    }
}
</script>

```

Struktura i zasada działania skryptu są w pewnym stopniu podobne do przedstawionych w przykładzie 21. Za rozpoczęcie odliczania odpowiada funkcja `btnStartClick`. Najpierw sprawdza, czy działa już timer oraz czy udało się pobrać odwołanie do pola `tf1`. Jeśli odpowiedź na jedno z tych pytań brzmi „nie”, działanie jest przerywane za pomocą instrukcji `return`.

Wartość zapisana w polu tekstowym jest pobierana i poddawana działaniu metody `split`, z separatorem ustalonym na znak `:` oraz maksymalną liczbą ciągów wynikowych ustaloną na 2. Tablica wynikowa jest zapisywana w zmiennej `arr`:

```
var arr = tf1.value.split(":", 2);
```

To oznacza, że jeżeli w polu znajdował się przykładowy zapis `1:28`, to powstanie tablica, w której w komórce o indeksie 0 będzie się znajdował ciąg `1`, a komórce o indeksie 1 — ciąg `28`.

Wartości komórek pobrane z tablicy `arr` są poddawane działaniu metody `parseInt` przetwarzającej je na wartości liczbowe oraz zapisywane w zmiennych `minuty` i `sekundy`. Następnie jest badane, czy operacje te zakończyły się sukcesem, czyli czy wartości zmiennych są różne od `NaN`. Są w tym celu używane funkcja `isNaN` oraz instrukcje warunkowe. Sprawdzane jest również to, czy wartości sekund i minut mieszczą się w prawidłowym przedziale (sekundy w zakresie od 0 do 59 i minuty nie mniejsze niż 0). W przypadku wykrycia nieprawidłowości za pomocą metody `alert` jest wyświetlany komunikat.

Jeżeli dane są poprawne, inicjowany jest licznik. Ponieważ odliczanie będzie się odbywać co sekundę, wyliczana jest całkowita liczba sekund:

```
licznik = minuty * 60 + sekundy;
```

Następnie wyłączane jest pole `tfl`, tak by w trakcie odliczania nie można było zmieniać jego zawartości:

```
tfl.disabled = true;
```

oraz za pomocą wywołania metody `setTimeout` uruchamiany jest timer. Przy czym to uruchomienie następuje tylko wtedy, gdy liczba sekund odliczenia jest większa od 0. Wtedy co sekundę będą wywoływane metody `licz` i `wyswietl`. W przeciwnym wypadku wywoływana jest jedynie metoda `licz`.

Uwaga! Należy pamiętać, że nie ma gwarancji, iż wywołania generowane za pomocą timera będą następowały ściśle w zadanym czasie (taka jest istota działania timerów ustawianych za pomocą `setTimeout` i `setInterval`). W przypadku chwilowego czy ciągłego znacznego obciążenia systemu mogą nastąpić opóźnienia. Rzeczywisty czas odliczania może być więc nieco dłuższy niż prezentowany. Oczywiście dla typowych zadań takie opóźnienia nie mają specjalnego znaczenia.

Zadaniem metody `licz` jest zmniejszenie wartości zmiennej `licznik` o 1 i sprawdzenie, czy osiągnęła wartość równą zero lub mniejszą, czyli czy należy zakończyć odliczanie. Jeśli odliczanie dobiegło końca (licznik mniejszy od zera bądź równy 0), licznik jest zerowany (`licznik = 0;`), działanie timera — przerywane (`clearInterval(timerId);`), zmienna `timerId` otrzymuje wartość `null`, wyświetlany jest komunikat o końcu odliczania (`alert("Upłynął zadany czas!");`) oraz włączana jest możliwość edycji pola tekstowego (`tfl.disabled = false;`). Oczywiście nic nie stoi na przeszkodzie, aby wykonać w tym miejscu inne dowolne czynności.

Funkcja `wyswietl` dotyczy uaktualniania zawartości pola tekstowego. Działa na takiej samej zasadzie jak funkcja o tej samej nazwie ze skryptu 21., choć oczywiście obliczenia wykonywane są nieco inaczej ze względu na to, że tym razem zliczane są sekundy, a nie centysekundy. W związku tym bieżącą liczbę sekund uzyskujemy dzięki dzieleniu modulo:

```
var sekundy = licznik % 60;
```

a liczbę minut dzięki zwykłemu dzieleniu i zaokrągleniu wyniku w dół:

```
var minuty = Math.floor(licznik / 60);
```

Skrypt 23. [C][F][E][O][S]

Odliczanie czasu do zadanej daty

W sieci często spotykamy skrypty odliczające liczbę dni pozostałych do jakiegoś wydarzenia, np. Nowego Roku, Wielkanocy, kalendarzowej wiosny czy też dowolnego innego. Konstrukcja takiego skryptu nie jest skomplikowana — wystarczy użyć obiektu typu `Date`, wykonać kilka prostych obliczeń i wyświetlić wynik na ekranie. Tak działający skrypt został zaprezentowany na listingu 2.8.

Listing 2.8. *Obliczanie pozostałej liczby dni*

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN"
"http://www.w3.org/TR/html4/strict.dtd">
<html lang="pl">
<head>
  <meta http-equiv="Content-Type" content="text/html; charset=utf-8">
  <title>Ile dni do...?</title>
  <script type="text/javascript">
    function ileDni(rok, miesiac, dzien)
    {
      var data1 = new Date();
      var data2 = new Date(rok, miesiac, dzien);
      var różnica = data2.getTime() - data1.getTime();
      return Math.floor(różnica / (1000 * 60 * 60 * 24));
    }

    function wyświetlLiczbęDni(e1, r, m, d)
    {
      var str = "";
      str += "Pozostało jeszcze " + ileDni(r, m, d);
      m = (m < 10 ? "0" : "") + m;
      d = (d < 10 ? "0" : "") + d;
      str += " do " + d + "-" + m + "-" + r + " r.";

      var e1 = document.getElementById(e1);
      if(e1) e1.innerHTML = str;
    }
  </script>
</head>
<body onload="wyświetlLiczbęDni('dataDiv', 2011, 1, 1);">
  <div id="dataDiv">
  </div>
</body>
</html>
```

Na stronie została umieszczona warstwa `dataDiv`, na której pojawi się informacja generowana przez skrypt. Odpowiada za to funkcja `wyświetlLiczbęDni` wywoływana po załadowaniu witryny do przeglądarki (funkcja jest procedurą obsługi zdarzenia `load` sekcji `<body>`). Przyjmuje cztery argumenty: nazwę elementu, na którym ma się pojawić komunikat, oraz rok, miesiąc i dzień docelowej daty. Jedynym zadaniem funkcji jest utworzenie komunikatu i przypisanie go właściwości `innerHTML` danego elementu witryny.

Ponieważ w komunikacie określenia dni i miesięcy powinny mieć format dwucyfrowy, w przypadku gdy te wartości są mniejsze niż 10, tworzony jest ciąg, w którym na początku dodawany jest znak 0. Ta czynność jest wykonywana za pomocą operatora warunkowego.

Wyliczeniem poszukiwanej liczby dni zajmuje się funkcja o nazwie `ileDni`. Przyjmuje ona trzy argumenty, którymi są kolejno: rok, miesiąc i dzień, określające datę w przeszłości. Zadaniem funkcji jest stwierdzenie, ile dni pozostało od dnia bieżącego do tej daty. Data bieżąca przechowywana jest w zmiennej `data1`, natomiast data przyszła — w zmiennej `data2`. Działanie:

```
var różnica = data2.getTime() - data1.getTime();
```

powoduje zapisanie w zmiennej `różnica` liczby milisekund, jaka upłynie między tymi datami. Kiedy liczba milisekund jest znana, wystarczy przeliczyć je na dni, zaokrąglając wynik, tak aby nie otrzymać ułamków (jeden dzień to $1000 \times 60 \times 60 \times 24$ milisekund). Za zaokrąglenie odpowiada metoda `floor` obiektu `Math`:

```
Math.floor(różnica / (1000 * 60 * 60 * 24));
```

Obliczony w ten sposób wynik jest zwracany jako rezultat działania funkcji za pomocą instrukcji `return`.

Należy jedynie zwrócić uwagę na to, że wynik ujemny oznacza liczbę dni, które upłynęły od daty docelowej (przekazanej funkcji w postaci argumentów). Zasadne może być zatem dodanie do funkcji `wyświetlLiczbeDni` instrukcji warunkowej badającej wynik i odpowiednio zmieniającej komunikat. To jednak zależy już od konkretnych potrzeb i przeznaczenia skryptu.

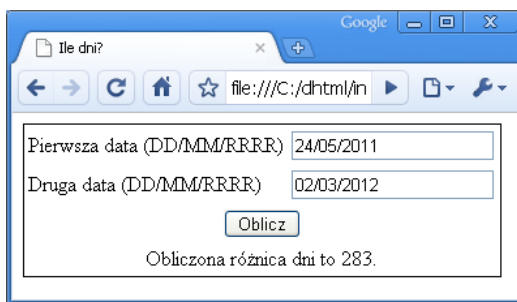
Skrypt 24. [C][E][F][O][S] Obliczanie liczby dni między podanymi datami

Korzystając z metod udostępnianych przez obiekt `Date`, można napisać skrypt pozwalający na obliczenie liczby dni występujących między dwoma datami. Na stronie zostaną umieszczone dwa pola tekstowe oraz przycisk. Kliknięcie przycisku będzie powodowało wykonanie obliczeń i wyświetlenie wyniku. Interfejs skryptu będzie wyglądał tak, jak zaprezentowano to na rysunku 2.5. Kod HTML przyjmie zatem postać przedstawioną na listingu 2.9.

Listing 2.9. Treść sekcji `<body>` skryptu 24

```
<body>
  <div>
    <table style="border:1px solid black;">
      <tr>
        <td>Pierwsza data (DD/MM/RRRR)</td>
```

Rysunek 2.5.
*Interfejs skryptu
do obliczania
liczby dni*



```

        <td><input type="text" id="tfd1" /></td>
    </tr>
    <tr>
        <td>Druga data (DD/MM/RRRR)</td>
        <td><input type="text" id="tfd2" /></td>
    </tr>
    <tr>
        <td colspan="2" style="text-align:center;">
            <input type="button" value="Oblicz" onclick="btnObliczClick();">
        </td>
    </tr>
    <tr>
        <td colspan="2" style="text-align:center;" id="wynik">
        </td>
    </tr>
</table>
</div>
</body>

```

Do formatowania elementów witryny została użyta tabela HTML generowana za pomocą znaczników `<table>`, `<tr>` i `<td>`. To pozwoliło na proste i szybkie ułożenie elementów interfejsu na witrynie. W komórkach tabeli umieszczone zostały dwa pola tekstowe oraz przycisk. Pola służą do wprowadzania dat, a przycisk do wywołania skryptu dokonującego obliczeń. Ostatnia komórka tabeli jest pusta. W niej będzie wyświetlany wynik wykonywanych operacji. Przy opisie pól tekstowych został podany format, w jakim będą musiały być wprowadzane daty. Jest to DD/MM/RRRR, np. 24/05/2011. Poprawność wprowadzonych danych będzie weryfikowana w skrypcie. Kod skryptu został przedstawiony na listingu 2.10.

Listing 2.10. *Skrypt obliczający różnicę między dwoma datami*

```

<script type="text/javascript">
    function ileDni(d1, d2)
    {
        var arr1 = d1.split("/");
        var arr2 = d2.split("/");

        var data1 = new Date(arr1[2], arr1[1], arr1[0]);
        var data2 = new Date(arr2[2], arr2[1], arr2[0]);

        var różnica = data2.getTime() - data1.getTime();
        if(różnica < 0) różnica = -różnica;
    }

```

```
        var dni = Math.floor(różnica / (1000 * 60 * 60 * 24));
        return dni;
    }

    function btnObliczClick()
    {
        var data1 = document.getElementById("tfD1").value;
        var data2 = document.getElementById("tfD2").value;

        if(/\\d{2}\\d{2}\\d{4}/.exec(data1) != data1){
            alert("Pierwsza data ma nieprawidłowy format.");
            return;
        }

        if(/\\d{2}\\d{2}\\d{4}/.exec(data2) != data2){
            alert("Druga data ma nieprawidłowy format.");
            return;
        }

        var str = "Obliczona różnica dni to ";
        str += ileDni(data1, data2);
        str += ".";

        var wynik = document.getElementById("wynik");
        wynik.innerHTML = str;
    }
</script>
```

Funkcja `btnObliczClick` jest wywoływana po kliknięciu przycisku *Oblicz*, a jej zadaniem jest weryfikacja danych, przygotowanie komunikatu i wyświetlenie go na stronie. Najpierw są odczytywane i zapisywane w zmiennych pomocniczych `data1` i `data2` wartości z pól tekstowych `tfD1` i `tfD2`. Następnie badana jest zgodność obu wartości z wyrażeniem regularnym w postaci:

```
\\d{2}\\d{2}\\d{4}/
```

To wyrażenie opisuje takie ciągi, które zaczynają się od dwóch cyfr, po których występuje ukośnik, za nim kolejne dwie cyfry, kolejny ukośnik, a za nim jeszcze cztery cyfry. Jest to więc wyrażenie opisujące datę w żądanym przez nas formacie.

Do wykonania testu poprawności danych używana jest metoda `exec`, która poszukuje w tekście przekazanym jej w postaci argumentu ciągów odpowiadających wzorcowi opisanemu w wyrażeniu regularnym. Użyta konstrukcja pozwala na stwierdzenie, czy w danym polu znajduje się tekst dokładnie odpowiadający wzorcowi (ten sposób testowania danych został dokładnie opisany w książce *JavaScript. Praktyczny kurs*, <http://helion.pl/ksiazki/jscpk.htm>).

Po sprawdzeniu poprawności danych jest tworzony komunikat, który ma się pojawić na stronie. Poszukiwana liczba dni między dwoma datami jest obliczana przez funkcję `ileDni`. Treść komunikatu jest zapisywana w komórce tabeli o identyfikatorze `wynik`. Odbywa się to przez przypisanie wartości zmiennej `str` (zawierającej komunikat) właściwości `innerHTML` obiektu `wynik` pobranego wcześniej przez wywołanie metody `getElementById`.

Funkcja `ileDni` otrzymuje argumenty `d1` i `d2` zawierające ciągi opisujące daty. Przyjmuje się przy tym, że dane te są prawidłowe, zostały bowiem zweryfikowane w funkcji `btnObliczClick`. Oba ciągi są rozbijane na części składowe za pomocą metody `split`. Jako separator jest używany znak `/`. Przy prawidłowych danych po rozbiciu ciągów otrzymujemy dwie tablice (`arr1` i `arr2`) zawierające po trzy komórki. Komórka o indeksie 0 zawiera dzień, komórka o indeksie 1 — miesiąc, a komórka o indeksie 2 — rok.

Dane z tablic `arr1` i `arr2` są używane do skonstruowania dwóch obiektów typu `Date`, a następnie jest obliczana liczba milisekund występująca między obiema datami. Zasada obliczenia jest taka sama jak w przykładzie 23. Różnica jest taka, że w przypadku ujemnego wyniku zmieniany jest jego znak:

```
if(różnica < 0) różnica = -różnica;
```

Zamiast instrukcji warunkowej można by w tym miejscu użyć też metody `abs` obiektu `Math`:

```
różnica = Math.abs(różnica);
```

Skrypt 25. Kalendarz

[C][E][F][O][S]

Na stronie WWW można umieścić dynamicznie wygenerowany kalendarz wskazujący aktualną datę. Przyjmuje on postać przedstawioną na rysunku 2.6 i będzie mógł być wyświetlony na dowolnie wskazanej warstwie. W rzeczywistości wszystkie elementy składowe będą częściami zwykłej tabeli, której wystrój zostanie ustalony za pomocą stylów CSS. Generowaniem danych zajmie się skrypt przedstawiony na listingu 2.11.

Rysunek 2.6.

Kalendarz wskazujący bieżącą datę

Grudzień 2009						
Pn	Wt	Śr	Cz	Pt	So	Nd
	1	2	3	4	5	6
7	8	9	10	11	12	13
14	15	16	17	18	19	20
21	22	23	24	25	26	27
28	29	30	31			

Listing 2.11. Skrypt generujący tabelę z kalendarzem

```
<script type="text/javascript">
function rokPrzestepny(rok)
{
    return ((rok % 4 == 0) && ((rok % 100 != 0) || (rok % 400 == 0)));
}
```

```
}  
function wyświetlKalendarz(e1)  
{  
    data = new Date();  
  
    var rok = data.getFullYear();  
    var miesiac = data.getMonth() + 1;  
    var dzienTygodnia = data.getDay();  
    var dzienMiesiaca = data.getDate();  
  
    var tempDate = new Date(rok, miesiac - 1, 1);  
    var pierwszyDzienMiesiaca = tempDate.getDay();  
  
    if(dzienTygodnia == 0) dzienTygodnia = 7;  
    if(pierwszyDzienMiesiaca == 0) pierwszyDzienMiesiaca = 7;  
  
    switch(miesiac){  
        case 1 : nazwaMiesiaca = "Styczeń";  
                 dniwMiesiacu = 31;  
                 break;  
        case 2 : nazwaMiesiaca = "Luty";  
                 dniwMiesiacu = rokPrzestepny(rok)?29:28;  
                 break;  
        case 3 : nazwaMiesiaca = "Marzec";  
                 dniwMiesiacu = 31;  
                 break;  
        case 4 : nazwaMiesiaca = "Kwiecień";  
                 dniwMiesiacu = 30;  
                 break;  
        case 5 : nazwaMiesiaca = "Maj";  
                 dniwMiesiacu = 31;  
                 break;  
        case 6 : nazwaMiesiaca = "Czerwiec";  
                 dniwMiesiacu = 30;  
                 break;  
        case 7 : nazwaMiesiaca = "Lipiec";  
                 dniwMiesiacu = 31;  
                 break;  
        case 8 : nazwaMiesiaca = "Sierpień";  
                 dniwMiesiacu = 31;  
                 break;  
        case 9 : nazwaMiesiaca = "Wrzesień";  
                 dniwMiesiacu = 30;  
                 break;  
        case 10 : nazwaMiesiaca = "Październik";  
                 dniwMiesiacu = 31;  
                 break;  
        case 11 : nazwaMiesiaca = "Listopad";  
                 dniwMiesiacu = 30;  
                 break;  
        case 12 : nazwaMiesiaca = "Grudzień";  
                 dniwMiesiacu = 31;  
                 break;  
    }  
  
    var str = "";
```

```

str += "<table border='1'><tr>";
str += "<td class='tdNaglowek' colspan='7'>";
str += nazwaMiesiaca + " " + rok;
str += "<\td><\tr>";

str += "<tr class='trNazwyDni'>";
str += "<td>Pn<\td>";
str += "<td>Wt<\td>";
str += "<td>Śr<\td>";
str += "<td>Cz<\td>";
str += "<td>Pt<\td>";
str += "<td>So<\td>";
str += "<td>Nd<\td>";
str += "<\tr><tr>";

var j = dniwMiesiacu + pierwszyDzienMiesiaca - 1;

for(var i = 0; i < j; i++){
    if(i < pierwszyDzienMiesiaca - 1){
        str += "<td class='tdEmpty'><\td>";
        continue;
    }
    if((i % 7) == 0){
        str += "<\tr><tr>";
    }
    var klasa = "tdDzien";
    if((i - pierwszyDzienMiesiaca + 2) == dzienMiesiaca){
        klasa = "tdBiezacyDzien";
    }
    str += "<td class='" + klasa + "'>";
    str += i - pierwszyDzienMiesiaca + 2;
    str += "<\td>";
}
str += "<\tr><\table>";

var el = document.getElementById(el);
if(el) el.innerHTML = str;
}
</script>

```

Podstawą konstrukcji kalendarza jest zwyczajna tabela HTML wygenerowana za pomocą znaczników `<table>`, `<tr>` i `<td>`. Przed rozpoczęciem budowy tabeli konieczne jest pobranie niezbędnych danych. Dlatego jest wykonywana seria operacji na obiekcie `Date`, a ich wynik zapisywany w następujących zmiennych:

- ♦ rok — aktualny rok (pobierany za pomocą metody `getYear()`);
- ♦ miesiac — aktualny miesiąc (pobierany za pomocą metody `getMonth()`);
- ♦ dzienTygodnia — numer dnia w tygodniu (pobierany za pomocą metody `getDay()`);
- ♦ dzienMiesiaca — numer dnia w miesiącu (pobierany za pomocą metody `getDate()`);
- ♦ pierwszyDzienMiesiaca — numer dnia tygodnia, którym jest pierwszy dzień bieżącego miesiąca.

Pierwszy dzień miesiąca jest ustalany przez utworzenie daty tymczasowej (`new Date(rok, miesiac - 1, 1)`), a następnie wywołaniem metody `getDay`.

Ponieważ numer dnia tygodnia zwracany przez metodę `getDay` jest określony w taki sposób, że niedziela ma numer 0, poniedziałek 1 itd., dla ułatwienia dalszych obliczeń numer niedzieli jest zmieniany na 7. Czyli jeśli zostanie wykryte, że zmienna `dzienTygodnia` lub `pierwszyDzienMiesiaca` ma wartość 0, jest zmieniana na 7.

Instrukcja wyboru `switch...case` służy do ustalenia polskiej nazwy miesiąca (zapisywanej w zmiennej `nazwaMiesiaca`) oraz liczby dni w miesiącu (zapisywanej w zmiennej `dniWmiesiacu`). W przypadku lutego trzeba dodatkowo sprawdzić, czy bieżący rok jest rokiem przestępnym. Używana jest w tym celu osobna funkcja o nazwie `rokPrzestepny`. Sprawdza ona, czy wartość przekazana jej jako argument (czyli `rok`) spełnia warunek przestępnosci. Rok przestępny jest bowiem wtedy, kiedy dzieli się on przez cztery, ale nie dzieli się przez 100, chyba że dzieli się przez 400. Za sprawdzenie odpowiada zatem wyrażenie logiczne w postaci:

```
(rok % 4 == 0) && ((rok % 100 != 0) || (rok % 400 == 0))
```

Znaczniki składające się na tabelę z kalendarzem są dopisywane do zmiennej `str`, której początkowo przypisywany jest pusty ciąg znaków. Pierwsza komórka tabeli ma zawierać jedynie określenie miesiąca oraz roku i składa się z połączonych siedmiu kolejnych komórek (atrybut `colspan='7'`), czyli pierwszy wiersz zawiera jedną komórkę powstałą z połączenia wszystkich siedmiu kolumn tabeli. Została jej przypisana klasa CSS `tdNaglowek`.

Drugi wiersz zawiera skrótowe określenie kolejnych dni: *Pn, Wt, Śr* itd. W tym przypadku klasa CSS (`trNazwyDni`) została przypisana do całego wiersza, dzięki czemu nie trzeba powtarzać atrybutu `class` przy każdej komórce. Komórki odziedziczą style po wierszu.

Komórki określające dni miesiąca są generowane w pętli `for`. Należy pamiętać, że każdy miesiąc może rozpoczynać się w innym dniu tygodnia, dlatego też najczęściej jedna lub kilka początkowych komórek powinny być puste. Takie puste komórki należy generować tak długo, jak zmienna iteracyjna `i` jest mniejsza od wartości `pierwszyDzienMiesiaca - 1`. Dlatego też dopóki prawdziwy jest warunek:

```
i < pierwszyDzienMiesiaca - 1
```

dopóty generowane są puste komórki, którym nadawane jest klasa CSS o nazwie `tdEmpty`.

Nowy wiersz tabeli, czyli nowy tydzień, należy rozpocząć, kiedy wartość `i % 7` jest równa 0, czyli kiedy reszta z dzielenia `i` przez 7 jest równa zero. Ponieważ aktualny dzień miesiąca ma zostać wyróżniony innym kolorem (np. żółtym), będzie potrzebna również instrukcja warunkowa sprawdzająca, czy `i` jest równe numerowi bieżącego dnia w miesiącu. Warunek taki ma postać:

```
(i - pierwszyDzienMiesiaca + 2) == dzienMiesiaca)
```

Gdy jest prawdziwy, bieżąca komórka otrzymuje klasę CSS `tdBiezacyDzien`, a gdy jest fałszywy — klasę `tdDzien`.

Po zakończeniu pętli zmienna `str` zawiera pełną tabelę. Warstwa, na której ma być umieszczona, jest przekazywana funkcji `wyświetlKalendarz` w postaci argumentu `el`. Dlatego też za pomocą metody `getElementById` pobierane jest odwołanie do tego elementu i jeśli ta operacja zakończyła się sukcesem, właściwości `innerHTML` jest przypisywana wartość zmiennej `str`. Dzięki temu kalendarz pojawia się na warstwie.

Przykładowy sposób umieszczenia kalendarza na witrynie został przedstawiony na listingu 2.12.

Listing 2.12. *Sekcja `<body>` strony z kalendarzem*

```
<body>
  <div id="divKalendarz">
  </div>
  <script type="text/javascript">wyświetlKalendarz('divKalendarz');</script>
</body>
```

W sekcji `<body>` została umieszczona pusta warstwa o identyfikatorze `divKalendarz`, a za nią skrypt wywołujący funkcję `wyświetlKalendarz`. Ponieważ w wywołaniu została przekazana nazwa warstwy, to na niej właśnie zostanie wyświetlony kalendarz. Oczywiście w takiej postaci będzie czarno-biały, bez żadnych wyróżnień. Aby go uatrakcyjnić, trzeba zdefiniować reguły dla klas CSS wymienionych w opisie skryptu. Przykładowe reguły zostały przedstawione na listingu 2.13.

Listing 2.13. *Reguły CSS dla kalendarza*

```
<style type="text/css">
  table{
    border:1px solid black;
    empty-cells:hide;
  }
  td{
    border:1px solid black;
    width:1.5em;
  }
  .tdNaglowek{
    background-color:yellow;
    text-align:center;
  }
  .trNazwyDni{
    background-color:pink;
  }
  .tdDzien{
    background-color:green;
  }
  .tdBiezacyDzien{
    background-color:yellow;
  }
</style>
```

Tabela oraz komórki otrzymały 1-pikselowe ciągle obramowanie (`1px solid black`). Zostało wyłączone wyświetlanie pustych komórek (`empty-cells:hide`), a szerokość pojedynczej komórki określona na `1.5em`. Klasy dotyczące różnych rodzajów komórek

(nagłówki, nazwy dni, numery dni, numer dnia bieżącego) otrzymały różne kolory tła (cecha background-color). Nagłówek oraz dzień bieżący jest prezentowany na tle żółtym (yellow), zwykłe dni — na zielonym (green), a skróty nazw dni — na różowym (pink).

Skrypt 26. [C][E][F][O][S] Kalendarz w dowolnym miejscu strony (drag & drop)

Przykład 25. umożliwiał umieszczenie na stronie WWW kalendarza wskazującego bieżącą datę. Kalendarz musiał się znajdować w miejscu ściśle określonym w kodzie HTML. Nic jednak nie stoi na przeszkodzie, aby przekształcić go w gadżet, którego położenie będzie mogło być dowolnie zmieniane przez użytkownika. Wystarczy do skryptu funkcje realizujące funkcjonalność typu drag & drop. Kod przyjmie wtedy postać przedstawioną na listingu 2.14.

Listing 2.14. *Przesuwalny kalendarz*

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN"
"http://www.w3.org/TR/html4/strict.dtd">
<html lang="pl">
  <head>
    <meta http-equiv="Content-Type" content="text/html; charset=utf-8">
    <title>Kalendarz</title>
    <style type="text/css">

      /* tutaj definicje stylów dla kalendarza */

      #divKalendarz{
        background-color:white;
        position:absolute;
        cursor:move;
        right:2px;
        top:2px;
        max-width:200px;
      }
    </style>
    <script type="text/javascript">
      var draggedDiv = null;
      var offset = null;
      document.onmouseup = mouseUp;
      document.onmousemove = mouseMove;

      /* tutaj funkcje dotyczące funkcjonalności drag & drop:
         mousePos, startDrag, mouseUp, mouseMove */

      /* tutaj funkcje dotyczące funkcjonalności kalendarza:
         rokPrzestepny i wyswietlKalendarz */
```

```
        </script>
    </head>
    <body>
        <!-- tutaj treść strony -->
        <div id="divKalendarz" onmousedown="startDrag(this);">
        </div>
        <script type="text/javascript">wyświetlKalendarz('divKalendarz');</script>
    </body>
</html>
```

Ta witryna to w istocie połączenie pomysłów przedstawionych w skryptach 25. i 62. W sekcji `<body>` została umieszczona warstwa o identyfikatorze `kalendarzDiv`, a za nią skrypt z wywołaniem funkcji `wyświetlKalendarz`. Warstwie została też przypisana procedura obsługi zdarzenia `mousedown` w postaci funkcji `startDrag`. To spowoduje, że będzie mogła być przemieszczana po stronie.

W sekcji `<head>` znajduje się znacznik `<style>` określający style użyte w witrynie. Oprócz stylów określających wygląd kalendarza została w nim umieszczona reguła z selektorem `#divKalendarz` dotycząca warstwy `divKalendarz`. Warstwa będzie miała biały kolor (`white`), kursor w postaci skrzyżowanych strzałek (`move`), bezwzględne pozycjonowanie (`absolute`) i zostanie umiejscowiona w odległości 2 pikseli od prawej (`right`) i górnej (`top`) krawędzi okna. Została również określona maksymalna szerokość warstwy na 200 pikseli (`max-width:200px`).

Część skryptowa to po prostu połączenie funkcji zaprezentowanych w przykładach 25. i 62. Nie trzeba w nich wprowadzać żadnych zmian. Na początku znalazły się definicje zmiennych obsługujących funkcjonalność `drag & drop`, dalej związane z tym funkcje, a na końcu — funkcje służące do generowania kalendarza. Tak przygotowany kalendarz będzie mógł być dowolnie przemieszczany po witrynie. Użytkownik sam zadecyduje, w którym miejscu ten gadżet ma się znaleźć.

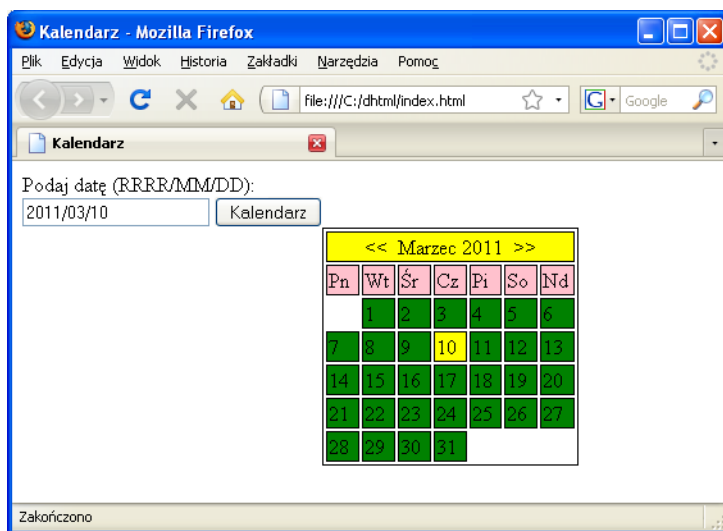
Skrypt 27.

[C][E][F][O][S]

Kalendarz typu pop-up pozwalający na wskazanie daty

Istnieje wiele sytuacji, w których na stronie WWW trzeba podać pewną datę, najczęściej wprowadza się ją do pola tekstowego (lub pól tekstowych). Przydatnym udogodnieniem jest możliwość wyświetlenia kalendarza i wskazanie daty za pomocą myszy. Nie trzeba wtedy podawać żadnych danych z klawiatury, a wprowadzony ciąg od razu będzie miał żądany format. Przykład takiego rozwiązania zaprezentowano na rysunku 2.7. Do pola należy wprowadzić datę w formacie `RRRR/MM/DD`. Wybór może być wspomagany kalendarzem wyświetlanym po kliknięciu przycisku *Kalendarz*. Kliknięcie wybranego dnia powoduje wprowadzenie danych do pola tekstowego.

Rysunek 2.7.
Kalendarz
umożliwiający
wybór daty



Po wyświetleniu na stronie kalendarz będzie wskazywał datę bieżącą. Umieszczone w górnej części symbole << i >> umożliwią natomiast przemieszczanie się o jeden miesiąc w przód i w tył. Dzień aktualnie wskazywany przez kursor myszy będzie podświetlany na żółto.

Nietrudno się zorientować, że podstawą kalendarza będzie odpowiednio zmodyfikowany skrypt 25. Trzeba też będzie rozwiązać problem wyświetlania warstwy z kalendarzem przy prawym dolnym rogu przycisku. W tym celu zostanie użyta technika podpowiedzi opisana dokładnie w skrypcie 59. Zaczniemy jednak od części HTML. Kod sekcji <body> przyjmie postać widoczną na listingu 2.15.

Listing 2.15. Kod sekcji <body> skryptu 27.

```
<body>
  <div>
    Podaj datę (RRRR/MM/DD):<br />
    <input type="text" id="tfData" />
    <input type="button" value="Kalendarz"
      onclick="pokażWarstwę(this, 'divKalendarz');" />
  </div>
  <div id="divKalendarz" style="display:none;">
    <script type="text/javascript">
      var data = new Date();
      wyświetlKalendarz(data.getFullYear(),
        data.getMonth(), data.getDate(), 'divKalendarz');
    </script>
  </div>
</body>
```

W kodzie znajdują się dwie warstwy. Pierwsza jest kontenerem dla elementów interfejsu: pola tekstowego i przycisku. Pole ma identyfikator tfData i służy do wprowadzania daty w opisanym formacie. Przycisk umożliwia wywołanie kalendarza, który pozwala na wybór daty za pomocą kliknięcia. Zdarzenie click przycisku ma przypisaną

procedurę obsługi w postaci funkcji `pokażWarstwę`. Wyświetla ona warstwę o nazwie wskazanej przez drugi argument wywołania przy prawym dolnym rogu elementu (obiektu), wskazanego za pomocą pierwszego argumentu. W tym kontekście wywołanie:

```
pokażWarstwę(this, 'divKalendarz');
```

oznacza wyświetlanie warstwy o nazwie `divKalendarz` w prawym dolnym rogu elementu inicjującego wywołanie, czyli przycisku *Kalendarz*.

Druga warstwa jest kontenerem dla kalendarza i początkowo nie jest wyświetlana. Jest to osiągnięte dzięki zastosowaniu stylu `display:none;`. Wewnątrz warstwy został umieszczony skrypt generujący kalendarz. To oznacza, że po załadowaniu strony bieżąca zawartość (znacznik `<script>`) zostanie zamieniona przez wygenerowaną dynamicznie treść. Generowaniem treści zajmuje się funkcja `wyświetlKalendarz` przyjmująca cztery argumenty. Pierwsze trzy określają dzień, miesiąc i rok, który ma się pojawić w kalendarzu, a czwarty — warstwę, na której ten kalendarz ma być wyświetlany. Bieżące dane są uzyskiwane dzięki wywołaniom metod `getFullYear`, `getMonth` i `getDate`.

Skrypt obsługujący witrynę będzie się składał z funkcji:

- a) generujących kalendarz: `rokPrzestępny`, `wyświetlKalendarz`;
- b) obsługujących wyświetlanie: `pokażWarstwę`, `schowajWarstwę`;
- c) obsługujących interfejs: `ustawDatę`.

Pierwsza grupa będzie w dużej mierze oparta na przykładzie 25., w funkcji `wyświetlKalendarz` nastąpi jednak kilka znaczących zmian. Zostaną omówione fragmentami ze względu na znaczną długość pełnego listingu. Pierwsze zmiany nastąpiły w deklaracji i pierwszej części kodu. Początek funkcji będzie wyglądał tak jak na listingu 2.16.

Listing 2.16. *Początek funkcji*

```
function wyświetlKalendarz(r, m, d, el)
{
    var data = new Date(r, m, d);
    if(data == "Invalid date") data = new Date();
    //dalszy kod funkcji
```

Funkcja otrzymuje cztery argumenty. Pierwsze trzy wskazują rok, miesiąc i dzień daty, która zostanie pokazana w kalendarzu. Te argumenty są używane do skonstruowania nowego obiektu typu `Date`. Ponieważ nie można mieć pewności, że dane są poprawne, następuje porównanie wartości zmiennej `data` i ciągu `Invalid date`. Jeżeli zostanie stwierdzona zgodność (co by oznaczało, że dane są nieprawidłowe), tworzony jest obiekt wskazujący datę bieżącą. Jeśli więc przekazane argumenty nie będą poprawne, w kalendarzu zostanie wyświetlona aktualna data.

Ostatni z argumentów wskazuje nazwę warstwy (elementu witryny), na którym będzie wyświetlany kalendarz. Będzie używany w dalszej części kodu.

Kolejne zmiany dotyczą fragmentu generującego nagłówkę, czyli pierwszą komórkę tabeli zawierającą nazwę miesiąca oraz rok. Konieczne jest dodanie interfejsu, dzięki

któremu będzie można przemieszczać się o jeden miesiąc w przód i w tył. Dlatego kod przyjmie postać przedstawioną na listingu 2.17.

Listing 2.17. Generowanie komórki nagłówkowej

```
var str = "";

str += "<table border='1'><tr>";
str += "<td class='tdNaglowek' colspan='7'>";
str += "<span class='clickable' onclick='wyświetlKalendarz(";
str += rok + "," + (miesiąc - 2) + "," + dzieńMiesiaca + ",\\"";
str += el + "\\");'>&lt;&lt;</span>&nbsp;&nbsp;&nbsp;";
str += nazwaMiesiaca + " " + rok + "&nbsp;&nbsp;&nbsp;";
str += "<span class='clickable' onclick='wyświetlKalendarz(";
str += rok + "," + (miesiąc) + "," + dzieńMiesiaca + ",\\"";
str += el + "\\');>&gt;&gt;</span>";
str += "</td></tr>";
```

Co prawda taki kod jest niezbyt czytelny, ale chodzi w nim o wygenerowanie początkowego fragmentu tabeli o następującej postaci (dodane zostało formatowanie ułatwiające orientację):

```
<table border='1'>
  <tr>
    <td class='tdNaglowek' colspan='7'>
      <span class='clickable'
        onclick='wyświetlKalendarz(rok, miesiąc - 1,dzień,"divKalendarz");'>
        &lt;&lt;</span>
        &nbsp;&nbsp;&nbsp; nazwa_miesiąca rok &nbsp;&nbsp;&nbsp;
      <span class='clickable'
        onclick='wyświetlKalendarz(rok,miesiąc + 1,dzień,"divKalendarz");'>
        &gt;&gt;&gt;</span>
    </td>
  </tr>
```

Jest to zatem nagłówkowa komórka złożona z połączonych siedmiu kolumn, zawierająca nazwę miesiąca oraz rok, a także znaki << i >> (tak jak zostało to zaprezentowane na rysunku 2.7). Symbol << (złożony z encji <) ma umożliwiać zmianę miesiąca o jeden w tył, a >> (złożony z encji >) — o jeden wprzód. Oba muszą więc reagować na kliknięcia. Dlatego została im przypisana procedura obsługi zdarzenia click w postaci funkcji wyświetlKalendarz. Argumenty przekazane funkcji odpowiadają właściwym przesunięciom miesięcznym. Ponieważ wcześniej w kodzie numer miesiąca został zwiększony o 1 (tak by styczeń miał numer 1, a grudzień 12¹), operacja *miesiąc - 1* w kodzie jest wykonywana jako (*miesiąc - 2*), a operacja *miesiąc + 1* — jako (*miesiąc*).

Zmiany trzeba wprowadzić także we fragmencie generującym komórki tabeli odzwierciedlające poszczególne dni miesiąca. Wewnątrz pętli for, za instrukcją warunkową

¹ Nie jest to konieczne. Można pozostawić nienaturalną numerację od 0, dokonując odpowiednich modyfikacji w kodzie. Wtedy numery miesięcy będą niezgodne z intuicją, ale wykonywane działania już tak. To jednak kwestia czysto techniczna.

if((i % 7) == 0){ odpowiedzialną za wygenerowanie kolejnego wiersza, należy umieścić kod zaprezentowany na listingu 2.18.

Listing 2.18. Generowanie komórek tabeli

```
var klasa = "tdDzien";
var dzien = i - pierwszyDzienMiesiaca + 2;
if(dzien == dzienMiesiaca &&
    (new Date()).toDateString() == data.toDateString()){
    klasa = "tdBiezacyDzien";
}
str += "<td class='" + klasa + "' onclick='ustawDate(";
str += rok + "," + miesiac + "," + dzien + ")';";
str += "schowajWarstwę(\"divKalendarz\")';>"
str += dzien;
str += "</td>";
```

W stosunku do pierwotnego kodu nastąpiło tu kilka znaczących zmian. Wartość wyrażenia określającego bieżący dzień została przypisana zmiennej pomocniczej dzien:

```
var dzien = i - pierwszyDzienMiesiaca + 2;
```

Dzięki temu nie będzie trzeba wielokrotnie wykonywać tego obliczenia. Niezbędne stało się także sprawdzenie, czy wyświetlanie dotyczy bieżącego miesiąca. Tylko wtedy bowiem należy wyróżniać numer bieżącego dnia. Wykonywana jest więc złożona instrukcja warunkowa if. Po pierwsze, bada ona, czy w danym przebiegu pętli aktualnie generowany numer dnia jest zgodny z dniem bieżącym (warunek dzien == dzienMiesiaca). Po drugie, stwierdza, czy bieżąca data jest zgodna z tą zapisaną w obiekcie data. Tylko wtedy gdy oba te warunki są spełnione jednocześnie, należy wyróżnić bieżącą komórkę. Odbywa się to przez przypisanie jej klasy tdBiezacyDzien.

Ponieważ każda komórka musi reagować na kliknięcia, niezbędne stało się użycie zdarzenia click (i atrybutu onclick). W odpowiedzi na to zdarzenie wywoływane są dwie funkcje: ustawDate oraz schowajWarstwę. Zadaniem pierwszej jest wprowadzenie bieżącej daty (przekazanej w postaci argumentów) do znajdującego się na stronie pola tekstowego tfData, zadaniem drugiej — ukrycie warstwy z kalendarzem.

Ogólny kod HTML komórki tabeli generowanej za pomocą przedstawionego kodu będzie miał następującą postać:

```
<td class='nazwa_klasy' onclick=
    'ustawDate(rok,miesiac,dzien);schowajWarstwę("divKalendarz");'
>numer_dnia</td>
```

Treść funkcji ustawDate została zaprezentowana na listingu 2.19.

Listing 2.19. Treść funkcji ustawDate

```
function ustawDate(r, m, d)
{
    var tfData = document.getElementById('tfData');
    if(tfData){
        m = m < 10 ? "0" + m : "" + m;
        d = d < 10 ? "0" + d : "" + d;
```



```
        tfData.value = r + "/" + m + "/" + d;
    }
}
```

Jej działanie jest bardzo proste. Otrzymuje trzy argumenty — *r*, *m*, *d* — określające dzień, miesiąc i rok, które mają się pojawić w polu tekstowym *tfData*. Odwołanie do pola jest pobierane za pomocą metody *getElementById*. Jeśli operacja ta zakończy się sukcesem, dane są przetwarzane do postaci dwucyfrowej oraz przypisywane właściwości *value* obiektu *tfData*, dzięki czemu pojawiają się w polu. Składowe daty oddzielane są ukośnikami.

Funkcje *pokażWarstwę* i *schowajWarstwę* zostały skonstruowane na takiej samej zasadzie jak funkcje *showTooltip* i *closeTooltip* z przykładu 59. (listing 7.3).

Do wykończenia skryptu brakuje jeszcze tylko stylów CSS. Ponieważ wygląd kalendarza nie był zmieniany w stosunku do przykładów 25. i 26., można użyć reguł zaprezentowanych na listingach 2.13 i 2.14. Warto jednak dodać do nich definicję klasy *clickable* oraz regułę z selektorem *td.tdDzien: hover*, tak jak zostało to przedstawione na listingu 2.20.

Listing 2.20. *Style dla wyskakującego kalendarza*

```
<style type="text/css">

    /* definicje stylów z listingu 2.13 i 2.14 */

    .clickable{
        cursor:pointer;
    }
    td.tdDzien: hover{
        background-color:yellow;
    }
</style>
```

Klasa *clickable* dotyczy elementów, które mogą być kliknięte. Typowym wyróżnikiem takich elementów jest kursor-wskaźnik — stąd też definicja cechy *cursor*. Warto też wyróżniać komórkę z numerem dnia, o ile znajdzie się nad nią kursor myszy. Odpowiada za to druga reguła. Kolor tła takiej komórki stanie się żółty (ten efekt nie będzie widoczny w przeglądarce Internet Explorer w wersji 6.).

Skrypt 28. [C][E][F][O][S]

Treść zależna od godziny (pory dnia)

Wygląd strony WWW można uzależnić od pory dnia. W tym celu należy wykorzystać obiekt *Date* do określania aktualnego czasu. Taki właśnie sposób został zastosowany w skrypcie widocznym na listingu 2.21. Doba została podzielona na 4 części: ranek, popołudnie, wieczór i noc. Przyjęto, że pora nocna trwa od godziny 22 do 4, poranna

od 4 do 12, popołudniowa od 12 do 18, a wieczorna od 18 do 22. Oczywiście podział może być dowolnie inny.

Listing 2.21. *Różne wersje strony zależne od pory dnia*

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN"
"http://www.w3.org/TR/html4/strict.dtd">
<html lang="pl">
  <head>
    <meta http-equiv="Content-Type" content="text/html; charset=utf-8">
    <title>Pora dnia</title>
    <script type="text/javascript">
      data = new Date();
      godzina = data.getHours();
      if(godzina > 22 && godzina < 4)
        document.location.href = "indexNoc.html";
      else if(godzina >= 4 && godzina < 12)
        document.location.href = "indexRano.html";
      else if(godzina >= 12 && godzina < 18)
        document.location.href = "indexPopoludnie.html";
      else if(godzina >= 18 && godzina <= 22)
        document.location.href = "indexWieczor.html";
    </script>
  </head>
  <body>
    <div>
      <a href="indexNoc.html">Nocna strona</a>
      <a href="indexRano.html">Poranna strona</a>
      <a href="indexPopoludnie.html">Popołudniowa strona</a>
      <a href="indexWieczor.html">Wieczorna strona</a>
    </div>
  </body>
</html>
```

Skrypt umieszczony w sekcji `<head>` w znaczniku `<script>` jest wykonywany od razu po załadowaniu witryny do przeglądarki. W przypadku gdyby przeglądarka nie obsługiwała skryptów (lub obsługa ta byłaby wyłączona), na ekranie pojawi się treść sekcji `<body>` zawierająca odnośniki do podstron dedykowanych poszczególnym porom dnia. W każdej zatem sytuacji użytkownik witryny będzie miał dostęp do jej treści.

W skrypcie aktualna godzina jest pobierana za pomocą metody `getHours` i zapisywana w zmiennej `godzina`. Następnie jest wykorzystywana złożona instrukcja warunkowa `if...else` do określenia przedziałów czasowych, które oczywiście można dobrać wedle indywidualnych potrzeb. W zależności od tego, w którym przedziale czasowym mieści się godzina zapisana w zmiennej `godzina`, jest ładowana inna strona WWW:

- ♦ *indexNoc* — dla pory nocnej,
- ♦ *indexRano* — dla poranka,
- ♦ *indexPopoludnie* — dla popołudnia,
- ♦ *indexWieczor* — dla pory wieczornej.

Strony są wczytywane przez przypisanie ich adresu (nazwy pliku) właściwości `href` obiektu `location`.

Skrypt 29. Treść zależna od daty

[C][E][F][O][S]

W skrypcie 28. treść witryny była uzależniona od pory dnia, w której strona została odwiedzona. Skoro można uzależnić zawartość od godziny, to można również od dnia tygodnia lub miesiąca. Przykładowo inna wersja witryny może pojawiać się w zwykły dzień, a inna w weekend. Tak właśnie działa skrypt przedstawiony na listingu 2.22. Używana jest metoda `getDay` obiektu `Date`, która zwraca następujące wartości określające dzień tygodnia: 0 — niedziela, 1 — poniedziałek, 2 — wtorek itd.

Listing 2.22. Strona zależna od dnia tygodnia

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN"
"http://www.w3.org/TR/html4/strict.dtd">
<html lang="pl">
  <head>
    <meta http-equiv="Content-Type" content="text/html; charset=utf-8">
    <title>Dzień tygodnia</title>
    <script type="text/javascript">
      var data = new Date();
      var dzień = data.getDay();
      if(dzień == 0 && dzień == 6)
        document.location.href = "indexWeekend.html";
      else
        document.location.href = "indexZwykly.html";
    </script>
  </head>
  <body>
    <div>
      <a href="indexWeekend.html">Strona na weekend</a>
      <a href="indexZwykly.html">Strona na zwykły dzień</a>
    </div>
  </body>
</html>
```

Ogólna konstrukcja strony jest taka sama jak w przykładzie 28. Po załadowaniu strony jest wykonywany skrypt zawarty między znacznikami `<script>` i `</script>`, a w przypadku gdyby wykonanie skryptu nie było możliwe, wyświetlana jest treść sekcji `<body>` zawierająca odnośniki do poszczególnych podstron. W skrypcie tworzony jest nowy obiekt typu `Date`, a następnie wykonywana metoda `getDay` pobierająca numer dnia tygodnia. Instrukcja warunkowa `if` bada, czy pobrana wartość to 0 (niedziela) lub 6 (sobota). Jeżeli tak, wczytywana jest strona zawarta w pliku *indexWeekend.html*, a jeżeli nie — strona zwarta w pliku *indexZwykly.html*.

Gdybyśmy chcieli, aby każdy dzień tygodnia miał przypisaną swoją własną wersję strony, zamiast instrukcji warunkowej równie dobrze sprawdziłaby się instrukcja wyboru `switch`. Treść skryptu mogłaby wtedy wyglądać tak, jak przedstawiono to na listingu 2.23.


```
        else if(godzina >= g1 && godzina <= g2)
            div.innerHTML = str2;
        else
            div.innerHTML = str1;
        setTimeout("sprawdź(" + g1 + "," + g2 + "," + divId + "');", timeout);
    }
</script>
</head>
<body onload="sprawdź(18, 20, 'dataDiv');">
    <div>
        Treść strony.
    </div>
    <div id="dataDiv">
    </div>
</body>
</html>
```

W sekcji `<body>` zostały umieszczone dwie warstwy. Na pierwszej ma znajdować się właściwa treść strony, a na drugiej — o identyfikatorze `dataDiv` — treść zależna od aktualnej godziny. Ustalaniem zawartości drugiej warstwy zajmuje się funkcja `sprawdź`, której pierwsze wywołanie następuje automatycznie po załadowaniu strony. Jest to wynikiem użycia zdarzenia `load` znacznika `<body>`.

Funkcja `sprawdź` przyjmuje trzy argumenty. Pierwsze dwa określają przedział czasowy (godzinę początkową — `g1` i końcową — `g2`), w którym ma być wyświetlana specjalna treść, a trzeci — identyfikator warstwy, na której ta treść ma być wyświetlana. Wywołanie `sprawdź(18, 20, 'dataDiv')` oznacza zatem, że między godziną 18 a 20 na warstwie `dataDiv` ma być wyświetlana treść specjalna.

To, co ma się znaleźć na warstwie w zwykłych godzinach, powinno być zapisane w zmiennej globalnej `str1`, a to, co się pojawi w godzinach specjalnych — w `str2`. Czas pomiędzy kolejnymi wywołaniami funkcji `sprawdź` jest również określany globalnie. Odpowiada za to zmienna `timeout`. Jej początkowa wartość to 60×1000 milisekund, czyli jedna minuta.

Początek kodu funkcji to utworzenie nowego obiektu typu `Date` i określenie na jego podstawie aktualnej godziny (wywołanie metody `getHours`). Odczytana wartość jest zapisywana w zmiennej `godzina`. Pobierane jest także i zapisywane w zmiennej `div` odwołanie do warstwy o identyfikatorze przekazanym w postaci argumentu `divId`. W przypadku gdyby odwołanie nie mogło być pobrane, wykonywanie funkcji jest kończone za pomocą instrukcji `return`.

Wyświetlanie treści na warstwie jest wykonywane przez przypisanie odpowiedniej wartości właściwości `innerHTML` obiektu wskazywanego przez `div`. Jeśli mamy do czynienia z przedziałem specjalnym określonym przez argumenty `g1` i `g2`, powinna to być zawartość zmiennej `str2`, a w przeciwnym przypadku — treść zmiennej `str1`. Istnieją jednak dwa możliwe rodzaje przedziałów specjalnych, które są badane w złożonej instrukcji warunkowej `if...else if`.

Zwykły przedział to np. godziny od 8 do 10 czy od 14 do 22. Wartość `g1` jest wtedy mniejsza niż `g2`. Z takim przedziałem mamy zatem do czynienia, gdy prawdziwy jest

warunek `godzina >= g1 && godzina <= g2`. Inaczej będzie jednak, gdy wartość `g1` jest większa niż `g2`. Przykładem może być przedział od 22 do 4 — oznacza to po prostu, że przekraczamy północ. Wtedy warunek powinien mieć postać: `godzina >= g1 || godzina <= g2`.

Funkcja jest wywoływana cyklicznie dzięki znajdującemu się na jej końcu wywołaniu metody `setTimeout`. Pierwszy argument metody to instrukcja powodująca kolejne wywołanie funkcji. Ponieważ musi ono zawierać te same argumenty co wywołanie pierwotne, argument jest składany z kilku ciągów przeplatanych wartościami kolejnych argumentów:

```
"sprawdź(" + g1 + ", " + g2 + ", '" + divId + "');"
```

Rozdział 3.

Style CSS

Style CSS były już używane w poprzednich rozdziałach, trudno dziś nawet o prostą stronę, która może obyć się bez tej techniki. Jednak rozdział trzeci poświęcony jest wyłącznie modyfikacjom stylów dokonywanym przy użyciu JavaScriptu. Wiadomo np., że nierzadko różne przeglądarki odmiennie interpretują reguły CSS, konieczne może być więc przygotowanie różnych wersji. Jak jednak dynamicznie wczytać zestaw reguł dla całej witryny, jak umożliwić użytkownikom zmianę wystroju strony, jak modyfikować pojedyncze cechy dotyczące elementów HTML? Odpowiedzi na te wszystkie pytania znajdują się na kolejnych stronach tego rozdziału.

Skrypt 31. [C][E][F][O][S]

Dynamiczna zmiana stylu strony

Wystrój strony, generowany zwykle za pomocą stylów CSS, jest sprawą gustu. Czasem warto więc przygotować kilka wersji i udostępnić użytkownikom wybór. Niech każdy sam zdecyduje, co mu się najbardziej podoba. Najlepiej, aby taki wybór mógł być wykonany dynamicznie, bez potrzeby przeładowania strony. Trzeba zatem będzie przygotować odpowiedni skrypt. Będzie umożliwiał wybór jednego z trzech stylów, a wybór będzie dokonywany za pomocą przycisków lub listy rozwijanej. Pracę zaczniemy od sekcji `<head>`, która przyjmie postać widoczną na listingu 3.1.

Listing 3.1. Sekcja `<head>` skryptu 31.

```
<head>
  <meta http-equiv="Content-Type" content="text/html; charset=utf-8">
  <title>Dynamiczna zmiana stylu</title>
  <link rel="stylesheet" title="styl1" type="text/css" href="style1.css">
  <link rel="stylesheet" title="styl2" type="text/css" href="style2.css">
  <link rel="stylesheet" title="styl3" type="text/css" href="style3.css">
  <script type="text/javascript">
    //tutaj treść skryptu
  </script>
</head>
```

Jak widać, w kodzie znalazły się trzy znaczniki `<link>` wskazujące trzy różne pliki ze stylami: *style1.css*, *style2.css* i *style3.css*. Każdy znacznik ma atrybut `title` określający nazwę danego stylu. Na podstawie tej wartości będzie dokonywany wybór. Ponieważ tak załadowane style będą obowiązywały jednocześnie, a w razie konfliktu reguł będzie obowiązywała ostatnio zdefiniowana, trzeba będzie je sprytnie wyłączyć tuż po załadowaniu witryny.

W kodzie strony trzeba umieścić interfejs pozwalający na zmianę stylów. Sekcja `<body>` będzie zatem zawierała elementy przedstawione na listingu 3.2.

Listing 3.2. *Interfejs pozwalający na zmianę stylów*

```
<body onload="zmieńStyl('');">
  <div>
    <input type="button" value="Reset"
      onclick="zmieńStyl('');" />
    <input type="button" value="Pierwszy styl"
      onclick="zmieńStyl('styl1');" />
    <input type="button" value="Drugi styl"
      onclick="zmieńStyl('styl2');" />
    <input type="button" value="Trzeci styl"
      onclick="zmieńStyl('styl3');" />
  </div>
  <div style="margin-top:10px;">
    <select onchange="listaChange(this);">
      <option value="">Wybierz styl</option>
      <option value="styl1">Pierwszy styl</option>
      <option value="styl2">Drugi styl</option>
      <option value="styl3">Trzeci styl</option>
    </select>
  </div>
</body>
```

W kodzie znajdują się dwie warstwy. Pierwsza zawiera zestaw trzech przycisków zdefiniowanych za pomocą znaczników `<input>`. Każdy z nich otrzymał procedurę obsługi zdarzenia w postaci funkcji `zmieńStyl`. A zatem ta funkcja będzie wywoływana po każdym kliknięciu dowolnego przycisku. Argumentem przekazywanym funkcji jest nazwa stylu, który ma być włączony, czyli wartość atrybutu `title` znacznika `<link>`. W ten sposób skrypt łatwo będzie mógł rozpoznać, o który styl chodzi. Funkcja będzie działała w taki sposób, że w przypadku przekazania jej argumentu niewskazującego żadnego ze stylów, wyłączy wszystkie dostępne. To oznacza, że wywołanie typu `zmieńStyl('')` wyłącza wszystkie style. To dlatego to wywołanie jest procedurą obsługi zdarzenia `onload` sekcji `<body>`, a także procedurą obsługi zdarzenia `click` przycisku *Reset*.

Druga warstwa zawiera listę (wykaz) zdefiniowaną za pomocą znacznika `<select>`. Procedurą obsługi zdarzenia `change` listy jest funkcja `listaChange`, która w postaci argumentu otrzymuje obiekt bieżący (`this`). Obiektem tym jest obiekt listy. Poszczególne pozycje zostały zdefiniowane za pomocą znaczników `<option>`. Każdy ma atrybut `value` wskazujący nazwę stylu, który przypisano danej opcji. W pierwszym przypad-

ku (pozycja *Wybierz styl*) wartością atrybutu jest pusty ciąg znaków (""), co oznacza, że będzie ona spełniała taką samą funkcję jak przycisk *Reset*.

Kody funkcji `zmieńStyl` oraz `listaChange` zostały przedstawione na listingu 3.3.

Listing 3.3. *Skrypt zmieniający style*

```
<script type="text/javascript">
  function zmieńStyl(nazwa)
  {
    var s = document.getElementsByTagName("link");
    if(!s) return;
    for(var i = 0; i < s.length; i++){
      if(s[i].title == nazwa)
        s[i].disabled = false;
      else
        s[i].disabled = true;
    }
  }
  function listaChange(lista)
  {
    if(lista){
      zmieńStyl(lista[lista.selectedIndex].value);
    }
  }
</script>
```

Funkcja `zmieńStyl` otrzymuje argument `nazwa` wskazujący nazwę stylu, który ma być włączony (wartość atrybutu `title` znacznika `link`). Za pomocą metody `getElementsByTagName` obiektu `document` pobiera odwołania do wszystkich znajdujących się w dokumencie obiektów typu `link` i zapisuje w zmiennej `s`. Jeśli operacja powiedzie się, rozpoczynana jest pętla `for` przeglądająca tablicę (kolekcję) zapisaną w zmiennej `s`. Wewnątrz pętli badane jest to, czy właściwość `title` bieżącego obiektu (o indeksie wskazywanym przez zmienną iteracyjną `i`) jest równa argumentowi `name`:

```
if(s[i].title == nazwa)
```

Jeśli tak jest, dany styl jest włączany:

```
s[i].disabled = false;
```

a w przeciwnym przypadku — wyłączany:

```
s[i].disabled = true;
```

Funkcja `listaChange` ma jeszcze prostszą konstrukcję. Otrzymuje w postaci argumentu obiekt listy, a jej zadaniem jest odczytanie wartości przypisanej aktywnemu elementowi wykazu i użycie tej wartości w wywołaniu funkcji `zmieńStyl`. Ponieważ indeks aktywnej pozycji listy jest zapisany we właściwości `selectedIndex`, a wartość — we właściwości `value`, wykonywana jest po prostu instrukcja:

```
zmieńStyl(lista[lista.selectedIndex].value);
```

Skrypt 32. [C][E][F][O][S]

Styl zależny od przeglądarki

Zdarza się, że przeglądarki różnie interpretują te same style. Może być więc konieczne przygotowanie różnych wersji plików CSS. Nie można jednak zmuszać użytkownika witryny, aby sam dobierał styl do własnej przeglądarki. Dopasowanie powinno się odbywać automatycznie. Można to osiągnąć, stosując odpowiedni skrypt. Kod mógłby wyglądać tak, jak zaprezentowano to na listingu 3.4.

Listing 3.4. *Style zależne od rozpoznanej przeglądarki*

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN"
"http://www.w3.org/TR/html4/strict.dtd">
<html lang="pl">
  <head>
    <meta http-equiv="Content-Type" content="text/html; charset=utf-8">
    <title>Styl zależny od przeglądarki</title>
    <script type="text/javascript">
      function getBrowserType()
      {
        //treść funkcji getBrowserType
      }
      function.setStyle(nazwa)
      {
        var styleEl = document.createElement("link");
        styleEl.setAttribute("rel", "stylesheet");
        styleEl.setAttribute("type", "text/css");
        styleEl.setAttribute("href", nazwa);
        var head = document.getElementsByTagName("head");
        if(head){
          head[0].appendChild(styleEl);
        }
      }
      function ustalStyl()
      {
        switch(getBrowserType()){
          case 'ff' : setStyle('ff.css');break;
          case 'ie' : setStyle('ie.css');break;
          case 'ch' : setStyle('ch.css');break;
          case 'op' : setStyle('op.css');break;
          case 'sa' : setStyle('sa.css');break;
          case 'un' : setStyle('def.css');break;
        }
      }
      ustalStyl();
    </script>
  </head>
  <body>
    <!-- treść witryny -->
  </body>
</html>
```

Treść witryny może być dowolna. Należy dodatkowo przygotować osobne pliki ze stylami dla każdej przeglądarki. Pliki powinny mieć nazwy zgodne ze schematem *kod.css*, gdzie kod oznacza kod przeglądarki: *ch* — Chrome, *ie* — Internet Explorer, *ff* — Firefox, *op* — Opera, *sa* — Safari. Np. nazwa pliku z definicjami stylów dla przeglądarki Firefox powinna mieć postać *ff.css*.

W trakcie ładowania strony zostanie wywołana funkcja `ustalStyl`. Znajduje się w niej instrukcja wyboru `switch`, która bada wartość zwróconą przez funkcję `getBrowserType`. Ta funkcja odpowiada za rozpoznanie typu przeglądarki (została przedstawiona i dokładnie opisana przy omawianiu skryptu 7.). W zależności od kodu zwróconego przez `getBrowserType` następuje wywołanie funkcji `setStyle` z argumentem wskazującym nazwę pliku ze stylami, który ma być załadowany.

Funkcja `setStyle` używa metody `createElement` obiektu `document` do utworzenia nowego obiektu typu `link`. Za pomocą metody `setAttribute` ustala także atrybuty nowego powstałego obiektu: `rel`, `type` i `href`. Używane są wartości charakterystyczne dla stylów CSS (`stylesheet` i `text/css`). Wartością atrybutu `href` jest wartość przekazana w postaci argumentu, czyli nazwa pliku ze stylami.

W ten sposób został utworzony element HTML dokładnie taki, jaki powstałby przy użyciu znacznika `<link>` w kodzie źródłowym strony. Musi on być dołączony do drzewa dokumentu HTML. Dlatego też za pomocą metody `getElementsByName` pobierane jest odwołanie do obiektu odzwierciedlającego sekcję `<head>`, a wspomniany obiekt jest do tej sekcji dodawany za pomocą metody `appendChild`. Po wykonaniu tych operacji dokument HTML będzie się zachowywał tak, jakby w sekcji `<head>` znajdował się zwykły znacznik `<link>`.

Należy zwrócić uwagę, że przyjęto milczące założenie, iż źródłowy dokument HTML jest prawidłowy i zawiera jedną sekcję typu `<head>`. Dlatego odwołanie do tej sekcji poprzez kolekcję zwróconą przez metodę `getElementsByName` ma postać `head[0]`. Ten zapis oznacza bowiem pierwszy (o indeksie 0) element kolekcji.

Skrypt 33. [C][E][F][O][S] Powiększanie i zmniejszanie tekstu

Większość przeglądarek umożliwia zwiększanie i zmniejszanie tekstu witryny czy to za pomocą menu, czy skrótów klawiaturowych. Taka funkcja może być jednak bez problemów zrealizowana programowo. Jeśli przygotujemy odpowiednią procedurę JavaScript, efekt będzie mógł być zarówno nadawany całej witrynie, jak i ograniczany tylko do wybranego elementu (elementów), np. wybranej warstwy. Co więcej, będzie można stosować różne jednostki miary, a także decydować o stopniu powiększenia w każdym kroku. Takie zadania realizuje kod przedstawiony na listingu 3.5.

Listing 3.5. *Funkcja zmieniająca rozmiar tekstu w wybranym elemencie witryny*

```
<script type="text/javascript">
  var rozmiarDomyślny = 10;
  var miaraDomyślna = "pt";
  var krokDomyślny = 1;
  function resizeText(nazwa, kierunek, krok, miara)
  {
    var el = document.getElementById(nazwa);
    if(!el) return;
    if(!krok) krok = krokDomyślny;
    if(!miara) miara = miaraDomyślna;
    var rt = parseInt(el.style.fontSize);
    if(isNaN(rt)) rt = rozmiarDomyślny;

    if(kierunek == "+")
      rt += krok;
    else
      rt -= krok;

    if(rt < 1) rt = 1;

    el.style.fontSize = rt + miara;
  }
</script>
```

Na początku znajdują się definicje trzech zmiennych globalnych, które określają parametry domyślne. Zostaną zastosowane, jeśli argumenty przekazane funkcji `resizeText` będą niepełne bądź nieprawidłowe. Znaczenie argumentów jest następujące:

- ♦ `nazwa` — identyfikator elementu strony, którego dotyczy zmiana wielkości tekstu;
- ♦ `kierunek` — znak określający kierunek zmian, '+' — zwieszanie, '-' — zmniejszanie;
- ♦ `krok` — wartość, o którą zmieni się rozmiar w pojedynczym kroku;
- ♦ `miara` — jednostka miary (np. px — piksele, pt — punkty drukarskie itp.).

Najpierw za pomocą metody `getElementById` pobierane jest odwołanie do elementu witryny o identyfikatorze przekazanym w postaci argumentu `nazwa`. Odwołanie jest zapisywane w zmiennej `el`. Gdyby ta operacja nie powiodła się, wykonywanie kodu jest przerywane za pomocą instrukcji `return`.

Następnie sprawdzane jest istnienie argumentów `krok` i `miara`. Jeżeli nie zostały przekazane, przypisywane są im wartości domyślne. Za pomocą metody `parseInt` jest też przetwarzany ciąg znajdujący się we właściwości `style` obiektu `font` elementu `el`. Oznacza to pobranie aktualnego rozmiaru czcionki i przetworzenie go na wartość całkowitoliczbową. Odczytana wartość jest przypisywana zmiennej `rt`. Gdyby ta operacja się nie udała, zmienna otrzymuje wartość domyślną:

```
if(isNaN(rt)) rt = rozmiarDomyślny;
```

Stanie się tak wtedy, gdy elementowi `el` (o identyfikatorze wskazywanym przez argument `nazwa`) nie przypisano cechy `font-size` lub też gdy cecha ta została określona w sposób inny niż liczbowy (np. `small`, `xx-large` itp.).

W zależności od stanu argumentu `kierunek` do zmiennej `rt` jest dodawana (kierunek równy `+`) lub też jest od niej odejmowana (kierunek różny od `+`) wartość zapisana w krok. W przypadku wykrycia, że uzyskana wartość jest mniejsza od 1, do `rt` jest przypisywane 1. Dzięki temu ograniczana jest możliwość zmniejszania tekstu do jednej jednostki danej miary. Powiększanie nie jest natomiast ograniczane w żaden sposób.

Wielkość tekstu jest zmieniana dzięki instrukcji:

```
el.style.fontSize = rt + miara;
```

Jest to przypisanie właściwości `fontSize` ciągu znaków powstałego z połączenia wartości zapisanej w `rt` z jednostką miary zapisaną w `miara`.

Przykład użycia funkcji `resizeText` został zaprezentowany na listingu 3.6.

Listing 3.6. *Użycie funkcji `resizeText`*

```
<body>
  <div id="divInterfejs">
    
    
  </div>
  <div id="dataDiv" style="font-size:12pt;">
    <!-- tutaj tekst warstwy -->
  </div>
</body>
```

Pierwsza warstwa, o identyfikatorze `divInterfejs`, zawiera interfejs sterujący powiększaniem i zmniejszaniem tekstu. Zostały w niej umieszczone dwa obrazy — są to strzałki w górę i w dół. Oba obrazy otrzymały procedurę obsługi zdarzenia `click` w postaci funkcji `resizeText`. W pierwszym przypadku parametry przekazane funkcji wskazują na powiększanie tekstu o 1 punkt drukarski, a w drugim — na zmniejszanie o 1 punkt. Operacje te będą wykonywane na warstwie o identyfikatorze `dataDiv`, która znajduje się w dalszej części kodu. Do kodu można dodać style zmieniające domyślny układ witryny, np. takie jak zaprezentowano w przykładzie 3. z rozdziału 1.

Należy też zwrócić uwagę, że warstwie, na której będzie zmieniany rozmiar tekstu (`dataDiv`), przypisany został atrybut `style` określający jawnie początkowy rozmiar czcionki (cecha `font-style`). Gdyby nie zostało to zrobione lub też rozmiar nie został określony liczbowo, pierwsze użycie funkcji `resizeText` spowodowałoby przypisanie rozmiaru domyślnego zapisanego w skrypcie. Wtedy dopiero kolejne wywołania powodowałyby właściwy efekt.

Skrypt 34. [C][E][F][O][S]

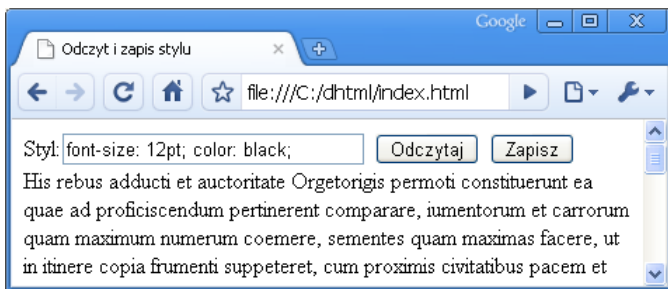
Przypisanie wybranemu elementowi stylu wprowadzonego przez użytkownika

Styl wybranego elementu witryny może być dowolnie odczytywany i zapisywany. Można zatem umieścić na stronie interfejs, który umożliwi wykonywanie takich operacji. Tego typu skrypt może posłużyć do wygodnego testowania różnych reguł CSS w rozmaitych przeglądarkach. Styl będzie przypisywany natychmiast, bezpośrednio z poziomu witryny, nie będzie więc konieczności ciągłego przełączania się między edytorem kodu a przeglądarką.

Na stronie wystarczy umieścić pole tekstowe i dwa przyciski oraz warstwę (lub dowolny inny element) podlegający zmianom. Przykładowy wygląd takiej strony jest widoczny na rysunku 3.1, a kod sekcji `<body>` został zaprezentowany na listingu 3.7.

Rysunek 3.1.

Strona pozwalająca na testowanie stylów



Listing 3.7. Kod sekcji `<body>` skryptu 34.

```
<body>
  <div id="divInterfejs">
    Styl:<input type="text" id="tfStyl" />
    <input type="button" onclick="odczytajStyl('divDane', 'tfStyl');"
      value="Odczytaj"/>
    <input type="button" onclick="przypiszStyl('divDane', 'tfStyl');"
      value="Zapisz"/>
  </div>
  <div id="divDane" style="font-size:12pt;color:black;">
    <!-- treść warstwy -->
  </div>
</body>
```

Pierwsza warstwa zawiera elementy interfejsu, a druga — treść, na której mają być testowane reguły CSS. Interfejs składa się z pola tekstowego oraz dwóch przycisków. Te elementy zostały zdefiniowane za pomocą znaczników `<input>`. Przyciski otrzymały procedury obsługi zdarzenia `click`. Dla przycisku *Odczytaj* jest to funkcja `odczytajStyl`,

a dla przycisku *Zapisz* — funkcja `przypiszStyl`. Obie funkcje otrzymują takie same argumenty. Pierwszy określa element strony, którego styl ma być modyfikowany, drugi — pole tekstowe z opisem stylu.

Treść funkcji powinna być taka, jak zaprezentowano to na listingu 3.8.

Listing 3.8. *Funkcje odczytujące i zapisujące style CSS*

```
<script type="text/javascript">
  function odczytajStyl(nazwa, pole)
  {
    var el = document.getElementById(nazwa);
    var tf = document.getElementById(pole);
    if(el && el.style && tf){
      tf.value = el.style.cssText;
    }
  }
  function przypiszStyl(nazwa, pole)
  {
    var el = document.getElementById(nazwa);
    var tf = document.getElementById(pole);
    if(el && el.style && tf){
      el.style.cssText = tf.value;
    }
  }
</script>
```

Obie funkcje mają bardzo podobną konstrukcję, choć wykonują nieco inne zadania¹. Najpierw pobierane są odwołania do elementów o identyfikatorach przekazanych w postaci argumentów `nazwa` i `pole`. Odwołanie do elementu podlegającego modyfikacji jest zapisywane w zmiennej `el`, a odwołanie do pola tekstowego zawierającego styl — w zmiennej `tf`. Następnie badane jest to, czy obie te operacje zakończyły się sukcesem oraz czy obiekt wskazywany przez `el` zawiera właściwość `style`. Jeżeli odpowiedzi na te pytania brzmią „tak”, wykonywana jest operacja odczytu stylu (w funkcji `odczytajStyl`) lub zapisu (w funkcji `zapiszStyl`).

Odczyt stylu odbywa się przez pobranie wartości zapisanej we właściwości `cssText` obiektu `style` i zapisanie jej w polu tekstowym `tf`, natomiast zapis to operacja odwrotna — przypisanie właściwości `cssText` obiektu `style` wartości pobranej z pola tekstowego. Przy zapisie nie jest badane to, czy styl wprowadzony do pola jest prawidłowy. Jeśli jednak nie będzie prawidłowy, po prostu nie zostanie uwzględniony.

Należy w tym miejscu przypomnieć, że zastosowana właściwość `cssText`, pierwotnie charakterystyczna dla przeglądarek z rodziny Internet Explorer, nie jest częścią oficjalnych standardów, jest jednak obecnie obsługiwana przez wszystkie popularne przeglądarki.

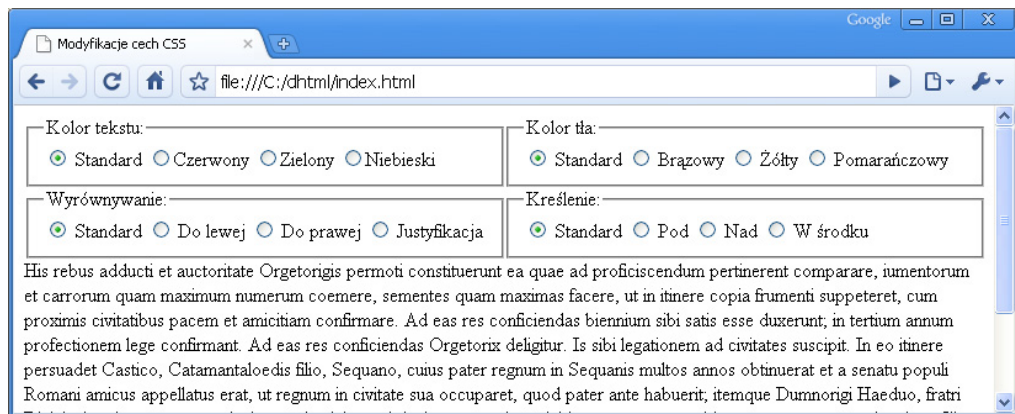
¹ Można również połączyć obie funkcje w jedną sterowaną dodatkowych argumentem określającym to, czy ma zostać wykonany zapis czy odczyt.

Skrypt 35.

Kompozycja stylu z wybranych elementów

[C][E][F][O][S]

Na stronie można umieścić interfejs pozwalający użytkownikowi komponować jej wygląd z różnych elementów. Będzie więc mógł zdecydować, jaki ma być kolor tekstu, tła, czy też jaki rodzaj wyrównywania zostanie zastosowany. Jakie cechy CSS będą podlegały modyfikacjom i jakie będą ich dopuszczalne wartości — to już zależy od naszych potrzeb, a dokładniej od koncepcji witryny. Założmy, że dopuścimy zmiany cech przedstawionych na rysunku 3.2. Prezentuje on interfejs skryptu 35. Taka strona jest generowana przez kod widoczny na listingu 3.9.



Rysunek 3.2. Interfejs pozwalający modyfikować poszczególne cechy CSS

Listing 3.9. Sekcja `<body>` skryptu 35.

```
<body>
  <div id="divInterfejs">
    <fieldset style="width:360px;float:left;">
      <legend>Kolor tekstu:</legend>
      <input type="radio" name="kolor" checked="checked"
        onclick="zmieńStyl('divDane', 'color', '');" />
        Standard
      <input type="radio" name="kolor"
        onclick="zmieńStyl('divDane', 'color', 'red');" />Czerwony
      <input type="radio" name="kolor"
        onclick="zmieńStyl('divDane', 'color', 'green');" />Zielony
      <input type="radio" name="kolor"
        onclick="zmieńStyl('divDane', 'color', 'blue');" />Niebieski
    </fieldset>
    <fieldset style="width:360px;">
      <legend>Kolor tła:</legend>
      <input type="radio" name="kolor_tla" checked="checked"
        onclick="zmieńStyl('divDane', 'backgroundColor', '');" />
        Standard
```



```

☒
    Standard
☒
    Standard
☐

```

Listing jest co prawda stosunkowo długi, nie zawiera jednak żadnych skomplikowanych konstrukcji. Na pierwszej warstwie (`divInterfejs`) zawarte zostały elementy interfejsu pozwalające na zmianę stylów warstwy drugiej (`divDane`). Ponieważ w omawianym przykładzie zmieniane będą style dotyczące tekstu, na warstwie drugiej najlepiej umieścić właśnie tekst. Interfejs składa się z czterech grup pól wyboru typu radio definiovanych za pomocą znaczników `<input>` atrybutem `type` ustawionym na `radio`. Pola są grupowane za pomocą znaczników `<fieldset>`, a grupom są nadawane tytuły za pomocą znaczników `<legend>`.

Każde pole otrzymało procedurę obsługi zdarzenia `click` w postaci funkcji `zmieńStyl`. Otrzymuje ona trzy argumenty. Pierwszy określa identyfikator elementu, którego styl będzie zmieniany, drugi — nazwę właściwości obiektu `style` odpowiadającej danej cecie CSS, trzeci — wartość, którą ta cecha ma otrzymać. Należy pamiętać, że w przypadku cech dwuczłonowych (np. `background-color`) właściwość CSS ma postać niepodzielną, a drugi człon należy rozpocząć wielką literą (np. `backgroundColor`).

Przykładowe wywołanie:

```
zmieńStyl('divDane', 'textDecoration', 'overline');
```

oznacza: zmodyfikuj styl elementu o identyfikatorze `divDane`; zmień właściwość `text-decoration`, nadając jej wartość `overline`.

W każdej grupie pól znajduje się jedno pozwalające usunąć daną cechę. Odbywa się to przez przekazanie w trzecim argumencie pustego ciągu znaków, np.:

```
zmieńStyl('divDane', 'textAlign', '');
```

Treść funkcji `zmieńStyl` została przedstawiona na listingu 3.10.

Listing 3.10. *Funkcja dokonująca zmiany stylu*

```
<script type="text/javascript">
  function zmieńStyl(nazwa, cecha, wartość)
  {
    var el = document.getElementById(nazwa);
    if(el && el.style){
      el.style[cecha] = wartość;
    }
  }
</script>
```

Funkcja najpierw pobiera odwołanie do elementu wskazanego przez argument `nazwa`. Jeśli ten element istnieje i zawiera właściwość `style`, następuje modyfikacja właściwości obiektu `style` o nazwie wskazanej przez argument `cecha`. Tym samym wartością cechy staje się wartość trzeciego argumentu funkcji.

Rozdział 4.

Obsługa formularzy

Formularze to elementy witryny pozwalające na wprowadzanie danych przez użytkownika. Dzięki nim gość odwiedzający naszą witrynę może wyrazić swoją opinię na pewien temat, dodać komentarz, wybrać jakąś opcję czy zapisać się do newslettera. Wszystko to jest możliwe dzięki elementom składowym, takim jak pola tekstowe, różne typy pól wyboru, listy, przyciski itp. Przetwarzaniem danych wprowadzanych do formularzy zwykle zajmują się skrypty działające po stronie serwera (np. skrypty PHP, ASP), jednak wiele funkcji związanych z obsługą tych elementów witryny jest też wykonywanych przez JavaScript. W tym rozdziale zostaną przedstawione skrypty wykonujące takie funkcje, jak weryfikacja danych, przeszukiwanie tekstu czy wskazywanie aktywnych elementów strony, a także usprawniające pracę z formularzami przez automatyczne przenoszenie kursora między kolejnymi elementami, czy też dostosowywanie wielkości pola tekstowego do umieszczonego w nim ciągu znaków.

Skrypt 36. Walidacja formularzy

[C][E][F][O][S]

Sprawdzanie, czy dane wpisane przez użytkownika w formularzu są poprawne, jest bardzo pożądaną czynnością. Bardzo często bowiem zdarza się, że brakuje pewnych wymaganych wpisów bądź też są one niepoprawne. Prosty skrypt dokonujący walidacji ustrzeże nas przed takimi pomyłkami i od razu poinformuje użytkownika, gdzie popełnił błąd. Załóżmy, że mamy do czynienia z formularzem widocznym na rysunku 4.1. Jest on generowany przez kod z listingu 4.1. To osiem pól, z których część jest obowiązkowa. Jeśli nie zostaną wypełnione, formularz nie zostanie wysłany, pojawi się za to komunikat dokładnie informujący o tym, gdzie brakuje danych.

Listing 4.1. Kod HTML generujący formularz

```
<body>
  <div id="divDane">
    <form name="form1" action="" method="get">
      <div>
```

Rysunek 4.1.
Wygląd formularza
z przykładu 36.

Weryfikacja formularza - Windows Internet Explorer

E:\dhtml\index.html

Live Search

Ulubione Weryfikacja formularza Strona

Proszę podać swoje dane:
(pola oznaczone * muszą zostać wypełnione)

Dane personalne:

imię: *

nazwisko: *

Adres:

ulica:

nr domu:

kod:

miasto:

kraj: *

tel.: *

Gotowe Mój komputer 100%

```

<span style="font-size:14pt;font-weight:bold;">
    Proszę podać swoje dane:</span><br />
<span style="font-size:smaller;font-style:italic;">
    (pola oznaczone * muszą zostać wypełnione)</span>
</div>
<table>
<tr>
    <td><b>Dane personalne:</b></td>
<td></td>
</tr>
<tr>
    <td>imię:</td>
    <td><input type="text" name="imie" />*</td>
</tr>
<tr>
    <td>nazwisko:</td>
    <td><input type="text" name="nazwisko" />*</td>
</tr>
<tr>
    <td><b>Adres:</b></td>
<td></td>
</tr>
<tr>
    <td>ulica:</td>
    <td><input type="text" name="ulica" /></td>
</tr>
<tr>
    <td>nr domu:</td>
    <td><input type="text" name="nrdomu" /></td>
</tr>
<tr>
    <td>kod:</td>

```

```

        <td><input type="text" name="kod" /></td>
    </tr>
    <tr>
        <td>miasto:</td>
        <td><input type="text" name="miasto" /></td>
    </tr>
    <tr>
        <td>kraj:</td>
        <td><input type="text" name="kraj" />*</td>
    </tr>
    <tr>
        <td>tel.:</td>
        <td><input type="text" name="tel" />*</td>
    </tr>
    <tr>
        <td style="text-align:center;" colspan="2">
            <input type="button" name="wyslij"
                value="    Wyślij!    " onclick="przetwarzaj_dane();" />
        </td>
    </tr>
</table>
</form>
</div>
</body>

```

Formularz składa się z typowych elementów HTML — znaczników `<form>` i `<input>`. W znaczniku `<form>` została pominięta wartość atrybutu `action`. W realnych warunkach pojawi się tam oczywiście adres skryptu działającego na serwerze, do którego będą wysyłane dane, np. `action="form.php"`. Do formatowania formularza została użyta tabela generowana za pomocą znaczników `<table>`, `<tr>` i `<td>`. To najprostszy sposób na szybkie wyrównanie elementów witryny (choć formalnie zalecanym sposobem wyrównywania jest używanie stylów CSS).

Każde pole tekstowe formularza ma swoją nazwę określoną przez parametr `name`. Dzięki temu można je będzie bez problemu zidentyfikować w skrypcie. Na samym dole zamiast przycisku typu `submit` (tak jak byłoby to w klasycznym formularzu) został umieszczony zwyczajny przycisk typu `button`. Została natomiast do niego dodana procedura obsługi zdarzenia `click` w postaci funkcji o nazwie `przetwarzaj_dane`.

Ta funkcja będzie miała postać przedstawioną na listingu 4.2.

Listing 4.2. Treść funkcji `przetwarzaj_dane`

```

<script type="text/javascript">
    function przetwarzaj_dane()
    {
        var brakuje_danych = false;
        var formularz = document.forms['form1'];
        if(!formularz) return;

        var komunikat = "";
        if(formularz.imie.value == ""){
            komunikat += "imie\n"

```

```
        brakuje_danych = true;
    }
    if(formularz.nazwisko.value == ""){
        komunikat += "nazwisko\n"
        brakuje_danych = true;
    }
    if(formularz.kraj.value == ""){
        komunikat += "kraj\n"
        brakuje_danych = true;
    }
    if(formularz.tel.value == ""){
        komunikat += "telefon\n"
        brakuje_danych = true;
    }
    if(!brakuje_danych)
        formularz.submit();
    else
        alert ("Nie wypełniono następujących pól:\n" + komunikat);
}
</script>
```

Funkcja `przetwarzaj_dane` pobiera odwołanie do obiektu formularza, używając przy tym jego nazwy. Odwołanie jest zapisywane w zmiennej `formularz` (dzięki temu dalsze odwołania będą krótsze):

```
var formularz = document.forms['form1'];
```

Ustala też wartość pomocniczej zmiennej `brakuje_danych` na `false`. Będzie ona wskazywała, czy w wymagalnych polach brakuje danych. W przypadku gdyby formularza o zadanej nazwie nie było w kodzie strony, wykonywanie kodu jest przerywane za pomocą instrukcji `return`.

Następnie za pomocą serii instrukcji warunkowych `if` sprawdzane są po kolei wartości interesujących nas pól. Wartość pola jest odczytywana za pomocą konstrukcji w postaci:

```
form1.nazwa_pola.value
```

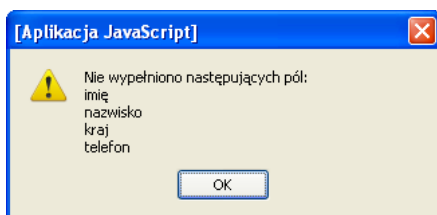
Jeżeli dane pole nie zostało wypełnione (zawiera pusty ciąg znaków), do zmiennej `komunikat` dodawana jest jego nazwa, a zmienna `brakuje_danych` otrzymuje wartość `true` (prawda).

Na samym końcu następuje sprawdzenie wartości zmiennej `brakuje_danych`. Jeżeli jest ona równa `true`, czyli brakuje jakichś danych, za pomocą metody `alert` wyświetlany jest odpowiedni komunikat zawierający nazwy niewypełnionych pól, pobrane ze zmiennej `komunikat`. Ten komunikat będzie miał zatem postać przedstawioną na rysunku 4.2. Jeżeli wartość zmiennej `brakuje_danych` nie została zmodyfikowana i jest równa `false`, wywoływana jest metoda `submit` powodująca przesłanie danych do serwera (do skryptu wskazywanego przez atrybut `action` znacznika `<form>`)¹.

¹ Obsługa danych wysyłanych z formularza do serwera została opisana w takich książkach jak *PHP5. Praktyczny kurs* (<http://helion.pl/ksiazki/php5pk>) i *PHP. 101 praktycznych skryptów* (<http://helion.pl/ksiazki/php102.htm>).

Rysunek 4.2.

*Komunikat
o braku wszystkich
wymaganych danych*



Skrypt 37. [C][E][F][O][S] Weryfikacja z uwzględnieniem formatu danych i wyróżnianiem błędnych pól

Czasem przy sprawdzaniu danych wprowadzonych do formularza chcemy mieć pewność, że mają one prawidłowy format. Przykładowo nie akceptujemy cyfr w imieniu i nazwisku czy liter w numerze telefonu. Dobrym pomysłem może też być wskazywanie użytkownikowi, w których polach popełnił błąd. Takie też zadania realizuje skrypt 37. Formularz będzie miał postać widoczną na rysunku 4.3. Pola zawierające nieprawidłowe dane będą wyróżniane przez zmianę koloru tła, natomiast komunikat informujący o błędzie będzie się pojawiał nad formularzem, na wyróżnionym tle. Kod (X)HTML możemy wziąć z przykładu 36., niezbędne będzie jednak użycie dodatkowej warstwy informacyjnej, tak jak zostało to pokazane na listingu 4.3.

Rysunek 4.3.

*Formularz
wyróżniający
źle wypełnione pola*

Listing 4.3. *Umieszczona w kodzie dodatkowa warstwa informacyjna*

```

<!-- początek kodu z listingu 4.1 -->
<form name="form1" action="" method="GET">
<div>
  <span style="font-size:14pt;font-weight:bold;">
    Proszę podać swoje dane:</span><br />
    <span style="font-size:smaller;font-style:italic;">
      (pola oznaczone * muszą zostać wypełnione)</span>
    </div>
<div id="divKomunikat" style="display:none;">
</div>
<table>
<!-- dalsza część kodu z listingu 4.1 -->

```

Dodatkowa warstwa (o identyfikatorze `divKomunikat`) znajduje się za tekstem opisującym formularz, ale przed jego pierwszymi polami. Początkowo nie zawiera żadnej treści, nie jest też wyświetlana. Wyświetlanie zostało wyłączone przez przypisanie stylu `display:none;`. Po wczytaniu do przeglądarki strona będzie się więc zachowywała tak, jakby warstwy `divKomunikat` nie było. Będzie ona włączana w razie potrzeby przez skrypt zaprezentowany na listingu 4.4.

Listing 4.4. *Skrypt weryfikujący formularz*

```

<script type="text/javascript">
  function przetwarzaj_dane()
  {
    var brakuje_danych = false;
    var formularz = document.forms['form1'];
    if(!formularz) return;

    var tfImie = formularz.imie;
    var tfNazwisko = formularz.nazwisko;
    var tfKraj = formularz.kraj;
    var tfTel = formularz.tel;

    tfImie.style.backgroundColor = "#FFFFFF";
    tfNazwisko.style.backgroundColor = "#FFFFFF";
    tfKraj.style.backgroundColor = "#FFFFFF";
    tfTel.style.backgroundColor = "#FFFFFF";

    var tylkoLiteryRE = /[a-zA-Z]{2,}/;
    var telefonRE = /[0-9 -]{5,}/;
    var komunikat = "";

    if(tylkoLiteryRE.exec(tfImie.value) != tfImie.value){
      komunikat += "imie";
      tfImie.style.backgroundColor = "#FFFF00";
      brakuje_danych = true;
    }
    if(tylkoLiteryRE.exec(tfNazwisko.value) != tfNazwisko.value){
      komunikat += komunikat ? ", nazwisko" : "nazwisko";
      tfNazwisko.style.backgroundColor = "#FFFF00";
    }
  }

```



```
        brakuje_danych = true;
    }
    if(tylkoLitereRE.exec(tfKraj.value) != tfKraj.value){
        komunikat += komunikat ? ", kraj" : "kraj";
        tfKraj.style.backgroundColor = "#FFFF00";
        brakuje_danych = true;
    }
    if(telefonRE.exec(tfTel.value) != tfTel.value){
        komunikat += komunikat ? ", telefon" : "telefon";
        tfTel.style.backgroundColor = "#FFFF00";
        brakuje_danych = true;
    }
    var divKomunikat = document.getElementById('divKomunikat');
    if(!brakuje_danych){
        if(divKomunikat)
            divKomunikat.style.display = "none";
        formularz.submit();
    }
    else{
        if(divKomunikat){
            var str = "W następujących polach są błędne dane: ";
            str += komunikat + ".";
            divKomunikat.innerHTML = str;
            divKomunikat.style.display = "block";
        }
    }
}
</script>
```

Na początku funkcji pobierane są odwołania do formularza oraz pól, których wypełnienie jest wymagane. Odwołania są zapisywane w zmiennych `tfImie`, `tfNazwisko`, `tfKraj` i `tfTel`. Następnie kolor tła wszystkich pól jest zmieniany na biały. Jest to konieczne, ponieważ przy poprzednim wywołaniu funkcji kolor ten mógł być zmieniony. W każdym bieżącym wywołaniu musi więc następować swoisty „reset”.

Poprawność danych wprowadzonych przez użytkownika jest badana za pomocą wyrażeń regularnych. Dla pól *Imię*, *Nazwisko* i *Kraj* będą dopuszczone wyłącznie litery (duże i małe), a wprowadzony ciąg będzie musiał mieć co najmniej dwa znaki. Te warunki będą badane za pomocą wyrażenia `/[a-zA-Z]{2,}/` przypisanego zmiennej `tylkoLitereRE`. W polu *Telefon* będą dopuszczalne wyłącznie cyfry oraz znak spacji i kreski poziomej (minusa), a minimalna długość ciągu będzie wynosiła 5 znaków. Ten warunek będzie badany za pomocą wyrażenia `/[0-9 -]{5,}/` przypisanego zmiennej `telefonRE`.

Badanie poprawności jest wykonywane za pomocą instrukcji warunkowej i wyrażenia o ogólnej postaci:

```
wyrażenie_regularne.exec(wartość_pola) != wartość_pola
```

Jeżeli to wyrażenie da w wyniku wartość `true`, oznacza to niepoprawność danych. Wtedy do zmiennej `komunikat` dodawana jest nazwa błędnego pola, tło pola jest zmieniane na żółte (`#FFFF00`), a zmienna `brakuje_danych` jest ustawiana na `true`.

Warto przy tym zwrócić uwagę na sposób uaktualniania zmiennej komunikat. Ponieważ nazwy błędnych pól powinny być od siebie oddzielone znakiem przecinka, ale oczywiście tylko wtedy, gdy są co najmniej dwie nazwy, używana jest konstrukcja warunkowa w postaci:

```
komunikat += komunikat ? ", nazwa_pola" : "nazwa_pola";
```

Ten zapis oznacza: jeżeli zmienna komunikat nie jest pusta, dodaj do niej przecinek, spację i nazwę pola, a jeżeli jest pusta (nie zawiera żadnych znaków), dodaj do niej tylko nazwę pola.

Opisana weryfikacja jest przeprowadzana dla każdego z wymaganych pól. Po jej zakończeniu w przypadku wykrycia braku danych zmienna `brakuje_danych` będzie miała wartość `true`, a zmienna `komunikat` będzie zawierała listę błędnych pól oddzielonych znakami przecinka. Jeżeli jednak dane będą prawidłowe, zmienna `brakuje_danych` będzie miała wartość `false`, a zmienna `komunikat` będzie zawierała pusty ciąg znaków.

Jeżeli zatem `brakuje_danych` jest równe `true`, w warstwie `divKomunikat` trzeba umieścić komunikat o braku danych:

```
divKomunikat.innerHTML = str;
```

a warstwa musi się pojawić na ekranie:

```
divKomunikat.style.display = "block";
```

W przeciwnym wypadku warstwę `divKomunikat` trzeba ukryć:

```
divKomunikat.style.display = "none";
```

a dane z formularza wysłać do serwera:

```
formularz.submit();
```

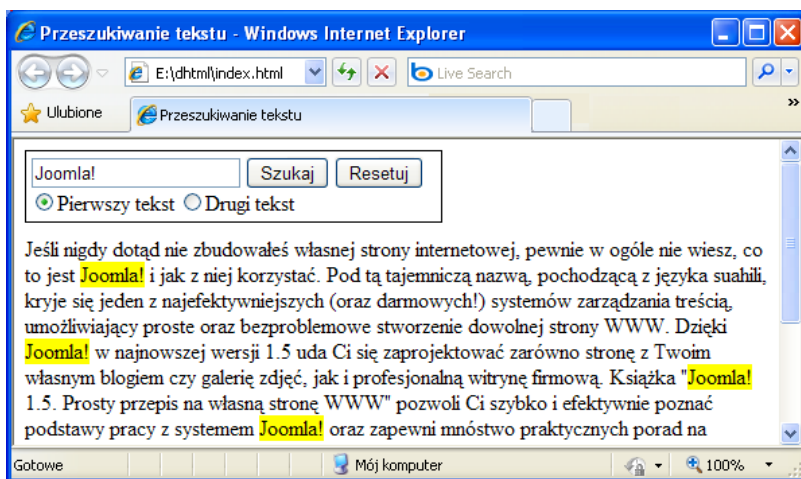
Skrypt 38. Wyszukiwanie frazy w tekście strony

[C][E][F][O][S]

Użytkownikom witryny można udostępnić funkcję przeszukiwania tekstu i wyróżniania wybranych fragmentów, której działanie może być ograniczane tylko do pewnych części strony. Przyjmijmy, że mamy dwie warstwy, obie zawierają tekst, ale mają być przeszukiwane oddzielnie. O tym, która część będzie w danej chwili uwzględniana, będzie można zdecydować za pomocą pól wyboru. Interfejs będzie więc wyglądał tak, jak zaprezentowano to na rysunku 4.4. Po wprowadzeniu do pola tekstowego poszukiwanej frazy i kliknięciu przycisku *Szukaj*, odnalezione fragmenty tekstu zostaną podświetlone (zmieni się ich kolor tła). Kod HTML sekcji `<body>` został przedstawiony na listingu 4.5.

Rysunek 4.4.

Odszukane
fragmenty tekstu
zostały wyróżnione
zmienionym
kolorem tła

**Listing 4.5.** Kod sekcji <body> skryptu 38.

```
<body>
  <div id="divInterfejs">
    <input type="text" id="tfFraza" />
    <input type="button" value="Szukaj" onclick="btnSzukajClick();" />
    <input type="button" value="Resetuj" onclick="resetuj();" />
    <br />
    <input type="radio" name="grupa1" id="rb1" checked="checked"
      />Pierwszy tekst
    <input type="radio" name="grupa1" id="rb2" />Drugi tekst
  </div>
  <div id="divDane1" class="divDane">
    <!-- tutaj pierwszy tekst -->
  </div>
  <div id="divDane2" class="divDane">
    <!-- tutaj drugi tekst -->
  </div>
</body>
```

Pierwsza warstwa, o identyfikatorze `divInterfejs`, zawiera pole tekstowe (`tfFraza`), przyciski *Szukaj* i *Resetuj* oraz pola wyboru typu radio *Pierwszy tekst* (`rb1`) i *Drugi tekst* (`rb2`). Pole wyboru *Pierwszy tekst* jest domyślnie zaznaczone dzięki użyciu atrybutu `checked`, natomiast oba tworzą grupę, dzięki czemu w danej chwili będzie mogło być zaznaczone tylko jedno z nich. Przyciski będą reagowały na kliknięcia dzięki procedurom obsługi zdarzenia `click`. W przypadku przycisku *Szukaj* jest to funkcja `btnSzukajClick`, a w przypadku przycisku *Resetuj* — funkcja `resetuj`.

Kluczową sprawą jest sposób wyszukania, a następnie wyróżnienia frazy wprowadzonej do pola tekstowego. Najlepiej użyć do tego celu operującej na wyrażeniach regularnych metody `replace`. Jej wywołanie ma postać:

```
ciąg_przeszukiwany.replace(wyr_reg, tekst)
```

Zwraca ona ciąg, w którym frazy ciągu przeszukiwanego zgodne z wyrażeniem regularnym *wyr_reg* zostały zamienione na ciąg przekazany w postaci argumentu *tekst*. Jeśli każdą poszukiwaną frazę zamienimy na ciąg o ogólnej postaci:

```
<span>fraz</span>
```

w prosty sposób będzie można ją wyróżnić na stronie. Wystarczy użyć odpowiednich stylów CSS. Równie łatwo będzie można usunąć zaznaczenia. Wystarczy pozbyć się z tekstu znaczników `` i ``.

Część skryptu odpowiedzialna za reakcję na kliknięcie przycisku *Szukaj* będzie miała postać przedstawioną na listingu 4.6.

Listing 4.6. Treść funkcji *btnSzukajClick* i *zaznacz*

```
function btnSzukajClick()
{
    var tfFraza = document.getElementById('tfFraza');
    var rb1 = document.getElementById('rb1');
    var rb2 = document.getElementById('rb2');

    if(!tfFraza || !rb1 || !rb2) return;

    var div;
    if(rb1.checked){
        div = document.getElementById('divDane1');
        reset('divDane1');
    }
    else if (rb2.checked){
        div = document.getElementById('divDane2');
        reset('divDane2');
    }
    else{
        alert('Wskaż tekst, który ma być przeszukiwany!');
        return;
    }
    if(div) div.innerHTML = zaznacz(tfFraza.value, div.innerHTML);
}
function zaznacz(co, gdzie)
{
    if(!co || !gdzie) return gdzie;
    re = new RegExp(co, "g");
    return gdzie.replace(re, "<span>"+co+"</span>");
}
```

Funkcja *btnSzukajClick* zajmuje się przygotowaniem danych, a za właściwe wyróżnienie fraz odpowiada funkcja *zaznacz*². Najpierw pobierane są odwołania do pola tekstowego oraz pól wyboru, a następnie badane jest to, czy te elementy faktycznie istnieją. Jeśli nie istnieją, działanie kodu jest kończone za pomocą instrukcji *return*. Następnie badane jest, które pole wyboru (*rb1* czy *rb2*) jest zaznaczone. W zależności od tego zmiennej pomocniczej *div* jest przypisywane odwołanie do warstwy *divData1* (zazna-

² To rozróżnienie jest czysto funkcjonalne. Ze względu na bardzo prostą konstrukcję funkcji *zaznacz* jej kod można by z powodzeniem włączyć do funkcji *btnSzukajClick*.

zione pole `rb1`) lub `divData2` (zaznaczone pole `rb2`). Resetowane jest także zaznaczenie w danej warstwie. Odpowiada za to funkcja `reset`, która w argumencie otrzymuje nazwę warstwy. Wywołanie `reset('divDane1')`; oznacza po prostu: usuń wszystkie zaznaczenia (znaczniki ``) z warstwy o identyfikatorze `divDane1`. Jeżeli żadne pole wyboru nie zostało zaznaczone, wyświetlany jest stosowny komunikat.

Na końcu funkcji następuje zamiana tekstu znajdującego się na warstwie wskazywanej przez `div`. Odpowiada za to instrukcja:

```
div.innerHTML = zaznacz(tfFraza.value, div.innerHTML);
```

A zatem we właściwości `innerHTML` (reprezentującej treść warstwy) zostaje zapisana wartość zwrócona przez wywołanie funkcji `zaznacz`. Pierwszym argumentem jest zawartość pola tekstowego `tfFraza`, a drugim — bieżąca zawartość warstwy wskazywanej przez `div`.

Funkcja `zaznacz` przyjmuje dwa argumenty: `co` i `gdzie`. Jej zadaniem jest wymiana w tekście reprezentowanym przez `gdzie` fraz reprezentowanych przez `co` na ciągi w postaci `co`. Najpierw z wartości przekazanej w postaci argumentu `co` tworzone jest wyrażenie regularne zapisywane w zmiennej `re`. Ponieważ wymiana ma dotyczyć wszystkich podciągów w tekście³, używany jest też atrybut (flaga, z ang. *flag*) wyrażenia `g`:

```
re = new RegExp(co, "g");
```

Następnie wywoływana jest opisana wyżej metoda `replace`, a wynik jej działania jest zwracany za pomocą instrukcji `return`.

Treść funkcji `resetuj` oraz związanej z nią dodatkowej funkcji `reset` znajduje się na listingu 4.7.

Listing 4.7. Treść funkcji `resetuj` i `reset`

```
function resetuj()
{
    var rb1 = document.getElementById('rb1');
    var rb2 = document.getElementById('rb2');

    if(!rb1 || !rb2) return;

    var div;
    if(rb1.checked)
        reset('divDane1');
    else if (rb2.checked)
        reset('divDane2');
    else{
        alert('Wskaż tekst, który ma być wyczyszczony!');
        return;
    }
}
```

³ Uwaga! Zwrot „wszystkie podciągi” oznacza dokładnie wszystkie podciągi, a więc również fragmenty wyrazów oraz frazy powtarzające się jedna za drugą. Jeśli zatem wprowadzona zostanie np. jedna litera, to zaznaczone zostaną dokładnie wszystkie wystąpienia tej litery. Jeżeli zaznaczane miałyby być np. tylko całe słowa, należałoby odpowiednio przeformułować wyrażenie regularne.

```
    }  
  }  
  function reset(id)  
  {  
    var div = document.getElementById(id);  
    var re = new RegExp("</?span>", "gi");  
    if(div){  
      div.innerHTML = div.innerHTML.replace(re, "");  
    }  
  }  
}
```

Funkcja `resetuj` ma za zadanie stwierdzić, które z pól wyboru jest zaznaczone oraz wywołać funkcję `reset`, przekazując jej identyfikator warstwy, z której mają być usunięte zaznaczenia. Pobiera zatem odwołania do pól `rb1` oraz `rb2` i w zależności od tego, które z nich ma właściwość `checked` równą `true` (czyli jest zaznaczone), dokonuje wywołania w postaci `reset('divDane1')` lub `reset('divDane2')`. Jeżeli żadne z pól nie jest zaznaczone, wyświetlany jest komunikat.

Funkcja `reset` otrzymuje w postaci argumentu identyfikator jednej z warstw. Pobiera odwołanie do niej, a następnie zamienia wszystkie występujące w tekście warstwy wystąpienia znaczników `` i `` na puste ciągi znaków (`""`), czyli po prostu usuwa je. Odbывается to za pomocą metody `replace`, na podobnej zasadzie jak było to w funkcji `zaznacz`. Tworzone wyrażenie regularne (zmienna `re`) zostało skonstruowane w taki sposób, aby obejmowało oba wymienione znaczniki. Konstrukcja `</?span>` oznacza bowiem takie ciągi, które zaczynają się od znaku `<`, po którym występuje bądź nie występuje jeden znak `/`, a dalej znajduje się ciąg `span>`.

Przy konstruowaniu wyrażenia został użyty atrybut (flaga) i oznaczający, że przy przetwarzaniu tekstu ma być ignorowana wielkość liter. Jest to konieczne, bowiem właściwość `innerHTML` zawiera treść danego znacznika zinterpretowaną przez przeglądarkę. To oznacza, że umieszczone w warstwie znaczniki zostaną przetworzone i niezależnie od tego, jakiej wielkości liter użyliśmy w kodzie źródłowym, mogą być one zmienione na małe (Chrome, Firefox, Safari) bądź wielkie (Internet Explorer, Opera).

Skrypt 39.

[C][E][F][O][S]

Automatyczne podświetlanie wszystkich wystąpień poszukiwanego ciągu znaków

W skrypcie 38. zostało pokazane, jak wyszukiwać w tekście frazy podane przez użytkownika witryny. Taka czynność może być jednak zautomatyzowana w ten sposób, że zaznaczenia będą się pojawiać natychmiast po wprowadzeniu każdego poszukiwanego znaku. Nie jest to trudne zadanie, wystarczy użyć zdarzenia `onkeyup`. Interfejs

może zostać znacznie uproszczony, na stronie pozostanie jedynie pole tekstowe. Sekcja `<body>` będzie zatem miała postać widoczną na listingu 4.8.

Listing 4.8. Treść sekcji `<body>` przykładu 39.

```
<body>
  <div id="divInterfejs">
    <input type="text" id="tfFraza" onkeyup="szukaj('divDane');" />
  </div>
  <div id="divDane" class="divDane">
    <!-- tutaj treść warstwy -->
  </div>
</body>
```

Zdarzeniu `onkeyup` pola tekstowego `tfFraza` została przypisana procedura obsługi w postaci funkcji `szukaj`. Funkcja otrzymuje jako argument wywołania nazwę warstwy, której ma dotyczyć przeszukiwanie. Ogólna zasada działania skryptu pozostanie taka sama jak w przypadku przykładu 38., kod zostanie jednak znacząco skrócony ze względu na brak konieczności obsługi elementów interfejsu. Kod skryptu został zaprezentowany na listingu 4.9.

Listing 4.9. Funkcje obsługujące przeszukiwanie

```
function szukaj(id)
{
  var tfFraza = document.getElementById('tfFraza');
  var div = document.getElementById(id);

  if(!tfFraza || !div) return;

  reset(id);
  div.innerHTML = zaznacz(tfFraza.value, div.innerHTML);
}
function zaznacz(co, gdzie)
{
  //treść funkcji zaznacz
}
function reset(id)
{
  //treść funkcji reset
}
```

Funkcje `zaznacz` i `reset` pozostały takie same jak w przypadku przykładu 38. Funkcja `zaznacz` została znacząco uproszczona. Obecnie otrzymuje argument `id` określający identyfikator warstwy, której zawartość ma być przeszukana. Na początku pobierane jest odwołanie do tej warstwy oraz do pola `tfFraza`, a następnie wywoływane są metody `reset` i `zaznacz`. Tym samym po każdym zwolnieniu klawisza w obrębie pola tekstowego (czyli po wpisaniu bądź usunięciu znaku) następuje wyczyszczenie aktualnych zaznaczeń i wprowadzenie nowych. Dzięki temu nastąpiła pełna automatyzacja.

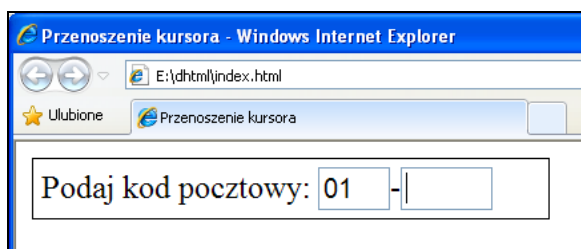
Skrypt 40. [C][E][F][O][S]

Automatyczne przenoszenie kursora między elementami formularza

Przydatną i ułatwiającą wpisywanie danych w formularzu funkcją jest automatyczne przemieszczanie kursora do kolejnego elementu. Oczywiście rozwiązanie takie jest sensowne tylko w przypadku danych, których format jest z góry określony. Przykładem może tu być kod pocztowy mający format dwie cyfry – trzy cyfry. Mógłby być więc wprowadzany w dwóch polach, tak jak zaprezentowano to na rysunku 4.5. Po wprowadzeniu dwóch pierwszych znaków do pierwszego pola kursor automatycznie przeskoczy do drugiego. Oczywiście to tylko przykład, a zaprezentowana niżej technika będzie mogła zostać użyta do dowolnych innych danych.

Rysunek 4.5.

Formularz umożliwiający wprowadzenie kodu pocztowego



Aby uzyskać opisany efekt, niezbędne jest wykrywanie liczby znaków aktualnie wpisanych do pola tekstowego. Najwygodniej jest wykorzystać w tym celu zdarzenie `keyup` wywoływane w momencie zwolnienia klawisza. Dodatkowo należałoby ograniczyć maksymalną liczbę znaków, które użytkownik może wprowadzić do pola. To można jednak zrobić z poziomu (X)HTML-a, wykorzystując atrybut `maxlength` znacznika `<input>`. Konstrukcja HTML powinna zatem wyglądać tak, jak przedstawiono to na listingu 4.10.

Listing 4.10. Pola tekstowe pozwalające wprowadzić kod pocztowy

```
<body>
  <div id="divDane">
    Podaj kod pocztowy:
    <input type="text" maxlength="2" id="tfKod1" size="2"
      onkeyup="sprawdźDługość('tfKod1', 2, 'tfKod2');">
    <input type="text" maxlength="3" id="tfKod2" size="3" />
  </div>
</body>
```

Oba pola tekstowe zostały zdefiniowane w standardowy sposób. Mają swoje własne identyfikatory (`tfKod1` i `tfKod2`) oraz atrybuty `maxlength` ograniczające maksymalną liczbę znaków, które będzie mógł wprowadzić użytkownik witryny. W przypadku pierwszego pola są to dwa znaki, w przypadku drugiego — trzy. Pole `tfKod1` ma również przypisaną procedurę obsługi zdarzenia `keyup` w postaci funkcji `sprawdźDługość`,

a zatem będzie ona wykonywana po każdym zwolnieniu klawisza (w obrębie tego pola). Funkcja przyjmuje trzy argumenty. Pierwszy wskazuje nazwę pola, którego dotyczy będzie sprawdzanie liczby wprowadzonych znaków, drugi — liczbę znaków, po których ma zostać przeniesiony kursor, trzeci — identyfikator pola, do którego ma zostać przeniesiony kursor. Wywołanie:

```
sprawdźDługość('tfKod1', 2, 'tfKod2');
```

można zatem potraktować jako polecenie: jeśli w polu tfKod1 znajdują się co najmniej dwa znaki, przenieś kursor do pola tfKod2.

Treść funkcji sprawdźDługość została przedstawiona na listingu 4.11.

Listing 4.11. *Kod funkcji sprawdźDługość*

```
<script type="text/javascript">
  function sprawdźDługość(pole1, ile, pole2)
  {
    var tf1 = document.getElementById(pole1);
    var tf2 = document.getElementById(pole2);
    if(!tf1 || !tf2 || !tf1.value) return;

    if (tf1.value.length >= 2)
      tf2.focus();
  }
</script>
```

Funkcja najpierw pobiera odwołania do elementów witryny przekazanych w postaci argumentów pole1 i pole2 i zapisuje je w zmiennych tf1 i tf2. Następnie sprawdza, czy istnieją obiekty tf1 i tf2 (co oznacza, że na stronie istnieją wymienione elementy) oraz czy obiekt wskazywany przez tf1 ma właściwość value. Jeśli odpowiedź na powyższe pytania jest twierdząca, za pomocą instrukcji warunkowej if badane jest to, czy liczba znaków zapisanych w polu tf1 (reprezentowanym przez zmienną tf1) jest większa bądź równa 2. Jeżeli tak jest, wywoływana jest metoda focus pola wskazywanego przez tf2, a tym samym jest do niego przenoszony kursor (pole otrzymuje fokus).

Skrypt 41. [C][E][F][S]

Blokada wpisywania w formularzu wybranych znaków (blokada klawiszy)

Przy pracy z formularzami może się zdarzyć, że będziemy chcieli zablokować wpisywanie przez użytkownika niektórych znaków. Dzięki temu będziemy mieli pewność, że osoba wprowadzająca dane nie popełni tak prostego błędu jak podanie liter w numerze telefonu czy kodzie pocztowym. Oczywiście aby tego rodzaju blokada była możliwa, musimy sprawdzać, jaki klawisz został wciśnięty. Niezbędne będzie zatem oprogramowanie zdarzenia keydown, keyup lub keypress pola tekstowego służącego do

wprowadzania danych. W kodzie funkcji będącej procedurą obsługi zdarzenia należy rozpoznać kod wciśniętego klawisza i jeśli będzie to klawisz, który chcemy zablokować, uniemożliwić wyświetlenie znaku.

Przyjmijmy zatem, że mamy do czynienia z formularzem służącym do wprowadzania kodu pocztowego i składa się on z dwóch pól. Ma więc taki sam wygląd jak w przykładzie 40. Różnica będzie taka, że oba pola tekstowe otrzymają procedurę obsługi zdarzenia `keydown`. Kod sekcji `<body>` przyjmie zatem postać widoczną na listingu 4.12.

Listing 4.12. *Formularz służący do podawania kodu pocztowego*

```
<body>
  <div id="divDane">
    Podaj kod pocztowy:
    <input type="text" maxlength="2" id="tfKod1"
      onkeydown="keyDown(event);"
    />-<input type="text" maxlength="3" id="tfKod2"
      onkeydown="keyDown(event);"
    />
  </div>
</body>
```

Wspomnianą procedurą obsługi zdarzenia `keydown` jest funkcja `keyDown`, która tym samym będzie wywoływana po każdym kliknięciu klawisza, gdy kursor znajduje się w obrębie danego pola tekstowego. Treść funkcji wraz z pozostałą częścią skryptu została przedstawiona na listingu 4.13.

Listing 4.13. *Skrypt blokujący wybrane klawisze*

```
<script type="text/javascript">
  var dopuszczoneKody = [8, 9, 37, 39];
  function inArray(arr, kod)
  {
    for(var i = 0; i < arr.length; i++){
      if(arr[i] == kod)
        return true;
    }
    return false;
  }
  function keyDown(evt)
  {
    var kod = evt.which ? evt.which : evt.keyCode ? evt.keyCode : 0;

    if((kod < 48 || kod > 57) && !inArray(dopuszczoneKody, kod)){
      evt.returnValue = false;
      if(evt.preventDefault) evt.preventDefault();
    }
    else{
      evt.returnValue = true;
    }
  }
</script>
```

Funkcja `keyDown` otrzyma w postaci argumentu `evt` obiekt zdarzenia zapoczątkowanego wciśnięciem klawisza. Kluczową sprawą jest pobranie kodu tego klawisza, tak by można było ustalić, czy ma być zablokowany, czy też nie. W zależności od przeglądarki, w której zostanie uruchomiony skrypt, kod może się znajdować we właściwości `which` (Chrome, Firefox, Safari) lub `keyCode` (Internet Explorer). Pierwsza instrukcja funkcji dokonuje więc odpowiedniego rozpoznania i przypisuje odczytany kod zmiennej `kod`. Zostało użyte złożone wyrażenie warunkowe, które można rozpisać za pomocą instrukcji warunkowej `if` w następujący sposób:

```
if(evt.which)
    kod = evt.which;
else if(evt.keyCode)
    kod = evt.keyCode;
else
    kod = 0;
```

Taki zapis jest czytelniejszy, ale też bardziej rozwlekły. Warto też zwrócić uwagę, że w przypadku gdy nie istnieje ani właściwość `which`, ani `keyCode`, zmienna `kod` przyjmuje wartość 0, co w praktyce oznacza, że klawisze nie będą blokowane.

Za pomocą instrukcji `if` sprawdzane jest następnie, czy uzyskany kod wykracza poza zdefiniowany przedział. Ponieważ chcemy wprowadzać tylko cyfry, warunek ma postać:

```
kod < 48 || kod > 57
```

wynika to z tego, że kodem klawisza 0 jest 48, kodem klawisza 1 — 49 itd., aż do klawisza 9 o kodzie 57. Ponieważ w taki sposób zablokowane zostaną także klawisze funkcyjne takie jak *Backspace* czy strzałki sterujące kursorem, wprowadzony został dodatkowy warunek:

```
!isArray(dopuszczoneKody, kod)
```

To oznacza, że blokada nie będzie następowała, gdy kod klawisza znajduje się w tablicy `dopuszczoneKody`. Tablica ta została zdefiniowana na początku skryptu i zawiera wartości: 8 — tabulacja, 9 — *Backspace*, 37 — strzałka w prawo, 39 — strzałka w lewo. W razie potrzeby można ją uzupełnić o kody innych klawiszy.

Sprawdzaniem, czy wskazany kod znajduje się w tablicy `dopuszczoneKody`, zajmuje się funkcja `isArray`. Jest w niej używana pętla `for` przebiegająca przez wszystkie komórki. Jeśli kod przekazany w postaci argumentu `kod` znajduje się w tablicy przekazanej w postaci argumentu `arr`, funkcja zwraca wartość `true`, a w przeciwnym przypadku — wartość `false`.

Samo uzyskanie blokady nie jest trudne. Jeśli klawisz ma być zablokowany, należy (w funkcji będącej procedurą obsługi zdarzenia) właściwości `evt.returnValue` przypisać wartość `false`, a w przeciwnym wypadku — wartość `true`. W przeglądarkach z rodziny Firefox dodatkowo należy wywołać metodę `preventDefault`. Dlatego też jeśli ta metoda jest dostępna (co jest sprawdzane za pomocą instrukcji `if(evt.preventDefault)`), jest wywoływana.

Oczywiście skrypt można modyfikować tak, by blokowane były inne zestawy znaków, np. umożliwić wprowadzanie tylko liter (alfabetu łacińskiego) i znaków o kodach umieszczonych w tablicy `dopuszczoneKody`. Należy wtedy użyć warunku:

```
if((kod < 65 || kod > 90) && !isArray(dopuszczoneKody, kod))
```

Skrypt 42. [C][E][F][O][S]

Dynamiczna weryfikacja danych w trakcie ich wprowadzania (według określonego wzorca)

Czasami chcemy na bieżąco informować użytkownika, czy wprowadzane przez niego dane są poprawne. Przykładowo w polach tekstowych mogą być zabronione pewne znaki, czy też konieczna jest określona długość tekstu. Użytkownik powinien cały czas widzieć, czy w obecnej chwili dane mają właściwy format, czy też nie. Sposób sygnalizacji jest sprawą dowolną, mogą to być np. odpowiedni kolor tła danego elementu strony albo też ikony sygnalizujące różne stany.

Przyjmijmy, że mamy do dyspozycji formularz, w którym do dwóch pól trzeba wprowadzić imię i nazwisko. Oba będą zatem mogły zawierać tylko litery, będą też musiały zawierać co najmniej dwa znaki. Przy prawidłowo wypełnionym polu będzie się znajdowała ikona ✓, a przy wypełnionym nieprawidłowo — ikona ✗. Formularz będzie miał zatem postać przedstawioną na rysunku 4.6. Jest on generowany przez kod HTML widoczny na listingu 4.14.

Rysunek 4.6.
Formularz
z dynamiczną
weryfikacją danych

Listing 4.14. Sekcja `<body>` dla skryptu 42.

```
<body>
  <div id="divDane">
    <form name="form1" action="" method="get">
      <div>
        <span style="font-size:14pt;font-weight:bold;">
          Proszę podać swoje dane:</span><br />
```

```

        <span style="font-size:smaller;font-style:italic;">
        (wolno podawać tylko litery, nie mniej niż 2 znaki)</span>
    </div>
    <table>
    <tr>
        <td><b>Dane personalne:</b></td>
        <td></td>
    </tr>
    <tr>
        <td>imię:</td>
        <td>
            <input type="text" name="imię"
                onkeyup="sprawdź(this.value, 'imgTfImię');" />
            
        </td>
    </tr>
    <tr>
        <td>nazwisko:</td>
        <td>
            <input type="text" name="nazwisko"
                onkeyup="sprawdź(this.value, 'imgTfNazwisko');" />
            
        </td>
    </tr>
    <tr>
        <td style="text-align:center;" colspan="2">
            <input type="submit" name="wyslij" value=" Wyślij! " />
        </td>
    </tr>
    </table>
</form>
</div>
</body>

```

Sposób budowania i formatowania formularza jest taki sam jak w przykładzie 36., zawiera on jednak mniej pól. Są tylko dwa: *imię* i *nazwisko*. Za polami znajdują się obrazy określające poprawność danych, zdefiniowane za pomocą znaczników ``. Dla pola *imię* jest to obraz o identyfikatorze `imgTfImię`, a dla pola *nazwisko* — obraz `imgTfNazwisko`.

Polom zostały przypisane procedury obsługi zdarzeń `keyup` w postaci funkcji `sprawdź`. Otrzymuje ona dwa argumenty. Pierwszy to wartość podlegająca sprawdzeniu, drugi — identyfikator obrazu przypisany danemu polu. Zapis `this.value` w tym kontekście oznacza wartość wprowadzoną do danego pola.

Treść funkcji `sprawdź` została przedstawiona na listingu 4.15.

Listing 4.15. Skrypt weryfikujący dane

```

<script type="text/javascript">
    var re = /[a-zA-Z]{2,}/;
    function sprawdź(str, imgId)
    {
        var img = document.getElementById(imgId);
    }

```

```
if(!img) return;

if(re.exec(str) != str)
    img.src = 'blad.png';
else
    img.src = 'ok.png';
}
</script>
```

Treść funkcji jest nadzwyczaj prosta. Pobiera ona odwołanie do obrazu o identyfikatorze przekazanym w postaci argumentu `imgId` i zapisuje je w zmiennej `img`. Jeśli dane (tekst) przekazane za pomocą argumentu `str` nie są poprawne, zmienia atrybut `src` obrazu `img` na wskazujący plik graficzny z ikoną błędu (*blad.png*). W przeciwnym wypadku przypisuje temu atrybutowi nazwę pliku z ikoną symbolizującą poprawność danych (*ok.png*).

Weryfikacja danych odbywa się za pomocą wyrażenia regularnego zapisanego w globalnej zmiennej `re` oraz metody `exec`. Ta technika była już stosowana w przykładach z tego rozdziału (np. skrypcie 37.). Zastosowane wyrażenie `/[a-zA-Z]{2,}/` za prawidłowe uzna tylko takie ciągi, które zawierają wyłącznie małe oraz wielkie litery i mają co najmniej dwa znaki. Oczywiście można je dowolnie modyfikować i dostosowywać do potrzebnych w danej witrynie wzorców danych (podane wyrażenie nie uzna za poprawne np. imienia dwuczłonowego).

Skrypt 43. [C][E][F][O][S]

Pole tekstowe automatycznie zmieniające swoją wielkość

Pole tekstowe definiowane za pomocą znacznika `<input>` ma stałą wielkość. Jeśli jednak używamy JavaScriptu, możemy modyfikować tę właściwość, osiągając ciekawy efekt dopasowywania rozmiaru pola do wpisywanego tekstu. Tak właśnie działa skrypt przedstawiony na listingach 4.16 i 4.17. Pierwszy listing zawiera treść sekcji `<body>`. Najlepszy efekt jest osiągany, jeśli pole tekstowe korzysta z czcionki o stałej szerokości znaku (np. Courier).

Listing 4.16. Sekcja `<body>` zawierająca pole tekstowe

```
<body>
  <div id="divDane">
    <input type="text" size="5" maxlength="20"
      onkeyup="sprawdźRozmiar(this);"
      style="font-family:Courier" />
  </div>
</body>
```

Zdarzeniu `keyup`, powstającemu po każdym zwolnieniu klawisza w obrębie pola tekstowego, została przypisana procedura obsługi w postaci funkcji `sprawdźRozmiar`. Funkcja otrzymuje jako argument odwołanie do pola tekstowego, którego rozmiar ma być modyfikowany. Odwołanie `this` oznacza pole bieżące — to, w którym znajduje się wywołanie funkcji. Treść funkcji została przedstawiona na listingu 4.17. Pole tekstowe ma zdefiniowaną wielkość bieżącą (5 znaków) oraz maksymalną (20 znaków). Jeżeli nie zostanie podana wartość maksymalna, zostanie użyta wartość zdefiniowana w skrypcie.

Listing 4.17. *Skrypt zmieniający rozmiar pola tekstowego*

```
<script type="text/javascript">
  var rozmiarMin = 5;
  var rozmiarMax = 20;
  function sprawdźRozmiar(pole)
  {
    if(!pole) return;
    if(pole.maxLength < 0 || pole.maxLength > 10000)
      pole.maxLength = rozmiarMax;
    if (pole.value.length <= rozmiarMin){
      pole.size = rozmiarMin;
    }
    else if(pole.value.length < pole.maxLength){
      pole.size = pole.value.length;
    }
    else{
      pole.size = pole.maxLength;
    }
  }
</script>
```

Na początku znajdują się deklaracje dwóch zmiennych globalnych `rozmiarMin` i `rozmiarMax`. Pierwsza określa minimalny rozmiar pola, poniżej którego jego wielkość nie będzie mogła być zmieniona, druga — rozmiar maksymalny, w sytuacji gdy wielkość ta nie została ustalona w kodzie HTML (atrybut `maxLength`). Bieżący rozmiar pola można odczytać dzięki odwołaniu `pole.size`, rozmiar maksymalny (wartość atrybutu `maxLength`) — dzięki odwołaniu `pole.maxLength`, a liczbę znaków tekstu znajdującego się w polu — dzięki odwołaniu `pole.value.length`.

Do prawidłowego działania skryptu niezbędne jest ustalenie maksymalnego rozmiaru pola. Powinno się to odbywać w kodzie HTML. Jeśli nie zostanie to zrobione, przeglądarka ustali własną wartość domyślną. Jest ona różna w różnych produktach (np. Firefox, Opera — -1, Internet Explorer — 2147483647, Chrome, Safari — 524288). Dlatego też stosowana jest instrukcja warunkowa:

```
if(pole.maxLength < 0 || pole.maxLength > 10000)
```

Założeniem jest, że nikt nie stosuje wartości atrybutu `maxLength` o wartości większej niż 10000 (atrybut `maxLength` w HTML odpowiada właściwości `maxLength` w JavaScriptcie). Jeśli jeden z warunków instrukcji jest prawdziwy, następuje użycie maksymalnej wartości zdefiniowanej w skrypcie (zmienna `rozmiarMax`).

Funkcja `sprawnRozmiar` porównuje liczbę znaków wpisanych do pola tekstowego z jego aktualnym rozmiarem i rozpatruje trzy różne sytuacje:

- ♦ Liczba wpisanych znaków jest mniejsza od minimalnego rozmiaru pola — polu tekstowemu jest przypisywany minimalny rozmiar (wartość zmiennej `rozmiarMin`).
- ♦ Liczba wpisanych znaków jest większa od minimalnego rozmiaru pola i mniejsza od maksymalnego — polu tekstowemu jest przypisywany rozmiar zgodny z aktualną liczbą wpisanych znaków (wartość właściwości `pole.value.length`).
- ♦ Liczba wpisanych znaków jest większa od maksymalnego rozmiaru pola — polu tekstowemu jest przypisywany maksymalny rozmiar (wartość zapisana we właściwości `maxLength`).

Skrypt 44. [C][E][F][O][S]

Ograniczenie liczby znaków wpisywanych do rozszerzonego pola tekstowego

O ile zwykle pole tekstowe ma atrybut `maxlength`, który pozwala ograniczyć maksymalną liczbę wpisywanych znaków, o tyle rozszerzone pole tekstowe generowane za pomocą znacznika `<textarea>` jest pozbawione takiego udogodnienia. Można je jednak zasymulować za pomocą JavaScriptu. To przydatna funkcjonalność, często bowiem nie można pozwolić użytkownikowi na wprowadzanie tekstu o nieograniczonej długości. Takie zadanie może być zrealizowane w sposób przedstawiony na listingu 4.18.

Listing 4.18. Ograniczenie liczby znaków w rozszerzonym polu tekstowym

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN"
"http://www.w3.org/TR/html4/strict.dtd">
<html lang="pl">
<head>
  <meta http-equiv="Content-Type" content="text/html; charset=utf-8">
  <title>Ograniczenie długości pola</title>
  <script type="text/javascript">
    function sprawnRozmiar(pole, max)
    {
      if(!pole) return;
      max = parseInt(max);
      if(isNaN(max)) return;

      if(pole.value.length >= max)
        pole.value = pole.value.substring(0, max);
    }
  </script>
```



```
</head>
<body>
  <div id="divDane">
    Wprowadź dane (maksymalnie 30 znaków):<br />
    <textarea rows="5" cols="30"
      onkeydown="sprawdźRozmiar(this, 5);"
    /></textarea>
  </div>
</body>
</html>
```

W sekcji `<body>` za pomocą znacznika zostało zdefiniowane rozszerzone pole tekstowe. Jego szerokość została określona na 30 znaków (`cols="30"`), a wysokość na 5 wierszy (`rows="5"`). Zdarzeniu `keydown` została przypisana procedura obsługi w postaci funkcji `sprawdźRozmiar`, której zadaniem jest ograniczanie liczby znaków wpisanych do pola. Funkcja przyjmuje dwa argumenty. Pierwszy określa pole, którego ma dotyczyć ograniczenie, a drugi — maksymalnie dopuszczalną liczbę znaków.

Kod funkcji zaczyna się od sprawdzenia, czy istnieje pierwszy argument — pole. Następnie za pomocą metody `parseInt` badane jest, czy wartość drugiego argumentu może być potraktowana jako liczba całkowita. To weryfikacja poprawności tego argumentu. Jeśli bowiem nie będzie on liczbą całkowitą (a dokładniej: nie będzie mógł być przetworzony na liczbę całkowitą), zostanie to wychwycone za pomocą instrukcji warunkowej `if(isNaN(max))` i kod funkcji zostanie przerwany.

Jeżeli jednak oba argumenty są prawidłowe, następuje sprawdzenie, czy liczba znaków w polu przekracza wartość maksymalną:

```
if(pole.value.length >= max)
```

Jeśli tak jest, za pomocą metody `substring` następuje usunięcie wszystkich nadmiarowych znaków, a ograniczony ciąg jest ponownie zapisywany w polu:

```
pole.value = pole.value.substring(0, max);
```

Skrypt 45. Wyróżnianie aktywnego elementu formularza (witryny)

[C][E][F][O][S]

Aktywny element formularza (a ogólniej — witryny) może być wyróżniany. Niektóre przeglądarki same stosują np. nieco inne wyglądy aktywnych pól tekstowych czy pól wyboru. O sposobie wyróżniania możemy jednak decydować samodzielnie, najlepiej za pomocą JavaScriptu i CSS. Załóżmy, że mamy do dyspozycji formularz o dwóch polach tekstowych ze skryptu 42. Chcemy, aby aktywne pole (to, które ma fokus) miało inny kolor tła, np. żółty. Można to osiągnąć, korzystając ze zdarzeń `focus` i `blur`. Kod HTML przyjmie więc postać przedstawioną na listingu 4.19.

Listing 4.19. Pola tekstowe z atrybutami *onfocus* i *onblur*

```
<!-- początek kodu z listingu 4.14 -->
<tr>
  <td>imię:</td>
  <td>
    <input type="text" name="imię"
      onfocus="wyróżnij(this, 'aktywne');"
      onblur="wyróżnij(this, 'nieaktywne');"/>
  </td>
</tr>
<tr>
  <td>nazwisko:</td>
  <td>
    <input type="text" name="nazwisko"
      onfocus="wyróżnij(this, 'aktywne');"
      onblur="wyróżnij(this, 'nieaktywne');"/>
  </td>
</tr>
<!-- dalsza część kodu z listingu 4.14 -->
```

Początek i koniec kodu będzie taki sam jak w przykładzie 42., dlatego też te fragmenty zostały pominięte. Zmieniła się natomiast definicja pól tekstowych. Znaczniki `<input>` otrzymały atrybuty `onfocus` i `onblur`. Oznacza to obsługę zdarzeń `focus` i `blur`. Pierwsze występuje wtedy, gdy element otrzymuje fokus (staje się elementem bieżącym, aktywnym), a drugie — gdy element traci fokus (przestaje być aktywny). Wystarczy zatem w odpowiedzi na te zdarzenia wywoływać funkcję dokonującą zmiany w danym elemencie. Najprościej modyfikować styl. Tak też działa funkcja `wyróżnij`. Pierwszym argumentem jest odwołanie do pola, które ma być modyfikowane (określenie `this` jest wskazaniem do pola będącego inicjatorem zdarzenia), a drugim — nazwa klasy CSS, która zostanie temu polu przypisana. Treść funkcji `wyróżnij` została przedstawiona na listingu 4.20.

Listing 4.20. Treść funkcji *wyróżnij*

```
<script type="text/javascript">
  function wyróżnij(pole, klasa){
    if(!pole) return;
    pole.className = klasa;
  }
</script>
```

Pierwszy argument funkcji wskazuje pole, którego styl będzie modyfikowany, a drugi — nazwę klasy CSS. Przypisanie klasy polu odbywa się przez modyfikację atrybutu `className`. Jaka będzie konstrukcja klasy CSS, to zależy od konkretnych potrzeb oraz sposobu, w jaki modyfikowany element ma być wyróżniony. Przykładowe style zostały przedstawione na listingu 4.21. Warstwa z formularzem (`divDane`) jest wyróżniana przez nadanie jej obramowania. Określone zostały także wypełnienie oraz maksymalna szerokość. Klasa `aktywne` definiuje żółty kolor tła, a klasa `nieaktywne` — biały. Tym samym dzięki działaniu skryptu aktywne pole formularza będzie miało żółty kolor tła, a nieaktywne — biały.

Listing 4.21. *Style CSS dla skryptu 45.*

```
<style type="text/css">
  #divDane{
    border:1px solid black;
    max-width:320px;
    padding:4px;
  }
  .aktywne{
    background-color:yellow;
  }
  .nieaktywne{
    background-color:white;
  }
</style>
```

Rozdział 5.

Różności

Istnieją skrypty, które czasem trudno zaliczyć do konkretnej kategorii, trudno też tworzyć osobny dział dla jednego lub dwóch przykładów. Stąd tytuł rozdziału 5. Znajduje się w nim siedem skryptów realizujących kilka różnych efektów. Zostanie pokazane, jak spowodować, by użytkownik potwierdzał chęć wykonania wybranych operacji, i jak zapewnić wykonanie operacji domyślnej w przypadku, gdy w zadanym czasie nie zostanie wykonana żadna akcja. Przedstawione zostaną proste sposoby na modyfikację informacji przedstawionych na pasku tytułowym i pasku stanu przeglądarki, a także na weryfikację formalnej poprawności adresu e-mail oraz na tworzenie sterowanych programowo pasków postępu.

Skrypt 46.

[C][E][F][O][S]

Potwierdzanie operacji przez użytkownika

Jeśli chcemy, aby użytkownik potwierdzał chęć wykonania pewnej operacji, możemy użyć metody `prompt` obiektu `window`. Wyświetla ona standardowe okno dialogowe z komunikatem oraz przyciskami *OK* i *Anuluj* (*Cancel*). Jeżeli zostanie kliknięty przycisk *OK*, metoda zwraca wartość `true`, a jeżeli zostanie kliknięty przycisk *Anuluj* lub okno zostało zamknięte w inny sposób — wartość `false`. Założmy zatem, że na stronie chcemy umieścić pewne dane, które mają być wyświetlane na wyraźne życzenie użytkownika — może to być np. wynik meczu. Kod HTML sekcji `<body>` przyjmie postać widoczną na listingu 5.1.

Listing 5.1. Kod sekcji `<body>` przykładu 46.

```
<body>
  <div id="divInterfejs">
    Kliknij przycisk, jeśli chcesz zobaczyć wynik.
    <input type="button" value="Wyświetl wynik"
      onclick="wyświetl('divWynik')" />
```

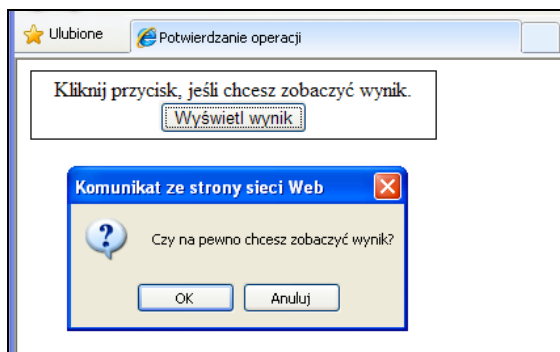
```

</div>
<div id="divWynik" style="display:none">
    Wynik meczu to 0:1.
</div>
</body>

```

Mamy tu dwie warstwy: `divInterfejs` i `divWynik`. Pierwsza zawiera przycisk, którego kliknięcie spowoduje wyświetlenie okna potwierdzającego chęć wykonania danej operacji. Będzie miało postać przedstawioną na rysunku 5.1. Druga zawiera dane, które początkowo mają być ukryte. Dlatego też przypisano jej styl z atrybutem `display` równym `none` (dzięki temu warstwa ta nie będzie wyświetlana). Wyświetlaniem wspomnianego okna oraz stwierdzeniem, który z przycisków został w nim kliknięty, zajmie się funkcja `wyświetl` będąca procedurą obsługi zdarzenia `click` przycisku. Jej treść została przedstawiona na listingu 5.2.

Rysunek 5.1.
Okno dialogowe
generowane przez
metodę `confirm`



Listing 5.2. Treść funkcji `wyświetl`

```

<script type="text/javascript">
    function wyświetl(id){
        if(!confirm("Czy na pewno chcesz zobaczyć wynik?"))
            return;
        var el = document.getElementById(id);
        if(!el) return;
        el.style.display = "block";
    }
</script>

```

Funkcja otrzymuje argument określający identyfikator elementu, który ma się pojawić na ekranie, o ile użytkownik potwierdził chęć wykonania tej operacji. Wywołuje więc metodę `confirm` z odpowiednim komunikatem i od razu za pomocą instrukcji warunkowej `if` sprawdza, jaki był efekt wywołania (jaka wartość została zwrócona). Jeżeli zwrócona została wartość `false` (co oznacza kliknięcie przycisku *Anuluj* bądź zamknięcie okna w inny sposób), działanie funkcji jest kończone za pomocą instrukcji `return`. Gdy jest to wartość `true` (co oznacza kliknięcie przycisku *OK*), pobierane jest odwołanie do elementu o identyfikatorze przekazanym w argumentcie `id` oraz element ten jest wyświetlany. Pojawia się na ekranie dzięki przypisaniu ciągu `block` właściwości `display` obiektu `style` (oznacza to zmianę sposobu wyświetlania na blokowy).

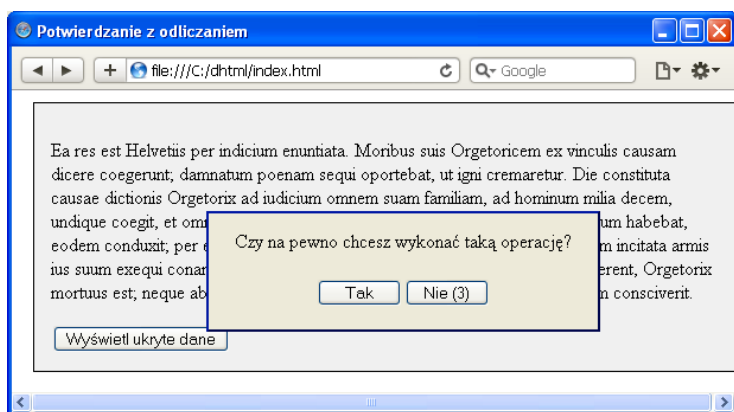
Skrypt 47. [C][E][F][O][S]

Okno potwierdzania z odliczaniem

Do potwierdzania chęci wykonania pewnej operacji najczęściej jest używana metoda `confirm`. Jej wadą jest niemożność zmiany wyglądu okna dialogowego, w tym także tekstu przycisków. Co więcej, skrypt musi czekać na odpowiedź użytkownika. Dużo większe możliwości dałoby w pełni konfigurowalne okno potwierdzeń. Aby je używać, trzeba jednak napisać własny kod, gdyż nie ma standardowej metody pozwalającej na coś takiego. Najlepiej skorzystać z pomysłu przedstawionego w skrypcie 9., czyli modalnego okna dialogowego. Jeśli umieścimy na nim dwa przyciski, dodamy kod obsługujący odliczanie oraz procedury wykonujące zadania odpowiednie dla danego przycisku, uzyskamy poszukiwany efekt.

Tak właśnie działa skrypt 47. Wygląd okna jest w pełni konfigurowalny i zależy tylko od naszej fantazji. Można je przygotować nawet tak, by wyglądało podobnie do standardowego okna systemowego. To oczywiście wymagałoby użycia odpowiedniej grafiki oraz stylów CSS. W prostszym przypadku mogłoby wyglądać tak jak na rysunku 5.2. Po wyświetleniu okna aktywowany będzie zegar odliczający czas. Jeżeli użytkownik strony nie kliknie żadnego z przycisków, po zadanej liczbie sekund przycisk domyślny zostanie kliknięty automatycznie. Na tym przycisku będzie oczywiście widoczny upływający czas.

Rysunek 5.2.
*Okno dialogowe
z odliczaniem czasu*



Kod sekcji `<body>` przyjmie postać widoczną na listingu 5.3. Po załadowaniu strony będzie widoczna tylko jej część. Kliknięcie przycisku spowoduje wyświetlenie omawianego okna dialogowego, pozwalającego podjąć decyzję, czy ma zostać pokazana ukryta treść.

Listing 5.3. *Kod sekcji `<body>` skryptu 47.*

```
<body>
  <div id="transparentDiv">
  </div>
  <div id="dialogDiv">
    Czy na pewno chcesz wykonać taką operację?<br /><br />
```

```



```

Struktura strony jest podobna do tej z przykładu 9. Warstwa `transparentDiv` służy do „przykrycia” właściwej strony. Warstwa `dialogDiv` reprezentuje okno dialogowe. Zawiera napis, który pojawi się w oknie oraz dwa przyciski odpowiadające opcjom *Tak* (`btnTak`) i *Nie* (`btnNie`). Kliknięcie dowolnego z przycisków będzie powodowało wywołanie funkcji `closeModal`. Otrzyma ona argument określający, który przycisk został kliknięty (`true` — przycisk *Tak*, `false` — przycisk *Nie*).

Warstwa `mainDiv` zawiera zwykłą treść witryny oraz przycisk pozwalający wyświetlić treść ukrytą znajdującą się na warstwie `hiddenDiv` (początkowo niewidocznej ze względu na cechę `display` o wartości `none`). Kliknięcie przycisku będzie powodowało wywołanie funkcji `showModal` o argumentach wskazujących rozmiar okna dialogowego (300×70 pikseli), czas, w którym będzie widoczne na ekranie (5 sekund) i przycisk domyślny (`false` — przycisk *Nie*).

Listing 5.4. Pierwsza część skryptu 47.

```

<script type="text/javascript">
  var dialogWidth = 300;
  var dialogHeight = 200;
  var timerId = null;
  function showModal(dW, dH, licznik, taknie)
  {
    // tutaj treść funkcji showModal z listingu 1.20
    odliczaj(licznik, taknie);
  }
  function closeModal(taknie)
  {
    if(timerId){
      clearTimeout(timerId);
      timerId = null;
    }

    // tutaj treść funkcji closeModal z listingu 1.20

    if(taknie){
      var el = document.getElementById('hiddenDiv');
      if(el) el.style.display = "block";
    }
  }
</script>

```


Ogólna struktura skryptu wraz z funkcjami `showModal` i `closeModal` pozostała taka sama jak w przykładzie 9. Na początku pojawiła się nowa zmienna globalna — `timerId` — która będzie przechowywała identyfikator timera związanego z odliczaniem czasu. Treść funkcji `showModal` pozostała prawie niezmienniona. Pojawiły się jedynie dwa nowe argumenty. Pierwszy określa czas, w którym okno będzie pozostawało na ekranie, jeżeli użytkownik nie podejmie żadnej akcji, drugi — akcję domyślną, czyli to, który z przycisków ma zostać automatycznie kliknięty, jeśli użytkownik nie podejmie samodzielnie decyzji. Oba te argumenty są przekazywane dalej do opisaną niżej funkcji `odliczaj`. A zatem wywołanie funkcji `showModal` spowoduje taką samą reakcję, jak w przypadku skryptu 9. — ukaże się okno dialogowe — oraz dodatkowo zostanie uruchomione odliczanie.

Funkcja `closeModal` jest wykonywana po kliknięciu jednego z przycisków okna dialogowego. Otrzymuje argument `taknie` wskazujący, który przycisk został kliknięty. Wartość `true` oznacza przycisk *Tak*, a `false` — przycisk *Nie*. Na początku sprawdza, czy jest aktywny timer (`timerId` różne od `null`). Jeśli tak, timer jest wyłączany przez wywołanie metody `clearTimeout`, oznacza to bowiem, że użytkownik samodzielnie kliknął przycisk i odliczanie nie powinno być kontynuowane. Dalej wykonywane są czynności związane z zamknięciem okna. Są takie jak w przykładzie 9. Na końcu kodu trzeba jednak wykonać akcję związaną z wyborem opcji *Tak* lub *Nie*. W tym przypadku dodatkowy kod jest wykonywany tylko wtedy, gdy został kliknięty przycisk *Tak* (gdy parametr `taknie` ma wartość `true`). Jest wtedy wyświetlana warstwa `hiddenDiv`, co odbywa się przez zmianę stylu wyświetlania na blokowy.

Do prawidłowego działania skryptu potrzebna jest jeszcze funkcja `odliczaj`. Jej treść została przedstawiona na listingu 5.5.

Listing 5.5. Treść funkcji `odliczaj`

```
function odliczaj(licznik, taknie)
{
    var btn = null;
    var tekst = "";
    if(taknie){
        btn = document.getElementById("btnTak");
        tekst = "Tak (" + licznik + ")";
    }
    else{
        btn = document.getElementById("btnNie");
        tekst = "Nie (" + licznik + ")";
    }
    if(licznik-- < 0){
        btn.click();
    }
    else{
        btn.value = tekst;
        timerId = setTimeout("odliczaj('"+licznik+"','"+taknie+"'", 1000);
    }
}
```

Funkcja przyjmuje dwa argumenty. Pierwszy to licznik wskazujący liczbę sekund pozostałych do kliknięcia domyślnego przycisku, drugi określa to, który z przycisków

jest domyślny. Jeżeli przyciskiem domyślnym jest *Tak* (takie równie *true*), zmiennej pomocniczej *btn* jest przypisywane odwołanie do elementu strony o identyfikatorze *btnTak*, a zmiennej pomocniczej *tekst* — ciąg *Tak* uzupełniony o aktualny stan zmiennej *licznik*. Jeżeli przyciskiem domyślnym jest *Nie* (takie różne od *true*), zmiennej pomocniczej *btn* jest przypisywane odwołanie do elementu strony o identyfikatorze *btnNie*, a zmiennej pomocniczej *tekst* — ciąg *Nie* uzupełniony o aktualny stan zmiennej *licznik*.

Następnie zmienna *licznik* jest zmniejszana o jeden i porównywana z wartością 0. Jeżeli jest mniejsza od zera, domyślny przycisk jest programowo klikany, czyli wywoływana jest metoda *click* obiektu *btn*. Jeżeli wartość zmiennej jest większa od zera, zmieniany jest tekst na przycisku (wskazuje kolejną fazę odliczania) oraz ustawiany jest timer powodujący po jednej sekundzie (1000 milisekund) kolejne wywołanie funkcji *odliczaj*.

Skrypt 48. [C][E][F][O][S]

Modyfikacja paska tytułowego

Zawartość paska tytułowego okna (i karty) przeglądarki może być modyfikowana za pomocą właściwości *title* obiektu *document*. Jeżeli używamy w przeglądarce widoku *kart* (czyli każdy dokument znajduje się w osobnej karcie), co obecnie jest powszechne, tytuł pojawia się zarówno w oknie przeglądarki, jak i na nagłówku karty z daną witryną. Modyfikacja tytułu powinna być jednak przeprowadzana tylko w takich przypadkach, gdy faktycznie wymaga tego projekt witryny, a nie dla samej „zabawy”. Sposób wykonania tej operacji został zobrazowany w skrypcie z listingów 5.6 i 5.7. Na listingu 5.6 jest widoczny kod HTML.

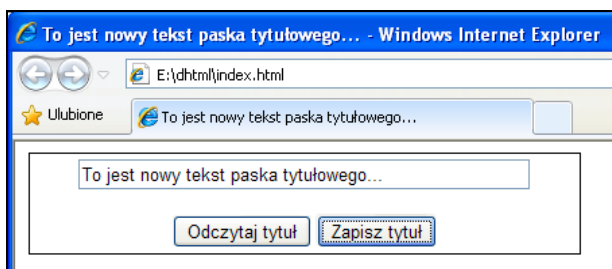
Listing 5.6. Część HTML skryptu 48.

```
<body>
  <div id="divInterfejs">
    <input type="text" id="tf1" size="50" /><br /><br />
    <input type="button" value="Odczytaj tytuł"
      onclick="odczytajTytuł('tf1')" />
    <input type="button" value="Zapisz tytuł"
      onclick="zapiszTytuł('tf1')" />
  </div>
</body>
```

W sekcji `<body>` została umieszczona warstwa `divInterfejs`, a w niej znajdują się pole tekstowe oraz dwa przyciski. Polu został nadany identyfikator, dzięki któremu możliwe będzie odwołanie do tego elementu w treści skryptu. Pierwszy przycisk służy do odczytywania zawartości paska tytułu (a dokładniej — tytułu danego dokumentu), natomiast drugi — do przypisywania mu dowolnej wartości. Strona będzie więc wyglądała tak, jak przedstawiono to na rysunku 5.3. Odczytem danych zajmie się funkcja `odczytajTytuł`, a zapisem — funkcja `zapiszTytuł`. Ich treść znajduje się na listingu 5.7.

Rysunek 5.3.

Wygląd strony
pozwalającej zmieniać
tekst paska tytułu

**Listing 5.7.** Funkcje odczytujące i zapisujące tekst paska tytułu

```
<script type="text/javascript">
  function odczytajTytuł(pole){
    var el = document.getElementById(pole);
    if(!el) return;
    el.value = document.title;
  }
  function zapiszTytuł(pole){
    var el = document.getElementById(pole);
    if(!el) return;
    document.title = el.value;
  }
</script>
```

Konstrukcja obu funkcji jest bardzo podobna. Jako argument przyjmują identyfikator pola, w którym ma być zapisany aktualny tytuł dokumentu lub z którego ma być pobrany nowy tytuł dokumentu. Za pomocą metody `getElementById` pobierają zatem odwołanie do elementu o identyfikatorze wskazywanym przez argument `pole`. Odwołanie jest zapisywane w zmiennej `el`. Jeśli ta operacja nie zakończy się sukcesem, wykonywanie kodu jest przerywane za pomocą instrukcji `return`. Jeśli odwołanie zostało pobrane, to:

- ♦ w funkcji `odczytajTytuł` wartość właściwości `title` obiektu `document` jest przypisywana właściwości `value` obiektu `el` (tytuł pojawia się w polu tekstowym);
- ♦ w funkcji `zapiszTytuł` wartość właściwości `value` obiektu `el` jest przypisywana właściwości `title` obiektu `document` (tytuł pojawia się na pasku tytułu).

Skrypt 49. Modyfikacja paska stanu

[C][E][F][O][S]

W skrypcie 48. pokazano sposób na modyfikację paska tytułu, a więc i tytułu danego dokumentu. W podobny sposób można też zmieniać napis widniejący na pasku stanu przeglądarki. Ten bardzo popularny niegdyś efekt nie jest już tak często spotykany, choć może być przydatny. Na pasku stanu należy prezentować faktycznie przydatne

informacje o statusie, adresy wskazywanych odnośników itp. Raczej nie należy natomiast umieszczać na nim żadnych animacji czy płynących tekstów, gdyż z reguły powoduje to jedynie irytację użytkownika witryny.

Aby zmodyfikować tekst paska, trzeba zmienić wartość właściwości status obiektu window. Należy przy tym pamiętać, że niektóre przeglądarki pozwalają na blokowanie zmian statusu i użytkownik witryny wcale nie musi zobaczyć efektu działania takiego skryptu. W skrypcie użyjemy interfejsu z listingu 5.6, zmieniając jedynie napisy na przyciskach na *Odczytaj status* i *Zapisz status* oraz wartości atrybutów onclick na `odczytajStatus('tf1');` i `zapiszStatus('tf1');`.

Treść funkcji `odczytajStatus` i `zapiszStatus`, wykonywanych po kliknięciach przycisków, została przedstawiona na listingu 5.8.

Listing 5.8. Funkcje zmieniające tekst na pasku stanu

```
<script type="text/javascript">
  function odczytajStatus(pole){
    var el = document.getElementById(pole);
    if(!el) return;
    el.value = window.status;
  }
  function zapiszStatus(pole){
    var el = document.getElementById(pole);
    if(!el) return;
    window.status = el.value;
  }
</script>
```

Konstrukcja funkcji jest taka sama jak w przykładzie z listingu 5.7 — operują jedynie na innej właściwości. Zamiast `document.title` jest to `window.status`. W przypadku odczytywania tej właściwości należy pamiętać, że domyślnie zawiera pusty ciąg znaków (mimo że zwykle na pasku widnieje napis typu *Zakończono*, *Gotowy* itp.). Co więcej, przypisanie właściwości pustego ciągu znaków jest przez niektóre przeglądarki (np. Firefox) traktowane jako sygnał, że została jej przekazana obsługa paska. W związku z tym, paradoksalnie, przypisanie pustego ciągu znaków może spowodować pojawienie się na pasku domyślnego napisu.

Skrypt 50. [C][E][F][O][S]

Pasek postępu (zdarzeniowy)

Niekiedy na stronie trzeba wykonać operacje, które zajmują więcej czasu. Można co prawda stosować wtedy wyświetlanie klepsydry i (lub) blokowanie elementów interfejsu, ale dobrze byłoby poinformować użytkownika o stanie zaawansowania danych operacji. Dlaczego więc nie użyć klasycznego paska postępu? Można go zbudować bardzo prosto, za pomocą dwóch zwykłych warstw i odpowiedniego skryptu sterującego ich zachowaniem.

Aby zademonstrować, jak działa taki pasek, na stronie zostaną umieszczone dodatkowo dwa przyciski pozwalające na jego zwiększanie i zmniejszanie. Witryna będzie zatem miała postać przedstawioną na rysunku 5.4. Kliknięcie pierwszego przycisku spowoduje zmniejszenie paska o 10 proc., a kliknięcie drugiego — zwiększenie o 10 proc. W sekcji `<body>` należy umieścić kod przedstawiony na listingu 5.9.

Rysunek 5.4.
*Pasek postępu
sterowany przyciskami*



Listing 5.9. *Sekcja `<body>` z paskiem postępu*

```
<body>
  <div id="divInterfejs" style="margin-bottom:4px;">
    <input type="button" value="Zmniejsz o 1/10"
      onclick="ustawPasek(bieżącyKrok - 10);" />
    <input type="button" value="Zwiększ o 1/10"
      onclick="ustawPasek(bieżącyKrok + 10);" />
  </div>
  <div id="divPasekZew">
    <div id="divPasekWew">
    </div>
  </div>
</body>
```

Pierwsza warstwa (`divInterfejs`) zawiera interfejs sterujący paskiem postępu, czyli dwa przyciski. Kliknięcie dowolnego przycisku wywołuje funkcję `ustawPasek`. Argumentem funkcji jest określona procentowo wartość, na jaką ma zostać zmienione aktualne wskazanie. Ponieważ każde kliknięcie ma zwiększyć lub zmniejszyć wskazanie o 10 proc. (1/10), stosowane są określenia `bieżącyKrok + 10` i `bieżącyKrok - 10`. Używana jest zmienna `bieżącyKrok`, która zostanie zdefiniowana w skrypcie.

Sam pasek składa się z dwóch warstw: zewnętrznej (`divPasekZew`) i wewnętrznej (`divPasekWew`). Aby był prawidłowo wyświetlany, niezbędne jest użycie odpowiednich stylów CSS. Warstwa zewnętrzna powinna zawierać obramowanie, a wewnętrzna mieć zdefiniowany kolor tła. Przykładowe style zostały przedstawione na listingu 5.10.

Listing 5.10. *Style CSS dla paska postępu*

```
<style type="text/css">
  #divPasekZew{
    border:1px solid black;
    width:400px;
    height:20px;
    padding:0px;
  }
  #divPasekWew{
```

```
border:none;
width:0%;
height:100%;
margin:0px;
background-color:#D0D0D0;
}
</style>
```

Rozmiary i obramowanie paska określone są w regule z selektorem `#divPasekZew` dotyczącej warstwy o identyfikatorze `divPasekZew`. Ramka będzie pełna, czarna, o grubości jednego piksela (`border:1px solid black;`). Pasek będzie miał długość 400 pikseli (`width:400px;`) i wysokość 20 pikseli (`height:20px;`). Wypełnienie (margines wewnętrzny) zostało określone na 0 pikseli (`padding:0px;`). W połączeniu z definicją `margin:0px` w selektorze `#divPasekWew` spowoduje to, że między warstwami tworzącymi pasek nie będzie żadnych odstępów.

W wewnętrznej warstwie paska (`divPasekWew`) nie będzie żadnego obramowania (`border:none;`). Jej wysokość została określona na 100 proc. (`height:100%;`), dzięki czemu będzie wypełniała całkowicie obszar paska w pionie, natomiast szerokość — na 0 proc. (`width:0%;`), co spowoduje, że początkowo stopień zaawansowania paska będzie ustawiony na 0 (warstwa `divPasekWew` po prostu nie będzie widoczna). Ostatnia definicja — `background-color:#D0D0D0;` — dotyczy koloru tła warstwy, czyli koloru paska. Może on być dowolny. W przykładzie został użyty kolor szary.

Skrypt sterujący paskiem został przedstawiony na listingu 5.11.

Listing 5.11. *Skrypt sterujący paskiem postępu*

```
<script type="text/javascript">
var bieżącyKrok = 0;
function ustawPasek(wartość)
{
    wartość = parseInt(wartość);
    if(isNaN(wartość))
        wartość = 0;
    if(wartość < 0) wartość = 0;
    if(wartość > 100) wartość = 100;
    bieżącyKrok = wartość;
    var wew = document.getElementById('divPasekWew');
    if(wew){
        wew.style.width = wartość + "%";
    }
}
</script>
```

Globalna zmienna `bieżącyKrok` przechowuje aktualny stan paska. Zmianą bieżącego ustawienia zajmuje się natomiast funkcja `ustawPasek`. Jako argument otrzymuje ona podaną w procentach wartość, która ma być ustawiona. Najpierw za pomocą metody `parseInt` argument `wartość` jest przetwarzany na liczbę całkowitą. Następuje też sprawdzenie, czy ta operacja zakończyła się sukcesem — badane jest, czy wartość zwrócona przez `parseInt` jest nieliczbą (`NaN`). Jeśli tak jest, czyli argument był nieprawidłowy,

parametr wartość otrzymuje wartość 0. Jeżeli argument był prawidłowy pod względem formalnym, za pomocą instrukcji warunkowych `if` sprawdzane jest, czy nie przekracza dopuszczalnego zakresu. Nie może być bowiem mniejszy od 0 ani większy od 100. Gdy wartość zapisana w `wartość` jest mniejsza od 0, jest zmieniana na 0, a gdy jest większa od 100, jest zmieniana na 100. Dzięki temu pasek nie wykroczy poza dopuszczane granice.

Po weryfikacji poprawności danych wartość przekazanego funkcji argumentu jest zapisywana w zmiennej globalnej `bieżącyKrok`. Dzięki temu stan paska jest zapamiętywany. Następnie za pomocą metody `getElementById` pobierane jest odwołanie do warstwy `divPasekWew`, a gdy ta operacja powiedzie się, zmieniana jest szerokość warstwy. Tym samym pasek przesunie się w prawo lub w lewo. Modyfikacja szerokości odbywa się przez przypisanie nowej wartości właściwości `width` obiektu `style`. Ponieważ szerokość ma być określana w procentach, do wartości wskazywanej przez parametr `wartość` dodawany jest znak `%`.

Oczywiście w praktyce nie steruje się paskiem postępu za pomocą przycisków, ale używa wywołań funkcji `ustawPasek` wewnątrz kodu wykonującego złożone lub wymagające czasu operacje, dzięki czemu użytkownik witryny jest informowany o postępach.

Skrypt 51. Pasek postępu (czasowy)

[C][E][F][O][S]

W skrypcie 50. został przedstawiony pasek postępu, który mógł być sterowany dowolnymi zdarzeniami. Czasem jednak chcemy tylko opóźnić jakiś proces o zadany czas lub też czas przetwarzania danych jest znany. Wtedy może nam się przydać pasek czasowy, czyli taki, który sam zmienia swój stan w określonym czasie. Tak właśnie będzie działał skrypt 51. Na stronie pojawi się przycisk i pasek postępu. Kliknięcie przycisku spowoduje animację paska, a po jej zakończeniu zostanie wykonana pewna funkcja. Jej zadaniem będzie wyświetlenie ukrytej początkowo warstwy. Kod będzie przy tym uniwersalny. To, jaka funkcja zostanie wywołana po zakończeniu odliczania, będzie podawane przy starcie całej procedury.

Treść sekcji `<body>` została przedstawiona na listingu 5.12.

Listing 5.12. Sekcja `<body>` skryptu 51.

```
<body>
  <div id="divInterfejs" style="margin-bottom:4px;">
    <input type="button" value="Rozpocznij"
      onclick="rozpocznijOdliczanie(100,10,wyswietl);" />
  </div>
  <div id="divPasekZew">
    <div id="divPasekWew">
    </div>
  </div>
```

```

<div id="dataDiv" style="display:none;margin-top:4px;">
  Dane zostały przetworzone.
</div>
</body>

```

W warstwie `div` Interfejs został tym razem zdefiniowany jeden przycisk, którego kliknięcie będzie powodowało wywołanie funkcji `rozpocznijOdliczanie`. Funkcja przyjmuje trzy argumenty: całkowitą liczbę przesunięć paska, czas pomiędzy kolejnymi przesunięciami i nazwę funkcji, która zostanie wywołana po zakończeniu odliczania. Wywołanie:

```
rozpocznijOdliczanie(100,10,wyświetl);
```

oznacza zatem: przesuwaj pasek 100 razy (czyli za każdym razem o jeden punkt procentowy) co 10 milisekund, a po ostatnim przesunięciu wywołaj funkcję `wyświetl` (oczywiście taka funkcja musi być zdefiniowana w kodzie skryptu).

Sposób definicji paska pozostał taki sam jak w przykładzie 50. Składa się on z dwóch warstw o identyfikatorach `divPasekZew` i `divPasekWew`. Aby pasek był prawidłowo wyświetlony, należy też zdefiniować odpowiednie style CSS. Z powodzeniem można jednak użyć definicji z listingu 5.10. Za warstwami określającymi pasek znajduje się jeszcze jedna (`dataDiv`). Początkowo jest ukryta dzięki użyciu właściwości CSS `display` o wartości `none`.

Skrypt sterujący czasowym paskiem postępu został przedstawiony na listingu 5.13.

Listing 5.13. *Skrypt sterujący czasowym paskiem postępu*

```

<script type="text/javascript">
  var ileKroków;
  var bieżącyKrok;
  var timeout;
  function ustawPasek(wartość)
  {
    // tutaj treść funkcji ustawPasek
  }
  function odliczaj(krok, funkcja)
  {
    ustawPasek(bieżącyKrok + krok);
    if(--ileKroków <= 0){
      if(typeof funkcja == 'function')
        funkcja();
    }
    else{
      setTimeout("odliczaj('"+krok+"','"+funkcja+"');", timeout);
    }
  }
  function rozpocznijOdliczanie(lKroków, czas, funkcja)
  {
    if(lKroków < 1) lKroków = 1;
    if(lKroków > 100) lKroków = 100;
    krok = Math.ceil(100 / lKroków);
    ileKroków = lKroków;
    timeout = czas;
  }

```



```
    bieżącyKrok = 0;
    setTimeout("odliczaj('+krok+', '+funkcja+')", timeout);
  }
  function wyświetl()
  {
    var div = document.getElementById('dataDiv');
    if(div) div.style.display = "block";
  }
</script>
```

Na początku zostały zdefiniowane trzy zmienne globalne:

- ◆ `ileKroków` — określa liczbę kroków, w których ma się przesunąć pasek;
- ◆ `bieżącyKrok` — określa bieżący stan paska (ma takie samo znaczenie jak w skrypcie 50.);
- ◆ `timeout` — określa w milisekundach czas pomiędzy kolejnymi przesunięciami paska.

Funkcja `ustawPasek` zmienia stan paska i ma taką samą postać jak w skrypcie 50., jej treść nie została więc uwzględniona na listingu.

Funkcja `rozpocznijOdliczanie` ustala wartości początkowe i sterujące skryptem. Przyjmuje trzy argumenty:

- ◆ `1Kroków` — liczba kroków, w których ma zostać przewinięty pasek postępu;
- ◆ `czas` — czas pomiędzy kolejnymi przesunięciami (podany w milisekundach), czyli między kolejnymi wywołaniami timera;
- ◆ `funkcja` — funkcja, która zostanie wykonana po zakończeniu odliczania.

Najpierw sprawdza, czy argument `1Kroków` ma wartość mieszczącą się we właściwych granicach. Krok minimalny to 1 proc., a maksymalny — 100 proc. Jeżeli wartość przekracza wyznaczone granice, jest odpowiednio modyfikowana. Następnie obliczane jest, o ile punktów procentowych należy zwiększyć pasek postępu w każdym kroku. Ta wartość wynika z prostego wzoru:

$$\frac{100}{\text{liczba kroków}}$$

i jest (po zaokrągleniu w dół za pomocą metody `ceil` obiektu `Math`) zapisywana w zmiennej `krok`.

Po dokonaniu opisanych wyliczeń ustalane są wartości zmiennych globalnych `ileKroków`, `timeout` i `bieżącyKrok` oraz rozpoczynany jest proces odliczania. Odbywa się to przez wywołanie metody `setInterval`. Pierwszy jej argument wskazuje instrukcję, jaka ma być wykonana po czasie określonym przez drugi argument. Tą instrukcją jest wywołanie funkcji `odliczaj`.

Zadaniem funkcji `odliczaj` jest ustawienie paska postępu w pozycji wskazywanej przez pierwszy argument wywołania i kontynuowanie tej operacji tak długo, aż wykonane

zostaną wszystkie kroki (czyli gdy liczba wywołań osiągnie wartość zapisaną w zmiennej `ileKroków`). Dodatkowo, gdy zostanie wykonany ostatni krok, ma zostać wywołana funkcja przekazana jako drugi argument.

Najpierw jest zatem wywoływana znana ze skryptu 50. funkcja `ustawPasek`, a przekazywana jej wartość wynika z dodawania bieżącyKrok + krok. Następnie zmienna `ileKroków` jest zmniejszana o 1 i jest badane, czy uzyskana w ten sposób wartość osiągnęła 0 (`if(--ileKroków <= 0){}`). Jeśli tak, oznacza to koniec odliczania. Wtedy sprawdzane jest, czy argument funkcja wskazuje prawidłową funkcję JavaScriptu (jest w tym celu używany operator `typeof` (`if(typeof funkcja == 'function')`). Gdy jest funkcją, jest ona wywoływana (`funkcja()`). W sytuacji gdy wartość zmiennej `ileKroków` nie osiągnęła 0, ponownie jest ustawiany timer powodujący kolejne wywołanie funkcji odliczaj:

```
setTimeout("odliczaj(++krok+','+funkcja+');", timeout);
```

Na końcu kodu została umieszczona ostatnia z funkcji — `wyświetl`. Ponieważ została przekazana jako trzeci argument w funkcji `rozpocznijOdliczanie` (a tym samym również jako drugi argument funkcji `odliczaj`), zostanie wywołana po zakończeniu odliczania. Jej zadaniem jest uaktywnienie warstwy `dataDiv`. Pobiera zatem odwołanie do niej, a następnie zmienia styl wyświetlania na blokowy. Odbyna się to przez przypisanie wartości `block` właściwości `display` obiektu `style`.

Skrypt 52.

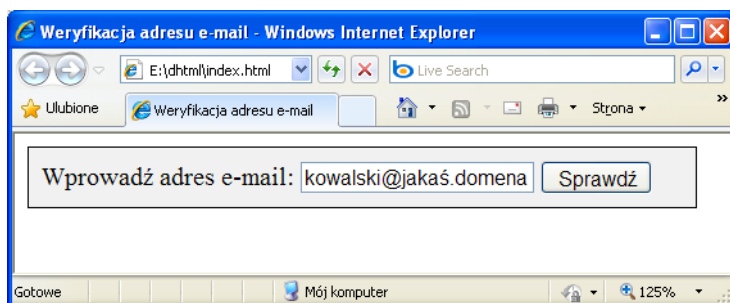
[C][E][F][O][S]

Weryfikacja adresu e-mail

Wykorzystując JavaScript, można w prosty sposób napisać funkcję, której zadaniem będzie weryfikacja podanego przez użytkownika adresu e-mail. Często się przecież zdarza, że ktoś się myli lub też wpisuje zupełnie nieprawdopodobne dane. Nie uda się co prawda sprawdzić, czy podany e-mail faktycznie istnieje, ale można zweryfikować jego formalną poprawność. Na stronie zostanie umieszczone pole tekstowe oraz przycisk, tak jak zostało to zaprezentowane na rysunku 5.5. Po kliknięciu przycisku zostanie wykonana funkcja, która wyświetli okno dialogowe z informacją, czy wprowadzone dane składają się na poprawny adres pocztowy, czy też nie. Kod HTML został przedstawiony na listingu 5.14.

Rysunek 5.5.

Strona pozwalająca zweryfikować poprawność adresu e-mail



Listing 5.14. Kod HTML skryptu 52.

```
<body>
  <div>
    Wprowadź adres e-mail:
    <input type="text" id="tfMail" />
    <input type="button" value="Sprawdź" onclick="sprawdźEmail();" />
  </div>
</body>
```

W sekcji `<body>` znalazła się warstwa zdefiniowana za pomocą znacznika `<div>`, a w niej pole tekstowe oraz przycisk. Polu został nadany identyfikator `tfMail`, z kolei przycisk otrzymał atrybut `onclick` zawierający wywołanie funkcji `sprawdźEmail`.

Aby sprawdzić poprawność adresu, trzeba znać obowiązujące w nim zasady. W praktyce korzysta się z uproszczonych reguł¹, które przedstawiają się następująco:

- ◆ w adresie mogą występować jedynie znaki cyfr, liter oraz myślnik, podkreślenie, kropka i `@`;
- ◆ w adresie musi występować dokładnie jeden znak `@` oddzielający część lokalną od domenowej;
- ◆ część lokalna musi zawierać co najmniej jeden znak;
- ◆ część lokalna i część domenowa nie mogą się zaczynać ani kończyć znakiem kropki;
- ◆ część domenowa musi składać się co najmniej z dwóch członów;
- ◆ ostatni człon nazwy domenowej musi zawierać od dwóch do sześciu liter (choć najczęściej dwie, trzy lub cztery).

Badanie, czy wprowadzony przez użytkownika adres spełnia te warunki, można przeprowadzić, wykorzystując serię instrukcji warunkowych, jednak lepszym i zdecydowanie prostszym w realizacji sposobem jest wykorzystanie wyrażeń regularnych. Wystarczy zbudować wyrażenie opisujące wymienione zasady i użyć metody `match`. Wyrażenie będzie miało postać:

```
/^[a-zA-Z0-9_-]+(\.[a-zA-Z0-9_-]+)*@([a-zA-Z0-9_-]+\.)+([a-zA-Z]{2,4})$/i;
```

Zawarty na jego końcu znak `i` oznacza, że analiza ma być przeprowadzona bez uwzględniania wielkości liter.

Kod skryptu weryfikującego został przedstawiony na listingu 5.15.

¹ Teoretycznie adres e-mail może być bardziej rozbudowany, niż wynika to z podanego opisu. Część lokalna może np. zawierać takie znaki jak tylda czy wykrzyknik, może być też ujęta w znaki podwójnego cudzysłowu. Z kolei część domenowa może być np. adresem IP ujętym w nawias kwadratowy. W praktyce jest to jednak prawie niespotykane. Wiele serwerów pocztowych nie obsługuje też tak rozbudowanych adresów (mimo ich formalnej poprawności).

Listing 5.15. *Skrypt weryfikujący poprawność adresu*

```
<script type="text/javascript">
  function sprawdźEmail()
  {
    var tfMail = document.getElementById('tfMail');
    if(!tfMail || !tfMail.value) return;
    var re = /^[a-zA-Z0-9_-]+(\.[a-zA-Z0-9_-]+)*@([a-zA-Z0-9_-]+\.)+([a-zA-Z]{2,4})$/i;
    if(tfMail.value.match(re) == null)
      alert('Ten adres jest nieprawidłowy.');
```

```
    else
      alert('Ten adres jest prawidłowy.');
```

```
  }
</script>
```

W funkcji `sprawdźEmail` najpierw pobierane jest i zapisywane w zmiennej `tfMail` odwołanie do pola tekstowego oraz sprawdzane, czy operacja ta zakończyła się powodzeniem (czyli czy istnieje element strony WWW o identyfikatorze `tfMail` oraz ma właściwość `value`, której wartość nie jest pustym ciągiem znaków).

Jeżeli dane są prawidłowe, powstaje wyrażenie regularne (o podanej wyżej postaci) i jest zapisywane w zmiennej `re`. Następnie w stosunku do ciągu zapisanego w `tfMail.value` (wartość wprowadzona do pola tekstowego) wykonywana jest metoda `match`, której w postaci argumentu przekazywane jest wyrażenie zapisane w `re`.

Jeśli `match` zwróci wartość `null`, oznacza to, że adres e-mail nie spełnia warunków opisanych wyrażeniem regularnym, a zatem jest nieprawidłowy. W przeciwnym wypadku, czyli kiedy `match` zwróci wartość różną od `null`, adres spełnia warunki opisane wyrażeniem regularnym, a zatem jest prawidłowy.

Informacja o poprawności bądź niepoprawności adresu jest wyświetlana w nowym oknie dialogowym dzięki wywołaniu metody `alert`.

Rozdział 6.

Pływające napisy

Na stronie WWW najważniejsza jest treść, nie można jednak zapominać o sposobie prezentacji witryny. Czasem warto do niej dodać sympatyczne gadżety, które uatrakcyjnią przekaz lub też pozwolą zwrócić uwagę na wybrane fragmenty. W rozdziale 6. zostały przedstawione różnego rodzaju pływające napisy, czyli animacje tekstu — zarówno te stosunkowo proste, gdy napis porusza się w zwykłym polu tekstowym, jak i bardziej złożone, gdy litery zmieniają położenie po zadanej ścieżce, np. sinusoidzie. Pokazany zostanie również efekt określany jako kolor pływący po statycznym tekście.

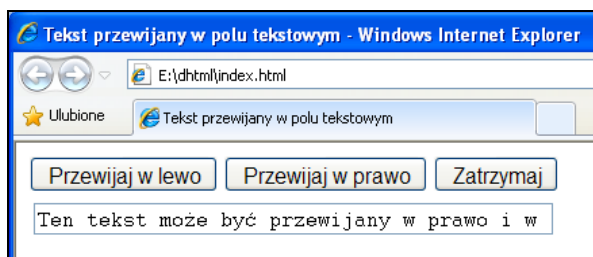
Skrypt 53.

[C][E][F][O][S]

Napis pływający w polu tekstowym

Napis znajdujący się w polu tekstowym może być animowany, czyli przewijany w prawo lub w lewo. Taki efekt nazywamy skrolowaniem (od ang. *scroll*). Wbrew pozorom uzyskuje się go w stosunkowo prosty sposób — przez odpowiednią modyfikację tekstu stanowiącego zawartość pola. Aby dobrze zademonstrować sposób działania skryptu, na stronie zostanie umieszczony interfejs pozwalający na włączanie i wyłączanie przewijania, a także zmianę kierunku przesuwu. Strona będzie wyglądała tak jak na rysunku 6.1, a kod HTML przyjmie postać zaprezentowaną na listingu 6.1.

Rysunek 6.1.
*Interfejs sterujący
przewijaniem tekstu*



Listing 6.1. Kod sekcji `<body>` skryptu 53.

```
<body>
  <div id="divInterfejs">
    <input type="button" value="Przewijaj w lewo"
      onclick="startScroll('tfText', 'lewo', 100);"/>
    <input type="button" value="Przewijaj w prawo"
      onclick="startScroll('tfText', 'prawo', 100);"/>
    <input type="button" value="Zatrzymaj"
      onclick="stopScroll();"/>
  </div>
  <div id="divDane">
    <input type="text" id="tfText"
      style="font-family:courier" size="45"
      value="Ten tekst może być przewijany w prawo i w lewo."
    />
  </div>
</body>
```

Pierwsza warstwa zawiera elementy interfejsu sterującego pracą skryptu, czyli trzy przyciski utworzone za pomocą znaczników `<input>` z atrybutem `type` ustawionym na `button`. Kliknięcie pierwszego przycisku będzie uruchamiało przewijanie tekstu w lewo, a drugiego — przewijanie w prawo. Będzie to realizowane przez wywołanie funkcji `startScroll`. Trzeci przycisk będzie odpowiadał za zatrzymywanie przewijania, a czynność ta będzie wykonywana za pomocą funkcji `stopScroll`. Wymienione funkcje zostały przypisane jako procedury obsługi zdarzenia `click`.

Na drugiej warstwie zostało umieszczone zwyczajne pole tekstowe o identyfikatorze `tfText`. Zawiera przykładowy tekst określony przez atrybut `value`. Użytkownik strony będzie mógł jednak wprowadzić dowolny inny napis, który też będzie przewijany.

Treść skryptu obsługującego przewijanie została przedstawiona na listingu 6.2.

Listing 6.2. Funkcje odpowiadające za przewijanie tekstu

```
<script type="text/javascript">
  var timeout = 0;
  var timerId = null;
  var kierunek = 'lewo';
  var pole = null;
  function startScroll(nazwaPola, lewoprawo, czas)
  {
    if(timerId != null && kierunek != lewoprawo)
      stopScroll();

    if(timerId != null) return;

    pole = document.getElementById(nazwaPola);
    if(!pole) return;

    timeout = czas;
    kierunek = lewoprawo;
    timerId = setInterval("scroll()", timeout);
```

```
}
function stopScroll()
{
    if(timerId != null){
        clearInterval(timerId);
        timerId = null;
    }
}
function scroll()
{
    if(!pole || !pole.value){
        stopScroll();
        return;
    }
    var text = pole.value;
    if(kierunek == 'lewo')
        pole.value = text.substr(1, text.length - 1) + text.substr(0,1);
    else
        pole.value = text.substr(text.length - 2, text.length - 1) +
            text.substr(0, text.length - 2);
}
</script>
```

Na początku kodu znajdują się definicje czterech zmiennych globalnych sterujących pracą skryptu:

- ◆ `timeout` — określa w milisekundach czas pomiędzy kolejnymi wywołaniami funkcji przesuwającej test, a tym samym szybkość animacji;
- ◆ `timerId` — identyfikator timera uruchomionego za pomocą metody `setInterval`;
- ◆ `kierunek` — określa kierunek przesuwu tekstu, prawo lub lewo;
- ◆ `pole` — określa pole tekstowe, w którym ma być przesuwany tekst.

Funkcja `startScroll` jest wywoływana po kliknięciu jednego z przycisków uruchamiających animację. Przyjmuje trzy argumenty:

- ◆ `nazwaPola` — identyfikator pola tekstowego;
- ◆ `lewoprawo` — ciąg znaków określający kierunek;
- ◆ `czas` — liczba milisekund między kolejnymi fazami animacji.

Działa w następujący sposób:

- ◆ jeśli animacja jest wyłączona, rozpoczyna przesuw w prawo lub w lewo (zgodnie z dyspozycją użytkownika);
- ◆ jeśli animacja trwa i został kliknięty przycisk wskazujący ten sam kierunek, nie robi nic;
- ◆ jeśli animacja trwa i został kliknięty przycisk wskazujący przeciwny kierunek, zmienia kierunek przesuwu.

Działanie rozpoczyna się od sprawdzenia, czy mamy do czynienia z ostatnim z wymienionych zdarzeń. Odpowiada za to instrukcja warunkowa:

```
if(timerId != null && kierunek != lewoprawo)
```

Jeżeli warunek jest prawdziwy, czyli gdy `timerId` jest różne od `null` (trwa animacja) i gdy `kierunek` jest różny od `lewoprawo`, animacja jest zatrzymywana przez wywołanie funkcji `stopScroll`. Kolejna instrukcja bada stan zmiennej `timerId`. Jeżeli jest on różny od `null`, wykonywanie funkcji jest kończone, oznacza to bowiem, że trwa animacja i nie należy zaczynać kolejnej.

Dalsze instrukcje funkcji `startScroll` ustawiają stan wymienionych wyżej zmiennych globalnych, których wartości zostały przekazane w postaci argumentów. Odwołanie do pola tekstowego o identyfikatorze wskazanym przez argument `nazwaPola` jest pobierane za pomocą metody `getElementById` i zapisywane w zmiennej `pole`.

Animacja jest rozpoczynana dzięki instrukcji:

```
timerId = setInterval("scroll()", timeout);
```

która powoduje cykliczne wywoływanie funkcji `scroll` co liczbę milisekund wskazywaną przez `timeout`.

Funkcja `scroll` ma spowodować przesuwanie w prawo bądź w lewo tekstu znajdującego się w polu tekstowym wskazywanym przez globalną zmienną `pole`. Po sprawdzeniu, że ta zmienna istnieje i że zawiera obiekt z właściwością `value`, następuje wykonanie pojedynczego kroku animacji. Przesuw o jeden znak w lewo odbywa się przez usunięcie pierwszego znaku tekstu (`text.substr(1, text.length - 1)`) i dopisanie go (`text.substr(0,1)`) na końcu tekstu:

```
pole.value = text.substr(1, text.length - 1) + text.substr(0,1);
```

Przesuw w prawo to czynność dokładnie odwrotna — usunięcie ostatniego znaku tekstu (`text.substr(0, text.length - 2);`) i dopisanie go (`text.substr(text.length - 2, text.length - 1)`) na początku:

```
pole.value = text.substr(text.length - 2, text.length - 1) +  
text.substr(0, text.length - 2);
```

Ostatnia z funkcji — `stopScroll` — odpowiada za zatrzymanie przesuwu. Czynność ta polega na wywołaniu metody `clearInterval` (w argumencie jest przekazywany identyfikator timera). Dodatkowo zmiennej `timerId` jest przypisywana wartość `null` wskazująca, że animacja nie jest aktywna. Opisane czynności są wykonywane tylko wtedy, gdy w momencie wywołania funkcji odbywało się przesuwanie tekstu, czyli gdy wartość `timerId` była różna od `null`.

Skrypt 54. Napis pływający w dowolnym miejscu witryny

[C][E][F][O][S]

Za pomocą skryptu 54. zostanie pokazane, jak można umieścić w dowolnym miejscu strony pływający napis. Zostanie zachowany interfejs sterujący z przykładu 53., jednak animowany tekst będzie umieszczany na warstwie, która z kolei będzie się mogła znaleźć w dowolnie wybranym położeniu. Przesuwanie będzie płynne, ustawienie tekstu będzie zmieniane o zadaną liczbę pikseli. Część HTML skryptu przyjmie postać zaprezentowaną na listingu 6.3.

Listing 6.3. Część HTML skryptu 54.

```
<body>
  <div id="divInterfejs">
    <input type="button" value="Przewijaj w lewo"
      onclick="startScroll('divDane', 'divTekst', 'lewo', 10, 1);"/>
    <input type="button" value="Przewijaj w prawo"
      onclick="startScroll('divDane', 'divTekst', 'prawo', 10, 1);"/>
    <input type="button" value="Zatrzymaj"
      onclick="stopScroll();"/>
  </div>
  <div id="divDane">
    <div id="divTekst" style="width:100px;position:relative;">
      To jest tekst.
    </div>
  </div>
</body>
```

Zawartość warstwy `divInterfejs` pozostała taka sama jak w przykładzie z listingu 6.1. To trzy przyciski służące do rozpoczynania i kończenia animacji. Inaczej wyglądają tylko wywołania funkcji `startScroll`, gdyż będzie ona miała inną postać i w związku z tym przyjmuje inne argumenty. Ponieważ przesuwany tekst nie może być swobodnym elementem witryny, należy go umieścić w warstwie pomocniczej (w przykładzie jest to warstwa `divTekst`), a dopiero tę warstwę w elemencie witryny, na którym ma „pływać” napis (w przykładzie jest to warstwa `divDane`). A zatem animowany tekst pojawi się na warstwie `dataDiv`, którą można umieścić w dowolnym miejscu strony.

Animacja będzie polegała po prostu na przesuwaniu warstwy `divTekst` po warstwie `divDane`. Pierwsza część skryptu została przedstawiona na listingu 6.4.

Listing 6.4. Pierwsza część skryptu obsługująca przesuwanie tekstu

```
<script type="text/javascript">
  var timeout = 0;
  var timerId = null;
  var kierunek = 'lewo';
```

```
var divGdzie = null;
var divTekst = null;
var krokpx;
function startScroll(divGdzieId, divTekstId, lewoprawo, czas, krok)
{
    if(timerId != null && kierunek != lewoprawo){
        kierunek = lewoprawo;
        return;
    }
    if(timerId != null) return;

    divGdzie = document.getElementById(divGdzieId);
    divTekst = document.getElementById(divTekstId);
    if(!divGdzie || !divTekst) return;

    kierunek = lewoprawo;
    timeout = isNaN(timeout = parseInt(czas)) ? 10: timeout;
    krokpx = isNaN(krokpx = parseInt(krok)) ? 1: krokpx;
    timerId = setInterval("scroll()", timeout);
}
function stopScroll()
{
    if(timerId != null){
        clearInterval(timerId);
        timerId = null;
    }
}
// dalsza część skryptu na listingu 6.5
```

Na początku kodu znajdują się zmienne globalne sterujące pracą skryptu i pozwalające na uniknięcie ciągłego pobierania danych i przekazywania argumentów w kolejnych wywołaniach funkcji wykonującej animację. Są to:

- ♦ `timeout` — określa w milisekundach czas pomiędzy kolejnymi fazami animacji (wywołaniami funkcji `scroll`);
- ♦ `timerId` — określa identyfikator timera (uruchomionego przez wywołanie metody `setInterval`);
- ♦ `kierunek` — określa kierunek przesuwu tekstu;
- ♦ `divGdzie` — zawiera odwołanie do warstwy, po której będzie przesuwany tekst;
- ♦ `divTekst` — zawiera odwołanie do warstwy, na której znajduje się przesuwany tekst;
- ♦ `krokpx` — określa liczbę pikseli, o jaką ma zostać przesunięty tekst w jednym kroku animacji.

Funkcja `startScroll` jest wywoływana w odpowiedzi na kliknięcie jednego z przycisków rozpoczynających animację (bądź zmieniających kierunek animacji). Jej zadaniem jest ustawienie właściwych parametrów, a więc ustalenie stanów wymienionych wyżej zmiennych globalnych, sprawdzenie ich poprawności i uruchomienie procesu przesuwania tekstu. Funkcja przyjmuje pięć argumentów:

- ◆ `divGdzieId` — identyfikator warstwy, po której będzie przesuwany tekst;
- ◆ `divTekstId` — identyfikator warstwy, na której znajduje się tekst;
- ◆ `lewoprawo` — ciąg znaków określający kierunek ruchu (lewo, prawo);
- ◆ `czas` — wartość parametru `timeout`, czyli szybkość animacji;
- ◆ `krok` — wartość parametru `krokp`, czyli liczba pikseli, o jaką będzie przesuwany tekst.

Pierwsza instrukcja warunkowa ma wykryć, czy mamy do czynienia ze zmianą kierunku animacji. Będzie tak, gdy zmienna `timerId` będzie różna od `null` oraz gdy wartość argumentu `prawolewo` będzie różna od bieżącej wartości zmiennej `kierunek`. Gdy oba warunki są spełnione jednocześnie, kierunek jest zmieniany przez modyfikację zmiennej `kierunek`, a wykonywanie funkcji jest przerywane. Warto porównać to rozwiązanie z zaproponowanym w skrypcie 53. W obu przypadkach rozwiązywany jest ten sam problem, ale w inny sposób.

W oparciu o argumenty `divGdzieId` i `divTekstId` pobierane są odwołania do warstw z tekstem. Gdyby się to nie udało, wykonywanie kodu zostanie przerwane, a tym samym animacja nie zostanie uruchomiona. Pozostałe argumenty są przypisywane odpowiednim zmiennym globalnym. W przypadku parametrów `czas` i `krok` wykonywana jest dodatkowa weryfikacja, co pozwala na stwierdzenie, czy zawierają wartości całkowite. Jeśli nie zawierają, zmiennym `timeout` i `krokp` przypisywane są wartości domyślne (odpowiednio: 10 i 1). Używana jest w tym celu złożona instrukcja przypisania z wyrażeniem warunkowym. Zapis:

```
krokp = isNaN(krokp = parseInt(krok)) ? 1: krokp;
```

należy rozumieć następująco: przypisz zmiennej `krokp` wynik działania funkcji `parseInt(krok)`; za pomocą funkcji `isNaN` sprawdź, czy wartość ta jest różna od `NaN`, czyli czy jest liczbą (`isNaN(krokp = parseInt(krok))`); jeśli nie jest, przypisz zmiennej `krokp` wartość 1, jeśli jest, pozostaw niezmienną wartość `krokp`¹.

Po wykonaniu opisanych czynności za pomocą metody `setInterval` ustawiany jest timer, który będzie powodował cykliczne wywołania funkcji `scroll`. Jego identyfikator (wartość zwrócona przez `setInterval`) jest przypisywany zmiennej `timerId`.

Za funkcją `startScroll` znajduje się funkcja `stopScroll`. Wykonuje zadanie przeciwne, czyli zatrzymuje animację. Odbyna się to przez wywołanie funkcji `clearInterval`, której w postaci argumentu jest przekazywany identyfikator bieżącego timera zapisany w zmiennej `timerId`. Wartość zmiennej jest też zerowana (jest jej przypisywana wartość `null`). Operacje te są wykonywane tylko wtedy, gdy wartość `timerId` jest różna od `null` (czyli gdy animacja jest aktywna).

Dalsza część skryptu została przedstawiona na listingu 6.5.

¹ Formalnie rzecz ujmując, wykonywana jest wtedy instrukcja `krokp = krokp`, co ma identyczne znaczenie.

Listing 6.5. *Druga część skryptu obsługująca przesuwanie tekstu*

```
// dalsza część skryptu z listingu 6.4
function scroll()
{
    if(!divGdzie || !divTekst){
        stopScroll();
        return;
    }
    var gdzieW = parseInt(divGdzie.offsetWidth);
    var tekstW = parseInt(divTekst.offsetWidth);

    var tekstL = parseInt(divTekst.style.left);
    if(isNaN(tekstL)) tekstL = 0;

    if(kierunek == 'lewo'){
        tekstL -= krokpx;
        if(tekstL < -tekstW) tekstL = gdzieW;
        divTekst.style.left = tekstL + "px";
    }
    else{
        tekstL += krokpx;
        if(tekstL > gdzieW) tekstL = -tekstW;
        divTekst.style.left = tekstL + "px";
    }
}
</script>
```

Funkcja `scroll`, której pojedyncze wywołanie odpowiada jednemu krokowi animacji, najpierw bada, czy istnieją obiekty `divGdzie` i `divTekst` odzwierciedlające warstwy związane z przesuwaniem tekstu. Jeśli ich nie ma, cały proces jest zatrzymywany przez wywołanie funkcji `stopScroll`.

Animacja polega na przesuwananiu warstwy `divTekst` po warstwie `divGdzie`. W tym celu należy modyfikować właściwość `left` (określającą położenie w poziomie od lewej strony) obiektu `style`. Trzeba jednak wziąć pod uwagę poziome rozmiary obu warstw. Rozmiar w pikselach można uzyskać za pomocą właściwości `offsetWidth`. Tworzone są trzy zmienne pomocnicze:

- ♦ `gdzieW` — szerokość warstwy, po której jest przesuwany tekst;
- ♦ `tekstW` — szerokość warstwy zawierającej tekst;
- ♦ `tekstL` — współrzędna x lewego brzegu warstwy zawierającej tekst.

Położenie warstwy (wartość zmiennej `tekstL`) jest określane przez przetworzenie na wartość całkowitą ciągu zapisanego we właściwości `divTekst.style.left`. Jest do tego używana metoda `parseInt`. Ponieważ może się zdarzyć, że warstwie `divTekst` nie została przypisana reguła CSS określająca położenie (cechę `left`), niezbędne jest zbadanie, czy wynik zwrócony przez `parseInt` jest różny od `NaN`. Jeśli nie jest, zmienna `tekstL` otrzymuje wartość domyślną — 0 (`tekstL = 0;`).

Animacja wykonywana jest w prosty sposób. Aby przesunąć warstwę z tekstem w lewo, należy od bieżącej wartości właściwości `left` (zmienna `tekstL`) odjąć wartość

zapisaną w krokp_x (tekst_L -= krokp_x). Przesuw w prawo jest wykonywany w sposób odwrotny — do bieżącej wartości tekst_L dodawana jest wartość zapisana w krokp_x (tekst_L += krokp_x).

Należy przy tym zadbać o właściwą obsługę wartości granicznych. Otóż przy przesuwaniu w lewo, jeśli cały tekst zniknie z lewej strony warstwy divGdzie, następnie powinien pojawić się u jej prawego brzegu. To oznacza, że jeśli prawdziwy jest warunek tekst_L < -tekst_W, należy dokonać przypisania tekst_L = gdzie_W. Przy przesuwaniu w prawo jest podobnie. Jeśli tekst zniknie w całości z prawej strony, powinien się pojawić z lewej. A zatem gdy prawdziwy jest warunek tekst_L > gdzie_W, należy dokonać przypisania tekst_L = -tekst_W.

Aby efekt animacji był prawidłowy, nie można zapomnieć o stylach CSS. Przede wszystkim pozycjonowanie warstwy z tekstem (divTekst) nie może być statyczne, ale względne lub bezwzględne. W tym drugim przypadku warstwa, po której będzie pływał tekst (divDane), również nie mogłaby mieć pozycjonowania statycznego. Należy też określić szerokość warstw, w szczególności warstwy divTekst — ten parametr został określony w kodzie na listingu 6.3. Dodatkowo warstwa, po której ma pływać tekst, powinna mieć zdefiniowaną cechę overflow o wartości hidden, a w przypadku Internet Explorera w wersji 6. i 7. nie może też mieć pozycjonowania statycznego. Przykładowe style dla warstwy divDane mogłyby zatem wyglądać następująco:

```
#divDane{
  width:300px;
  height:20px;
  border:1px solid black;
  position:relative;
  overflow:hidden;
}
```

Skrypt 55. [C][E][F][O][S] Napis płynnie zmieniający kolor

Jeśli w krótkich odstępach czasu będziemy zmieniać kolor danego elementu strony, osiągniemy efekt płynnej zmiany barw. Oczywiście każdy kolejny odcień musi być wtedy zbliżony do poprzedniego, inaczej zmiany byłyby skokowe, co spowodowałoby zwykle nieprzyjemne migotanie. Kolor można zmieniać, po prostu modyfikując właściwość color obiektu style. W przypadku gdybyśmy chcieli manipulować kolorem tła, należałoby modyfikować właściwość backgroundColor. Załóżmy, że na stronie znalazła się warstwa z tekstem, a tekst ma płynnie zmieniać barwę. Kod sekcji <body> ma zatem postać widoczną na listingu 6.6.

Listing 6.6. Warstwa z tekstem umieszczona w sekcji <body>

```
<body onload="start('divDane', 1);">
  <div id="divDane" style="font-size:36pt;font-weight:bold">
    To jest przykładowy tekst.
```

```
</div>  
</body>
```

Warstwa została zdefiniowana w standardowy sposób za pomocą znacznika `<div>`. Przypisano jej styl powodujący zwiększenie i pogrubienie czcionki, tak by efekt był lepiej widoczny, oraz identyfikator `divDane`. Animacja kolorów będzie następowała automatycznie po załadowaniu strony dzięki przypisaniu zdarzeniu `load` sekcji `<body>` procedury obsługi w postaci funkcji `start`. Funkcja przyjmuje dwa argumenty. Pierwszy wskazuje identyfikator elementu witryny, którego kolor ma być zmieniany, a drugi — czas, w którym ma być widoczny pojedynczy odcień. Inaczej mówiąc, drugi parametr określa szybkość zmian kolorów. Wartość 1 oznacza największą możliwą prędkość.

Treść funkcji `start` wraz z pozostałą częścią skryptu została przedstawiona na listingu 6.7.

Listing 6.7. *Skrypt zmieniający kolory*

```
<script type="text/javascript">  
  var tabKolorów = new Array();  
  var el = null;  
  var timerId = null;  
  var poz = 0;  
  var krok = 1;  
  function przygotujKolory()  
  {  
    for(var i = 0; i <= 255; i++){  
      var kolor = "rgb(" + i + "," + (255 - i) + "," + "  
        kolor += 0 + ")";  
      tabKolorów[i] = kolor;  
    }  
  }  
  function start(id, timeout)  
  {  
    el = document.getElementById(id);  
    if(!el) return;  
    przygotujKolory();  
    timerId = setInterval('zmieniaj()', timeout);  
  }  
  function zmieniaj()  
  {  
    if(!el){  
      clearInterval(timerId);  
      return;  
    }  
    el.style.color = tabKolorów[poz];  
    poz += krok;  
    if(poz >= tabKolorów.length){  
      poz = tabKolorów.length - 1;  
      krok = -krok;  
    }  
    if(poz <= 0){  
      poz = 0;  
      krok = -krok;  
    }  
  }  
}</script>
```

Na początku kodu znajdują się deklaracje zmiennych globalnych:

- ◆ `tabKolorów` — tablica zawierająca kody kolorów;
- ◆ `el` — odniesienie do elementu witryny, którego kolor będzie zmieniany;
- ◆ `timerId` — identyfikator timera;
- ◆ `poz` — wskaźnik aktualnej pozycji w tablicy kolorów;
- ◆ `krok` — wartość, o którą zostanie zmieniony wskaźnik pozycji w każdym kroku animacji.

Funkcja `start` przyjmuje argument `id` określający identyfikator elementu, którego kolor będzie zmieniany, oraz `timeout`, określający czas między kolejnymi wywołaniami funkcji `zmieniaj` dokonującej faktycznej zmiany koloru. Poprzez wywołanie metody `getElementById` pobiera odwołanie do elementu wskazywanego przez `id` i zapisuje je w globalnej zmiennej `el`. Dalsze kroki są wykonywane tylko wtedy, gdy ta czynność zakończyła się sukcesem. Będzie to wywołanie funkcji `przygotujKolory` oraz ustawienie timera, powodującego cykliczne wywołania funkcji `zmieniaj` w czasie określonym przez `timeout`.

Zadaniem funkcji `przygotujKolory` jest zainicjowanie globalnej tablicy `tabKolorów`. W jej komórkach mają się znaleźć określenia kolejnych barw, które będzie przyjmował modyfikowany elementy witryny. Kod tej funkcji należy zatem dostosować do własnych potrzeb. W prezentowanym przykładzie kolor będzie się zmieniał od zielonego do czerwonego. Stosowane jest określenie koloru w formacie *rgb(R,G,B)*, gdzie *R* — to składowa czerwona, *G* — zielona, a *B* — niebieska. W pętli `for` generowanych jest 256 kodów kolorów, w których:

- ◆ składowa *R* zmienia się od 0 do 255;
- ◆ składowa *G* zmienia się od 255 do 0;
- ◆ składowa *B* nie zmienia się i jest cały czas równa 0.

Pierwszy kod ma więc postać `rgb(0, 255, 0)`, drugi — `rgb(1, 254, 0)`, trzeci — `rgb(2, 253, 0)` itd., aż do ostatniego — `rgb(255, 0, 0)`.

Zmianą koloru pierwszoplanowego (koloru tekstu) elementu witryny wskazywanego przez zmienną `el` (w prezentowanym przykładzie jest to warstwa `divDane`) zajmuje się wywoływana cyklicznie funkcja `zmieniaj`. Najpierw sprawdza, czy zmienna `el` jest pusta. Jeśli jest, wyłącza timer (`clearInterval(timerId)`) i kończy wykonywanie kodu. Jeśli nie jest, przypisuje właściwości `color` obiektu `style` elementu `el` kolor pobrany z tablicy `tabKolorów` znajdujący się pod indeksem wskazywanym przez zmienną `poz`. To powoduje zmianę koloru.

W kolejnym kroku indeks `poz` jest zwiększany o wartość zapisaną w zmiennej `krok`. W klasycznym przypadku `krok` powinna mieć wartość 1, co spowoduje wykorzystanie wszystkich kolorów z tablicy. Jeśli jednak chcemy zwiększyć szybkość zmian, a tym samym zmniejszyć płynność przejść tonalnych, można tę wartość zwiększyć.

Ponieważ tablica kolorów ma ściśle określoną liczbę komórek, po zmianie wartości zmiennej `poz` trzeba stwierdzić, czy nie przekracza dopuszczalnych granic. Indeks nie może być mniejszy od 0 ani większy niż długość tablicy minus 1. W przykładzie przyjęto, że kierunek zmian indeksu po osiągnięciu danego krańca tablicy zmienia się na przeciwny. Jeżeli zatem po tablicy przesuwano się w górę (zwiększano indeks), zacznie się on zmniejszać, a jeśli przesuwano się w dół — zwiększać. Stąd dwie instrukcje warunkowe badające stan zmiennej `poz` i dokonujące, w razie potrzeby, odpowiednich korekt. Kierunek przesuwania się po tablicy jest modyfikowany przez odwrócenie znaku zmiennej `krok`:

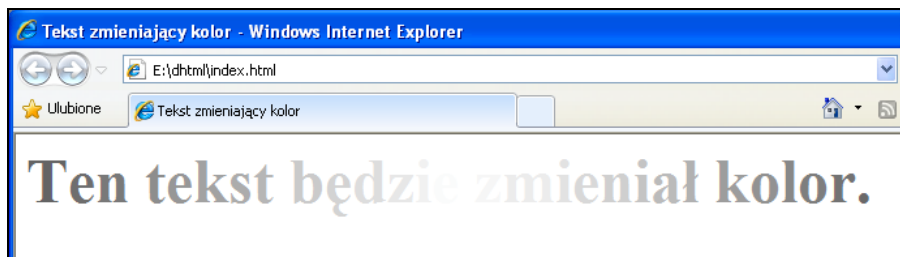
```
krok = -krok;
```

Jeżeli zatem była dodatnia, a ma nastąpić zmiana, stanie się ujemna, i odwrotnie — jeśli była ujemna, stanie się dodatnia.

Skrypt 56. [C][E][F][O][S]

Kolor płynący po napisie

W skrypcie 56. został osiągnięty efekt płynnej zmiany koloru całego napisu. Równie ciekawy będzie kolor „płynący po napisie”. Poszczególne litery będą zmieniały swoją barwę w taki sposób, aby wydawało się, że następuje przesuwanie barwy w prawą lub w lewą stronę. Pojedyncza faza takiej animacji mogłaby wyglądać tak, jak przedstawiono to na rysunku 6.2. Sekcja `<body>` HTML skryptu wykonującego takie zadanie została zaprezentowana na listingu 6.8.



Rysunek 6.2. Litery o generowanych dynamicznie różnych kolorach

Listing 6.8. Sekcja `<body>` skryptu 56.

```
<body onload="start('divDane', 'lewo', 20);">
  <div id="divDane" style="font-size:36pt;font-weight:bold">
    Ten tekst będzie zmieniał kolor.
  </div>
</body>
```

W kodzie została umieszczona zwykła warstwa generowana za pomocą znacznika `<div>`. W warstwie znajduje się przykładowy tekst, został jej też nadany styl powodujący zwiększenie i pogrubienie czcionki. Animacja będzie się rozpoczynała tuż po

załadowaniu strony, bowiem atrybutowi `onload` znacznika `<body>` zostało przypisane wywołanie funkcji `start` rozpoczynającej ten proces. Funkcja przyjmuje trzy argumenty: identyfikator warstwy z tekstem, określenie kierunku ruchu i określenie szybkości animacji.

Zadaniem skryptu będzie przetworzenie tekstu z dowolnej warstwy o identyfikatorze przekazanym jako argument funkcji `start` na postać, która pozwoli na zmianę koloru poszczególnych liter oraz wykonania animacji stwarzającej wrażenie płynięcia koloru po napisie. Można to osiągnąć poprzez ujęcie każdej litery w znacznik `` z własnym identyfikatorem. Wtedy będzie można manipulować jego dowolnymi parametrami, w tym również i kolorem.

Początek kodu skryptu został przedstawiony na listingu 6.9.

Listing 6.9. Pierwsza część kodu skryptu 56.

```
<script type="text/javascript">
  var tabKolorów = new Array();
  var el = null;
  var timerId = null;
  var poz = 0;
  var długośćTekstu;
  var kierunek = 'lewo';
  function przygotujKolory()
  {
    var licznik = 0;
    for(var i = 0; i <= 255; i += 10){
      tabKolorów[licznik] = "rgb(" + 0 + "," + i + "," + 0 + ")";
      tabKolorów[26 + licznik] =
        "rgb(" + 0 + "," + (255 - i) + "," + 0 + ")";
      licznik++;
    }
  }
  function start(id, lewoprawo, timeout)
  {
    el = document.getElementById(id);
    if(!el) return;
    var str = "";
    długośćTekstu = el.innerHTML.length;
    for(var i = 0; i < długośćTekstu; i++){
      str += "<span id='" + i + "'>";
      str += el.innerHTML.charAt(i);
      str += "</span>";
    }
    el.innerHTML = str;
    przygotujKolory();
    kierunek = lewoprawo;
    timerId = setInterval('zmieniaj()', timeout);
  }
  //dalsza część kodu skryptu
```

Na początku kodu znajdują się deklaracje zmiennych globalnych. Zmienne `tabKolorów`, `el`, `timerId` i `poz` mają takie samo znaczenie jak w przypadku skryptu 55. Pojawiły się natomiast dwie nowe zmienne:

- ♦ `długośćTekstu` — określająca długość tekstu, czyli liczbę liter;
- ♦ `kierunek` — określająca kierunek przesuwu (prawo lub lewo).

Zadaniem funkcji `przygotujKolory` jest przygotowanie tablicy kolorów zapisanej w zmiennej globalnej `tabKolorów`. Można ją przygotować dowolnie, według własnych upodobań. Aby efekt był najlepszy, odcienie powinny zmieniać się płynnie, a zmiany barw powinny być zapętlone, tzn. najlepiej, by końcówka tablicy była odwróceniem jej początku. W zaprezentowanym rozwiązaniu druga połowa jest prawie dokładnym lustrzanym odbiciem pierwszej (odcienie różnią się minimalnie). Komórki z początku i końca definiują kolory zbliżone do czarnego, a komórki środkowe — odcienie zielonego. Spowoduje to efekt czarnego napisu, po którym „przepływa” kolor zielony.

Funkcja `start` jest wykonywana tuż po załadowaniu strony do przeglądarki. Otrzymuje trzy argumenty:

- ♦ `id` — określa identyfikator warstwy zawierającej tekst, po którym będzie płynął kolor;
- ♦ `lewoprawo` — określa kierunek przepływu;
- ♦ `timeout` — określa szybkość animacji (czas pomiędzy kolejnymi fazami, czyli wywołaniami funkcji `zmieniaj`).

Odwołanie do elementu wskazywanego przez argument `id` jest pobierane za pomocą metody `getElementById` i zapisywane w zmiennej `el`. Długość zawartego w nim tekstu jest określana za pomocą odwołania do właściwości `length` i zapisywana w zmiennej `długośćTekstu`. Następnie w pętli `for` każda litera jest ujmowana w znacznik ``, a przetworzony ciąg jest dopisywany do zmiennej pomocniczej `str`. Każdy znacznik otrzymuje swój własny identyfikator, według schematu:

```
<span id='1Numer'>litera</span>
```

gdzie *Numer* to kolejny numer litery. Po zakończeniu pętli zawartość zmiennej `str` jest ponownie zapisywana w warstwie. Przykładowo, jeśli w warstwie o identyfikatorze wskazanym przez argument `id` znajduje się tekst `abc`, to po przetworzeniu przez pętlę przyjmie postać:

```
<span id='10'>a</span><span id='11'>b</span><span id='12'>c</span>
```

Zawartość warstwy jest przy tym uzyskiwana za pomocą odwołania do właściwości `innerHTML` elementu wskazywanego przez zmienną `el`, natomiast dostęp do pojedynczej litery — za pomocą metody `charAt`. Metoda ta przyjmuje indeks ciągu, a zwraca znak znajdujący się pod tym indeksem. W pętli nie zostało użyte bardziej naturalne odwołanie w postaci:

```
str += el.innerHTML[i];
```

w którym ciąg jest traktowany jak tablica (a także kolekcja lub obiekt z właściwością domyślną), bowiem taka konstrukcja nie jest dostępna w przeglądarkach Internet Explorer w wersjach poniżej 8.

Po zakończeniu pętli wywoływana jest jeszcze funkcja `przygotujKolory` oraz ustalany jest stan globalnej zmiennej `kierunek` określającej kierunek przepływu koloru. Na zakończenie za pomocą metody `setInterval` ustawiany jest timer powodujący cykliczne wywołania funkcji `zmieniaj`, która wykonuje właściwą animację. Treść funkcji `zmieniaj` została przedstawiona na listingu 6.10.

Listing 6.10. Treść funkcji `zmieniaj`

```
//początek skryptu z listingu 6.9
function zmieniaj()
{
    if(!el){
        clearInterval(timerId);
        return;
    }
    for(var i = 0; i < długośćTekstu; i++){
        document.getElementById("l" + i).style.color =
            tabKolorów[(poz + i) % tabKolorów.length];
    }
    if(kierunek == 'lewo'){
        if(++poz >= tabKolorów.length) poz = 0;
    }
    else{
        if(--poz < 0) poz = tabKolorów.length;
    }
}
</script>
```

Najpierw sprawdzane jest, czy zmienna `el` istnieje i jest niepusta — tylko wtedy mają sens dalsze czynności. W przypadku wykrycia nieprawidłowości wyłączany jest timer i funkcja kończy działanie. Głównym zadaniem jest jednak zmiana koloru liter, czyli przypisywanie kolejnym znacznikom ``, o identyfikatorach w postaci `lNumer`, kolorów z tablicy `tabKolorów`. Odbывается to w pętli `for`. Pobierane jest odwołanie do elementu strony o identyfikatorze w podanym formacie i zmieniana jest właściwość `color` obiektu `style`. Aktualna pozycja w tablicy kolorów jest wskazywana przez zmienną `poz`. Ponieważ nie można dopuścić do przekroczenia zakresu tablicy, bieżący indeks jest wyliczany ze wzoru:

$$(poz + i) \% tabKolorów.length$$

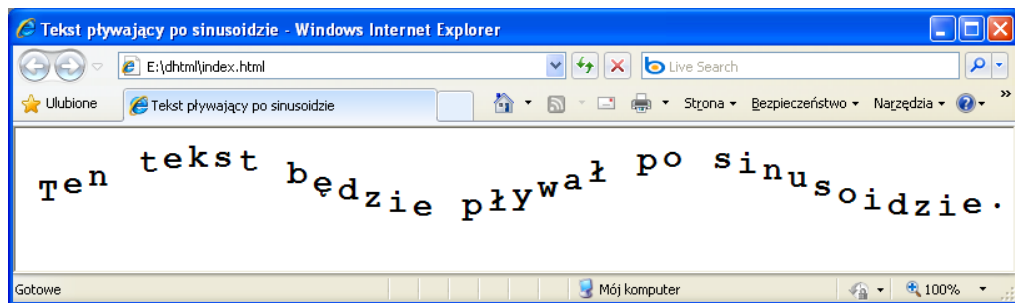
Użyte w nim dzielenie modulo powoduje odpowiednie zapętlenie, gdy wartość `poz + i` jest większa niż rozmiar tablicy (`tabKolorów.length`).

W każdym wywołaniu funkcji `zmieniaj` musi nastąpić zwiększanie lub zmniejszanie wartości zmiennej `poz`, czyli odpowiednie przesunięcie bieżącej pozycji w tablicy kolorów. Jeżeli kolor ma być przesuwany w lewo, wartość `poz` jest zwiększana (`++poz`), a w przypadku przekroczenia zakresu (`++poz >= tabKolorów.length`) — zerowana (`poz = 0`). Gdy kolor ma być przesuwany w prawo, wartość `poz` jest zmniejszana (`--poz`), a w przypadku przekroczenia zakresu (`--poz < 0`) — otrzymuje wartość maksymalną (`poz = tabKolorów.length`).

Skrypt 57. Napis na sinusoidzie

[C][E][F][O][S]

Skrypt 57. pozwoli utworzyć animowany napis, którego litery będą pływały po sinusoidzie. Da to bardzo przyjemny dla oka efekt wizualny. Jedna z faz takiej animacji została przedstawiona na rysunku 6.3. Tekst będzie mógł być dowolny i będzie podawany jako argument w funkcji rozpoczynającej procedurę „wyginania” napisu. W związku z tym treść sekcji `<body>` będzie bardzo prosta i przyjmie postać zaprezentowaną na listingu 6.11.



Rysunek 6.3. Jedna z faz animacji ze skryptu 57.

Listing 6.11. Treść sekcji `<body>` skryptu 57.

```
<body onload="start('divDane',
  'Ten tekst będzie pływał po sinusoidzie.', 20, 20, 10);">
  <div id="divDane">
    </div>
</body>
```

W kodzie została umieszczona pusta warstwa o identyfikatorze `divDane`. To na niej po załadowaniu strony do przeglądarki pojawi się animowany tekst. Stanie się tak dzięki przypisaniu atrybutowi `onload` znacznika `<body>` wywołania funkcji `start`, która dokona odpowiednich przypisań i rozpocznie właściwą animację. Początek skryptu zawierającego tę funkcję został przedstawiony na listingu 6.12.

Listing 6.12. Początek skryptu animującego tekst

```
<script type="text/javascript">
  var timerId = null;
  var długośćTekstu;
  var odstęp = 20;
  var tabPozycji = new Array();
  var okres = 2 * Math.PI;
  var amplituda = 10;
  var yPos = 0;
  function przygotujPozycje()
  {
    var x = 0;
    var j = 0;
```

```
for(i = 0; i < długośćTekstu; i++){
    tabPozycji[j++] = Math.sin(x) * amplituda + yPos;
    x += okres / długośćTekstu;
}
}
function start(id, tekst, amplituda, yPos, timeout)
{
    el = document.getElementById(id);
    if(!el) return;
    window.yPos = yPos;
    window.amplituda = amplituda;
    długośćTekstu = tekst.length;
    var str = "";
    var xPos = 0;
    for(var i = 0; i < tekst.length; i++){
        str += "<div class='litera' id='d" + i + "' ";
        str += "style='left:" + xPos + "px;top:0px;'>";
        str += tekst.charAt(i);
        str += "</div>";
        xPos += odstęp;
    }
    el.innerHTML = str;
    przygotujPozycje();
    timerId = setInterval('zmieniaj()', timeout);
}
//dalsza część skryptu
```

Pierwsza część kodu rozpoczyna się od definicji zmiennych globalnych:

- ♦ timerId — określa identyfikator timera (uruchomionego przez wywołanie metody setInterval);
- ♦ długośćTekstu — określa długość tekstu, czyli liczbę liter;
- ♦ odstęp — określa w pikselach odstęp pomiędzy kolejnymi literami;
- ♦ tabPozycji — tablica określająca pionowe pozycje liter;
- ♦ okres — długość sinusoidy, na której będą rozmieszczone litery;
- ♦ amplituda — rozciągnięcie sinusoidy w pionie;
- ♦ yPos — położenie sinusoidy w pionie (przesunięcie względem górnego brzegu warstwy z napisem).

Funkcja start zajmuje się wstępnym przygotowaniem danych oraz uruchomieniem timera. Przyjmuje pięć argumentów:

- ♦ id — identyfikator warstwy, na której pojawi się animowany tekst;
- ♦ tekst — tekst, który ma być animowany;
- ♦ amplituda — rozciągnięcie sinusoidy w pionie;
- ♦ yPos — położenie (przesunięcie) sinusoidy w pionie;
- ♦ timeout — określa szybkość animacji (czas pomiędzy kolejnymi fazami, czyli wywołaniami funkcji zmieniaj).

Najpierw za pomocą metody `getElementById` jest pobierane i zapisywane w zmiennej `el` odwołanie do elementu wskazanego za pomocą argumentu `id`. Jeżeli takiego elementu nie będzie w kodzie witryny, wykonywanie funkcji zostanie przerwane za pomocą instrukcji `return`. Jeżeli element będzie dostępny, zostaną ustalone wartości globalnych zmiennych: `yPos`, `amplituda` i `długośćTekstu`. Będą one używane w dalszej części kodu przy obliczaniu pozycji liter.

Aby zwykły napis utworzył sinusoidę, każda litera musi być oddzielnie pozycjonowana. Dlatego też na warstwie wskazanej przez `id` zamiast napisu pojawi się seria dodanych warstw — każda litera otrzyma swoją własną. A więc zamiast litery zostaną użyte ciągi w postaci:

```
<div id='lNumer' atrybuty>litera</div>
```

gdzie *Numer* to kolejny numer litery. Oprócz identyfikatora zostanie także użyty atrybut `style` określający cechy CSS: `left` i `top`. Pierwsza określa położenie w poziomie i jest wyliczana przez cykliczne dodawanie do zmiennej `xPos` wartości zapisanej w odstęp. Tym samym za pomocą globalnej zmiennej `odstęp` można regulować odległość między kolejnymi literami. Wartość cechy `top` jest stała i równa 0. Została użyta tylko w celu umieszczenia właściwości `top` w każdym z obiektów odpowiadających warstwom zawierającym litery².

Ostatecznie po wykonaniu pętli tekst zostanie zamieniony na serię warstw. Przykładowo dla napisu `abc` i odległości między literami równej 20 pikseli w warstwie pojawił się kod:

```
<div class='litory' id='d0' style='left:0px;top:0px;'>a</div>
<div class='litory' id='d1' style='left:20px;top:0px;'>b</div>
<div class='litory' id='d2' style='left:40px;top:0px;'>c</div>
```

Do uzyskania pojedynczej litery tekstu jest przy tym używane wywołanie metody `charAt`:

```
str += tekst.charAt(i);
```

Powód jest taki sam jak w przypadku skryptu 56. — przeglądarka Internet Explorer w wersji 6. i 7. nie obsługuje dostępu do poszczególnych pozycji w ciągu znaków za pomocą składni z nawiasem kwadratowym. W innych przeglądarkach dopuszczalne byłoby również użycie instrukcji:

```
str += tekst[i];
```

Po przygotowaniu zawartości warstwy jest wywoływana funkcja `przygotujPozycje` wypełniająca danymi tablicę `tabPozycji` oraz jest uruchamiany timer powodujący cykliczne wywołania funkcji `zmieniaj`, która przesuwa litery, powodując efekt ich pływania po sinusoidzie. Kluczową sprawą dla uzyskania dobrego efektu jest zatem odpowiedni kod wspomnianej funkcji. W rzeczywistości jest on bardzo prosty. Otóż należy rozmieścić litery tekstu (dokładniej warstwy z literami) równomiernie po sinusoidzie. O tym, jak bardzo będą one zagęszczone, decyduje wartość zmiennej `okres`. W przykładzie została użyta wartość 2π (`2 * Math.PI`). Im większa będzie ta wartość, tym większe będzie zagęszczenie liter na sinusoidzie.

² Nie jest to formalnie konieczne w prezentowanym przykładzie, ale może być przydatne w przypadku wprowadzania modyfikacji, które odczytywałyby wartość tej właściwości.

Ponieważ litery mają być rozmieszczone równomiernie, oznacza to, że odstęp między nimi wynika ze wzoru:

$$\frac{\text{okres}}{\text{liczba liter}}$$

natomiast położenie w pionie wyliczymy ze wzoru: $y = \sin(x) * \text{amplituda} + \text{przesunięcie w pionie}$. Przy tym im większa amplituda, tym większe rozciągnięcie sinusoidy w pionie. Kolejne pozycje są wyliczane na podstawie podanych wzorów i zapisywane w globalnej tablicy `tabPozycji`.

Wykonaniem animacji zajmuje się funkcja `zmieniaj`. Została zaprezentowana na listingu 6.13.

Listing 6.13. Treść funkcji `zmieniaj`

```
//początek skryptu z listingu 6.12
function zmieniaj()
{
    if(!el){
        clearInterval(timerId);
        return;
    }
    for (i = 0; i < długośćTekstu; i++)
        document.getElementById('d' + i).style.top = tabPozycji[i] + "px";
    var x = tabPozycji[0];
    for (i = 0; i < długośćTekstu - 1; i++)
        tabPozycji[i] = tabPozycji[i + 1];
    tabPozycji[długośćTekstu - 1] = x;

}
</script>
```

Funkcja sprawdza najpierw, czy istnieje warstwa (dokładniej: element witryny), na której znajduje się animowany tekst. To wyznacznik tego, czy mają być wykonywane dalsze czynności. Jeśli warstwy nie ma, timer jest zatrzymywany, a wykonywanie kodu — kończone.

W pętli `for` w kolejnych warstwach, na których znajdują się litery (warstwy te mają identyfikatory w postaci `d0`, `d1`, `d2`... itd.) modyfikowana jest właściwość `top` obiektu `style`. Tym samym zmienia się położenie warstwy w pionie. Właściwości `top` przypisywana jest wartość korespondującej z nią komórki tabeli (tabela ma tyle komórek, ile znaków miał pierwotny tekst).

Po ustaleniu pozycji liter cała zawartość tabeli jest przepisywana w taki sposób, że do komórki o indeksie 0 przypisywana jest wartość komórki o indeksie 1, do komórki o indeksie 1 wartość komórki o indeksie 2 itd. A więc następuje przesunięcie wszystkich komórek o jedną do tyłu (komórka pierwsza staje się przy tym ostatnią). Ponieważ dzieje się to cyklicznie, w każdym wywołaniu funkcji `zmieniaj`, warstwy (litery) wciąż zmieniają swoje położenie, a napis „tańczy” po sinusoidzie.

Ostatnim niezbędnym elementem skryptu jest ustalenie bezwzględnego pozycjonowania dla warstw zawierających litery, czyli zdefiniowanie klasy `litera`. Musi ona zawierać definicję dla cechy `position` o wartości `absolute`. Dodatkowo można też zmienić krój czcionki, jej wielkość czy grubość. Przykładowa reguła mogłaby mieć zatem następującą postać:

```
.litera{
  position:absolute;
  font-family:Courier New;
  font-size:20pt;
  font-weight:bold;
}
```

Skrypt 58. [C][E][F][O][S]

Litery pojawiające się pojedynczo (symulacja pisania na klawiaturze)

W skrypcie 58. litery będą się pojawiały na stronie z pewnym opóźnieniem, co da wrażenie, jakby ktoś pisał na klawiaturze. Aby efekt był bardziej realistyczny, opóźnienie będzie losowe w wybranym przedziale czasu. W kodzie witryny należy umieścić warstwę, na której będzie się pojawiał napis, a po załadowaniu strony — wywołać funkcję realizującą opisane zadanie. Kod sekcji `<body>` mógłby więc przyjąć postać przedstawioną na listingu 6.14.

Listing 6.14. Sekcja `<body>` z warstwą, na której pojawi się tekst

```
<body onload="start(
  'dataDiv', 'Ten tekst pojawi się na warstwie.', 50, 200);">
  <div id="dataDiv">
  </div>
</body>
```

Warstwa o identyfikatorze `dataDiv` będzie służyła do prezentacji tekstu. Początkowo będzie jednak pusta. Litery zaczną się pojawiać dzięki wywołaniu funkcji `start` tuż po załadowaniu witryny do przeglądarki. Argumenty funkcji określają parametry skryptu. Jego kod został zaprezentowany na listingu 6.15.

Listing 6.15. Skrypt symulujący pisanie na klawiaturze

```
<script type="text/javascript">
  var długośćTekstu;
  var licznik;
  var div = null;
  var tekst = "";
  var czas1 = 100;
  var czas2 = 200;
  function start(id, tekst, czas1, czas2)
```



```
{
  div = document.getElementById(id);
  if(!div || !tekst) return;
  window.czas1 = isNaN(czas1 = parseInt(czas1))?100:czas1;
  window.czas2 = isNaN(czas2 = parseInt(czas2))?200:czas2;
  długośćTekstu = tekst.length;
  licznik = 0;
  window.tekst = tekst;
  pisz();
}
function pisz()
{
  if(!div) return;
  div.innerHTML += tekst.charAt(licznik);
  if(++licznik >= długośćTekstu)
    return;
  else{
    var timeout = Math.floor((czas2 - czas1 - 1) * Math.random()) + czas1;
    setTimeout("pisz()", timeout);
  }
}
</script>
```

Na początku skryptu znajdują się definicje zmiennych globalnych:

- ◆ długośćTekstu — liczba znaków w prezentowanym tekście;
- ◆ licznik — indeks aktualnie wyświetlanego znaku;
- ◆ div — warstwa, na której ma być wyświetlany tekst;
- ◆ tekst — wyświetlany tekst;
- ◆ czas1 — dolny zakres czasu między wyświetlaniem kolejnych znaków;
- ◆ czas2 — górny zakres czasu między wyświetlaniem kolejnych znaków.

Funkcja start, której zadaniem jest przygotowanie parametrów i rozpoczęcie procesu wyświetlania, przyjmuje cztery argumenty:

- ◆ id — identyfikator warstwy, której dotyczy efekt;
- ◆ tekst — ciąg znaków, który ma zostać wyświetlony;
- ◆ czas1 — dolny zakres czasu między wyświetlaniem kolejnych znaków;
- ◆ czas2 — górny zakres czasu między wyświetlaniem kolejnych znaków.

Najpierw jest pobierane i zapisywane w globalnej zmiennej `div` odwołanie do warstwy o identyfikatorze wskazanym przez argument `id`. Badane jest także to, czy zmienne `tekst` i `div` są puste. Jeśli są, wykonywanie kodu jest przerywane za pomocą instrukcji `return`. Następnie weryfikowane są wartości przekazane za pomocą argumentów `czas1` i `czas2`. Jeśli są to liczby całkowite, są przypisywane globalnym zmiennym o nazwach `czas1` i `czas2`, jeśli argumenty są nieprawidłowe, zmienne otrzymują wartości domyślne (100 i 200).

Ponieważ nazwy zmiennych i parametrów są takie same, w celu dostępu do zmiennych globalnych używana jest konstrukcja `window.zmienna`. Weryfikacja przeprowadzana jest za pomocą złożonej konstrukcji z operatorem warunkowym oraz metod `parseInt` i `isNaN`. Jej znaczenie zostało dokładniej wyjaśnione przy omawianiu skryptu 54.

Po wypełnieniu wartości pozostałych zmiennych globalnych (`długośćTekstu`, `licznik` i `tekst`) wywoływana jest funkcja `pisz` rozpoczynająca prezentację tekstu.

Głównym zadaniem funkcji `pisz` jest dopisywanie przy każdym wywołaniu do aktualnej zawartości warstwy kolejnego znaku tekstu. Znak ten jest wskazywany przez zmienną `licznik`. To zadanie jest realizowane za pomocą instrukcji:

```
div.innerHTML += tekst.charAt(licznik);
```

Proces ten ma trwać tak długo, aż zmienna `licznik` osiągnie wartość wskazywaną przez zmienną `długośćTekstu`. Oczywiście w każdym wywołaniu `licznik` jest zwiększany o 1. Jeśli zostanie osiągnięta wartość graniczna (co oznacza, że cały tekst pojawił się na ekranie), wykonywanie kodu jest przerywane (jest to ostatnie wywołanie funkcji `pisz`), w przeciwnym wypadku ustawiany jest timer powodujący kolejne wywołanie funkcji `pisz`.

Ponieważ czas pomiędzy pojawianiem się kolejnych znaków ma być za każdym razem inny, jest on losowany z zakresu określonego przez zmienne `czas1` i `czas2`. Do używania wartości losowej jest używana metoda `random` obiektu `Math`. Ponieważ zwracana przez nią wartość jest z przedziału $<0,1$), musi być przetworzona na właściwy zakres. Jest przy tym używany wzór:

```
Math.floor((czas2 - czas1 - 1) * Math.random()) + czas1;
```

gdzie `Math.floor` to zaokrąglenie wartości dół.

Rozdział 7.

Boksy reklamowe, podpowiedzi itp.

Boksy to po prostu niewielkie kontenery dla treści, zwykle budowane za pomocą warstw. Można w nich umieszczać praktycznie dowolne dane. Mogą to być reklamy, wiadomości, porady dnia, różnego rodzaju opisy itp. W tym rozdziale zostanie opisanych kilka technik tworzenia takich elementów witryny. Zostaną przedstawione m.in. sposoby tworzenia podpowiedzi i związane z nimi techniki określania położenia elementów witryny względem siebie, gadżety dowolnie przesuwane po ekranie za pomocą myszy i związane z tym określanie współrzędnych kliknięcia uwzględniające rozmiary elementów sąsiadujących, a także różnego rodzaju animacje umożliwiające efektowne pokazywanie i chowanie treści.

Skrypt 59.

[C][E][F][O][S]

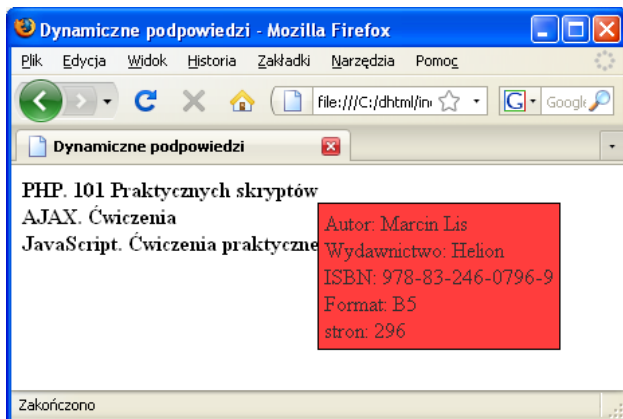
Wyskakująca podpowiedź

Skrypt 59. to ilustracja systemu dynamicznych podpowiedzi czy też dodatkowych informacji wyświetlanych po najechaniu kursorem myszy na wybrane elementy witryny. Dla przykładu na stronie umieścimy kilka tytułów książek. Wskazanie dowolnego tytułu będzie powodowało pojawienie się przy jego prawym dolnym rogu nowej warstwy zawierającej dodatkowe informacje, tak jak jest to widoczne na rysunku 7.1. Kod sekcji `<body>` takiej witryny został zaprezentowany na listingu 7.1.

Listing 7.1. *Kod sekcji `<body>` dla dynamicznych podpowiedzi*

```
<body>
  <div id="tooltipDiv" class="tooltipDiv">
  </div>
  <div id="divDane">
    <div><span onmouseover="showTooltip(1, this)" onmouseout="closeTooltip();"
      class="tytuły">
        PHP. 101 Praktycznych skryptów </span></div>
```

Rysunek 7.1.
*Dynamiczna
 podpowiedź*



```
<div><span onmouseover="showTooltip(2, this)" onmouseout="closeTooltip();"
      class="tytuły">
  AJAX. Ćwiczenia </span></div>
<div><span onmouseover="showTooltip(3, this)" onmouseout="closeTooltip();"
      class="tytuły">
  JavaScript. Ćwiczenia praktyczne </span></div>
</div>
</body>
```

Każdy tytuł znajduje się na osobnej warstwie i jest ujęty w znacznik ``, któremu przypisane zostały procedury obsługi zdarzeń `mouseover` i `mouseout`. Są to funkcje `showTooltip` oraz `closeTooltip`, powodujące wyświetlenie oraz schowanie podpowiedzi. Funkcja `showTooltip` przyjmuje dwa argumenty. Pierwszy to identyfikator podpowiedzi, która ma się ukazać, a drugi — wskazanie na element `` wywołujący funkcję. Podpowiedź będzie wyświetlana na warstwie o identyfikatorze `tooltipDiv`, która domyślnie powinna być niewidoczna (należy o to zadbać w definicji stylu `tooltipDiv`, ustawiając właściwość `display` na `none`). Ta warstwa została zdefiniowana na samym początku kodu i początkowo jest pusta. Zawartość będzie generowana dynamicznie przez skrypt.

Przykładowe style CSS, które można użyć w witrynie, zostały przedstawione na listingu 7.2.

Listing 7.2. *Przykładowe style CSS dla skryptu 59.*

```
<style type="text/css">
  .tytuły{
    font-weight:bold;
    cursor:pointer;
  }
  .tooltipDiv
  {
    color: #333333;
    position: absolute;
    background-color: #FF4040;
    border: 1px solid black;
    margin: 0px;
```

```
padding: 4px;
display:none;
}
</style>
```

Najważniejsze reguły dotyczą klasy `tooltipDiv`. Niezbędne jest ustalenie pozycjonowania bezwzględne (`position: absolute;`), a także zapobieżenie wyświetlaniu warstwy poprzez przypisanie właściwości `display` wartości `none`. Pozostałe cechy można definiować dowolnie według potrzeb i tego, jak ma wyglądać podpowiedź w danym serwisie. W prezentowanym przykładzie jest to po prostu prostokąt o zadanym kolorze i obramowaniu.

Treść funkcji `showTooltip` i `closeTooltip`, które zajmują się wyświetlaniem i chowaniem podpowiedzi, została zaprezentowana na listingu 7.3.

Listing 7.3. *Treść funkcji sterujących podpowiedziami*

```
<script type="text/javascript">
var tabPodpowiedzi = [
    "Pierwsza podpowiedź",
    "Druga podpowiedź",
    "Trzecia podpowiedź"
];
function showTooltip(id, obj)
{
    if(!obj) return;

    id = isNaN(id = parseInt(id))?0:id;
    if(id < 1 || id > tabPodpowiedzi ) return;

    xOff = obj.offsetLeft/1;
    yOff = obj.offsetTop/1;

    while(obj.offsetParent){
        xOff += obj.offsetLeft/1;
        yOff += obj.offsetTop/1;
        obj = obj.offsetParent;
    }

    var tooltipDiv = document.getElementById("tooltipDiv");
    if(!tooltipDiv) return;
    tooltipDiv.innerHTML = tabPodpowiedzi[id - 1];
    tooltipDiv.style.top = yOff + "px";
    tooltipDiv.style.left = xOff + "px";
    tooltipDiv.style.display = "block";
}

function closeTooltip()
{
    var tooltipDiv = document.getElementById("tooltipDiv");
    if(tooltipDiv)
        tooltipDiv.style.display = "none";
}
</script>
```

Wszystkie podpowiedzi, które będą używane na stronie, należy zapisać w znajdującej się na początku skryptu tablicy `tabPodpowiedzi`. Funkcja `showTooltip` zajmuje się ich wyświetlaniem. Kod zaczyna się od sprawdzenia, czy istnieje obiekt przekazany w postaci drugiego argumentu oraz czy przekazany numer podpowiedzi (pierwszy argument) jest liczbą całkowitą i zawiera się w prawidłowym zakresie. Nie może być bowiem mniejszy od 1 (numer pierwszej podpowiedzi) ani większy niż rozmiar tablicy `tabPodpowiedzi` (numer ostatniej podpowiedzi). W celu weryfikacji, czy wartość argumentu `id` jest całkowita, używa się techniki opisanej w skrypcie 54.

Ponieważ podpowiedź ma być wyświetlana w prawym dolnym rogu elementu, którego dotyczy, trzeba odnaleźć właściwe współrzędne. Muszą to być jednak współrzędne bezwzględne, poczynwszy od lewego górnego rogu okna przeglądarki. Aby je odnaleźć, wykonywane są dwie operacje. Po pierwsze, przez odwołanie się do właściwości `offsetWidth` i `offsetHeight` elementu odczytywana jest jego szerokość i wysokość. Dane te są zapamiętywane w zmiennych `xOff` i `yOff`, a następnie w pętli `while` są do nich dodawane współrzędne `x` (właściwość `left`) i `y` (właściwość `top`) każdego elementu nadrzędnego (w hierarchii otaczania) do obiektu `obj`.

W ten sposób uzyskiwane są współrzędne, w których należy umieścić lewy górny róg podpowiedzi. Pobierane jest więc odwołanie do warstwy `tooltipDiv` oraz modyfikowany obiekt `style`. Pozycja jest ustalana przez modyfikację właściwości `left` i `top`. Aby warstwa pojawiła się na ekranie, właściwość `display` jest przypisywana wartość `block`. Treść warstwy (właściwość `innerHTML`), czyli podpowiedź, jest pobierana z tablicy `tabPodpowiedzi` z użyciem indeksu wskazywanego przez parametr `id`.

Skrypt 60. [C][E][F][O][S]

Wysuwana podpowiedź (opis)

W skrypcie 59. zostało pokazane, jak wyposażyć elementy witryny w dynamiczną podpowiedź (czy też opis). W oparciu o techniki przedstawione w tamtym kodzie można napisać skrypt, w którym podpowiedzi nie będą się pojawiały od razu po wskazaniu danego elementu kursorem myszy, ale będą stopniowo wysuwane. Taki efekt można osiągnąć poprzez stopniowe zwiększanie rozmiarów warstwy zawierającej podpowiedź — `tooltipDiv`. Niezbędne będzie oczywiście użycie timera.

Kod HTML może w całości pozostać w postaci zaprezentowanej na listingu 7.1, a style CSS w postaci z listingu 7.2. Do selektora dotyczącego warstwy `tooltipDiv` należy jedynie dodać regułę:

```
overflow:hidden;
```

Dzięki temu zawartość warstwy nie będzie wystawała poza jej brzegi. Znaczące zmiany muszą natomiast zająć w kodzie skryptu. Jego pierwsza część została przedstawiona na listingu 7.4.

Listing 7.4. Zmienne globalne i treść funkcji `showTooltip`

```
<script type="text/javascript">
  var tabPodpowiedzi = [/*definicje podpowiedzi*/];
  var lKroków = 10;
  var timeout = 10;
  var krokX = 0;
  var krokY = 0;
  var timerId = null;
  function showTooltip(id, obj)
  {
    // tutaj początek funkcji showTooltip

    var tooltipDiv = document.getElementById("tooltipDiv");
    if(!tooltipDiv) return;

    tooltipDiv.style.width = "";
    tooltipDiv.style.height = "";

    tooltipDiv.innerHTML = tabPodpowiedzi[id - 1];
    tooltipDiv.style.top = yOff + "px";
    tooltipDiv.style.left = xOff + "px";
    tooltipDiv.style.display = "block";

    krokX = Math.floor(tooltipDiv.offsetWidth / lKroków);
    krokY = Math.floor(tooltipDiv.offsetHeight / lKroków);

    tooltipDiv.style.width = "0px";
    tooltipDiv.style.height = "0px";
    timerId = setTimeout("show("+lKroków+", 'show');", timeout);
  }
  // dalsza część skryptu
```

Tablica `tabPodpowiedzi` powinna zawierać listę podpowiedzi, tu nic nie zmieniło się w stosunku do skryptu 59. Dalej znajdują się jednak definicje nowych zmiennych globalnych:

- ◆ `lKroków` — liczba kroków, w których podpowiedź przejdzie ze stanu zminimalizowanego (niewidocznego) do pełnego (pełnych rozmiarów), innymi słowy: liczba faz animacji;
- ◆ `timeout` — czas pomiędzy kolejnymi fazami animacji;
- ◆ `krokX` — liczba pikseli, o którą należy zwiększyć szerokość warstwy z podpowiedzią w każdej fazie animacji (w każdym kroku);
- ◆ `krokY` — liczba pikseli, o którą należy zwiększyć wysokość warstwy z podpowiedzią w każdej fazie animacji;
- ◆ `timerId` — identyfikator timera obsługującego animację.

Początek kodu funkcji `showTooltip` jest taki sam jak na listingu 7.3. W dalszej części pojawiło się jednak kilka znaczących zmian. Po pobraniu odwołania do warstwy `tooltipDiv` i zapisaniu go w zmiennej `tooltipDiv` usuwane są właściwości `width` i `height` warstwy. Odbywa się to przez przypisanie właściwościom pustych ciągów znaków:

```
tooltipDiv.style.width = "";
tooltipDiv.style.height = "";
```

To bardzo ważne. Dzięki temu w kolejnym kroku, po przypisaniu warstwie treści pobranej z tablicy `tabPodpowiedzi`, wysokość i szerokość zostaną dopasowane automatycznie. Następnie ustalana jest pozycja warstwy i jest ona wyświetlana. W kolejnym kroku ustalany jest stan zmiennych `krokX` i `krokY`. Ich wartości wynikają ze wzorów:

$$krokX = \frac{\text{szerokość warstwy}}{\text{liczba kroków}} \quad \text{ i } \quad krokY = \frac{\text{wysokość warstwy}}{\text{liczba kroków}}$$

Szerokość warstwy w pikselach jest pobierana z właściwości `offsetWidth`, a wysokość — z właściwości `offsetHeight`. Dopiero po wykonaniu obliczeń można ustalić początkową szerokość i wysokość na 0 pikseli, co jest wykonywane przez dwie kolejne instrukcje.

Na końcu kodu funkcji jest uruchamiany timer, który po czasie wskazanym przez `timeout` wywoła funkcję `show` wykonującą właściwą animację. Identyfikator timera jest zapisywany w zmiennej `timerId`, dzięki czemu animacja będzie mogła być przerwana w każdej chwili. Funkcja `show` otrzymuje w postaci pierwszego argumentu wartość określającą liczbę faz animacji (kroków). Drugi argument (ciąg `show`) wskazuje, że podpowiedź ma się pojawiać na ekranie (a nie zniknąć).

Treść funkcji `show` została przedstawiona na listingu 7.5.

Listing 7.5. *Treść funkcji `show`*

```
function show(licznik, showhide)
{
    var obj = document.getElementById("tooltipDiv");
    if(!obj) return;
    if(showhide == 'show'){
        var width = krokX + parseInt(obj.style.width);
        var height = krokY + parseInt(obj.style.height);
        obj.style.width = width + "px";
        obj.style.height = height + "px";
    }
    else{
        var width = parseInt(obj.style.width) - krokX;
        var height = parseInt(obj.style.height) - krokY;
        if(width < 0) width = 0;
        if(height < 0) height = 0;
        obj.style.width = width + "px";
        obj.style.height = height + "px";
    }
    if(--licznik > 0){
        timerId = setTimeout("show("+licznik+", '"+showhide+"');", timeout);
    }
    else if(showhide != 'show'){
        obj.style.display = "none";
    }
}
```

Funkcja przyjmuje dwa argumenty. Pierwszy (licznik) wskazuje, ile jeszcze pozostało wywołań do całkowitego zmaksymalizowania lub zminimalizowania warstwy. Drugi (showhide) to ciąg znaków określający, czy warstwa ma się pojawiać (show) czy znikać (hide). Najpierw pobierane jest odwołanie do warstwy tooltipDiv, a następnie modyfikowane są właściwości width (szerokość) i height (wysokość) obiektu style. Wartości, które mają być przypisane wymienionym właściwościom, znajdują się w pomocniczych zmiennych width i height.

Jeżeli warstwa ma się pojawiać, czyli zwiększać swoje rozmiary (argument showhide równy show), do aktualnej szerokości (parseInt(obj.style.width)) jest dodawana wartość globalnej zmiennej krokX, a do aktualnej wysokości (parseInt(obj.style.height)) — wartość globalnej zmiennej krokY. Gdy warstwa ma znikać z ekranu, czyli zmniejszać swoje rozmiary (argument showhide równy hide¹), od aktualnej szerokości jest odejmowana wartość krokX, a od aktualnej wysokości — wartość krokY. Przy odejmowaniu mogą się pojawić wartości ujemne, a takie nie mogą być używane we właściwościach width i height, i jeśli taka sytuacja zostanie wykryta, są przeprowadzane odpowiednie korekty.

Na końcu kodu zmienna licznik jest zmniejszana o 1. Gdy po zmniejszeniu jest wciąż większa od 0, oznacza to, że należy kontynuować wywoływanie funkcji show, dlatego też ponownie jest ustawiany timer. Jeżeli jednak po modyfikacji zmienna licznik jest mniejsza od 0 lub równa 0, a jednocześnie parametr showhide wskazuje, że warstwa jest chowana (showhide różne od show), wyświetlanie warstwy jest wyłączane przez przypisanie wartości none właściwości display obiektu style.

Ostatnią z funkcji jest closeTooltip. Jej treść została zaprezentowana na listingu 7.6.

Listing 7.6. Treść funkcji show

```
function closeTooltip()
{
    var tooltipDiv = document.getElementById("tooltipDiv");
    if(!tooltipDiv) return;
    if(timerId){
        clearTimeout(timerId);
        timerId = null;
    }
    krokX = tooltipDiv.offsetWidth / 1Kroków;
    krokY = tooltipDiv.offsetHeight / 1Kroków;
    timerId = setTimeout("show('+"+1Kroków+"','hide');", timeout);
}
```

Funkcja jest wywoływana w odpowiedzi na zdarzenie mouseout, czyli gdy kursor myszy opuścił obszar danego elementu i w związku z tym odpowiedź ma być schowana. Ponieważ przy wywołaniu tej funkcji nie można mieć pewności, że warstwa została całkowicie rozwinięta (wciąż może trwać proces rozwijania), po stwierdzeniu, że identyfikator timera (timerId) nie jest pusty, następuje zatrzymanie zegara przez wywołanie metody clearInterval, a dopiero potem są wykonywane dalsze czynności. Są to:

¹ Formalnie rzecz ujmując, chodzi o parametr showhide zawierający ciąg znaków różny od ciągu show.

- ♦ obliczenie wartości dla zmiennych `krokX` i `krokY`;
- ♦ uruchomienie timera wywołującego funkcję `show`.

Te czynności są wykonywane na takiej samej zasadzie jak w funkcji `showTooltip`. Ponieważ jednak w tym przypadku warstwa ma być chowana (zwijana), drugi argument wywołania funkcji `show` ma wartość `hide`.

Skrypt 61. [C][E][F][O][S] Pojawiająca się podpowiedź

W skrypcie 59. podpowiedź pojawiała się natychmiast po wskazaniu kursorem danego elementu witryny. Można jednak spowodować, aby ukazywała się stopniowo. Wystarczy odpowiednio manipulować przezroczystością warstwy `tooltipdiv`. Takie właśnie zadanie wykonuje skrypt 61. Jego kod został oparty na pomysłach z przykładów 59. i 60. Kody HTML i CSS pozostaną takie same jak w przykładzie 59. (listingi 7.1 i 7.2). Zmienne globalne oraz funkcja `showTooltip` będą natomiast miały postać przedstawioną na listingu 7.7.

Listing 7.7. Początek skryptu 61.

```
<script type="text/javascript">
  var tabPodpowiedzi = [/*definicje podpowiedzi*/];
  var timeout = 30;
  var krok = 0.1;
  var bieżącyKrok = 0;
  var timerId = null;
  function showTooltip(id, obj)
  {
    // tutaj cała treść funkcji showTooltip z listingu 7.3

    bieżącyKrok = 0;

    tooltipDiv.style.opacity = "0.0";
    tooltipDiv.style.filter = "alpha(opacity=0)";
    timerId = setTimeout("show('show');", timeout);
  }
  // dalsza część skryptu
```

Zmienne globalne `tabPodpowiedzi`, `timeout` i `timerId` mają takie samo znaczenie jak w przykładzie 60. Zmienna `krok` określa stopień zmiany przezroczystości w każdym kroku animacji, natomiast `bieżącyKrok` — bieżącą wartość atrybutu określającego przezroczystość warstwy. Im mniejsza wartość zapisana w `krok`, tym mniejsze różnice między kolejnymi stopniami przezroczystości i dłuższa animacja (zakładając tę samą wartość parametru `timeout`).

Cały początek funkcji `showTooltip` jest identyczny z tym z listingu 7.3 — wykonywane jest bowiem takie samo zadanie. Na końcu konieczne było jednak dodanie kilku

instrukcji. Pierwsza ustala początkową wartość zmiennej `bieżącyKrok`. Jest to 0, bowiem warstwa ma się pojawiać (0 to pełna przezroczystość warstwy). Kolejne dwie instrukcje ustalają początkową przezroczystość. Dzięki nim przy pierwszym wywołaniu nie nastąpi nieprzyjemne „mrugnięcie” warstwy. Zamiast nich można też zdefiniować bezpośrednio style CSS.

Modyfikowane są właściwości `opacity` i `filter` obiektu `style`. Jest to konieczne, bowiem przeglądarki z rodziny Internet Explorer obsługują tylko niestandardową właściwość `filter`, a nie rozpoznają prawidłowej właściwości `opacity`. Przypisanie ciągu 0.0 do `opacity` lub ciągu `alpha(opacity=0)` do `filter` oznacza, że warstwa ma być całkowicie przezroczysta.

Na końcu kodu ustawiany jest timer wywołujący funkcję `show`. Jej zadaniem jest wykonanie animacji, czyli stopniowe zwiększanie lub zmniejszanie przezroczystości (zależnie od stanu argumentu). Treść funkcji `show` została przedstawiona na listingu 7.8.

Listing 7.8. Treść funkcji `show`

```
function show(showhide)
{
    var obj = document.getElementById("tooltipDiv");
    if(!obj) return;
    if(showhide == 'show'){
        bieżącyKrok += krok;
    }
    else{
        bieżącyKrok -= krok;
    }
    if(bieżącyKrok > 1) bieżącyKrok = 1;
    if(bieżącyKrok < 0) bieżącyKrok = 0;

    obj.style.opacity = bieżącyKrok.toFixed(2);
    obj.style.filter = "alpha(opacity:" +
        (bieżącyKrok * 100).toFixed(0) + ")";
    if(bieżącyKrok != 0 && bieżącyKrok != 1)
        timerId = setTimeout("show('"+showhide+"');", timeout);
    else timerId = null;
}
```

Funkcja wykonuje operacje na warstwie `tooltipDiv`, pobiera więc odwołanie do niej, a następnie w zależności od stanu parametru `showhide` zmienia wartość zmiennej `bieżącyKrok`. Jeżeli `showhide` jest równe `show`, oznacza to, że podpowiedź ma się pojawiać na ekranie, a zatem do `bieżącyKrok` jest dodawana wartość zapisana w `krok`. W przeciwnym wypadku od wartości zapisanej w `bieżącyKrok` jest odejmowana wartość zapisana w `krok`.

Po wykonaniu tej operacji niezbędne jest stwierdzenie, czy nie nastąpiło przekroczenie zakresu. Przezroczystość jest opisywana przez wartości od 0 do 1, gdzie 0 oznacza całkowitą przezroczystość, a 1 — brak przezroczystości. Jeżeli zatem zmienna `bieżącyKrok` jest większa od 1, jest jej przypisywana wartość 1, a jeśli jest mniejsza od 0, jest jej przypisywana wartość 0. Po dokonaniu tych korekt można dokonać modyfikacji właściwości obiektu `style`.

Właściwości `opacity` jest przypisywana wartość zapisana w `bieżącyKrok`. Aby uniknąć przypisania nieprawidłowych danych wynikających z niedokładności reprezentacji liczb rzeczywistych, za pomocą metody `toFixed` wartość jest zaokrąglana do dwóch miejsc po przecinku. Podobna sytuacja ma miejsce w przypadku właściwości `filter`. Tu dodatkowo wartość zapisana w `bieżącyKrok` jest mnożona przez 100 (w Internet Explorerze całkowita przezroczystość to 0, a brak przezroczystości to 100). Ponieważ uzyskana wartość ma być całkowita, używane jest wywołanie `toFixed(0)`.

Po dokonaniu przypisań badane jest to, czy proces zmiany przezroczystości ma być kontynuowany. Będzie tak wtedy, gdy `bieżącyKrok` jest różne i od 0, i od 1. W takiej sytuacji wykonywane jest ponowne wywołanie metody `setTimeout`. W przeciwnym wypadku zerowany jest timer (zmiennej `timerId` jest przypisywana wartość `null`).

Ostatnia z funkcji — `closeTooltip` — zajmuje się ukryciem warstwy `tooltipDiv`, czyli rozpoczęciem procesu stopniowego zmniejszania przezroczystości. Jej treść jest widoczna na listingu 7.9.

Listing 7.9. Treść funkcji `closeTooltip`

```
function closeTooltip()
{
    var tooltipDiv = document.getElementById("tooltipDiv");
    if(!tooltipDiv) return;
    if(timerId){
        clearTimeout(timerId);
        timerId = null;
    }
    timerId = setTimeout("show('hide');", timeout);
}
```

Po pobraniu odwołania do warstwy i stwierdzeniu, że ona istnieje, badany jest stan timera. Gdy jest aktywny, co oznaczałoby, że trwa odkrywanie warstwy, timer jest wyłączany za pomocą metody `clearTimeout`, a zmienna `timerId` otrzymuje wartość `null`. Proces ukrywania warstwy jest rozpoczynany przez wywołanie metody `setInterval`. W tym wywołaniu funkcji `show` jest przekazywany parametr `hide`.

Skrypt 62. [C][E][F][O][S]

Warstwa (opis, okno)

przesuwana za pomocą myszy

Na stronie można umieścić element, który będzie mógł być dowolnie przesuwany przez użytkownika witryny. Będzie się to odbywało za pomocą myszy. Element otrzyma więc funkcjonalność typu przeciągnij i upuść (drag & drop). Najlepiej, aby była to zwykła warstwa. Jej treść może być dowolna. W części HTML wystarczy więc umieścić kod przedstawiony na listingu 7.10.

Listing 7.10. Kod sekcji `<body>` skryptu 62.

```
<body>
  <div id="divDane" onmousedown="startDrag(this);">
    Tę warstwę można przesuwać za pomocą myszy!
  </div>
  <!-- dalsza część witryny -->
</body>
```

Warstwa została zdefiniowana za pomocą standardowego znacznika `<div>` i zawiera przykładowy tekst. Wygląda zatem bardzo niepozornie. Ważne jest, że atrybut `onmousedown` zawiera wywołanie funkcji `startDrag`, a funkcji tej jest przekazywany obiekt bieżący (`this`), czyli wskazanie do warstwy. To spowoduje, że będzie ona mogła być przesuwana po oknie przeglądarki. Oczywiście niezbędny jest do tego odpowiedni skrypt. Treść jego pierwszej części znajduje się na listingu 7.11.

Listing 7.11. Pierwsza część skryptu umożliwiającego przesuwanie warstwy

```
<script type="text/javascript">
  var draggedBox = null;
  var offset = null;

  document.onmouseup = mouseUp;
  document.onmousemove = mouseMove;

  function mousePos(evt)
  {
    evt = evt || window.event;

    if(evt.pageX){
      return {x:evt.pageX, y:evt.pageY};
    }
    else if(evt.clientX){
      return{
        x:evt.clientX + document.body.scrollLeft + document.body.clientLeft,
        y:evt.clientY + document.body.scrollTop - document.body.clientTop
      };
    }
    else{
      return {x:0, y:0};
    }
  }
  //dalsza część skryptu
```

Na początku kodu właściwościom `onmouseup` i `onmousemove` obiektu `document` przypisywane są procedury obsługi:

```
document.onmouseup = mouseUp;
document.onmousemove = mouseMove;
```

Dzięki temu po zwolnieniu przycisku myszy zostanie wykonana funkcja `mouseUp`, a przy każdej zmianie położenia — funkcja `mouseMove` (obie zostaną omówione w dalszej części opisu). Tworzone są też dwie zmienne globalne:

- ♦ `draggedBox` — odwołanie do przesuwanej warstwy;
- ♦ `offset` — obiekt określający przesunięcie kursora myszy.

Dalej została umieszczona pomocnicza funkcja `mousePos`. Ma ona odczytać współrzędne kliknięcia myszy i zwrócić je w postaci obiektu o właściwościach `x` i `y`. Jest to konieczne, bowiem skoro mamy przesuwać warstwę, musimy wiedzieć, jaka jest aktualna pozycja myszy.

Niestety, poszczególne przeglądarki rządzą się tu swoimi własnymi prawami. Przede wszystkim obiekt zawierający dane dotyczące zdarzenia może być przekazywany funkcji obsługującej (tym charakteryzuje się np. Firefox), ale może też być właściwością event obiektu `window` (tym charakteryzuje się np. Internet Explorer). Aby to ujednolicić, stosujemy instrukcję:

```
evt = evt || window.event;
```

Po jej wykonaniu zmienna `evt` będzie zawierała obiekt związany ze zdarzeniem. To jednak nie koniec kłopotów. Otóż właściwe współrzędne mogą się znajdować we właściwościach `pageX` i `pageY`:

```
if(evt.pageX){
```

ale również we właściwościach `clientX` i `clientY`. To jednak wciąż nie wszystko. Otóż w tym drugim przypadku (Internet Explorer) podane dane uwzględniają jedynie widoczne okno dokumentu, a nie cały dokument, pominięte są również rozmiary ramki. Niezbędne jest więc wykonanie dodatkowych obliczeń. Stąd współrzędna `x` obliczana jest ze wzoru:

```
evt.clientX + document.body.scrollLeft + document.body.clientLeft
```

a współrzędna `y`:

```
evt.clientY + document.body.scrollTop - document.body.clientTop
```

Dopuszczana jest także taka sytuacja, że mamy do czynienia z przeglądarką, która nie udostępnia żadnego z opisanych obiektów. Nie można wtedy określić współrzędnych, dlatego też otrzymują wartość 0:

```
return {x:0, y:0};
```

Na listingu 7.12 została przedstawiona funkcja `startDrag` wywoływana, gdy nad warstwą, która ma być przesuwana, zostanie wciśnięty przycisk myszy.

Listing 7.12. Treść funkcji `startDrag`

```
function startDrag(obj)
{
    if(!obj) return;
    draggedBox = obj;
    xOff = 0;
    yOff = 0;
    while(obj.offsetParent){
        xOff += obj.offsetLeft;
        yOff += obj.offsetTop;
    }
}
```

```
    obj = obj.offsetParent;
  }

  obj.onmousedown = function(evt)
  {
    coords = mousePos(evt);
    xOff = coords.x - xOff;
    yOff = coords.y - yOff;
    offset = {x:xOff, y:yOff};
  }
}
```

Funkcja przypisuje zmiennej globalnej `draggedBox` odwołanie do klikniętego obiektu (w tym przypadku warstwy `divDane`). Dzięki temu pozostałe funkcje dostaną informacje, czy i co mają przesuwać. Jej drugim (i dużo bardziej złożonym) zadaniem jest natomiast obliczenie współrzędnych kliknięcia względem lewego górnego rogu tej warstwy (innymi słowy, trzeba się dowiedzieć, w którym miejscu warstwy nastąpiło kliknięcie) i zapisanie tych wartości w obiekcie `offset`. W pętli `while` odnajdywane są zatem bezwzględne współrzędne lewego górnego rogu warstwy (odbywa się to przez dodawanie rozmiarów wszystkich kolejnych elementów sąsiadujących z nią) i zapisywane w zmiennych `xOff` i `yOff`.

Następnie niezbędne jest uzyskanie współrzędnych kliknięcia. Trzeba więc przypisać zdarzeniu `onmousedown` obiektu `obj` funkcję obsługującą zdarzenie `mousedown` (jest to anonimowa funkcja wewnętrzna):

```
obj.onmousedown = function(evt)
```

W tej funkcji odczytywane są bezwzględne współrzędne kliknięcia:

```
coords = mousePos(evt);
```

oraz obliczane współrzędne względne (względem lewego górnego rogu warstwy):

```
xOff = coords.x - xOff;
yOff = coords.y - yOff;
```

Wyliczone dane są zapisywane w zmiennej `offset`:

```
offset = {x:xOff, y:yOff};
```

Do obsługi funkcjonalności *drag & drop* potrzebne są jeszcze dwie funkcje: `mouseUp` i `mouseMove`. Zostały przedstawione na listingu 7.13.

Listing 7.13. Treść funkcji `mouseUp` i `mouseMove`

```
function mouseUp(evt)
{
  draggedBox = null;
}

function mouseMove(evt)
{
  coords = mousePos(evt);
```

```
if(draggedBox){
    draggedBox.style.position = "absolute";
    draggedBox.style.top = (coords.y - offset.y) + "px";
    draggedBox.style.left = (coords.x - offset.x) + "px";
    return false;
}
}
```

Zadaniem funkcji `mouseUp` jest zakończenie przesuwania warstwy. Ta czynność jest wykonywana w wyjątkowo prosty sposób — zmiennej `draggedBox` jest przypisywana wartość `null`.

Ostatnia funkcja to `mousemove`. Jej zadanie to zmiana współrzędnych obiektu wskazywanego przez `draggedBox`, tak aby przesuwał się wraz z ruchami kursora myszy. Współrzędne pobierane są przez wywołanie funkcji `mousePos` i zapisywane w zmiennej `coords`, a następnie, po sprawdzeniu, czy zmienna `draggedBox` jest różna od `null` (czyli wskazuje jakiś obiekt i ma on być przesuwany), modyfikowane są właściwości `left` i `top`. Właściwości `top` (współrzędna *y* lewego górnego rogu) przypisywana jest wartość wynikająca z odjęcia od współrzędnej *y* kliknięcia wartości wskazywanej przez `offset.y`, a właściwości `left` (współrzędna *x* lewego górnego rogu) — wartość wynikająca z odjęcia od współrzędnej *x* kliknięcia wartości wskazywanej przez `offset.x`.

Skrypt 63.

[C][E][F][O][S]

Boks ze zmienną treścią (reklamy, wiadomości itp.)

W skrypcie 63. zostanie pokazane, w jaki sposób umieścić na witrynie boks ze zmieniającą się treścią. Może on zawierać reklamy, wiadomości, porady dnia i temu podobne treści. Boksem będzie po prostu odpowiednio przygotowana warstwa, której zawartość będzie wymieniana cyklicznie przez skrypt JavaScript. Kod HTML proponowanego rozwiązania został zamieszczony na listingu 7.14.

Listing 7.14. Kod sekcji `<body>` skryptu 63.

```
<body onload="start('advBox');">
  <div id="mainDiv">
    To jest główna warstwa danych.
  </div>
  <div id="advBox">
  </div>
</body>
```

W sekcji `<body>` zostały umieszczone dwie warstwy: `mainDiv`, służąca jako główna warstwa strony, oraz `advBox`, służąca jako boks do wyświetlania — powiedzmy — reklam. Zdarzeniu `load` sekcji `<body>` została przypisana procedura obsługi w postaci funkcji

start, która rozpoczyna proces wyświetlania danych. Treść tej funkcji, wraz z definicjami zmiennych globalnych, została przedstawiona na listingu 7.15.

Listing 7.15. Treść funkcji start

```
<script type="text/javascript">
  var reklamy = [
    "To jest pierwsza reklama.",
    "To jest druga reklama.",
    "To jest trzecia reklama."
  ];
  var timeout = 3;
  var licznik = 0;
  function start(id)
  {
    var div = document.getElementById(id);
    if(!div) return;
    div.innerHTML = reklamy[licznik];
    if(++licznik >= reklamy.length) licznik = 0;
    setTimeout("start('"+id+"');", timeout * 1000);
  }
</script>
```

Wyświetlane reklamy powinny zostać zdefiniowane w globalnej tablicy reklamy. Może być ich dowolnie wiele, mogą też zawierać kod HTML (np. odnośniki czy też znaczniki formatujące tekst). Oprócz tablicy istnieją dwie dodatkowe zmienne globalne: timeout i licznik. Pierwsza określa w sekundach czas, w którym dana reklama ma pozostawać na ekranie, a druga — indeks aktualnie wyświetlanej reklamy.

Funkcja start przyjmuje jeden argument określający identyfikator warstwy, na której ma się pojawiać treść pobierana z tablicy reklamy. Wskazanie do warstwy jest pobierane za pomocą metody getElementById i zapisywane w zmiennej pomocniczej div. Zawartość pobranego elementu witryny jest wymieniana przez modyfikację właściwości innerHTML. Jest jej przypisywana komórka tabeli reklamy wskazywana przez indeks licznik:

```
div.innerHTML = reklamy[licznik];
```

Następnie licznik jest zwiększany i badane jest to, czy osiągnął wartość maksymalną (rozmiar tablicy reklamy minus jeden). Jeśli tak jest, licznik jest zerowany:

```
if(++licznik >= reklamy.length) licznik = 0;
```

Na zakończenie za pomocą wywołania metody setTimeout ustawiany jest timer powodujący ponowne wywołanie funkcji start po czasie określonym jako timeout * 1000. Dzięki temu mnożeniu zmienna timeout może wyrażać czas w sekundach, a nie milisekundach, co jest wygodniejsze.

Boks reklamowy powinien być odpowiednio sformatowany i umieszczony w zadanym miejscu strony. Najlepiej więc przygotować dodatkowo odpowiednie style CSS. Mogą mieć postać przedstawioną na listingu 7.16. Zostały zdefiniowane kolor tła (background-color) oraz obramowanie (border), a także wysokość (height) i szerokość

(width). Warstwa została zaś umieszczona w prawym (right) górnym (top) rogu. W przypadku przeglądarki Internet Explorer w wersji 6. zamiast pozycjonowania ustalonego (position: fixed;), trzeba użyć bezwzględnego (position: absolute;).

Listing 7.16. Przykładowe style CSS dla boksu reklamowego

```
<style type="text/css">
#advBox
{
    position: fixed;
    background-color: #F0F0F0;
    border: 1px solid black;
    margin: 0px;
    padding: 4px;
    right: 10px;
    top: 10px;
    width: 120px;
    height: 60px;
}
</style>
```

Skrypt 64. [C][E][F][O][S]

Boks z efektem skrolowania

W przykładzie 63. został zaprezentowany boks ze zmienną treścią. Skrypt 64. również realizuje takie zadanie, tym razem jednak pojawi się tzw. efekt przejścia, w związku z czym nowa wiadomość będzie zastępowała starą, najeżdżając na nią np. od lewej strony. Taka animacja uatrakcyjni wizualną stronę witryny. Kod HTML można pozostawić w dokładnie takiej samej postaci jak na listingu 7.14. Reguła CSS dla warstwy advBox również będzie taka sama, jak było to w skrypcie 63. (listing 7.16), dodać należy jedynie definicję dla cechy overflow:

```
overflow: hidden;
```

Dzięki niej zawartość boksu nie będzie wystawała poza jego krawędzie. Potrzebna też będzie reguła definiująca klasę advBoxDiv:

```
.advBoxDiv{
    background-color: #F0F0F0;
    position: absolute;
}
```

Ta klasa będzie dotyczyła pomocniczych warstw, dzięki którym zostanie zrealizowany cały efekt. Każda reklama otrzyma bowiem swoją warstwę, która będzie musiała mieć pozycjonowanie bezwzględne (position: absolute) oraz kolor tła odpowiadający kolorowi warstwy advBox. Dlatego też warstwom z reklamami będzie przypisywana przedstawiona wyżej klasa advBoxDiv.

Pierwsza część skryptu została zaprezentowana na listingu 7.17.

Listing 7.17. *Początek skryptu 64.*

```
<script type="text/javascript">
  var reklamy = [/*definicje reklam*/];
  var currItem = 0;
  var timeout = 5;
  var slideSpeed = 10;
  var slideStep = 3;
  var advBox = null;
  var divs = null;
  var padding = 4;
  function start(id)
  {
    advBox = document.getElementById(id);
    if(!advBox) return;

    var count = 0;
    for(var i = 0; i < reklamy.length; i++){
      var div = document.createElement("div");
      if(count > 0) div.style.display = "none";
      else div.style.display = "block";

      div.id = "advBoxDiv" + count++;
      div.className = "advBoxDiv";
      div.innerHTML = reklamy[i];

      advBox.appendChild(div);
      div.style.width = "100%";
      div.style.height = "100%";
    }
    divs = advBox.getElementsByTagName("div");
    setTimeout("startShow();", timeout * 1000);
  }
  //dalsza część kodu skryptu
```

Na początku kodu znajdują się definicje ośmiu zmiennych globalnych:

- ◆ **reklamy** — tablica z definicjami reklam, wiadomości lub innych treści;
- ◆ **currItem** — numer bieżącej reklamy;
- ◆ **timeout** — czas w sekundach pomiędzy kolejnymi zmianami treści;
- ◆ **slideSpeed** — czas pomiędzy kolejnymi fazami animacji (szybkość przesuwu warstw);
- ◆ **slideStep** — liczba pikseli, o którą należy przesunąć warstwę z treścią w jednej fazie animacji;
- ◆ **advBox** — wskazanie do warstwy z boksem;
- ◆ **divs** — tablica zawierająca warstwy z reklamami;
- ◆ **padding** — parametr określający wypełnienie zastosowane w boksie (w praktyce oznacza to liczbę pikseli pomiędzy lewą krawędzią boks a prezentowaną treścią).

Zadaniem funkcji `start` jest utworzenie warstw z treścią odpowiadającą poszczególnym reklamom pobranym z tablicy `reklamy`, dołączenie ich do boksu (warstwy) o identyfikatorze przekazanym w postaci argumentu `id` oraz rozpoczęcie prezentacji.

Po pobraniu odwołania do warstwy przekazanej w postaci parametru `id`, zapisaniu go w zmiennej `advBox` i sprawdzeniu, że taki element strony faktycznie istnieje, rozpoczynana jest pętla `for` generująca warstwy dla poszczególnych reklam. Pojedyncza warstwa jest tworzona za pomocą metody `createElement` obiektu `document` i jest zapisywana w zmiennej pomocniczej `div`. Każdej warstwie jest nadawany identyfikator (wartość właściwości `id` obiektu) o nazwie zgodnej ze schematem `advBoxDivX`, gdzie `X` jest kolejnym numerem warstwy (wartość zmiennej `count`).

Treść warstwy reprezentowana przez właściwość `innerHTML` jest pobierana z tablicy `reklamy` spod indeksu wskazywanego przez zmienną iteracyjną `i`. Atrybut `className` otrzymuje za każdym razem tę samą wartość — `advBoxDiv` — dzięki czemu ogólny styl warstw będzie mógł być modyfikowany przez zmiany w jednej tylko klasie CSS. W kodzie są jednak nadawane właściwości `width` i `height`, obie określone na 100 proc. Ponieważ początkowo na ekranie powinna być wyświetlana tylko pierwsza warstwa, jej właściwość `display` jest ustawiana na `block`, a we wszystkich pozostałych przypadkach ta właściwość otrzymuje wartość `none`.

Za pomocą metody `appendChild` warstwy są dodawane do elementu wskazywanego przez `advBox`:

```
advBox.appendChild(div);
```

Po zakończeniu pętli wszystkie warstwy dodane do elementu `advBox` są pobierane i zapisywane w tablicy (kolekcji) `divs`:

```
divs = advBox.getElementsByTagName("div");
```

Ostatnia instrukcja wywołuje timer rozpoczynający proces wyświetlania. Spowoduje to wywołanie funkcji `startShow` po czasie wskazywanym przez `timeout`. Ponieważ wartość globalnej zmiennej `timeout` ma wyrażać czas w sekundach, a w wywołaniu `setTimeout` należy podać milisekundy, wykonywane jest mnożenie `timeout * 1000`.

Treść funkcji `startShow` została przedstawiona na listingu 7.18.

Listing 7.18. Treść funkcji `startShow`

```
function startShow()
{
    if(!advBox || !divs || divs.length < 1) return;

    if(++currItem >= divs.length) currItem = 0;
    if(currItem == 0)
        prevItem = divs.length - 1;
    else
        prevItem = currItem - 1;

    var currDiv = divs[currItem];
    var prevDiv = divs[prevItem];

    var xPos = parseInt(currDiv.parentNode.offsetWidth);
```

```
currDiv.style.left = xPos + "px";
currDiv.style.display = "block";

currDiv.style.zIndex = 2;
prevDiv.style.zIndex = 1;

slideDiv(prevItem, currItem);
}
```

Zadaniem funkcji jest określenie, która z warstw jest aktualnie wyświetlona, a która ma się za chwilę pojawić na ekranie, ustalenie związanych z tym parametrów oraz wywołanie funkcji wykonującej animację. Najpierw sprawdzane jest, czy istnieją obiekty `advBox` i `divs` oraz czy liczba zapisanych w `divs` warstw jest większa od 1. Jeśli dane są poprawne, wykonywane są dalsze czynności. Zwiększana jest zmienna `currItem` wskazująca aktualnie wyświetlaną warstwę (`++currItem`). Jeśli osiągnie przy tym wartość graniczną (`divs.length`), jest zerowana (`currItem = 0`). Dzięki temu pokaz zostanie zapętlony, czyli po warstwie ostatniej ponownie zostanie zaprezentowana pierwsza, a więc proces wyświetlania rozpocznie się od nowa.

Po ustaleniu numeru warstwy, która ma być wyświetlona (`currItem`), określany jest numer warstwy, która obecnie widnieje na ekranie, a zostanie przykryta przez nową (`prevItem`). Możliwe są przy tym dwa przypadki. Jeżeli warstwa do wyświetlenia ma indeks 0, to warstwa wyświetlana ma indeks ostatniej komórki tablicy `divs` (`divs.length - 1`). W każdym innym przypadku warstwa aktualnie wyświetlana będzie miała numer wynikający z odejmowania `currItem - 1`. Po ustaleniu numerów odwołania do warstw są pobierane z tablicy `divs` i zapisywane w zmiennych `currDiv` i `prevDiv`.

Warstwa z nową reklamą ma wjechać do boks od jego prawej krawędzi. Zatem pozycja początkowa musi być przy prawym brzegu warstwy nadrzędnej (`advBox`). To oznacza, że właściwość `left` warstwy `currDiv` powinna mieć wartość równą długości (`offsetWidth`) warstwy nadrzędnej (`parentNode`). Właściwość `offsetWidth` jest zatem odczytywana, przetwarzana na wartość całkowitą i zapisywana w zmiennej pomocniczej `xPos`. Następnie wartość `xPos` uzupełniona od ciąg `px` jest przypisywana właściwości `left` obiektu `style` warstwy `currDiv`.

Konieczne jest też ustalenie kolejności wyświetlania warstw. Ta, która stanie się bieżącą (`currDiv`), musi się znajdować nad warstwą aktualnie wyświetlaną (`prevDiv`). Dlatego właściwość `zIndex` obiektu `style` w warstwie `currDiv` musi być większa niż w warstwie `prevDiv`:

```
currDiv.style.zIndex = 2;
prevDiv.style.zIndex = 1;
```

Na końcu kodu wywoływana jest funkcja `slideDiv`, która wykonuje animację najazdu jednej warstwy na drugą. Kod funkcji został przedstawiony na listingu 7.19.

Listing 7.19. Treść funkcji `slideDiv`

```
function slideDiv(prevItem, currItem)
{
    if(!advBox || !divs) return;

    var currDiv = divs[currItem];
```

```

var prevDiv = divs[prevItem];

var xPos = parseInt(currDiv.style.left);
xPos -= slideStep;
if(xPos <= padding){
    xPos = padding;
    prevDiv.style.display = "none";
    currDiv.style.left = xPos + "px";
    setTimeout("startShow();", timeout * 1000);
}
else{
    currDiv.style.left = xPos + "px";
    setTimeout("slideDiv(prevItem, currItem);", slideSpeed);
}
}

```

Najpierw sprawdzane jest to, czy istnieją obiekty `advBox` i `divs`. Jeśli istnieją, z tablicy `divs` pobierane są i zapisywane w pomocniczych zmiennych `prevDiv` i `currDiv` odwołania do warstw: aktualnie wyświetlanej (`prevDiv`) i nowej — tej, która ma być wyświetlona (`currDiv`). Następnie pozioma pozycja warstwy `currDiv` (właściwość `left`) jest odczytywana, przetwarzana na wartość całkowitą i zapisywana w zmiennej `xPos`. Od tej wartości odejmowana jest ta zapisana w `slideStep`, czyli liczba pikseli, o którą warstwa ma być przesunięta w danej fazie ruchu.

Dalej znajduje się instrukcja warunkowa `if`, która stwierdza, czy ruch ma być kontynuowany czy ma się zakończyć. Zakończenie ma nastąpić wtedy, gdy wartość `xPos` będzie mniejsza lub równa wartości zapisanej w `padding`, czyli gdy warstwa `currDiv` znajdzie się w odległości `padding` pikseli od lewej krawędzi boks (warstwy) `advBox`. Wtedy wyłączane jest wyświetlanie poprzedniej warstwy:

```
prevDiv.style.display = "none";
```

oraz wyrównywane położenie warstwy aktualnej:

```
xPos = padding;
```

Włączany jest także timer powodujący kolejne wywołanie funkcji `startShow` po czasie `timeout` sekund.

Jeżeli jednak ruch warstwy ma być kontynuowany, właściwości `left` obiektu `style` przypisywana jest bieżąca wartość zmiennej `xPos`. Tym samym warstwa jest przesuwana w lewo. Ustawiany jest także timer powodujący ponowne wywołanie funkcji `slideDiv` po czasie `slideSpeed` milisekund.

Skrypt 65. Boks z efektem przejścia

[C][E][F][O][S]

Ostatni skrypt w tym rozdziale realizuje efekt boks, w którym treść jest zmieniana poprzez efekt przenikania. Bieżące dane będą znikły, a w tym samym czasie będą się pojawiały nowe. Łatwo się domyślić, że w tym celu trzeba będzie po prostu odpowied-

nio manipulować przezroczystością warstw. Ogólna struktura może pozostać taka sama jak w przykładzie 64., trzeba będzie jednak dokonać modyfikacji w zmiennych globalnych oraz zmienić funkcję realizującą efekt wymiany warstw. Początek skryptu będzie więc wyglądał tak, jak zaprezentowano to na listingu 7.20.

Listing 7.20. *Początek kodu skryptu 65.*

```
<script type="text/javascript">
  var reklamy = [/*definicje reklam*/];
  var currItem = 0;
  var timeout = 2;
  var speed = 100;
  var advBox = null;
  var divs = null;

  var krok = 0.05;
  var bieżącyKrok = 0;

  function start(id)
  {
    //kod funkcji start z listingu 7.17
  }

  function startShow()
  {
    //kod funkcji startShow z listingu 7.18
    //z usuniętymi instrukcjami dotyczącymi
    //ustalania pozycji warstwy
  }
  //dalsza część skryptu
```

Zmienne globalne reklamy, currItem, timeout, advBox i divs mają takie samo znaczenie jak w skrypcie 64. (listing 7.17). Zmienna speed określa szybkość przenikania, czyli czas pomiędzy wywołaniami funkcji wykonującej animację, zmienna krok — zmianę przezroczystości w jednym kroku, a zmienna bieżącyKrok — bieżącą wartość przezroczystości. Funkcja start pozostała w takiej samej postaci jak w przykładzie 64. (listing 7.17). Kod funkcji startShow jest bardzo podobny do jej odpowiednika z poprzedniego skryptu (65.), jednak zostały usunięte instrukcje dotyczące ustalania pozycji warstwy — w tym przypadku położenie warstw będzie przecież stałe.

Całkowitemu przeobrażeniu uległa funkcja slideDiv, której obecnym zadaniem nie jest przemieszczanie warstw, ale wykonanie efektu przenikania. Jej treść została zaprezentowana na listingu 7.21.

Listing 7.21. *Treść funkcji slideDiv*

```
//początek skryptu
function slideDiv(prevItem, currItem)
{
  if(!advBox || !divs) return;

  var currDiv = divs[currItem];
  var prevDiv = divs[prevItem];
```

```
bieżącyKrok += krok;

if(bieżącyKrok > 1) bieżącyKrok = 1;
currDiv.style.opacity = bieżącyKrok.toFixed(2);
currDiv.style.filter = "alpha(opacity:" +
    (bieżącyKrok * 100).toFixed(0) + ")";
if(bieżącyKrok != 1){
    setTimeout("slideDiv(prevItem, currItem);", speed);
}
else{
    bieżącyKrok = 0;
    setTimeout("startShow();", timeout * 1000);
}
}
</script>
```

Po pobraniu odwołania do warstwy aktualnie wyświetlanej (prevDiv) i tej, która zostanie wyświetlona (currDiv), zwiększana jest zmienna bieżącyKrok odzwierciedlająca stopień przezroczystości. Przezroczystość zmienia się od 0 do 1, w każdym kroku o wartość zdefiniowaną w zmiennej krok. Im większa wartość tej zmiennej, tym szybsze zmiany. Ponieważ 1 to wartość maksymalna, w przypadku stwierdzenia, że bieżącyKrok zawiera większą, zmiennej jest przypisywana wartość 1.

Przezroczystość jest modyfikowana wyłącznie dla warstwy currDiv, czyli tej, która ma się pojawić na ekranie, przesłaniając warstwę bieżącą (poprzednią). Jest to wystarczające, bowiem warstwa currDiv znajduje się zawsze nad warstwą prevDiv, o czym decyduje właściwość zIndex odpowiednio modyfikowana w funkcji startShow. To działanie jest wykonywane dokładnie w taki sam sposób, jak było to w przypadku skryptu 64.

Zmiana przezroczystości jest wykonywana przez modyfikację właściwości opacity i filter obiektu style. Ta druga właściwość jest zmieniana ze względu na niestandardową obsługę przezroczystości w przeglądarkach z rodziny Internet Explorer. Sposób zmiany przezroczystości działa na takiej samej zasadzie, jaka została zaprezentowana przy omawianiu skryptu 61.

Po wykonaniu opisanych czynności konieczne jest stwierdzenie tego, czy proces zamiany warstw został zakończony. Będzie tak wtedy, gdy zmienna bieżącyKrok będzie równa 1. W takiej sytuacji (blok else) należy tej zmiennej przypisać wartość 0 oraz ustawić timer wywołujący funkcję startShow po czasie timeout milisekund. Jeżeli jednak zamiana warstw wciąż trwa (bieżącyKrok różna od 1, blok if), należy ponownie wywołać funkcję slideShow po czasie wskazywanym przez zmienną speed.

Rozdział 8.

Odnośniki

Gdyby trzeba było wskazać element, bez którego sieć WWW w żadnym wypadku nie mogłaby funkcjonować, z pewnością byłyby to odnośniki. To podstawa i istota światowej pajęczyny. Bez nich nie można byłoby przenosić się ze strony na stronę czy też przemieszczać między częściami tego samego serwisu. W rozdziale 8. znajduje się zestaw skryptów związanych tematycznie z hiperłączami. Odnośniki nie muszą bowiem mieć zawsze klasycznej postaci generowanej za pomocą znacznika <a>. Tak naprawdę taką funkcję może pełnić praktycznie dowolny element witryny. Można je też wyposażać w dodatkowe opisy, wyróżnienia, potwierdzenia, decydować o tym, jak i gdzie mają być otwarte itd.

Skrypt 66.

[C][E][F][O][S]

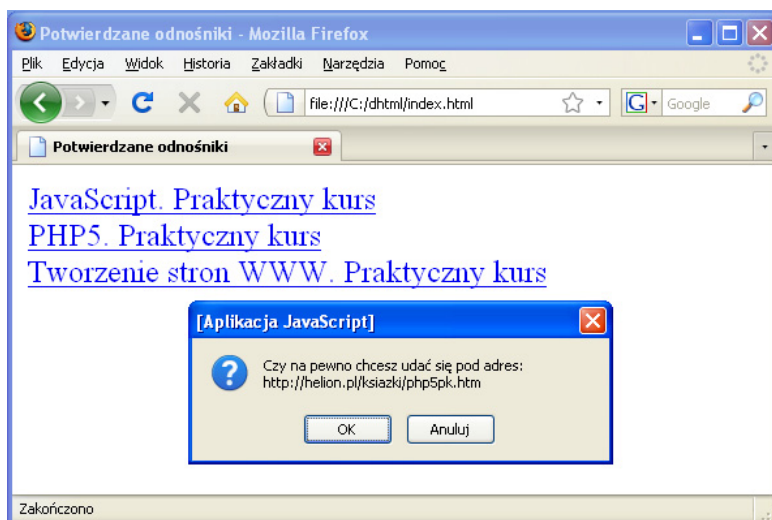
Odnośnik z potwierdzeniem

Kliknięcie odnośnika natychmiast przenosi pod zawarty w nim adres. Czasem jednak chcielibyśmy, aby taka akcja była potwierdzana przez użytkownika witryny. Może to być przydatne na przykład w przypadku odsyłaczy do stron o kontrowersyjnej treści. Wtedy zamiast od razu przenosić na wybraną witrynę, najpierw należy wyświetlić okno z odpowiednim komunikatem. Tak działa skrypt 66. Kliknięcie linku spowoduje pojawienie się okna dialogowego, widocznego na rysunku 8.1, z prośbą o potwierdzenie chęci wykonania operacji oraz dokładnym adresem strony, która ma zostać wczytana. Kod takiego skryptu został przedstawiony na listingu 8.1.

Listing 8.1. Kod skryptu 66.

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN"
"http://www.w3.org/TR/html4/strict.dtd">
<html lang="pl">
  <head>
    <meta http-equiv="Content-Type" content="text/html; charset=utf-8">
    <title>Potwierdzane odnośniki</title>
    <script type="text/javascript">
      function potwierdź(link)
```

Rysunek 8.1.
*Kliknięcie odnośnika
 spowodowało
 wyświetlenie okna
 z prośbą
 o potwierdzenie*



```
{
    var komunikat = "Czy na pewno chcesz udać się pod adres:\n";
    komunikat += link.href;
    return confirm(komunikat));
}
</script>
</head>
<body>
    <div id="mainDiv">
        <a href="http://helion.pl/ksiazki/jscpk.htm"
          onclick="return potwierdź(this);">JavaScript. Praktyczny kurs</a><br />
        <a href="http://helion.pl/ksiazki/php5pk.htm"
          onclick="return potwierdź(this);">
            >PHP5. Praktyczny kurs</a><br />
        <a href="http://helion.pl/ksiazki/twspk.htm"
          onclick="return potwierdź(this);">
            >Tworzenie stron WWW. Praktyczny kurs</a>
    </div>
</body>
</html>
```

W sekcji <body> znajduje się warstwa mainDiv, a w niej są trzy odnośniki zdefiniowane za pomocą znaczników <a>. Konstrukcja odnośników jest standardowa, adresy zostały zdefiniowane w atrybutach href, a wyświetlane na stronie teksty — pomiędzy znacznikami <a> i . Każdy odsyłacz ma jednak dodatkowy atrybut onclick definiujący kod, który ma być wykonany po kliknięciu go. Zawarta wewnątrz instrukcja wygląda specyficznie:

```
return potwierdź(this);
```

Wykorzystywany jest tu fakt, że jeśli procedura obsługi (rozumiana jako treść atrybutu onclick) zwróci wartość true, to zostanie wykonana domyślna akcja danego zdarzenia (w tym przypadku będzie to wczytanie strony wskazywanej przez odnośnik), a gdy

zwróci wartość `false`, domyślna akcja zostanie zaniechana (strona nie zostanie wczytana). A zatem powyższa konstrukcja oznacza, że jeśli funkcja `potwierdź`, która jako argument otrzymuje wskazanie do klikniętego odnośnika, zwróci wartość `true`, to odnośnik ma być wczytany, a jeśli zwróci wartość `false`, odnośnik nie ma być wczytany.

Zadanie wspomnianej funkcji, umieszczonej w nagłówku strony pomiędzy znacznikami `<script>` i `</script>`, jest zatem proste. Musi po prostu zwrócić wynik działania metody `confirm`, która wyświetla okno dialogowe widoczne na rysunku 8.1 oraz zwraca wartość `true`, gdy kliknięty został przycisk *OK*, a wartość `false`, gdy kliknięty został przycisk *Cancel* lub okno zostało zamknięte w inny sposób. W komunikacie wyświetlanym w oknie używany jest adres strony, którą wskazuje odnośnik. Ten adres jest pobierany z właściwości `href` obiektu przekazanego funkcji w postaci argumentu (czyli z obiektu odzwierciedlającego kliknięty odnośnik).

Skrypt 67. [C][E][F][O][S] Element strony jako odnośnik (symulacja odnośników)

Rolę odnośnika, którego kliknięcie spowoduje wczytanie nowej witryny, może pełnić dowolny element strony. Wystarczy użyć zdarzeń i w odpowiedzi na kliknięcie wywołać odpowiednią instrukcję JavaScriptu. W skrypcie 67. przedstawionym na listingu 8.2 rolę odnośników pełnią: przycisk oraz akapit. W sekcji `<body>` została umieszczona warstwa, a w niej elementy zdefiniowane za pomocą znaczników `<input>` i `<p>`. Oba znaczniki otrzymały atrybuty obsługujące zdarzenia `click`, `mouseover` i `mouseout`.

Listing 8.2. Symulacja odnośników

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN"
"http://www.w3.org/TR/html4/strict.dtd">
<html lang="pl">
  <head>
    <meta http-equiv="Content-Type" content="text/html; charset=utf-8">
    <title>Element strony jako odnośnik</title>
  </head>
  <body>
    <div id="mainDiv">
      <input type="button"
        onclick="location.href='http://helion.pl/ksiazki/php102.htm';"
        onmouseover="window.status='http://helion.pl/ksiazki/php102.htm';"
        onmouseout="window.status='';"
        value="PHP. 101 praktycznych skryptów" style="cursor:pointer;"
      />
      <p onclick="location.href='http://helion.pl/ksiazki/joowww.htm';"
        onmouseover="window.status='http://helion.pl/ksiazki/joowww.htm';"
        onmouseout="window.status='';" style="cursor:pointer;"
      >
        Joomla! 1.5. Prosty przepis na własną stronę WWW
      </p>
```

```

</div>
</body>
</html>

```

Atrybut `onclick` zawiera kod wykonywany po kliknięciu danego elementu (w tym przypadku przycisku lub akapitu). Jest to instrukcja przypisująca wybrany adres WWW właściwości `href` obiektu `location`:

```
location.href='http://adres.strony';
```

To powoduje wczytanie do przeglądarki strony wskazywanej przez ten adres. Ponieważ w przypadku zwykłego odnośnika najechanie kursorem myszy na tekst powoduje wyświetlanie adresu na pasku stanu przeglądarki, ten efekt jest symulowany dzięki zdarzeniom `mouseover` i `mouseout`. W przypadku pierwszego z nich wykonywana jest instrukcja w postaci:

```
window.status='http://adres.strony';
```

powodująca, że adres pojawia się na pasku stanu. W drugim przypadku wykonywana jest instrukcja przypisująca właściwości `status` obiektu `window` pusty ciąg znaków:

```
window.status='';
```

To powoduje, że napis na pasku stanu wraca do postaci domyślnej (obsługa paska jest przekazywana przeglądarce).

Skrypt 68. Wybór odnośnika z listy rozwijanej (manualny)

[C][E][F][O][S]

Skrypt 68. pozwala umieścić odnośniki na liście rozwijanej, przy czym przeniesienie na nową witrynę będzie następowało manualnie po kliknięciu przycisku (automatyczne wczytywanie wskazanej na liście nowej strony zostanie pokazane w przykładzie 69.). Strona będzie więc miała postać przedstawioną na rysunku 8.2. Jest ona generowana przez kod zaprezentowany na listingu 8.3.

Listing 8.3. Odnośniki na liście rozwijanej

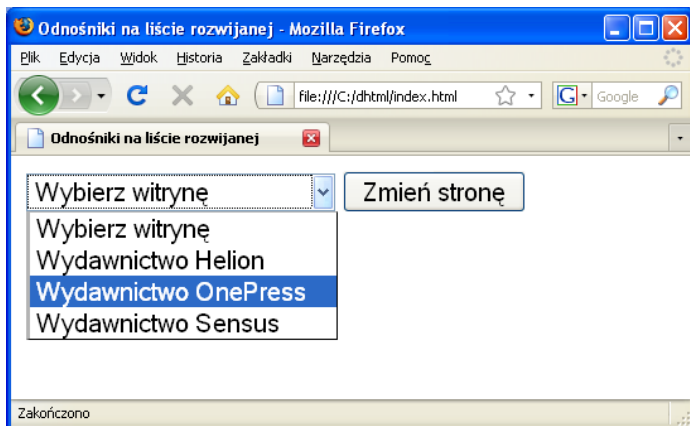
```

<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN"
"http://www.w3.org/TR/html4/strict.dtd">
<html lang="pl">
<head>
<meta http-equiv="Content-Type" content="text/html; charset=utf-8">
<title>Odnośniki na liście rozwijanej</title>
<script type="text/javascript">
function zmieńStronę()
{
var lista = document.getElementById('lista');
if(!lista) return;

```

Rysunek 8.2.

*Lista rozwijana
zawierająca odnośniki
do innych witryn*



```
var link = lista[lista.selectedIndex].value;
if(link)
    location.href = link;
}
</script>
</head>
<body>
<div id="mainDiv">
<select id="lista" size="1">
<option value="">Wybierz witrynę
<option value="http://helion.pl">Wydawnictwo Helion
<option value="http://onepress.pl">Wydawnictwo OnePress
<option value="http://sensus.pl">Wydawnictwo Sensus
</select>
<input type="button" value="Zmień stronę" onclick="zmieńStronę();" />
</div>
</body>
</html>
```

W sekcji `<body>` została umieszczona warstwa, a w niej jest lista rozwijana oraz przycisk. Lista została utworzona za pomocą znacznika `<select>` z parametrem `size` ustawionym na 1 (będzie więc rozwijana), a poszczególne pozycje — za pomocą znaczników `<option>`. Każdy znacznik `<option>` ma atrybut `value` definiujący wartość przypisaną danej pozycji, czyli powiązany z nią adres WWW. Należy zwrócić uwagę, że w przypadku kodu XHTML znaczniki należałoby domknąć, dodając na końcu element `</option>`.

Przycisk będzie reagował na kliknięcie ze względu na użycie atrybutu `onclick` zawierającego wywołanie funkcji `zmieńStronę`. Zadaniem tej funkcji jest odczytanie wartości przypisanej do wybranego przez użytkownika elementu listy oraz wczytanie danego adresu do przeglądarki. Treść funkcji znajduje się w części nagłówkowej kodu. Najpierw jest pobierane i zapisywane w zmiennej `lista` odwołanie do listy o identyfikatorze `lista`. Jeżeli zmienna ta nie jest pusta, czyli gdy taki element strony istnieje, pobierana jest wartość przypisana pozycji wybranej przez użytkownika. Indeks zaznaczonej pozycji znajduje się we właściwości `selectedIndex`, a wartość — we właściwości `value`. Wartość zaznaczonego elementu listy jest zatem uzyskiwana za pomocą konstrukcji `lista[lista.selectedIndex].value`.

Po odczytaniu wartości powiązanej z wybraną pozycją listy uzyskany adres jest przypisywany właściwości href obiektu location:

```
location.href = link;
```

co powoduje wczytanie do przeglądarki nowej witryny. Wcześniej jest jednak sprawdzane, czy zmienna link jest zdefiniowana oraz czy nie zawiera pustego ciągu znaków:

```
if(link)
```

Dzięki temu unikamy przypisania danych właściwości href, gdy nie powiódł się odczyt odnośnika lub gdy została wybrana pierwsza z opcji zawierająca nie adres, ale pusty ciąg znaków (<option value="">Wybierz witrynę).

Skrypt 69. [C][E][F][O][S] Wybór odnośnika z listy rozwijanej (automatyczny)

Skrypt przedstawiony na listingu 8.4 realizuje podobne zadanie jak ten z przykładu 68. (listingu 8.3), z tym że wczytanie nowej strony odbywa się w pełni automatycznie, tuż po wybraniu danej pozycji z listy rozwijanej. Z jednej strony jest to wygodniejsze, gdyż użytkownik nie musi dodatkowo klikać przycisku, czyli potwierdzać chęci wykonania tej operacji, z drugiej jednak strony w przypadku błędnego wyboru nie będzie czasu na korektę (powrót do bieżącej witryny będzie możliwy dopiero po kliknięciu w przeglądarce przycisku *Wstecz*). W praktyce na różnych witrynach spotyka się oba te rozwiązania.

Listing 8.4. Odnośniki na liście rozwijanej z automatycznym wyborem

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN"
"http://www.w3.org/TR/html4/strict.dtd">
<html lang="pl">
<head>
<meta http-equiv="Content-Type" content="text/html; charset=utf-8">
<title>Odnośniki na liście rozwijanej</title>
<script type="text/javascript">
function zmieńStronę(lista)
{
if(!lista) return;
var link = lista[lista.selectedIndex].value;
if(link)
location.href = link;
}
</script>
</head>
<body>
<div id="mainDiv">
<select id="lista" size="1" onchange="zmieńStronę(this);">
<option value="">Wybierz witrynę
```

```

<option value="http://helion.pl">Wydawnictwo Helion
<option value="http://onepress.pl">Wydawnictwo OnePress
<option value="http://sensus.pl">Wydawnictwo Sensus
</select>
</div>
</body>
</html>

```

Ogólna struktura kodu pozostała taka sama jak w przykładzie 68. Tym razem nie ma jednak przycisku, a znacznik `<select>` otrzymał atrybut `onchange`. Dzięki temu po zmianie aktywnej pozycji listy zostanie wywołana funkcja `zmieńStronę`. W postaci argumentu otrzyma ona wskazanie do listy. W związku z tym w kodzie funkcji nie ma już potrzeby pobierania odwołania do listy, jest tylko sprawdzane, czy argument jest niepusty. Dalsze instrukcje są identyczne jak w przykładzie z listingu 8.3.

Skrypt 70. [C][E][F][O][S] Odnośnik z dodatkowym opisem

Umieszczone na stronie odnośniki można wyposażać w dodatkowe opisy. Mogą się one pojawiać dynamicznie, na przeznaczony do tego celu warstwie, po najechaniu na odsyłacz kursorem myszy. Wyglądałoby to tak, jak przedstawiono na rysunku 8.3. Kod HTML działającej witryny znajduje się na listingu 8.5.

Rysunek 8.3.
*Odnośniki
z dynamicznymi
opisami*



Listing 8.5. *Kod HTML skryptu 70.*

```

<body>
  <div id="mainDiv">
    <a href="http://helion.pl/ksiazki/jscpk.htm"
      onmouseover="ustawOpis(1, 'divOpisy');"
      onmouseout="ustawOpis(0, 'divOpisy');">
      JavaScript. Praktyczny kurs</a><br />
    <a href="http://helion.pl/ksiazki/php5pk.htm"
      onmouseover="ustawOpis(2, 'divOpisy');">

```

```

        onmouseout="ustawOpis(0, 'divOpisy');"
      >PHP5. Praktyczny kurs</a><br />
      <a href="http://helion.pl/ksiazki/twospk.htm"
        onmouseover="ustawOpis(3, 'divOpisy');"
        onmouseout="ustawOpis(0, 'divOpisy');"
      >Tworzenie stron WWW. Praktyczny kurs</a>
    </div>
    <div id="divOpisy">
  </div>
</body>

```

W sekcji `<body>` znajdują się dwie warstwy. Pierwsza (`mainDiv`) zawiera listę odnośników. Druga (`divOpisy`) początkowo jest pusta — będą się na niej pojawiały podpowiedzi generowane przez skrypt. Odnośniki zostały zdefiniowane za pomocą typowych znaczników `<a>`, które mają określone atrybuty `onmouseover` i `onmouseout`. W przypadku obu zdarzeń zostanie wykonana funkcja `ustawOpis`. Jej pierwszy argument wskazuje numer podpowiedzi, drugi — identyfikator warstwy, na której ma się pojawić podpowiedź. Treść funkcji wraz z pozostałą częścią skryptu została przedstawiona na listingu 8.6.

Listing 8.6. Skrypt wyświetlający opisy

```

<script type="text/javascript">
  var tabOpisy = [
    "",
    "Strona książki <em>JavaScript. Praktyczny kurs</em>",
    "Strona książki <em>PHP5. Praktyczny kurs</em>",
    "Strona książki <em>Tworzenie stron WWW. Praktyczny kurs</em>"
  ];
  function ustawOpis(id, nazwaWarstwy)
  {
    var divOpis = document.getElementById(nazwaWarstwy);
    if(!divOpis) return;
    id = isNaN(id = parseInt(id))?0:id;
    if(id < 0 || id >= tabOpisy.length) id = 0;
    divOpis.innerHTML = tabOpisy[id];
  }
</script>

```

Opisy skryptów zostały zdefiniowane w globalnej tablicy `tabOpisy`. Zostało przyjęte, że pierwszy będzie wyświetlany w odpowiedzi na usunięcie kursora z obszaru odnośnika, czyli w sytuacji, gdy z warstwy ma zniknąć opis odpowiadający danemu odnośnikowi. Dlatego początkowym elementem tablicy `tabOpisy` jest pusty ciąg znaków (mógłby to być też inny tekst domyślny). Jak widać, w opisach zostały też użyte znaczniki HTML. Jest to możliwe, bowiem opisy będą zwykłą treścią warstwy `divOpisy`.

Funkcja `ustawOpis` otrzymuje argument określający numer opisu (`id`) oraz identyfikator warstwy, na której opis ma się pojawić (`nazwaWarstwy`). W tym przykładzie za każdym razem jest to warstwa `divOpisy`. Odwołanie do warstwy jest pobierane za pomocą metody `getElementById` i zapisywane w zmiennej `divOpis`. Następnie za pomocą konstrukcji z operatorem warunkowym weryfikowana jest poprawność argumentu `id`, który powinien być liczbą całkowitą. Jeśli nie jest, przypisuje mu się wartość domyśl-

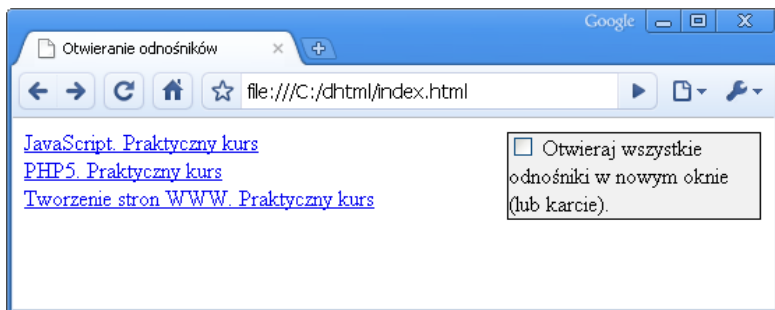
ną — 0. W kolejnej instrukcji badane jest też to, czy wartość `id` mieści się w dopuszczalnym zakresie od 0 do długości tablicy `tabOpisy` pomniejszonej o 1. Jeżeli zakres jest przekroczony, parametr otrzymuje wartość 0.

Po zakończeniu weryfikacji opis do wyświetlenia jest pobierany z tablicy `tabOpisy` spod indeksu wskazywanego przez `id`, a następnie zapisywany we właściwości `innerHTML` obiektu `divOpis`. Dzięki temu pojawia się na ekranie.

Skrypt 71. [C][E][F][O][S] Zdecyduj, gdzie otwierać odnośniki

Skrypt 71. umożliwia użytkownikowi zdecydowanie, gdzie mają być otwierane wszystkie odnośniki z danej strony: w bieżącym oknie (karcie) przeglądarki czy też nowym. Wybór będzie się odbywał za pomocą pola wyboru typu `checkbox` umieszczonego w dowolnym miejscu witryny, na dodatkowej warstwie, np. w prawym górnym rogu, tak jak zaprezentowano to na rysunku 8.4. Kod HTML takiego rozwiązania znajduje się na listingu 8.7.

Rysunek 8.4.
*Umieszczona
w rogu strony
dodatkowa warstwa
z polem wyboru*



Listing 8.7. Kod sekcji `<body>` skryptu 71.

```
<body>
  <div id="mainDiv">
    <a href="http://helion.pl/ksiazki/jscpk.htm">
      JavaScript. Praktyczny kurs</a><br />
    <a href="http://helion.pl/ksiazki/php5pk.htm">
      >PHP5. Praktyczny kurs</a><br />
    <a href="http://helion.pl/ksiazki/twospk.htm">
      >Tworzenie stron WWW. Praktyczny kurs</a>
  </div>
  <div id="divInterfejs">
    <input type="checkbox" onclick="zmień(this)" />
    Otwieraj wszystkie odnośniki w nowym oknie (lub karcie).
  </div>
</body>
```

Mamy tu dwie warstwy. Pierwsza zawiera trzy standardowe odnośniki zdefiniowane za pomocą znacznika <a>. Mogą one wskazywać dowolne adresy w sieci. Takie odnośniki będą standardowo otwierane w tym samym oknie przeglądarki co bieżąca strona (chyba że została zmieniona konfiguracja przeglądarki). Będzie to można zmienić za pomocą pola wyboru umieszczonego na drugiej z warstw. Polu nie został nadany identyfikator, ale ma ono atrybut onclick definiujący instrukcję JavaScriptu, która zostanie wykonana po kliknięciu. Jest to wywołanie funkcji *zmień* i przekazanie jej w postaci argumentu odwołania do obiektu inicjującego zdarzenie (czyli pola wyboru).

Treść funkcji *zmień* została przedstawiona na listingu 8.8.

Listing 8.8. *Treść funkcji *zmień**

```
<script type="text/javascript">
  function zmień(pole)
  {
    if(!pole) return;
    var links = document.getElementsByTagName("a");
    for(var i = 0; i < links.length; i++){
      if(pole.checked)
        links[i].target = "_blank";
      else
        links[i].target = "";
    }
  }
</script>
```

Funkcja otrzymuje argument wskazujący pole wyboru, którego stan ma zbadać. Najpierw sprawdza, czy otrzymana wartość jest pusta. Jeśli tak, kończy swoje działanie. Jeśli nie, wykonywane są dalsze czynności. Pobierane są odwołania do wszystkich umieszczonych w dokumencie odnośników. Odbywa się to przez wywołanie metody `getElementsByTagName` obiektu `document`. Uzyskana tablica (kolekcja) odnośników jest przypisywana zmiennej pomocniczej `links`. Następnie w pętli `for` modyfikowana jest właściwość `target` każdego obiektu zapisanego w tablicy, czyli każdego odnośnika. Jeżeli pole wyboru `pole` jest zaznaczone (właściwość `checked` pola jest równa `true`), właściwość `target` otrzymuje wartość `_blank`. To oznacza, że odnośnik ma być otwierany w nowym oknie (nowej karcie). Jeśli pole wyboru `pole` nie jest zaznaczone (właściwość `checked` pola jest równa `false`), właściwość `target` otrzymuje pusty ciąg znaków `""`. To oznacza, że odnośnik ma być otwierany w bieżącym oknie (bieżącej karcie).

Rozdział 9.

Obrazy

Rozdział 9. zawiera kilkanaście skryptów związanych z prezentacją i przetwarzaniem obrazów. Trudno przecież dziś znaleźć stronę WWW, która obywa się bez grafiki. Przedstawione zostaną zarówno przykłady związane z techniczną obsługą witryny, np. sposoby na wstępne załadowanie plików graficznych, jak i efekty związane z prezentacją, takie jak pokaz slajdów, galeria grafiki czy skalowanie obrazów. Nie zostaną pominięte także takie tematy jak poszukiwanie obrazów ze względu na ich opis, powiększanie fragmentów grafiki, czyli efekt lupy, a także elementy animacji.

Skrypt 72.

[C][E][F][O][S]

Zmiana obrazu

po najechaniu myszą

Bardzo popularnym i często spotykanym efektem jest zamiana obrazu po najechaniu na niego kursorem myszy. To znaczy, że znajdujący się na witrynie obraz (obrazy) ma dwa stany: spoczynkowy i aktywny (wskazany). Zamiana ze stanu spoczynkowego na aktywny odbywa się przy zdarzeniu `mouseover`, a zamiana ze stanu aktywnego na spoczynkowy — przy zdarzeniu `mouseout`. Oczywiście niezbędne jest przygotowanie dwóch plików graficznych reprezentujących oba stany obrazu. Należy również uwzględnić problem opóźnienia ładowania obrazów z sieci, szczególnie dobrze widoczny w przypadku wolniejszych lub mocno obciążonych łączy. Wtedy pierwsza zamiana (zanim obraz reprezentujący stan aktywny znajdzie się w pamięci podręcznej przeglądarki) odbywa się z widocznym opóźnieniem. Problem ten rozwiązuje się dzięki wcześniejszemu załadowaniu wszystkich obrazów przez skrypt. Tak też działa kod przedstawiony na listingu 9.1.

Listing 9.1. Pełna treść skryptu 72.

```

<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN"
"http://www.w3.org/TR/html4/strict.dtd">
<html lang="pl">
  <head>
    <meta http-equiv="Content-Type" content="text/html; charset=utf-8">
    <title>Zamiana obrazów</title>
    <script type="text/javascript">
      function zmieńObraz(obraz, nazwa)
      {
        if(!obraz || !nazwa) return;
        obraz.src = nazwa;
      }
      var obraz = new Image();
      obraz.src = "obrazlon.png";
      obraz.src = "obrazloff.png";
    </script>
  </head>
  <body>
    <div id="mainDiv">
      
    </div>
  </body>
</html>

```

W sekcji `<body>` znajduje się warstwa `mainDiv`, a w niej obraz zdefiniowany za pomocą standardowego znacznika ``. Parametr `src` wskazuje plik graficzny *obrazloff.png*, zatem w tym pliku powinna być zawarta grafika w stanie nieaktywnym. Atrybuty `onmouseover` i `onmouseout` zawierają wywołanie funkcji `zmieńObraz`, która zamienia plik graficzny przypisany znacznikowi ``. Pierwszy argument funkcji wskazuje obiekt obrazu, drugi — nazwę pliku. A zatem gdy kursor myszy znajdzie się nad obszarem obrazu (zdarzenie `mouseover`), zostanie wczytana grafika z pliku *obrazlon.png*, a gdy opuście obszar obrazu — grafika z pliku *obrazloff.png*.

Kod funkcji `zmieńObraz` jest bardzo prosty. Argument `obraz` odzwierciedla element strony wygenerowany za pomocą znacznika ``, a argument `nazwa` — nazwę pliku graficznego. Po sprawdzeniu, że oba argumenty nie są puste (niezdefiniowane), następuje przypisanie wartości atrybutu `nazwa` właściwości `src` obiektu `obraz`:

```
obraz.src = nazwa;
```

To powoduje zmianę grafiki na stronie.

Za kodem funkcji zostały umieszczone trzy instrukcje wykonywane w trakcie ładowania strony do przeglądarki. Powodują one wstępne załadowanie danych z plików graficznych *obrazlon.png* i *obrazloff.png*, dzięki czemu zamiana grafik powinna się odbywać bez opóźnień już przy pierwszym wywołaniu funkcji `zmieńObraz`. Pierwsza instrukcja tworzy nowy obiekt typu `Image`, a dwie kolejne zmieniają właściwość `src` wskazującą lokalizację pliku graficznego. To wymusza załadowanie obu obrazów do pamięci podręcznej.

Skrypt 73.

Zautomatyzowana zamiana obrazów

[C][E][F][O][S]

Klasyczny sposób zamiany obrazów z użyciem kursora myszy wymaga używania atrybutów `onmouseover` i `onmouseout` znacznika `` oraz wskazywania w kodzie HTML funkcji JavaScriptu, która zostanie wykonana w odpowiedzi na zdarzenie. Tak jednak być nie musi. O tym, który obraz będzie reagował na wskazanie kursorem, można zdecydować za pomocą atrybutu... `class`! Oczywiście bez kodu JavaScript się nie obejdzie, ale część HTML mogłaby wyglądać tak jak na listingu 9.2.

Listing 9.2. Sekcja `<body>` skryptu 73.

```
<body onload="przygotujObrazy('xyz');">
  <div id="mainDiv">
    
    
  </div>
</body>
```

W warstwie `mainDiv` za pomocą znaczników `` zostały zdefiniowane dwa obrazy. Zwróćmy uwagę, że nie zawierają nawet jednego znaku kodu JavaScript. Mimo to obrazy te będą reagowały na wskazanie ich kursorem myszy. Łatwo się domyślić, że będą za to odpowiedzialne specyficznie zdefiniowane atrybuty `class` oraz funkcja `przygotujObrazy` wywoływana po załadowaniu strony do przeglądarki (dzięki umieszczeniu wywołania funkcji w atrybucie `onload` znacznika `<body>`).

Postępowanie powinno być następujące: każdy obraz, który ma podlegać efektowi wymiany na inny po kliknięciu myszą, powinien mieć zdefiniowany atrybut `class` w postaci:

```
class="prefiks nazwa_pliku"
```

Prefiksem może być dowolny ciąg znaków (np. użyty w przykładzie `xyz`), nie może jednak występować w nazwach zwykłych klas powiązanych z obrazami. Powinien być charakterystyczny tylko dla obrazów korzystających z omawianego efektu. Za prefiksem, po znaku odstępu, należy umieścić nazwę pliku z grafiką, która ma się ukazać, gdy kursor myszy znajdzie się nad danym obrazem. Zapis:

```

```

oznacza zatem, że na stronie pojawi się obraz z pliku *obraz1off.png*, który będzie zamieniany na obraz z pliku *obraz1on.png*.

Prefiksu z atrybutu `class` należy następnie użyć w wywołaniu funkcji `przygotujObrazy`:

```
<body onload="przygotujObrazy('xyz');">
```

Treść tej funkcji została przedstawiona na listingu 9.3.

Listing 9.3. Treść funkcji przygotuj obrazy

```
<script type="text/javascript">
  function przygotujObrazy(prefiks)
  {
    if(!prefiks) return;
    var img = new Image();
    var imgs = document.getElementsByTagName("img");
    for(var i = 0; i < imgs.length; i++){
      if(imgs[i].className &&
        imgs[i].className.substr(0, prefiks.length) == prefiks){
        imgs[i].nazwaOn = imgs[i].className.slice(prefiks.length);
        imgs[i].nazwaOff = imgs[i].src;
        img.src = imgs[i].nazwaOn;
        imgs[i].onmouseover = function(){
          this.src = imgs[i].nazwaOn;
        }
        imgs[i].onmouseout = function(){
          this.src = imgs[i].nazwaOff;
        }
      }
    }
  }
</script>
```

Funkcja sprawdza, czy został jej przekazany niepusty parametr prefiks, tworzy pomocniczy obiekt typu `Image` (zmienna `img`) służący do wstępnego ładowania obrazów oraz za pomocą metody `getElementsByTagName` pobiera i zapisuje w zmiennej `imgs` odwołania do wszystkich elementów typu `img` (wygenerowanych przez znaczniki ``). Ze wszystkich elementów `img` należy wyodrębnić te, które mają atrybut `class`, a pierwsze znaki tego atrybutu odpowiadają prefiksowi. Odbywa się to w pętli `for`.

Atrybut `class` jest zdefiniowany wtedy, gdy w obiekcie odpowiadającym obrazowi jest właściwość `className`. Stąd warunek `imgs[i].className`. Drugi warunek ma postać:

```
imgs[i].className.substr(0, prefiks.length) == prefiks
```

Za pomocą metody `substr` z wartości atrybutu `className` jest wyodrębnianych tyle pierwszych znaków, ile ma argument `prefiks` (`prefiks.length`), a uzyskany ciąg jest porównywany z wartością zapisaną w `prefiks`. Jeżeli zgodność występuje, oznacza to obraz, który ma być poddany efektowi wymiany grafiki, są więc wykonywane dalsze czynności.

We właściwości `nazwaOn` (jest w tym miejscu tworzona) danego obrazu zapisywana jest nazwa pliku pobrana z atrybutu `class` (właściwości `className`). Do wyodrębnienia nazwy z ciągu jest używana metoda `slice` pobierająca podciąg zaczynający się w znaku o indeksie przekazanym jako argument, a kończący się wraz z końcem ciągu. We właściwości `nazwaOff` zapisywana jest nazwa pobrana z właściwości `src`, czyli nazwa pliku graficznego wyświetlanego, gdy obraz jest w stanie nieaktywnym.

Aby obraz reagował na pojawianie się na jego obszarze kursora myszy, trzeba oczywiście obsłużyć zdarzenia `mouseover` i `mouseout`. Zatem właściwościom `onmouseover` i `onmouseout` przypisywane są funkcje anonimowe zmieniające wartość właściwości `src` danego obrazu. Używane są w nich wartości z właściwości `nazwaOn` i `nazwaOff`.

Skrypt 74. [C][E][F][O][S] Przesuwanie obrazu po stronie

Skrypt 74. wyposaża witrynę w funkcję swobodnego przemieszczania obrazów po całej powierzchni strony. Wystarczy kliknąć obraz i przesunąć go w dowolne miejsce za pomocą myszy. Wykorzystany został pomysł zaprezentowany przy omawianiu przykładu 62. w rozdziale 7. Kod proponowanego rozwiązania jest widoczny na listingu 9.4.

Listing 9.4. Skrypt pozwalający na swobodne przesuwanie obrazów

```

<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN"
"http://www.w3.org/TR/html4/strict.dtd">
<html lang="pl">
<head>
  <meta http-equiv="Content-Type" content="text/html; charset=utf-8">
  <title>Obraz przesuwany za pomocą myszy</title>
  <style type="text/css">
    img{
      cursor:move;
    }
  </style>
  <script type="text/javascript">
    //treść skryptu
  </script>
</head>
<body>
  <div id="divDane">
    
    
    
  </div>
</body>
</html>
```

W sekcji `<body>` znajduje się warstwa `divDane` zawierająca trzy obrazy, które zostały zdefiniowane za pomocą typowych znaczników ``. Każdy znacznik otrzymał atrybut `onmousedown`, w którym znajduje się wywołanie funkcji `startDrag`. Dokładna postać tej funkcji została omówiona w rozdziale 7. (skrypt 62.). Przekazywanym argumentem jest `this`, czyli wskazanie do bieżącego obrazu (nad którym został naciśnięty przycisk myszy). Wykonywana jest także instrukcja `return false`, która zapobiega standardowej obsłudze zdarzenia `mousedown`.

Skrypt 75. [C][E][F][O][S]

Zmiana rozmiarów obrazu z podaniem nowych wartości

Skrypt 75. pozwala zmieniać rozmiary obrazu umieszczonego na stronie WWW. W witrynie pojawią się dwa pola tekstowe pozwalające wprowadzić nowe wielkości (wysokość i szerokość) oraz przycisk wprowadzający zmiany. Interfejs będzie zatem wyglądał tak, jak przedstawiono to na rysunku 9.1. Wprowadzane do pól tekstowych dane będą weryfikowane, ograniczona będzie też maksymalna rozdzielczość. W związku z tym po zmianie rozmiarów pod obrazem będzie się pojawiała informacja o tym, jaka faktyczna rozdzielczość została zastosowana. Kod HTML tak działającego skryptu znajduje się na listingu 9.5.

Rysunek 9.1.

Strona pozwalająca na zmianę rozmiarów obrazu



Listing 9.5. Część HTML skryptu 75.

```
<body>
  <div id="mainDiv">
    Nowa szerokość:
    <input type="text" id="tfSzer" style="width:30px;" />
    Nowa wysokość:
    <input type="text" id="tfWys" style="width:30px;" />
    <input type="button" value="Wykonaj" onclick="zmienRozmiar('img1');" />
  </div>
  <div id="imgDiv">
    <img src='obraz.jpg' alt='okladka' id="img1" /><br />
    <div id="divInfo"> </div>
  </div>
</body>
```


Sekcja <body> składa się z dwóch głównych warstw. Pierwsza zawiera interfejs, czyli dwa pola tekstowe oraz przycisk, druga — obraz podlegający skalowaniu i warstwę pomocniczą (divInfo) służącą do wyświetlania komunikatów. Pola tekstowe zostały zdefiniowane za pomocą znaczników input z atrybutem type ustawionym na text. Otrzymały też identyfikatory tfSzer (pole reprezentujące szerokość) i tfWys (pole reprezentujące wysokość). Przyciskowi została przypisana procedura zdarzenia click w postaci funkcji zmieńRozmiar, której w postaci argumentu jest przekazywany identyfikator obrazu. Obraz został zdefiniowany w klasyczny sposób, za pomocą znacznika , w którym atrybut src wskazuje plik z grafiką do wyświetlenia.

Funkcja zmieńRozmiar wraz z pozostałą częścią skryptu została przedstawiona na listingu 9.6.

Listing 9.6. Skrypt zmieniający rozmiary obrazu o wskazanym identyfikatorze

```
<script type="text/javascript">
  var maxSzer = 250;
  var maxWys = 250;
  function zmieńRozmiar(id)
  {
    var img = document.getElementById(id);
    var tfSzer = document.getElementById('tfSzer');
    var tfWys = document.getElementById('tfWys');
    var divInfo = document.getElementById('divInfo');
    if(!img || !tfSzer || !tfWys || !divInfo) return;
    var szer = tfSzer.value;
    var wys = tfWys.value;
    szer = isNaN(szer = parseInt(szer))?0:szer;
    wys = isNaN(wys = parseInt(wys))?0:wys;
    if(szer > maxSzer) szer = maxSzer;
    if(wys > maxWys) wys = maxWys;
    if(szer < 0) szer = 0;
    if(wys < 0) wys = 0;
    img.width = szer;
    img.height = wys;
    divInfo.innerHTML = "Nowe wymiary: " + szer + "x" + wys;
  }
</script>
```

Na początku skryptu zostały zdefiniowane dwie zmienne globalne określające maksymalną dopuszczalną szerokość (maxSzer) i wysokość (maxWys). Funkcja zmieńRozmiar przyjmuje jeden argument o nazwie id, który powinien zawierać identyfikator obrazu. Pierwsze cztery instrukcje korzystają z metody getElementById w celu pobrania odwołań do elementów strony: pól tekstowych (zmienne tfSzer i tfWys), warstwy divInfo (zmienna divInfo) oraz obrazu (zmienna img). Następnie sprawdzane jest, czy te operacje zakończyły się sukcesem. Jeśli nie, jest wykonywana instrukcja return kończąca działanie funkcji.

Wartości z pól tekstowych tfSzer i tfWys są zapisywane w zmiennych szer i wys. Kolejny zestaw instrukcji bada poprawność danych. Przede wszystkim wysokość i szerokość muszą być wartościami całkowitymi. Jest to badane za pomocą metod parseInt i isNaN oraz konstrukcji z operatorem warunkowym. Jej znaczenie zostało dokładniej

opisane w skrypcie 54. Gdy dane otrzymają postać wartości całkowitych, badane jest, czy przekraczają dopuszczalny zakres. Wysokość i szerokość nie mogą być bowiem mniejsze od 0 ani też większe od wartości granicznych zapisanych w zmiennych `maxSzer` i `maxWys`. W przypadku wykrycia przekroczenia zakresu następuje korekta danych.

Zmiana rozmiarów obrazu jest wykonywana w bardzo prosty sposób. Modyfikowane są właściwości `width` (reprezentująca szerokość) i `height` (reprezentująca wysokość). W warstwie `divInfo` (właściwość `innerHTML`) jest też zapisywany komunikat informujący o tym, jakie są bieżące rozmiary obrazu.

Skrypt 76. [C][E][F][O] Skalowanie obrazu za pomocą myszy

Skrypt 76. umożliwia skalowanie obrazów za pomocą ruchów myszy. Kliknięcie w obszarze obrazu i przesuwanie kursora w lewo i w górę będzie powodowało zmniejszania rozmiarów, a przesuwanie kursora w prawo i w dół — zwiększanie. Obrazy należy umieścić na stronie w zwyczajny sposób, tak jak zostało to zaprezentowane na listingu 9.7.

Listing 9.7. Sekcja `<body>` zawierająca trzy obrazy podlegające skalowaniu

```
<body>
  <div id="divDane">
    
    
    
  </div>
</body>
```

W sekcji `<body>` została umieszczona warstwa `divDane`, a w niej trzy obrazy zdefiniowane za pomocą znaczników ``. Każdy znacznik otrzymał atrybut `onmousedown`, czyli został określony kod wykonywany po naciśnięciu nad danym obrazem przycisku myszy. Jest to wywołanie funkcji `startScaling` i przekazanie jej w postaci argumentu bieżącego elementu (czyli danego obrazu), a także zwrócenie wartości `false`. Ta druga instrukcja zapobiega standardowej obsłudze zdarzenia w przeglądarce.

Treść funkcji `startScaling` wraz z pozostałą częścią skryptu została przedstawiona na listingu 9.8.

Listing 9.8. Kod skryptu 76.

```
<script type="text/javascript">
  var activeImage = null;
  var lastXY = null;
  var stepX = 1;
  var stepY = 1;
```

```
document.onmouseup = mouseUp;
document.onmousemove = mouseMove;

function mousePos(evt)
{
    //treść funkcji mousePos
}

function startScaling(obj)
{
    if(!obj) return;
    activeImage = obj;
    lastXY = {x:0, y:0};
}

function mouseUp(evt)
{
    activeImage = null;
}

function mouseMove(evt)
{
    if(activeImage){
        var currXY = mousePos(evt);
        if(currXY.x > lastXY.x) activeImage.width += stepX;
        else if(currXY.x < lastXY.x) activeImage.width -= stepX;
        if(currXY.y < lastXY.y) activeImage.height -= stepY;
        else if(currXY.y > lastXY.y) activeImage.height += stepY;
        lastXY = currXY;
        return false;
    }
}
</script>
```

Na początku kodu znajdują się deklaracje zmiennych globalnych:

- ◆ `activeImage` — przechowuje odwołanie do aktywnego obrazu;
- ◆ `lastXY` — przechowuje poprzednie współrzędne kursora myszy;
- ◆ `stepX` — określa liczbę pikseli, o którą przy każdym ruchu ma być zmieniona szerokość obrazu;
- ◆ `stepY` — określa liczbę pikseli, o którą przy każdym ruchu ma być zmieniona wysokość obrazu.

A zatem zmienne `stepX` i `stepY` określają po prostu szybkość i dokładność, z jaką będzie następowało zmniejszanie bądź zwiększanie obrazu.

Kolejne dwie instrukcje przypisują procedury obsługi zdarzeń `mouseup` (zwolnienie klawisza myszy) i `mousemove` (ruch myszy), niezbędne jest bowiem rozpoznawanie, kiedy klawisz myszy został zwolniony oraz kiedy mysz jest przemieszczana. W pierwszym przypadku procedurą obsługi będzie funkcja `mouseUp`, a w drugim — `mouseMove`. Określaniem współrzędnych bieżącego położenia kursora zajmie się funkcja `mousePos`, której treść została zapożyczona z przykładu 62. (listing 7.11).

Zadaniem funkcji `startScaling`, otrzymującej w postaci argumentu `obj` odwołanie do obrazu, nad którym został przyciśnięty klawisz myszy, jest zainicjowanie procesu skalowania (kod jest wykonywany tylko wtedy, gdy argument `obj` jest niepusty). Odbывается przez przypisanie wartości `obj` globalnej zmiennej `activeImage`, a także zainicjowanie zmiennej `lastXY` zerami (co oznacza przypisanie współrzędnym `x` i `y` wartości 0).

Zadaniem funkcji `mouseUp` jest zakończenie procesu skalowania. Zatem jedyną występującą w niej instrukcja przypisuje po prostu wartość `null` zmiennej `activeImage`. To sygnał dla funkcji `mouseMove`, że ma zakończyć skalowanie.

Właściwe skalowanie jest przeprowadzane w funkcji `mouseMove` wywoływanej przy każdym ruchu myszy. Jest to wykonywane tylko wtedy, gdy zmienna `activeImage` jest niepusta (`if(activeImage){}`). Funkcja otrzymuje argument `evt`, który jest związany ze zdarzeniem `mousemove` i określa m.in. bieżące współrzędne. Ze względu na różną obsługę zdarzeń w przeglądarkach obiekt ten jest przekazywany funkcji `mousePos`, która pobiera i zwraca właściwe współrzędne. Otrzymane wartości `x` i `y` są zapisywane w zmiennej pomocniczej `currPos`.

Następnie należy stwierdzić, czy obraz ma być zmniejszany, zwiększany, czy też ma pozostać niezmieniony. Oczywiście w tym ostatnim przypadku nie trzeba robić nic. Do rozpatrzenia pozostają zatem cztery sytuacje:

- ♦ Bieżąca wartość `x` jest większa od poprzedniej (`currXY.x > lastXY.x`)
— należy zwiększyć szerokość (właściwość `width`) o wartość zapisaną w `stepX` (`activeImage.width += stepX;`).
- ♦ Bieżąca wartość `x` jest mniejsza od poprzedniej (`currXY.x < lastXY.x`)
— należy zmniejszyć szerokość (właściwość `width`) o wartość zapisaną w `stepX` (`activeImage.width -= stepX;`).
- ♦ Bieżąca wartość `y` jest mniejsza od poprzedniej (`currXY.y < lastXY.y`)
— należy zmniejszyć wysokość (właściwość `height`) o wartość zapisaną w `stepY` (`activeImage.height -= stepY;`).
- ♦ Bieżąca wartość `y` jest większa od poprzedniej (`currXY.y > lastXY.y`)
— należy zwiększyć wysokość (właściwość `height`) o wartość zapisaną w `stepY` (`activeImage.height += stepY;`).

Po wykonaniu tych czynności pozostaje jeszcze zapisać bieżące współrzędne w zmiennej `lastXY`. Zatem w kolejnym kroku (ruchu) aktualne współrzędne staną się poprzednimi.

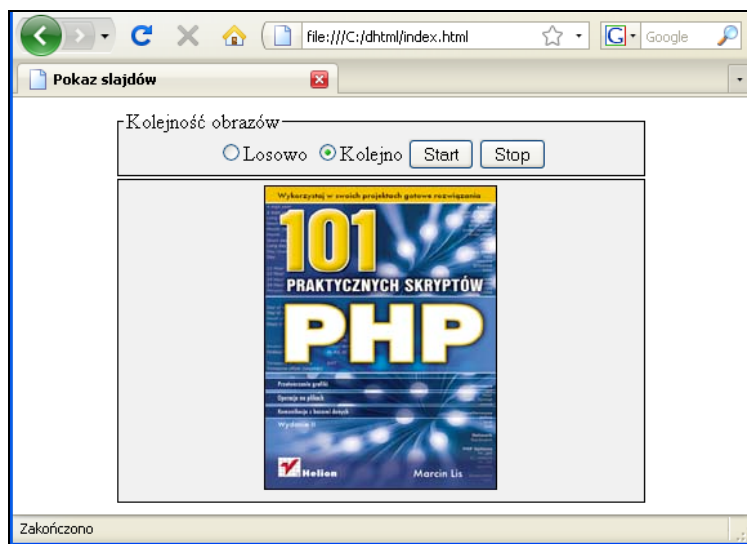
Skrypt 77. Pokaz slajdów

[C][E][F][O][S]

Pokaz slajdów, czyli po prostu obrazy zmieniające się w zadanym czasie, często można spotkać na rozmaitych witrynach. Takie właśnie zadanie realizuje skrypt 77. Pozwala on na decydowanie o sposobie prezentacji obrazów — będą mogły pojawiać się w kolejności określonej w skrypcie bądź też losowo. Wybór będzie dokonywany za pomo-

cą pól wyboru typu radio widocznych na rysunku 9.2. Pokaz będzie też mógł być rozpoczynany i kończony w dowolnym momencie — sterowanie będzie się odbywało za pomocą przycisków. Część HTML skryptu została przedstawiona na listingu 9.9.

Rysunek 9.2.
Interfejs skryptu 77.



Listing 9.9. Sekcja `<body>` skryptu 77.

```
<body>
  <div id="mainDiv">
    <fieldset id="fs1">
      <legend>Kolejność obrazów</legend>
      <input type="radio" name="kolejnosc" value="rand" id="chbRand" />Losowo
      <input type="radio" name="kolejnosc" value="ord" id="chb0rd"
        checked="checked" />Kolejno
      <input type="button" value="Start" onclick="btnStartClick();" />
      <input type="button" value="Stop" onclick="btnStopClick();" />
    </fieldset>
  </div>
  <div id="imgDiv">
    <img src='cwjas2.jpg' alt='okladka' id="img1" /><br />
  </div>
</body>
```

W sekcji `<body>` zostały umieszczone dwie warstwy `mainDiv` i `imgDiv`. Pierwsza zawiera interfejs sterujący pokazem slajdów, a druga — element `` służący do prezentacji obrazów (atrybut `id` pozwoli na identyfikację obrazu w skrypcie). Interfejs składa się z zestawu pól zgrupowanych dzięki znacznikowi `<fieldset>`. Są to tworzące jedną grupę (ta sama wartość atrybutu `name`) dwa pola wyboru typu radio, pozwalające na określenie kolejności prezentowanych grafik, oraz dwa przyciski umożliwiające rozpoczęcie i zakończenie prezentacji. Pola wyboru otrzymały identyfikatory `chbRand` (dla losowej kolejności prezentacji) i `chb0rd` (dla uporządkowanej kolejności prezentacji). Pole `chb0rd` jest domyślnie zaznaczone dzięki użyciu atrybutu `checked`. Przyciski `Start` i `Stop` będą reagowały na kliknięcia, bowiem zostały użyte atrybuty `onclick`.

W pierwszym przypadku zostanie wywołana funkcja `btnStartClick`, a w drugim — `btnStopClick`.

Treść skryptu zmieniającego obrazy została przedstawiona na listingu 9.10.

Listing 9.10. *Skrypt realizujący funkcję pokazu slajdów*

```
<script type="text/javascript">
  var imgs = new Array(
    "cwjas2.jpg", "cwjav2.jpg", "php102.jpg", "joowww.jpg",
    "jscpk.jpg", "aja101.jpg"
  );
  var timeout = 2;
  var licznik = 0;
  var timerId = null;
  function zmieńObraz(id)
  {
    var img = document.getElementById(id);
    if(!img) return;
    var chbRand = document.getElementById("chbRand");
    if(chbRand && chbRand.checked){
      licznik = Math.floor((imgs.length) * Math.random());
      img.src = imgs[licznik];
    }
    else{
      if(licznik >= imgs.length) licznik = 0;
      img.src = imgs[licznik++];
    }
    timerId = setTimeout("zmieńObraz('"+id+"')", timeout * 1000);
  }
  function btnStartClick()
  {
    if(!timerId) zmieńObraz('img1');
  }
  function btnStopClick()
  {
    if(timerId){
      clearTimeout(timerId);
      timerId = null;
    }
  }
</script>
```

Na początku kodu znajdują się deklaracje zmiennych globalnych. Są to:

- ♦ `imgs` — tablica zawierająca nazwy plików graficznych, które mają być prezentowane (jeśli pliki znajdują się w innej lokalizacji niż skrypt, należy podać również ścieżkę dostępu bądź nawet pełny URL);
- ♦ `timeout` — czas (w sekundach), w którym obraz ma pozostawać na ekranie;
- ♦ `licznik` — indeks bieżącego obrazu;
- ♦ `timerId` — identyfikator timera zainicjowanego za pomocą metody `setTimeout`.

Funkcja `btnStartClick` jest wykonywana po kliknięciu przycisku *Start*, a jej zadaniem jest uruchomienie prezentacji. To jednak należy zrobić tylko wtedy, jeśli pokaz nie jest aktualnie uruchomiony. Dlatego też najpierw sprawdzane jest to, czy zmienna `timerId` jest pusta (`if(!timerId)`). Jeśli jest pusta, wywoływana jest funkcja `zmieńObraz` otrzymująca w postaci argumentu identyfikator elementu ``, w którym ma być prezentowana grafika (w tym przypadku jest to `img1`).

Funkcja `btnStopClick` jest wykonywana po kliknięciu przycisku *Stop*, a jej zadaniem jest zatrzymanie prezentacji. Tę czynność należy wykonać, gdy prezentacja jest aktualnie aktywna, a więc gdy zmienna `timerId` nie jest pusta (jest różna od `null`). Jeśli tak jest, wywoływana jest metoda `clearTimeout` zatrzymująca timer, a zmienna `timerId` otrzymuje wartość `null`.

Za wymianę obrazów odpowiada funkcja `zmieńObraz`. Jako argument otrzymuje identyfikator znacznika ``, czyli elementu strony, w którym ma się pojawiać grafika. Odwołanie do tego elementu jest pobierane za pomocą metody `getElementById` i zapisywane w zmiennej `img`. Jeśli ta operacja się powiedzie (istnieje element witryny o wskazanym identyfikatorze), wykonywane są dalsze czynności. Jeśli się nie powiedzie (brak elementu o identyfikatorze określonym przez `id`), wykonywanie funkcji jest przerywane.

Ponieważ konieczne jest stwierdzenie, czy zmiana obrazów ma się odbywać losowo czy też w kolejności w jakiej występują w tablicy `imgs`, pobierane jest i zapisywane w zmiennej `rbRand` odwołanie do pola tekstowego `rbRand`. Przyjęto następującą zasadę: jeśli to pole istnieje i jest zaznaczone (właściwość `checked` równa `true`), kolejność ma być losowa, a w każdym innym przypadku — określona. Jeżeli zatem prawdziwy jest warunek:

```
chbRand && chbRand.checked
```

zmiennej `licznik` jest przypisywana wartość losowa uzyskana dzięki metodzie `rand` obiektu `Math` (wartość ta jest mnożona przez długość tablicy `length` i zaokrąglana w dół). Tak uzyskany indeks jest używany do pobrania nazwy pliku z tablicy `imgs` i zapisania jej we właściwości `src` obiektu `img`. To powoduje wyświetlenie wskazanego obrazu na stronie.

W przypadku prezentowania obrazów w kolejności definicji zmienna `licznik` wskazuje indeks aktualnej nazwy w tablicy `imgs`. Najpierw jest zatem sprawdzane, czy indeks nie przekracza dopuszczalnej wartości. Jeśli przekracza (indeks większy bądź równy długości tablicy), jest zerowany, co oznacza powrót do pierwszego obrazu. Wartość zmiennej jest następnie używana do pobrania nazwy obrazu z tablicy `imgs`, w tej samej instrukcji zmienna jest też powiększana o jeden.

Na zakończenie za pomocą metody `setTimeout` jest ustawiany timer powodujący kolejne wywołanie funkcji `zmieńObraz` po czasie wskazywanym przez `timeout`. Ponieważ wartość zmiennej `timeout` ma być podawana w sekundach, a argument metody musi być określony w milisekundach, wykonywane jest też mnożenie `timeout * 1000`.

Skrypt 78. [C][E][F][O][S]

Obraz wyświetlany na nowej warstwie przykrywającej zawartość strony

Obrazy mogą być wyświetlane na dodatkowej modalnej warstwie, która przykrywa istniejącą zawartość strony. Najczęściej na witrynie widnieją wtedy miniatury, a na wspomnianej warstwie pojawiają się obrazy w pełnej rozdzielczości. Aby napisać taki skrypt, najlepiej wykorzystać pomysł przedstawiony przy omawianiu przykładu 9. w rozdziale 1. Należy tylko zadbać, aby rozmiary warstwy dialogowej (dialogDiv) nie były określone z góry, ale na podstawie rozdzielczości obrazu. W treści warstwy powinien się wtedy znaleźć element `img` zawierający wybrany plik graficzny. Część HTML skryptu będzie zatem wyglądała tak, jak zaprezentowano to na listingu 9.11.

Listing 9.11. Część HTML skryptu 78.

```
<body>
  <div id="transparentDiv">
  </div>
  <div id="dialogDiv" onclick="closeModal();">
  </div>
  <div id="mainDiv">
    
    
    
  </div>
</body>
```

Warstwy `transparentDiv` oraz `dialogDiv` realizują efekt modalności. Ich zadania są takie same jak w skrypcie 9. Różnica jest taka, że warstwa `dialogDiv` nie zawiera żadnej treści — treść będzie generowana dynamicznie — ma natomiast atrybut `onclick` wskazujący wykonywaną po kliknięciu funkcję JavaScriptu. Ponieważ jest to funkcja `closeModal`, kliknięcie warstwy (czy też umieszczonego na niej obrazu) będzie powodowało jej zamknięcie.

Warstwa `mainDiv` to główna warstwa witryny zawierająca treść. Zostały na niej umieszczone trzy znaczniki `` z trzema różnymi obrazami. Każdy znacznik otrzymał atrybut `onclick`, dzięki czemu obrazy będą reagowały na kliknięcia. W każdym przypadku zostanie wtedy wywołana funkcja `loadPic`. Argument przekazany funkcji wskazuje plik z obrazem o większej rozdzielczości, który pojawi się na ekranie.

Skrypt przyjmie postać zaprezentowaną na listingu 9.12.

Listing 9.12. Treść skryptu 78.

```
<script type="text/javascript">
  function loadPic(nazwa)
  {
    var dialogDiv = document.getElementById("dialogDiv");
```



```
if(!dialogDiv) return;
var img1 = document.getElementById('img1');
if(img1) dialogDiv.removeChild(img1);

var img = document.createElement("img");
img.setAttribute('id', 'img1');
img.src = nazwa;
dialogDiv.appendChild(img);
if(img.complete) showModal();
else img.onload = showModal;
}
function showModal(nazwa)
{
    var dialogDiv = document.getElementById("dialogDiv");
    var transparentDiv = document.getElementById("transparentDiv");
    if(!dialogDiv || !transparentDiv) return;

    dialogDiv.style.display = "block";
    var dW = dialogDiv.offsetWidth;
    var dH = dialogDiv.offsetHeight;

    //dalsza część funkcji showModal
}
function closeModal()
{
    //treść funkcji closeModal
}
}</script>
```

Ogólna zasada działania kodu pozostała podobna do przypadku przedstawionego w skrypcie 9. Wprowadzone zmiany dotyczą tylko tego, że tym razem rozmiary warstwy `dialogDiv` muszą być ustalane dynamicznie, gdyż każdy obraz może mieć inną rozdzielczość. Niezbędne jest także wstępne utworzenie obrazu. Funkcja `loadPic` otrzymuje w postaci argumentu nazwę (adres) pliku graficznego. Najpierw pobiera odwołanie do warstwy `dialogDiv`, na której ów obraz ma się pojawić, a następnie, jeżeli ta warstwa istnieje w kodzie, pobiera i zapisuje w zmiennej `img1` odwołanie do elementu o identyfikatorze `img1`. Jeżeli ten element istnieje, jest usuwany z warstwy za pomocą metody `removeChild`. Po wykonaniu tej operacji warstwa będzie pusta.

Następnie tworzony jest nowy element typu `img` oraz ustalane są jego atrybuty `src` i `id`. Atrybutowi `src` jest przypisywana wartość argumentu `nazwa` i w tym momencie rozpoczyna się ładowanie pliku graficznego do przeglądarki. Ponieważ aby prawidłowo wyświetlić obraz, musi on być w pełni załadowany (muszą być znane jego wymiary — na podstawie tego ustalane jest położenie warstwy `dialogDiv`), funkcja `showModal` jest wywoływana bezpośrednio tylko wtedy, gdy właściwość `complete` ma wartość `true`. W przeciwnym przypadku wywołanie jest przypisywane jako procedura obsługi zdarzenia `load` obrazu (ta technika została opisana dokładniej w skrypcie 83.).

Funkcja `showModal` nie przyjmuje tym razem (w porównaniu do przykładu 9.) żadnych parametrów, jest bowiem wywoływana, gdy ustalona jest już treść warstwy `dialogDiv`, a tym samym znane są już jej rozmiary. Można je odczytać z właściwości `offsetWidth`

(szerokość) i `offsetHeight` (długość), ale tylko wtedy, kiedy warstwa jest wyświetlana (cecha `display` obiektu `style` nie może być równa `none`). Dlatego też instrukcja włączająca warstwę:

```
dialogDiv.style.display = "block";
```

została przeniesiona na początek kodu. Odczytane wartości są zapisywane w zmiennych o takich samych nazwach jak argumenty pierwotnej wersji funkcji (zmiennie `dw` i `dH`). Dlatego też dalsze obliczenia będą miały taką samą postać jak w przykładzie z listingu 1.20. Dokładnie tak samo będzie też wyglądała funkcja `closeModal`.

Ponieważ ciągle usuwanie i tworzenie elementów typu `img` jest dosyć rozrzutnym rozwiązaniem (pod względem wykorzystania zasobów systemowych), można też zmienić kod, tak aby obraz tworzony był tylko raz. Funkcja `loadPic` mogłaby więc wyglądać np. tak:

```
function loadPic(nazwa)
{
    var dialogDiv = document.getElementById("dialogDiv");
    if(!dialogDiv) return;
    var img = document.getElementById('img1')
    if(!img){
        img = document.createElement("img");
        img.setAttribute('id', 'img1');
        dialogDiv.appendChild(img);
    }
    img.src = nazwa;
    if(img.complete) showModal();
    else img.onload = showModal;
}
```

Jednak wtedy przy wymianie obrazów w niektórych przeglądarkach może być przez chwilę widoczny poprzednio wyświetlany obraz. W kodach źródłowych dołączonych do książki uwzględnione zostały oba rozwiązania.

Skrypt 79. [C][E][F]

Obraz wyświetlany w nowym oknie

Skrypt 79. pozwala na otwieranie obrazów w nowym oknie przeglądarki. O ile samo otwarcie nowego okna wymaga po prostu wywołania metody `open` z odpowiednimi parametrami, co zostało dokładnie opisane w przykładzie 9., o tyle problemem jest dopasowanie jego rozmiarów do rozmiarów grafiki. Trzeba więc napisać odpowiednią funkcję JavaScriptu. W sekcji `<body>` witryny zostaną umieszczone obrazy w taki sposób, jak zaprezentowano to na listingu 9.13.

Listing 9.13. Sekcja `<body>` skryptu 79.

```
<body>
  <div id="mainDiv">
    
```

```


</div>
</body>
```

Użyte zostały zwyczajne znaczniki ``, którym przypisano atrybuty `onclick`. To oznacza, że kliknięcie dowolnego obrazu spowoduje wywołanie funkcji o nazwie `showImage` i przekazanie jej w postaci argumentu nazwy pliku z obrazem, który ma się pojawić w nowym oknie. Treść funkcji `showImage` została przedstawiona na listingu 9.14.

Listing 9.14. Skrypt otwierający obraz w nowym oknie

```
<script type="text/javascript">
function showImage(nazwa)
{
    var str = "status=no,location=no,menubar=no,toolbar=no.";
    str += "directories=no,scrollbars=no,resizable=no";
    var wnd = open(nazwa, "", "width=1,height=1,"+str);
    var img = new Image();
    img.onload = function(){
        if(wnd.innerWidth && wnd.innerHeight){
            wnd.innerWidth = img.width + 20;
            wnd.innerHeight = img.height + 20;
        }
        else
            wnd.resizeTo(img.width - 10, img.height);
    }
    img.src = nazwa;
}
</script>
```

Na początku w zmiennej pomocniczej `str` jest zapisywany ciąg opisujący parametry okna. Należy go dopasować do własnych potrzeb. Niezbędne wartości zostały przedstawione w rozdziale 1. przy opisie skryptu 1. Ciąg używany jest w metodzie `open` otwierającej nowe okno (odwołanie do okna jest zapisywane w zmiennej `wnd`). Zastosowany został uproszczony sposób umieszczania grafiki w oknie. Nie jest generowany żaden kod HTML, zamiast tego jako pierwszy argument została przekazana wartość parametru `nazwa`, czyli nazwa pliku graficznego. W takiej sytuacji przeglądarka po prostu wczytuje plik (nie mamy jednak kontroli nad jego prezentacją).

Niestety, metoda `open` wymaga podania konkretnych rozmiarów okna i sama nie dopasuje go do wymiarów obrazu. Dlatego też pierwotny rozmiar został określony jako `width=1,height=1`, co *de facto* oznacza najmniejszy możliwy wymiar dla danej przeglądarki (w każdej przeglądarce może być nieco inny). Nie ma to jednak znaczenia, gdyż właściwe rozmiary zostaną nadane dzięki kolejnym instrukcjom.

Aby uzyskać rozmiary obrazu, tworzony jest nowy obiekt typu `Image`:

```
var img = new Image();
```

Odpowiedni plik zostanie wczytany do takiego obiektu po przypisaniu nazwy pliku właściwości `src`:

```
img.src = nazwa;
```

Niestety, rozmiarów nie można odczytywać zaraz po takim przypisaniu, gdyż wczytanie danych graficznych zawsze zajmuje pewną chwilę i trzeba się najpierw upewnić, że proces ten został zakończony. Dlatego też zamiast bezpośredniego odczytu została zastosowana anonimowa funkcja obsługująca zdarzenie `load`. Zostanie wykonana dopiero wtedy, gdy dane graficzne zostaną pobrane z sieci (lub pamięci cache przeglądarki)¹. Dlatego też dopiero w tej funkcji następuje właściwa zmiana rozmiarów okna.

Szerokość obrazu odczytamy z właściwości `width`, a wysokość — z właściwości `height`. Modyfikacja wymiarów odbywa się w dwóch wariantach. W przeglądarkach, które obsługują właściwości `innerWidth` (szerokość wewnętrznego obszaru okna) i `innerHeight` (wysokość wewnętrznego obszaru okna), modyfikowane są właśnie te właściwości (przeprowadzana jest też drobna korekta wymiarów), w pozostałych przeglądarkach (chodzi głównie o rodzinę Internet Explorer) używana jest standardowa funkcja zmieniająca wymiary (`resizeTo`).

Skrypt 80. Lupa (powiększanie fragmentów obrazu)

[C][E][F][O][S]

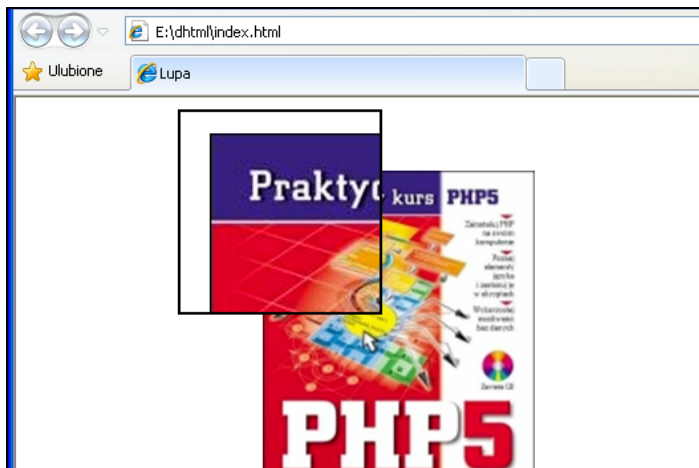
Ciekawym uatrakcyjnieniem strony WWW jest dodanie możliwości powiększania fragmentów obrazu za pomocą tzw. lupy w postaci prostokąta przesuwanego nad grafiką. Będzie to wyglądało tak jak na rysunku 9.3. Taki efekt można bez problemów zrealizować za pomocą JavaScriptu. Jak łatwo się domyślić, w charakterze lupy wystąpi osobna warstwa. Istnieją dwie główne metody realizacji tego typu skryptu. Pierwsza to umieszczenia na warstwie z lupą osobnego, odpowiednio skalowanego obrazu. Druga polega na manipulowaniu pozycją obrazu tła — ta właśnie technika jest podstawą skryptu 80. Kod HTML będzie bardzo prosty — został przedstawiony na listingu 9.15.

Listing 9.15. Kod HTML skryptu 80.

```
<body onload="setup('obraz2.jpg', 2);">
  <div id="imgDiv">
    
  </div>
  <div id="lupaDiv" onmousedown="startDrag(this);">
  </div>
</body>
```

¹ Metoda `open` mogłaby być wywoływana bezpośrednio w funkcji anonimowej, jednak większość przeglądarek blokuje możliwość bezpośredniego otwierania okien tworzonych w taki sposób. Użytkownik witryny zamiast obrazu zobaczyłby więc informację o blokadzie.

Rysunek 9.3.
Działanie lupy
powiększającej obraz



W sekcji `<body>` znajdują się dwie warstwy. Pierwsza zawiera obraz, który się pojawi na stronie po jej załadowaniu do przeglądarki. Obraz został zdefiniowany w typowy sposób, za pomocą znacznika ``. Co ważne, za pomocą atrybutu `id` został mu nadany identyfikator. To pozwoli na odwołanie się do tego elementu w kodzie skryptu. Druga warstwa ma identyfikator `lupaDiv` i będzie odpowiadała za realizację efektu lupy. Ponieważ musi być swobodnie przemieszczana po ekranie, atrybutowi `onmousedown` zostało przypisane wywołanie funkcji `startdrag` znanej ze skryptu 62. Znacznik `<body>` zawiera z kolei atrybut `onload`, któremu zostało przypisane wywołanie funkcji `setup`. Ta funkcja wykona czynności wstępne. Otrzymuje dwa argumenty. Pierwszy to adres pliku z przeskalowanym obrazem, a drugi — współczynnik skalowania.

A zatem, aby osiągnąć prawidłowy efekt przy danych podanych na listingu, należy przygotować dwa obrazy: *obraz.jpg* (ten pojawi się na stronie) i dwukrotnie powiększony *obraz2.jpg* (współczynnik skalowania równy 2).

Pierwsza część skryptu obsługującego efekt lupy została przedstawiona na listingu 9.16.

Listing 9.16. Skrypt realizujący efekt lupy

```
<script type="text/javascript">
var magnify = 0;
// tutaj początek skryptu z przykładu 62.
// tutaj treść funkcji mousePos, startDrag i mouseUp

function setup(imgName, multiple)
{
    var lupaDiv = document.getElementById('lupaDiv');
    if(!lupaDiv) return;
    lupaDiv.style.backgroundImage = "url('"+imgName+"')";
    lupaDiv.style.backgroundPosition = "-1000px -1000px";
    magnify = multiple;
}
// dalsza część skryptu
```

Technikę przemieszczania warstwy po ekranie można zapożyczyć bezpośrednio ze skryptu 62., dlatego też początek skryptu będzie bardzo podobny. Została jedynie dodana jedna zmienna globalna — `magnify` — przechowująca dane o wielkości powiększenia obrazu (jej wartość zostanie ustawiona przez funkcję `setup`). Funkcje `mousePos`, `startDrag` i `mouseUp` pozostały bez zmian. Pojawiła się natomiast nowa funkcja — `setup`. Przyjmuje dwa argumenty: `imgName`, określający adres (nazwę) pliku z obrazem, i `multiple`, określający wielokrotność powiększenia.

Funkcja `setup` pobiera odwołanie do warstwy z lupą (`lupaDiv`), a następnie, o ile operacja ta zakończyła się sukcesem, modyfikuje obiekt `style`. Ustala wartości właściwości `backgroundImage` i `backgroundPosition`. Pierwsza powoduje przypisanie warstwie obrazu tła, a druga przemieszcza tło tak, aby na pewno nie było widoczne (współrzędne `x` i `y` równe `-1000`). Na zakończenie wartość drugiego argumentu jest przypisywana globalnej zmiennej `magnify`.

W dalszej części skryptu należy umieścić funkcję `mouseMove`, która w tym przykładzie będzie się zajmowała nie tylko ustalaniem pozycji przesuwanej warstwy, ale także pozycji obrazu tła. Treść funkcji została przedstawiona na listingu 9.17.

Listing 9.17. Treść funkcji `mouseMove`

```
function mouseMove(evt)
{
    if(!draggedBox) return true;
    coords = mousePos(evt);

    draggedBox.style.position = "absolute";
    draggedBox.style.top = (coords.y - offset.y) + "px";
    draggedBox.style.left = (coords.x - offset.x) + "px";

    var imgOrg = document.getElementById('imgOrg');
    var lupaDiv = document.getElementById('lupaDiv');
    if(!imgOrg || !lupaDiv) return true;

    xPos = imgOrg.offsetLeft;
    yPos = imgOrg.offsetTop;
    lupaDiv.style.backgroundPosition =
        -((coords.x - Math.round(offset.x / 2) - xPos) * magnify) + "px " +
        -((coords.y - Math.round(offset.y / 2) - yPos) * magnify) + "px";
    return false;
}
```

Pierwsza część funkcji `mouseMove` pełni takie samo zadanie jak w przypadku przykładu 62., choć konstrukcja jest nieco inna. Odwrócony został warunek i tym razem po stwierdzeniu, że warstwa nie jest aktualnie przemieszczana (`!draggedBox`) działanie jest kończone, a w przeciwnym przypadku — kontynuowane. Kolejne cztery wiersze za instrukcją warunkową mają taką samą postać jak na listingu 7.12, tyle że nie znajdują się wewnątrz instrukcji (ze względu na odwrócenie warunku), ale poza nią.

Dalsze instrukcje odpowiadają za realizację efektu lupy. Najpierw pobierane jest odwołanie do obrazu znajdującego się na stronie (`imgOrg`) oraz do warstwy z lupą (`lupaDiv`).

Jeżeli któregoś z tych elementów nie będzie w kodzie strony, efekt nie będzie obsługiwany (funkcja jest opuszczana za pomocą instrukcji `return`). Następnie tworzone są zmienne pomocnicze `xPos` i `yPos`, którym jest przypisywana aktualna pozycja obrazu. Formalnie nie są one jednak konieczne. Zostały użyte wyłącznie ze względu na skrócenie kolejnej instrukcji kodu, której inaczej nie dałoby się rozbić na części funkcjonalne mieszczące się w osobnych wierszach na listingu.

Oczywiście do osiągnięcia dobrego efektu kluczowe jest odpowiednie pozycjonowanie obrazu tła w warstwie `lupaDiv`. Odpowiada za to kolejna instrukcja w postaci:

```
lupaDiv.style.backgroundPosition ="Xpx Ypx ";
```

Obliczenie konkretnych wartości X i Y zależy od danego projektu i tego, jak dokładnie ma się zachowywać lupa. W omawianym przypadku uwzględniane jest bieżące położenie myszy, pozycja kliknięcia warstwy z lupą, pozycja obrazu oraz oczywiście stopień powiększenia. Należy przy tym pamiętać, że standardowo tło pojawia się w lewym górnym rogu elementu, do którego zostało przypisane (czyli pozycja startowa to $(0,0)$). To oznacza, że należy je przesunąć w górę i w lewo poza obszar lupy. Stąd zastosowanie wartości ujemnych. Ostatecznie zostały użyte wzory:

```
left = -((coords.x - Math.round(offset.x / 2) - xPos) * magnify)
top = -((coords.y - Math.round(offset.y / 2) - yPos) * magnify)
```

gdzie:

- ◆ `coords.x` — współrzędna x aktualnego położenia kursora myszy;
- ◆ `coords.y` — współrzędna y aktualnego położenia kursora myszy;
- ◆ `offset.x` — współrzędna x miejsca kliknięcia warstwy `lupaDiv`;
- ◆ `offset.y` — współrzędna y miejsca kliknięcia warstwy `lupaDiv`;
- ◆ `xPos` — współrzędna x lewego górnego rogu oryginalnego obrazu (`imgOrg`);
- ◆ `yPos` — współrzędna y lewego górnego rogu oryginalnego obrazu (`imgOrg`);
- ◆ `magnify` — wielokrotność powiększenia.

Skrypt 81. [C][E7][F][O][S] Logo stale widoczne w wybranym miejscu strony

Skrypt 81. pozwala zamieścić na stronie logo, które będzie miało stałą pozycję względem okna przeglądarki. Do pozycjonowania wystarczyłyby tylko style CSS. Jeśli jednak chcielibyśmy dodatkowo uzyskać efekt stopniowego pojawiania się i (lub) znikania, potrzebny będzie też kod JavaScript. Sekcja `<body>` witryny będzie wyglądała tak, jak zaprezentowano to na listingu 9.18.

Listing 9.18. *Sekcja <body> skryptu 81.*

```
<body onload="initLogo('divLogo', 4);">
  <div id="mainDiv">
    Treść strony.
  </div>
  <div id="divLogo">
    To jest logo strony.
  </div>
</body>
```

W treści strony znajdują się dwie warstwy. Pierwsza (mainDiv) zawiera główną treść witryny, a druga (divLogo) służy do prezentacji logo. Zawartość drugiej warstwy może być dowolna. W prezentowanym przypadku jest to zwykły napis, ale można w niej umieścić również grafikę czy nawet prostą animację. Zdarzeniu load sekcji <body> została przypisana procedura obsługi w postaci funkcji initLogo. Ta funkcja zostanie zatem wywołana tuż po załadowaniu witryny do przeglądarki. Pierwszy argument wywołania określa identyfikator warstwy z logo, a drugi — po jakim czasie (w sekundach) logo ma zniknąć ze strony. Wartość 0 będzie oznaczała, że ma być stale obecne na witrynie. Style CSS będą miały postać przedstawioną na listingu 9.19.

Listing 9.19. *Style CSS dla skryptu 81.*

```
<style type="text/css">
  #divLogo{
    position:fixed;
    bottom:10px;
    right:10px;
    background-color:#F0F0F0;
    border:1px solid #A0A0A0;
    font-size:20pt;
    font-weight:bold;
    padding:4px;
  }
</style>
```

Najważniejsze są trzy pierwsze definicje. Stałe położenie warstwy z logo (divLogo) względem okna przeglądarki zapewnia cecha position o wartości fixed. Konkretna pozycja jest określana dzięki cechom bottom (odległość od dolnego brzegu dokumentu) i right (odległość od prawego brzegu dokumentu). Pozostałe cechy mają znaczenie dekoracyjne. Ustalony został kolor tła (background-color), obramowanie (border), rozmiar czcionki (font-size), grubość czcionki (font-weight) oraz wypełnienie (inaczej margines wewnętrzny — padding).

Na listingu 9.20 została przedstawiona pierwsza część skryptu zawierająca funkcję initLogo.

Listing 9.20. *Początek skryptu obsługującego warstwę z logo*

```
<script type="text/javascript">
  var opacity = 0.8;

  var timeout = 50;
```



```
var krok = 0.025;
var bieżącyKrok = 0;

function initLogo(id, x)
{
    var div = document.getElementById(id);
    if(!div) return;

    bieżącyKrok = 0;

    div.style.opacity = "0.0";
    div.style.filter = "alpha(opacity=0)";
    showLogo('show', id);

    if(x > 0) setTimeout("showLogo('hide', '"+id+"');", x * 1000);
}
//dalsza część skryptu
```

Na początku kodu znajdują się definicje zmiennych globalnych. Pierwsza z nich określa docelową przezroczystość warstwy `logoDiv`, czyli wartość atrybutu `opacity` obiektu `style`. Pozostałe będą używane w omówionej dalej funkcji `showLogo` przy wykonywaniu animacji pojawiania się i znikania logo. Ich znaczenie jest takie samo, jak było to w skrypcie 61.

Funkcja `initLogo` otrzymuje dwa argumenty. Pierwszy wskazuje identyfikator warstwy z logo, czyli tej, która ma się pojawić na ekranie i utrzymać swoją stałą pozycję. Drugi określa to, czy — a jeśli tak, to po jakim czasie — warstwa z logo ma zniknąć z ekranu. Najpierw za pomocą metody `getElementById` pobierane jest odwołanie do warstwy wskazywanej przez `id`. Następnie zerowana jest zmienna `bieżącyKrok` wskazująca aktualny stopień przezroczystości, a także ustalone są początkowe wartości cechy `opacity` (dla przeglądarek innych niż Internet Explorer) i `filter` (dla przeglądarek z rodziny Internet Explorer). Tym samym tuż po załadowaniu strony do przeglądarki warstwa `divLogo` będzie w pełni przezroczysta. Za doprowadzenie jej do stopnia przezroczystości wskazywanego przez zmienną `opacity` odpowiada funkcja `showLogo`.

Ostatnia instrukcja warunkowa bada stan argumentu `x`. Jeśli jest większy od zera, jest ustawiany timer powodujący wywołanie funkcji `showLogo` po czasie wskazywanym przez `x` (dzięki mnożeniu `x * 1000`, `x` może być wyrażane w sekundach). W tym wywołaniu wartością pierwszego argumentu jest ciąg `hide`, a zatem warstwa zostanie ukryta. Dzięki temu, jeśli `x` będzie większe od 0, logo zostanie ukryte po zadanim czasie, a jeśli będzie równe 0 (lub mniejsze od zera, ale taka sytuacja nie powinna mieć miejsca), logo będzie cały czas widoczne. Należy przy tym zwrócić uwagę, aby wartość `x` nie była zbyt mała (przy zadanych parametrach powinna być równa przynajmniej 2), gdyż inaczej nastąpi zablokowanie zmian przezroczystości przez dwa niezależne timery.

Treść funkcji `showLogo` została przedstawiona na listingu 9.21.

Listing 9.21. Treść funkcji `showLogo`

```
function showLogo(showhide, id)
{
    var obj = document.getElementById(id);
```

```
// dalsze instrukcje funkcji

if(bieżącyKrok > opacity) bieżącyKrok = opacity;
if(bieżącyKrok < 0) bieżącyKrok = 0;

obj.style.opacity = bieżącyKrok.toFixed(2);
obj.style.filter = "alpha(opacity:" +
    (bieżącyKrok * 100).toFixed(0) + ")";
if(bieżącyKrok != 0 && bieżącyKrok != opacity)
    timerId = setTimeout("showLogo('"+showhide+"','"+id+"');", timeout);
}
```

Funkcja `showLogo` wykonuje takie samo zadanie jak funkcja `show` z przykładu 61. (listing 7.8), czyli ma sprawić, aby warstwa stopniowo pojawiała się lub zniknęła z ekranu. Decyduje o tym parametr `showhide`. Gdy zawiera ciąg `show`, warstwa ma mieć zmniejszaną przezroczystość (czyli zwiększaną wartość cechy `opacity`), a w każdym innym przypadku przezroczystość ma być zwiększana (co jest równoznaczne ze zmniejszaniem cechy `opacity`). Ogólna konstrukcja kodu jest bardzo podobna do tej z listingu 7.8, występuje jednak kilka znaczących różnic. Przede wszystkim identyfikator warstwy, której dotyczy efekt, jest przekazywany jako argument `id` i w związku z tym ten identyfikator jest używany w wywołaniu metody `getElementById`. Po drugie, docelowa przezroczystość warstwy `divLogo` jest określana przez zmienną `opacity`. Dlatego też to wartość tej zmiennej, a nie wartość `1.0` jest używana w przypisaniu:

```
bieżącyKrok = opacity;
```

oraz warunku:

```
bieżącyKrok != opacity
```

Trzecia zmiana to postać pierwszego argumentu metody `setTimeout`. Ponieważ zawiera on wywołanie funkcji `showLogo`, a ta wersja funkcji przyjmuje dwa argumenty, konieczne było wprowadzenie odpowiednich modyfikacji i dodanie w wywołaniu argumentu `id`. Nieco inaczej wygląda również ostatnia instrukcja warunkowa. Została pozbawiona bloku `else`, bowiem tym razem nie są zapamiętywane identyfikatory timerów, nie ma więc również potrzeby ich zerowania.

Skrypt 82. Obraz płynący po stronie

[C][E][F][O][S]

W skrypcie 82. został pokazany sposób uzyskania efektu obrazu płynącego po witrynie, a dokładniej po jej wybranym fragmencie. Grafika będzie bowiem animowana wewnątrz warstwy, odbijając się od jej brzegów. Oczywiście sprawą kluczową dla uzyskania tego efektu jest umiejętność cyklicznego przemieszczania elementów witryny, co sprawia wrażenie płynnego ruchu. Uzyskuje się to za pomocą techniki timerów oraz modyfikacji cech CSS. Struktura kodu HTML (sekcje `<head>` i `<body>`) będzie taka, jak zaprezentowano to na listingu 9.22.

Listing 9.22. *Kod HTML skryptu do animacji obrazów*

```
<head>
  <meta http-equiv="Content-Type" content="text/html; charset=utf-8">
  <title>Obraz pływający po stronie</title>
  <style type="text/css">
    #mainDiv{
      width:600px;
      height:400px;
      border:1px solid black;
    }
    #img1{
      position:relative;
      border:1px solid black;
      cursor:pointer;
    }
  </style>
</head>
<body>
  <div id="mainDiv">
    
  </div>
</body>
</html>
```

W sekcji <body> znajduje się warstwa o identyfikatorze mainDiv, a w niej obraz img1. Obraz będzie reagował na kliknięcia dzięki użyciu atrybutu onclick. Każde kliknięcie spowoduje wywołanie funkcji startstop i przekazanie jej w postaci argumentu odwołania do klikniętego obrazu (this). Pierwsze kliknięcie będzie rozpoczynało ruch obrazu, a ponowne — jego zatrzymanie.

W sekcji <head> znajdują się definicje stylu CSS. Ponieważ ruch będzie się odbywał wewnątrz warstwy mainDiv, zostały określone jej wymiary oraz obramowanie (cechy width, height i border selektora #mainDiv). Można je oczywiście dowolnie modyfikować. W regule z selektorem #img1 zostały natomiast określone cechy obrazu. Otrzymał obramowanie, zmieniony został także kształt kursora. Co ważne, sposób pozycjonowania został określony na względny (relative). Dzięki temu pozycja obrazu będzie ustalana względem warstwy (elementu) nadrzędnej (czyli w tym przypadku warstwy mainDiv).

Do sekcji <head> trzeba jeszcze dodać skrypt zawierający definicję funkcji startstop. Jego pierwsza część została przedstawiona na listingu 9.23.

Listing 9.23. *Pierwsza część skryptu wykonującego animację obrazu*

```
<script type="text/javascript">
  var timerId = null;
  var activeImg = null;
  var timeout = 10;
  var stepX = 1;
  var stepY = 1;
  function startstop(img)
  {
    if(!img) return;
```

```

    if(activeImg){
        clearInterval(timerId);
        timerId = null;
        activeImg = null;
    }
    else{
        activeImg = img;
        timerId = setInterval("move()", timeout);
    }
}
//dalsza część skryptu

```

Na początku kodu znajdują się definicje zmiennych globalnych sterujących pracą skryptu. Są to:

- ♦ timerId — identyfikator timera używanego do wywołań cyklicznych;
- ♦ activeImg — wskazanie do przesuwanego obrazu;
- ♦ timeout — czas pomiędzy kolejnymi przesunięciami obrazu (wywołaniami opisanej niżej funkcji move);
- ♦ stepX — liczba pikseli, o którą obraz zostanie przesunięty w poziomie w jednej fazie ruchu;
- ♦ stepY — liczba pikseli, o którą obraz zostanie przesunięty w pionie w jednej fazie ruchu.

Funkcja startstop otrzymuje argument img wskazujący obraz podlegający animacji. Najpierw sprawdza więc, czy argument nie jest pusty. Jeśli jest, wykonywanie kodu jest przerywane. Zadaniem funkcji jest rozpoczęcie lub zatrzymanie ruchu obrazu. Obraz należy zatrzymać, jeśli aktualnie jest w ruchu, czyli wtedy, gdy zmienna activeImg jest różna od null (prawdziwy jest warunek if(activeImg){}). W takiej sytuacji za pomocą metody clearInterval wyłączany jest timer o identyfikatorze zapisanym w zmiennej timerId, a zmienne timerId i activeImg są zerowane (jest im przypisywana wartość null). Jeżeli jednak activeImage nie jest pusta, należy uruchomić timer (rozpocząć ruch). Zmiennej activeImg jest wtedy przypisywana wartość argumentu img oraz jest wykonywana metoda setInterval. Spowoduje ona cykliczne wywoływanie funkcji move co timeout milisekund.

Treść funkcji move została przedstawiona na listingu 9.24.

Listing 9.24. Treść funkcji move

```

function move()
{
    if(!activeImg || !activeImg.parentNode) return;
    var maxX = activeImg.parentNode.offsetWidth - activeImg.offsetWidth;
    var maxY = activeImg.parentNode.offsetHeight - activeImg.offsetHeight;
    var x, y;
    x = isNaN(x = parseInt(activeImg.style.left))?0:x;
    y = isNaN(y = parseInt(activeImg.style.top))?0:y;

    x = x + stepX;

```

```
y = y + stepY;

if(x >= maxX){
    x = maxX;
    stepX = -stepX;
}
if(y >= maxY){
    y = maxY;
    stepY = -stepY;
}
if(x <= 0){
    x = 0;
    stepX = -stepX;
}
if(y <= 0){
    y = 0;
    stepY = -stepY;
}
activeImg.style.left = x + "px";
activeImg.style.top = y + "px";
}
```

Funkcja `move` ma przesuwać obraz wskazywany przez globalną zmienną `activeImg` w obrębie elementu nadrzędnego (czyli warstwy `mainDiv`). Najpierw zatem sprawdza, czy istnieje obraz `activeImg` (czy zmienna nie jest pusta) oraz czy istnieje element nadrzędny (`activeImg.parentNode`). Jeżeli jeden z tych dwóch warunków nie jest prawdziwy, wykonywanie kodu jest przerywane za pomocą instrukcji `return`.

W zmiennych pomocniczych `maxX` i `maxY` zapisywane są maksymalne współrzędne położenia poziomego i pionowego obrazu — nie może on bowiem wykroczyć poza ramy warstwy, w której jest zawarty. Obliczenia są wykonywane za pomocą prostych wzorów:

```
maxX = szerokość warstwy - szerokość obrazu
maxY = wysokość warstwy - wysokość obrazu
```

Szerokość danego elementu w pikselach można odczytać z właściwości `offsetWidth`, a wysokość — z właściwości `offsetHeight`. A więc np. szerokość obrazu to `activeImg.offsetWidth`, a wysokość warstwy to `activeImg.parentNode.offsetHeight`.

W zmiennych `x` i `y` są zapisywane współrzędne aktualnego położenia obrazu. Współrzędna `x` odczytywana jest z właściwości `left` (`activeImg.style.left`), a współrzędna `y` — z właściwości `top` obiektu `style` (`activeImg.style.top`). Ponieważ należy uwzględnić sytuację, że te cechy CSS nie zostały zdefiniowane w kodzie HTML, dane są weryfikowane za pomocą konstrukcji z wyrażeniem warunkowym (opisanej w przy omawianiu skryptu 54.). W przypadku wykrycia nieprawidłowości (brak definicji dla cech `top` i `left`) współrzędne otrzymują wartość 0, co oznacza umiejscowienie obrazu w lewym górnym rogu warstwy.

Po dokonaniu wszystkich opisanych obliczeń współrzędna `x` (zmienna `x`) jest zwiększana o wartość zapisaną w `stepX`, a współrzędna `y` (zmienna `y`) — o wartość zapisaną w `stepY`. Dzięki użyciu dwóch zmiennych można osobno definiować przesunięcie

w poziomie i w pionie. Po zwiększeniu wartości zmiennych jest badane, czy ich wartości nie przekroczyły dopuszczalnych granic. Sprawdzane są warunki:

- ♦ $x \geq \max X$ — czy obraz wykracza poza prawą krawędź warstwy;
- ♦ $y \geq \max Y$ — czy obraz wykracza poza dolną krawędź warstwy;
- ♦ $x \leq 0$ — czy obraz wykracza poza lewą krawędź warstwy;
- ♦ $y \leq 0$ — czy obraz wykracza poza górną krawędź warstwy.

Jeżeli nastąpiło przekroczenie granicy, przeprowadzane są korekty położenia oraz zmieniany jest kierunek ruchu, czyli znak odpowiedniej zmiennej. To znaczy, jeśli obraz dotarł do prawego bądź lewego brzegu warstwy, trzeba zmienić kierunek ruchu w poziomie (zmienić znak zmiennej $\text{step}X$), a jeżeli obraz dotarł do górnego bądź dolnego brzegu warstwy, trzeba zmienić kierunek ruchu w pionie (zmienić znak zmiennej $\text{step}Y$).

Kiedy wszelkie korekty zostaną wprowadzone, następuje zmiana pozycji obrazu `activeImg`, co odbywa się przez przypisanie wartości zmiennej `x` właściwości `left` obiektu `style`, a wartości zmiennej `y` właściwości `top` obiektu `style`.

Skrypt 83. Ładowanie obrazów z paskiem postępu (I)

[C][E][F][O][S]

Czasami na stronie trzeba zamieścić większą liczbę obrazów czy też tylko kilka, ale o dużej objętości. Co zrobić, jeśli chcemy, aby grafiki pojawiły się na ekranie dopiero po ich całkowitym załadowaniu, ale użytkownik nie opuścił naszej witryny? Najlepiej poinformować go, że trwa proces ładowania danych. Można do tego użyć techniki paska postępu zaprezentowanej w skrypcie 50. W skrypcie 83. zobrazowano, jak sterować paskiem podczas ładowania plików graficznych i jak rozpoznać, kiedy dany plik został załadowany.

Część HTML witryny będzie miała postać przedstawioną na listingu 9.25.

Listing 9.25. Sekcja `<body>` skryptu 83.

```
<body>
  <div id="divLoading">
    Trwa ładowanie obrazów. Proszę czekać...
    <div id="divPasekZew">
      <div id="divPasekWew">
      </div>
    </div>
  </div>
  <div id="imgDiv">
    
    
```

```

</div>
<script type="text/javascript">
  start('imgDiv', 'divLoading');
</script>
</body>
```

Pierwsza warstwa, o identyfikatorze `divLoading`, zawiera napis informujący o procesie ładowania obrazów oraz warstwy stanowiące pasek postępu. Postać i działanie paska są takie same jak w skrypcie 50. Druga warstwa — `imgDiv` — zawiera obrazy, które mają się pojawić na stronie dopiero po ich pełnym wczytaniu przez przeglądarkę. Za warstwą został umieszczony znacznik `<script>` zawierający instrukcję wywołującą funkcję `start`. Otrzymuje ona dwa argumenty. Pierwszy to identyfikator warstwy zawierającej obrazy, drugi — identyfikator warstwy zawierającej pasek postępu.

Początkowo na stronie będzie widoczna tylko warstwa z paskiem postępu (warstwa z obrazami zostanie ukryta przez skrypt). Po załadowaniu obrazów zostanie pokazana warstwa z obrazami, a warstwa z paskiem zostanie ukryta. Obsługą paska postępu zajmie się skrypt, którego pierwsza część została przedstawiona na listingu 9.26.

Listing 9.26. *Pierwsza część skryptu obsługującego pasek postępu*

```
<script type="text/javascript">
  var bieżącyKrok = 0;
  var krok = 0;
  var licznik = 0;
  var imgDiv, barDiv;
  function ustawPasek(wartość)
  {
    //treść funkcji ustawPasek
  }
  function start(id1, id2)
  {
    imgDiv = document.getElementById(id1);
    barDiv = document.getElementById(id2);
    if(!imgDiv || !barDiv) return;
    imgDiv.style.display = "none";
    barDiv.style.display = "block";

    var imgs = imgDiv.getElementsByTagName('img');
    licznik = imgs.length;
    krok = licznik != 0?Math.ceil(100 / licznik):100;
    for(var i = 0; i < imgs.length; i++){
      if(imgs[i].complete){
        onimgload();
      }
      else{
        imgs[i].onload = onimgload;
      }
    }
  }
}
//dalsza część skryptu
```

Na początku kodu znajdują się definicje zmiennych globalnych:

- ♦ `bieżącyKrok` — aktualna wartość paska stanu;
- ♦ `krok` — wartość, o którą ma być zwiększony pasek po załadowaniu jednego obrazu;
- ♦ `licznik` — liczba załadowanych obrazów;
- ♦ `imgDiv` — odwołanie do warstwy z obrazami;
- ♦ `barDiv` — odwołanie do warstwy z paskiem postępu.

Funkcja `start` jest wykonywana po wczytaniu przez przeglądarkę warstwy `imgDiv` i rozpoczęciu ładowania obrazów. Przyjmuje dwa argumenty `id1` i `id2`. Pierwszy określa identyfikator warstwy z obrazami (tu `imgDiv`), drugi — identyfikator warstwy z paskiem postępu. Odwołania do tych warstw są pobierane za pomocą metody `getElementById` i zapisywane w zmiennych globalnych `imgDiv` i `barDiv`. Dzięki temu odwołań nie trzeba będzie pobierać ponownie w dalszej części kodu. Wyświetlanie warstwy `imgDiv` jest wyłączane, a wyświetlanie warstwy `barDiv` — włączane. Odbyna się to przez zmianę wartości właściwości `display` obiektu `style` (`block` — warstwa obecna na ekranie, `none` — brak warstwy na ekranie).

Liczba obrazów, które mają się pojawić na stronie, nie jest w żaden sposób limitowana ani ograniczana przez skrypt i zależy wyłącznie od liczby znaczników `` na warstwie o identyfikatorze przekazanym w postaci argumentu `id1` (warstwa `imgDiv` w kodzie HTML). To oznacza, że skrypt musi samodzielnie odnaleźć wszystkie obrazy i pobrać odwołania do nich. Jest to wykonywane za pomocą metody `getElementsByTagName`:

```
var imgs = imgDiv.getElementsByTagName('img');
```

Po wykonaniu powyższej instrukcji zmienna `imgs` będzie zawierała tablicę (kolekcję) z wszystkimi obrazami (elementami zdefiniowanymi w kodzie HTML za pomocą znacznika ``) znajdującymi się w warstwie wskazywanej przez `imgDiv`. To pozwala ustalić stan zmiennych sterujących: `licznik` i `krok`. Pierwsza będzie zawierała całkowitą liczbę obrazów pobraną z właściwości `length` tablicy `imgs`, druga — wartość, o którą ma zostać zwiększony pasek postępu w jednym kroku (po załadowaniu jednego obrazu). Wartość zmiennej `krok` wynika oczywiście z działania

$$\frac{100}{\text{liczba obrazów}}$$

Ponieważ w przypadku braku obrazów na warstwie `imgDiv` wartość zmiennej `krok` byłaby równa 0 i powyższe działanie spowodowałoby błąd skryptu, używana jest złożona konstrukcja z operatorem warunkowym, korygująca dane w takiej sytuacji (gdy `licznik` będzie równa 0, `krok` otrzyma wartość 100)².

² Ze względu na użycie zaokrąglenia w górę (`Math.ceil`), jeśli obrazów będzie więcej niż 100, pasek osiągnie pełną wartość już po 100 obrazach.

Po przygotowaniu wszystkich opisanych danych można przystąpić do badania stanu obrazów i przesuwania paska postępu. Odbyna się to w pętli `for`. Po załadowaniu każdego z obrazów należy wywołać funkcję `onimageLoad`, która uaktualni pasek postępu, zmniejszy licznik i stwierdzi, czy wszystkie obrazy są już gotowe. Nie można tego jednak zrobić bezpośrednio, ale w przypadku każdego z obrazów zapisanych w tablicy `imgs` należy rozpoznać dwie sytuacje. Albo obraz jest już w całości załadowany (np. wczytany z pamięci podręcznej przeglądarki), albo też wciąż trwa jego ładowanie (bądź czeka on na swoją kolej do załadowania). To, z którą z sytuacji mamy do czynienia, stwierdzamy, badając stan właściwości `complete` (`if(imgs[i].complete){}`).

Gdy `complete` jest równa `true`, oznacza to, że obraz jest załadowany. Wtedy po prostu jest wywoływana funkcja `onimgLoad`. Jeśli jednak `complete` jest równe `false`, oznacza to, że obraz nie jest gotowy, a funkcja `oncomplete` powinna być wywołana dopiero po jego pobraniu. Na całe szczęście obiekty typu `Image` mają właściwość `onload` pozwalającą na przypisanie funkcji, która ma być wykonana po załadowaniu obrazu. Zatem gdy `complete` jest równa `false`, właściwości `onload` danego obrazu jest przypisywana funkcja `onimgLoad`:

```
imgs[i].onload = onimgLoad;
```

To powoduje, że w przypadku tego obrazu funkcja `onimgLoad` zostanie wykonana dopiero, gdy przeglądarka pobierze i przetworzy wszystkie dane graficzne.

Treść funkcji `onimgLoad` została przedstawiona na listingu 9.27.

Listing 9.27. *Treść funkcji `load`*

```
function onimgLoad()
{
    ustawPasek(biezącyKrok + krok);
    if(--licznik <= 0){
        imgDiv.style.display = "block";
        barDiv.style.display = "none";
    }
}
```

Funkcja jest wykonywana po załadowaniu każdego obrazu umieszczonego na warstwie `imgDiv`. Najpierw za pomocą funkcji `ustawPasek` przesuwa pasek postępu o jeden krok. Następnie zmniejsza licznik obrazów o jeden (`--licznik`) i sprawdza, czy bieżący obraz był ostatnim. Będzie tak wtedy, gdy wartość zmiennej `licznik` osiągnie 0. W takiej sytuacji chowana jest warstwa z paskiem postępu (`barDiv`), a odkrywana warstwa z obrazami (`imgDiv`). Tym samym kończy się też działanie skryptu.

Uwaga! Aby zaobserwować efekt działania skryptu, strona powinna być umieszczona w sieci i najlepiej, by pliki z obrazami miały różne wielkości — dane graficzne nie mogą się też znajdować w pamięci podręcznej przeglądarki. Inaczej wszystkie obrazy zostaną wczytane tak szybko, że pasek nie będzie mógł być dostrzeżony.

Skrypt 84.

Ładowanie obrazów z paskiem postępu (II)

[C][E][F][O][S]

W skrypcie 83. zostało pokazane, jak użyć paska postępu informującego o stopniu zaawansowania operacji ładowania obrazów umieszczonych na wybranej warstwie danej strony WWW. Często wykonywana jest też jednak nieco inna operacja — ładowanie plików graficznych bezpośrednio w skrypcie i dopiero późniejsze używanie ich na witrynie. W takiej sytuacji też można zastosować pasek postępu. Jak to zrobić, zostanie pokazane w tym właśnie skrypcie. Zastosowana zostanie też zupełnie inna technika sprawdzania stopnia zaawansowania operacji pobierania danych. Będzie to zliczanie i badanie cykliczne.

Część HTML strony została przedstawiona na listingu 9.28. Będzie służyła jedynie do demonstracji całego efektu.

Listing 9.28. *Część HTML skryptu 84.*

```
<body>
  <div id="divLoading">
    <input type="button" value="Ładuj obrazy" id="btnLoad"
      onclick="btnLoadClick(załadowane);" />
    <div id="divPasekZew">
      <div id="divPasekWew">
      </div>
    </div>
  </div>
  <div id="infoDiv">
  </div>
</body>
```

W sekcji `<body>` znalazły się dwie główne warstwy: `divLoading` oraz `infoDiv`. Pierwsza zawiera przycisk uruchamiający procedurę ładowania, czyli wywołujący funkcję `btnLoadClick` (w konkretnym projekcie ta funkcja zostałaby wywołana automatycznie, w trakcie ładowania strony) oraz warstwy tworzące pasek postępu (`divPasekZew` i `divPasekWew`). Funkcja `btnLoadClick` otrzymuje w postaci argumentu nazwę funkcji, która ma być wykonana po załadowaniu danych.

Druga warstwa jest początkowo pusta. Na niej pojawi się komunikat informujący o zakończeniu procesu pobierania obrazów. Początek skryptu obsługującego stronę został przedstawiony na listingu 9.29.

Listing 9.29. *Początek skryptu ładującego obrazy*

```
<script type="text/javascript">
  var nazwy = new Array("joowww_big.jpg", "php102_big.jpg", "php5pk_big.jpg");
  var obrazy = new Array();
  var bieżącyKrok = 0;
  var krok = 0;
```

```
var timerId = null;
function ustawPasek(wartość)
{
    //treść funkcji ustaw pasek
}
function btnLoadClick(func)
{
    var btnLoad = document.getElementById('btnLoad');
    if(btnLoad) btnLoad.disabled = true;
    bieżącyKrok = 0;
    krok = nazwy.length != 0?Math.ceil(100 / nazwy.length):100;
    for(var i = 0; i < nazwy.length; i++){
        obrazy[i] = new Image();
        obrazy[i].src = nazwy[i];
    }
    var timerId = setInterval("aktualizujDane("+func+")", 250);
}
//dalsza część skryptu
```

Na początku kodu znajdują się definicje zmiennych globalnych:

- ◆ nazwy — tablica zawierająca nazwy plików graficznych (jeśli pliki znajdują się w innej lokalizacji niż skrypt, należy uwzględnić ścieżkę dostępu lub cały URL);
- ◆ obrazy — tablica zawierająca obiekty wczytanych obrazów;
- ◆ bieżącyKrok — aktualny stan paska postępu;
- ◆ krok — wartość, o którą ma być zwiększony pasek po załadowaniu jednego obrazu;
- ◆ timerId — identyfikator timera używanego do cyklicznego badania liczby załadowanych obrazów.

Funkcja *ustawPasek* zajmuje się modyfikacją paska postępu i ma taką samą postać jak w przykładzie 50. Funkcja *btnLoadClick* jest wykonywana po kliknięciu przycisku *Ładuj obrazy*, a jej zadaniem jest ustalenie parametrów skryptu i rozpoczęcie ładowania obrazów. Otrzymuje jeden argument — *func* — wskazujący funkcję, która ma być wykonana po zakończeniu procedury pobierania danych.

Na początku za pomocą metody *getElementById* pobierane jest odwołanie do przycisku *btnLoad* (*Ładuj obrazy*), a przycisk jest wyłączany przez przypisanie wartości *true* właściwości *disabled*. Następnie zerowana jest zmienna *bieżącyKrok* oraz wyliczana wartość zmiennej *krok*. Obliczenie odbywa się na tej samej zasadzie jak w przypadku skryptu 83. — wartość 100 jest dzielona przez liczbę obrazów (pobieraną tym razem z właściwości *length* tablicy *nazwy*).

W pętli *for* tworzona jest tablica z obrazami (*obrazy*). Każdy obraz jest tworzony przez wywołanie konstruktora typu *Image* i zapisywany pod indeksem wskazywanym przez zmienną iteracyjną *i*. Następnie właściwości *src* (wskazuje źródło obrazu) jest przypisywana wartość pobrana z tablicy *nazwy*, a wskazująca nazwę (lub adres) pliku graficznego. To rozpoczyna proces ładowania obrazu przez przeglądarkę.

Po zakończeniu pętli za pomocą metody `setInterval` ustawiany jest timer wywołujący cyklicznie funkcję `aktualizujDane`, która zajmie się badaniem stanu procesu pobierania obrazów. Funkcji tej przekazywany jest argument `func` otrzymany przez funkcję `btnLoadClick`.

Treść funkcji `aktualizujDane` wraz z dalszą częścią skryptu została przedstawiona na listingu 9.30.

Listing 9.30. *Dalsza część skryptu 84.*

```
function aktualizujDane(func)
{
    var licznik = 0;
    for(var i = 0; i < obrazy.length; i++){
        if(obrazy[i].complete) licznik++;
    }
    ustawPasek(krok * licznik);
    if(licznik >= obrazy.length){
        clearTimeout(timerId);
        if(typeof(func) == 'function') func();
    }
}
function załadowane()
{
    var btnLoad = document.getElementById('btnLoad');
    if(btnLoad) btnLoad.disabled = false;
    var div = document.getElementById('infoDiv');
    if(div) div.innerHTML = "Obrazy zostały załadowane.";
}
```

Funkcja `aktualizujDane` przyjmuje argument wskazujący, jaka funkcja ma być wykonana po wykryciu, że wszystkie obrazy zostały załadowane. W omawianym przykładzie będzie to przedstawiona niżej funkcja `załadowane` (jest uwzględniona w procedurze zdarzenia `click` przycisku *Ładuj obrazy*, 4. linia na listingu 9.28). W każdym wywołaniu funkcji `aktualizujDane` trzeba policzyć, ile obrazów jest aktualnie załadowanych. Odbywa się to w pętli `for`, która przebiega całą tablicę `obrazy`. We wnętrzu pętli badany jest stan właściwości `complete` każdego elementu tablicy (każdego obrazu). Jeśli w danym przypadku `complete` jest równe `true` (`if(obrazy[i].complete)`), zmienna `licznik` jest zwiększana o 1 (`licznik++`). To oznacza, że po zakończeniu pętli zmienna `licznik` będzie zawierała liczbę wczytanych obrazów.

Gdy zmienna `licznik` zawiera liczbę gotowych grafik, jest używana do ustawienia paska postępu. Aktualny stan paska jest wyliczany ze wzoru `krok * licznik`. Następnie badane jest, czy wczytane zostały wszystkie obrazy, czyli czy `licznik` jest równy wielkości tablicy `obrazy` zapisanej we właściwości `length` (`obrazy.length`). Jeśli tak jest, za pomocą metody `clearTimeout` jest wyłączany timer, a także jest wywoływana funkcja przekazana w postaci argumentu `func`. To ostatnie działanie jest wykonywane tylko wtedy, jeśli typem argumentu jest faktycznie funkcja (jest to sprawdzane za pomocą operatora `typeof`).

Funkcja załadowane jest wykonywana po zakończeniu ładowania obrazów. Zajmuje się włączeniem wyłączzonego wcześniej przycisku `btnLoad` (*Ładuj obrazy*), a także wyświetleniem na warstwie `infoDiv` komunikatu o zakończeniu pobierania danych. Komunikat staje się treścią warstwy, ponieważ jest przypisywany właściwości `innerHTML`.

Skrypt 85. [C][E][F][O][S] Galeria obrazów z podpisami

Wszelkiego rodzaju galerie obrazów to bardzo popularny element stron WWW. Taką właśnie funkcjonalność realizuje skrypt 85. Będzie to galeria obrazów z podpisami. W jednej chwili na stronie będzie prezentowany plik graficzny, jego opis, informacja o numerze obrazu i całkowitej liczbie obrazów, a także elementy nawigacyjne pozwalające na przeniesienie do obrazu pierwszego, poprzedniego, następnego i ostatniego. Fragment takiej witryny został przedstawiony na rysunku 9.4. Część HTML kodu będzie bardzo prosta. Została przedstawiona na listingu 9.31.

Rysunek 9.4.
*Interfejs galerii
ze skryptu 85.*



Listing 9.31. Sekcja `<body>` skryptu 85.

```
<body onload="zmieńObraz(0);">
  <div id="imgDiv">
    <img src="" alt="" id="img1" />
    <div id="descDiv"></div>
    <div id="navDiv"></div>
  </div>
</body>
```

W sekcji `<body>` znajduje się warstwa `imgDiv` zawierająca elementy galerii. Są to: znacznik `` — definiujący obraz, warstwa `descDiv` — przechowująca opis, i warstwa `navDiv` — przechowująca elementy nawigacyjne. Wszystkie elementy są początkowo puste. Obraz nie ma przypisanego adresu grafiki, a warstwy `descDiv` i `navDiv` nie mają żadnej zawartości. Cała treść będzie generowana dynamicznie przez skrypt. Taka strona tuż po załadowaniu do przeglądarki byłaby pusta. Dlatego też atrybut `onload` znacznika

<body> zawiera wywołanie funkcji o nazwie `zmieńObraz`. Ta funkcja otrzymuje w postaci argumentu numer obrazu, który ma się znaleźć na stronie. Jej treść wraz z pozostałą częścią skryptu została przedstawiona na listingu 9.32.

Listing 9.32. Skrypt galerii obrazów

```
<script type="text/javascript">
  var imgObjs = new Array(
    {src:"php102_big.jpg",
     desc:"Okładka książki PHP. 101 praktycznych skryptów"},
    {src:"joowww_big.jpg",
     desc:"Okładka książki Joomla! 1.5. Prosty przepis na własną stronę WWW"},
    {src:"php5pk_big.jpg",
     desc:"Okładka książki PHP5. Praktyczny kurs"}
  );
  function zmieńObraz(nr)
  {
    if(!imgObjs || imgObjs.length < 1) return;
    nr = isNaN(nr = parseInt(nr))?0:nr;
    if(nr < 0 || nr >= imgObjs.length) nr = 0;

    var img = document.getElementById("img1");
    var descDiv = document.getElementById("descDiv");
    var navDiv = document.getElementById("navDiv");
    if(!img || !descDiv || !navDiv) return;

    img.src = imgObjs[nr].src;
    descDiv.innerHTML = imgObjs[nr].desc;
    var last = imgObjs.length - 1;
    var prev = nr > 0?nr - 1 : 0;
    var next = nr < imgObjs.length - 1?nr + 1:imgObjs.length - 1;

    var str = "Obraz " + (nr + 1) + " z " + imgObjs.length + "<br />";
    str += "<span onclick='zmieńObraz(0)'>Pierwszy</span> ";
    str += "<span onclick='zmieńObraz(\"+prev+\")'>Poprzedni</span> ";
    str += "<span onclick='zmieńObraz(\"+next+\")'>Następny</span> ";
    str += "<span onclick='zmieńObraz(\"+last+\")'>Ostatni</span>";
    navDiv.innerHTML = str;
  }
</script>
```

Na początku kodu znajduje się definicja tablicy `imgObjs`, która przechowuje nazwy (adresy) plików graficznych oraz ich opisy. Każdy element tablicy to osobny obiekt w formacie JSON³ składający się z dwóch pól (właściwości): `src` (zawiera nazwę i ewentualnie adres pliku graficznego) i `desc` (zawiera opis obrazu).

Funkcja `zmieńObraz` przyjmuje argument `nr` wskazujący indeks komórki tablicy `imgObjs`, z której mają być pobrane dane. Inaczej mówiąc, to numer obrazu pomniejszony o 1 (bo- wiem tablica jest numerowana od 0, a obrazy zazwyczaj numerujemy od 1). Pierwszą

³ Dokładny opis notacji JSON można znaleźć m.in. pod adresem <http://www.json.org/>. Informacje o tym formacie zostały również zawarte w publikacjach: *JavaScript. Praktyczny kurs* (<http://helion.pl/ksiazki/jscpk.htm>) oraz *AJAX. Ćwiczenia* (<http://helion.pl/ksiazki/cajax.htm>).

wykonywaną czynnością jest sprawdzenie, czy istnieje tablica `imgObjs` (a dokładniej, czy zmienna `imgObjs` jest niepusta) oraz czy wielkość tablicy jest większa od 0 (czyli czy zawiera co najmniej jeden obiekt z danymi obrazu). Jeżeli odpowiedź na jedno z tych pytań brzmi „nie”, wykonywanie kodu jest przerywane za pomocą instrukcji `return`.

Kolejne czynności to weryfikacja stanu argumentu `nr`. Najpierw za pomocą złożonej instrukcji z wyrażeniem warunkowym oraz metodami `parseInt` i `isNaN` jest on przetwarzany na wartość całkowitą. Gdyby przekazana wartość nie była całkowita, argument otrzymuje wartość 0 (ta procedura została opisana przy skrypcie 54.). Następnie badane jest, czy `nr` przekracza dopuszczalny zakres, czyli czy jest mniejszy od 0 lub większy od liczby obrazów pomniejszonej o 1. Innymi słowy, czy przekracza dopuszczalny indeks tablicy `imgObjs`. Jeżeli zostanie wykryte przekroczenie, wartość jest korygowana do 0.

Dalsze instrukcje to pobranie odwołań do elementów witryny i zapisanie ich w zmiennych pomocniczych: `img` (odwołanie do elementu ``), `descDiv` (odwołanie do warstwy `descDiv`) i `navDiv` (odwołanie do warstwy `navDiv`). Badane jest także to, czy te czynności zakończyły się sukcesem (czy wszystkie wymienione zmienne są niepuste). Jeżeli któregoś elementu brakuje, działanie funkcji jest kończone.

Adres (nazwa) pliku graficznego jest pobierany z właściwości `src` obiektu znajdującego się w tablicy `imgObjs` pod indeksem `nr` i przypisywany właściwości `src` obiektu `imgDiv`:

```
img.src = imgObjs[nr].src;
```

Dzięki temu obraz pojawia się na witrynie. W podobny sposób wyświetlany jest opis. Treść opisu jest pobiera z właściwości `desc`, a przypisywana właściwości `innerHTML` warstwy `descDiv`:

```
descDiv.innerHTML = imgObjs[nr].desc;
```

Nieco więcej pracy wymaga utworzenie panelu nawigacyjnego pozwalającego na wyświetlanie obrazu pierwszego, poprzedniego, następnego i ostatniego. Najpierw obliczane są wartości indeksów. Obraz pierwszy znajduje się pod indeksem 0 — tu nie trzeba nic wyliczać. Obraz ostatni znajduje się pod indeksem wynikającym z działania liczba obrazów - 1 (`imgObjs.length - 1`) — uzyskana wartość jest zapisywana w zmiennej `last`. Indeks obrazu poprzedniego w stosunku do bieżącego (zmienna `prev`) wynika z działania `nr - 1`. Jednak to działanie ma sens tylko wtedy, gdy bieżący obraz nie jest pierwszym. Stąd pełna instrukcja ma postać:

```
var prev = nr > 0?nr - 1 : 0;
```

Posobnie jest z indeksem obrazu następnego w stosunku do bieżącego (zmienna `next`). Wartość wynika z działania `nr + 1`, ale tylko wtedy, gdy obraz bieżący nie jest ostatnim. Dlatego instrukcja wykonująca obliczenie ma postać:

```
var next = nr < imgObjs.length - 1?nr + 1:imgObjs.length - 1;
```

Utworzone zmienne służą do stworzenia serii znaczników `` pozwalających na nawigację między poszczególnymi obrazami. Ogólna postać takiego znacznika to:

```
<span onclick='zmieńObraz("numer")'>tekst</span>
```

Kliknięcie każdego z nich będzie powodowało wywołanie funkcji `zmieńObraz` i przekazanie jej odpowiedniego numeru. Treść znaczników jest zapisywana w zmiennej pomocniczej `str`, która na zakończenie jest przypisywana właściwość `innerHTML` warstwy `navDiv`. To powoduje wyświetlenie na ekranie aktualnego panelu nawigacyjnego.

Do kodu strony warto jeszcze dodać style CSS, np. takie jak zaprezentowano na listingu 9.33. Warstwa `imgDiv` otrzymała cechę `text-align` o wartości `center`, co spowoduje wycentrowanie całej jej zawartości. Z kolei wszystkie znaczniki `` znajdujące się na warstwie `navDiv` będą wyświetlane za pomocą pogrubionej czcionki (`font-weight: bold`) o niebieskim kolorze (`color:blue`). Kursor umieszczony nad tekstem każdego z tych znaczników będzie miał natomiast kształt wskaźnika (`color:blue`).

Listing 9.33. *Style CSS dla skryptu 85.*

```
<style type="text/css">
  #imgDiv{
    text-align:center;
  }
  #navDiv span{
    cursor:pointer;
    font-weight:bold;
    color:blue;
  }
</style>
```

Skrypt 86. [C][E][F][O][S]

Wyszukiwanie obrazów po opisie

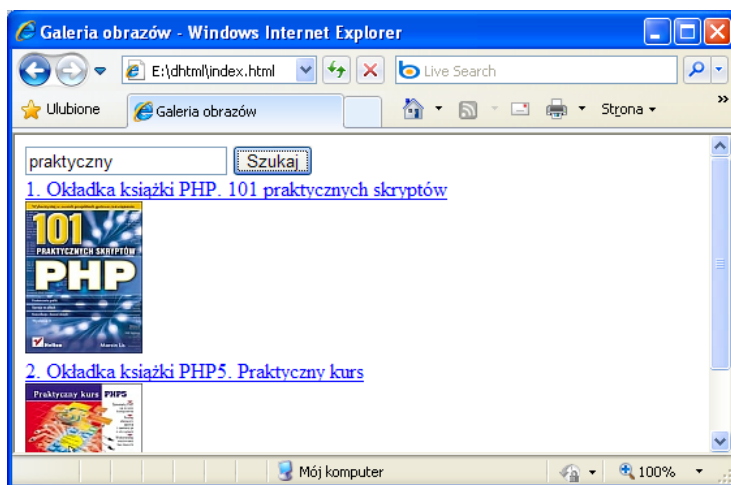
W skrypcie 86. została zrealizowana funkcja wyszukiwania obrazów po opisach. Na stronie zostaną umieszczone pole tekstowe oraz przycisk rozpoczynający procedurę przeszukiwania dowolnej frazy w tekstach opisów przypisanych obrazom. W wynikach wyszukiwania pojawiają się zarówno opisy, jak i miniatury obrazów (o ile zostały zdefiniowane). Strona wynikowa będzie więc wyglądała tak, jak zaprezentowano to na rysunku 9.5. Kliknięcie opisu lub miniatury spowoduje wczytanie do przeglądarki obrazu w pełnej rozdzielczości.

Kod HTML proponowanego rozwiązania takiej funkcjonalności został przedstawiony na listingu 9.34.

Listing 9.34. *Sekcja `<body>` skryptu 86.*

```
<body>
  <div id="interfejsDiv">
    <input type="text" id="tfStr" />
    <input type="button" value="Szukaj" onclick="btnSzukajClick();" />
  </div>
  <div id="wynikDiv">
  </div>
</body>
```


Rysunek 9.5.
*Rezultaty
 wyszukiwania
 obrazów*



W treści sekcji <body> zostały umieszczone dwie warstwy: `interfejsDiv` i `wynikDiv`. Pierwsza zawiera elementy interfejsu, na który składa się pole tekstowe oraz przycisk. Polu został nadany identyfikator `tfStr`, dzięki czemu będzie można się do niego odwoływać w kodzie skryptu, natomiast przycisk otrzymał atrybut `onclick`, dzięki czemu będzie reagował na kliknięcia. Każde kliknięcie będzie powodowało wywołanie funkcji o nazwie `btnSzukajClick`. Na drugiej warstwie będą wyświetlane wyniki wyszukiwania. Początkowo nie zawiera żadnej treści.

Skrypt obsługujący witrynę został przedstawiony na listingu 9.35.

Listing 9.35. *Skrypt wyszukujący obrazy po opisach*

```
<script type="text/javascript">
var imgObjs = new Array(
    {src:"php102_big.jpg", minsrc:"php102_min.jpg",
     desc:"Okładka książki PHP. 101 praktycznych skryptów"},
    {src:"joowww_big.jpg", minsrc:"joowww_min.jpg",
     desc:"Okładka książki Joomla! 1.5. Prosty przepis na własną stronę WWW"},
    {src:"php5pk_big.jpg", minsrc:"php5pk_min.jpg",
     desc:"Okładka książki PHP5. Praktyczny kurs"}
);
function btnSzukajClick()
{
    var tfStr = document.getElementById('tfStr');
    if(!tfStr) return;
    var div = document.getElementById('wynikDiv');
    if(div) div.innerHTML = szukaj(tfStr.value);
}
function szukaj(ciąg)
{
    if(!imgObjs || imgObjs.length < 1 || ciąg === "") return "";
    var str = "";
    var licznik = 1;
    ciąg = ciąg.toLowerCase();
    for(var i = 0; i < imgObjs.length; i++){
        if(imgObjs[i].desc.toLowerCase().indexOf(ciąg) != -1){
```

```

        str += "<a href='" + imgObjs[i].src + "'>";
        str += licznik++ + ". " + imgObjs[i].desc + "<br />";
        if(imgObjs[i].minsrc)
            str += "<img src='" + imgObjs[i].minsrc + "' alt='' /><br />";
        str += "</a>";
    }
}
return str;
}
</script>

```

Na początku kodu znajduje się definicja tablicy przechowującej obiekty z informacjami o obrazach i ich opisach. Konstrukcja tablicy jest bardzo podobna do przedstawionej przy omawianiu skryptu 85. Różnica jest taka, że każdy obiekt otrzymał dodatkowe pole (właściwość) określające nazwę (adres) pliku z miniaturą obrazu. Uwzględniane są zatem trzy właściwości:

- ♦ src — adres pliku z obrazem;
- ♦ minsrc — adres pliku z miniaturą obrazu;
- ♦ desc — opis obrazu.

Jeżeli dana grafika nie ma miniatury, właściwość minsrc powinna zawierać pusty ciąg znaków, np.:

```

{src:"php102_big.jpg", minsrc:"",
 desc:"Okładka książki PHP. 101 praktycznych skryptów"}.

```

Funkcja `btnSzukajClick` jest wykonywana po kliknięciu przycisku *Szukaj*. Najpierw pobiera odwołanie do pola tekstowego `tfStr`. Jeśli się to uda, pobiera odwołanie do warstwy `wynikDiv`. Jeżeli i ta czynność zakończy się sukcesem, właściwości `innerHTML` warstwy przypisywany jest wynik działania funkcji `szukaj`, której w postaci argumentu przekazywana jest wartość znajdująca się w polu tekstowym.

Funkcja `szukaj` otrzymuje argument ciąg określający poszukiwany ciąg znaków. Najpierw sprawdza, czy istnieje tablica `imgObjs`, czy jej długość jest większa niż 1 oraz czy argument ciąg jest niepusty. Jeśli którykolwiek z wymienionych warunków jest fałszywy, zwracany jest pusty ciąg znaków, nie ma bowiem czego szukać (alternatywnie można zwrócić komunikat o błędzie).

Jeżeli jednak dane są prawidłowe, tworzone są zmienne pomocnicze `str` i `licznik`. Pierwsza będzie przechowywała kod HTML z wynikami wyszukiwania, który zostanie zwrócony w wyniku działania funkcji, druga — liczbę odnalezionych obrazów. Tekst zapisany jest też przekształcany na małe litery (dzięki temu wielkość liter nie będzie miała znaczenia). Poszukiwanie ciągu znaków w opisach obrazów odbywa się w pętli `for`, która przebiega całą tablicę `imgObjs`.

To, czy poszukiwany ciąg znaków znajduje się w opisie danego obrazu (w polu `desc`), jest badane przez złożoną instrukcję warunkową:

```

if(imgObjs[i].desc.toLowerCase().indexOf(ciąg) != -1){

```

Wartość pola `desc` obiektu odczytanego z komórki o indeksie `i` jest tu pobierana i poddawana działaniu metody `toLowerCase()`. W wyniku tego działania zostanie zwrócony ciąg, w którym wszystkie litery zostały zamienione na małe. Na rzecz tego wynikowego ciągu jest wywoływana metoda `indexOf`, której w postaci argumentu jest przekazywany parametr `ciąg`. A zatem sprawdzane jest to, czy w opisie występuje ciąg znaków zapisany w `ciąg`. Jeżeli występuje, metoda `indexOf` zwraca wartość różną od `-1` (indeks wystąpienia tego ciągu), co jest sprawdzane za pomocą operatora warunkowego `!=`.

Jeżeli warunek instrukcji warunkowej `if` jest prawdziwy, do zmiennej `str` jest dopisywany kod HTML zawierający kolejny numer obrazu, jego opis, miniaturę (o ile istnieje) — wszystko ujęte w znacznik `<a>` zawierający odniesienie do obrazu w pełnej rozdzielczości. Ogólna struktura tego kodu jest następująca:

```
<a href='adres_pełnego_obrazu'>  
kolejny numer. opis obrazu<br />  
<img src='adres_miniatury' alt='' />  
<br /></a>
```

Adres pełnego obrazu jest pobierany z właściwości `src` (`imgObjs[i].src`), adres miniatury — z właściwości `minsrc` (`imgObjs[i].minsrc`), opis — z właściwości `desc` (`imgObjs[i].desc`), a kolejny numer to wartość zmiennej `licznik`.

Po zakończeniu pętli wartość zmiennej `str` jest zwracana jako wynik działania funkcji za pomocą instrukcji `return`.

Rozdział 10.

Menu

Rozdział 10. poświęcony jest różnym systemom menu. Zawiera je przecież praktycznie każda witryna składająca się z więcej niż jednej strony. Najprostsze menu mogą być skonstruowane ze zwykłych, umieszczonych jeden pod drugim odnośników. W dzisiejszych czasach używa się jednak bardziej zaawansowanych konstrukcji, które po prostu są bardziej atrakcyjne. Niektóre menu korzystają wyłącznie ze stylów CSS, dzięki czemu działają nawet w przeglądarkach, w których wyłączono obsługę skryptów. Najczęściej jednak łączy się techniki CSS i JavaScript, co pozwala na zastosowanie bardziej złożonych efektów.

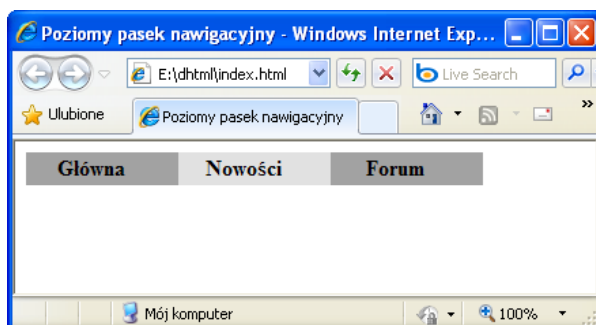
Skrypt 87.

Pasek nawigacyjny w CSS

[C][F][E][O][S]

Utworzenie paska czy też belki nawigacyjnej nie jest trudnym zadaniem. Wystarczy odpowiednie połączenie HTML i CSS. Poszczególne elementy paska będą wskazywały na różne podstrony serwisu, a menu, nad którym znajdzie się kursor myszy, będzie podświetlane (zmieni się jego kolor tła). Przykładowy wygląd paska został zobrazowany na rysunku 10.1. Taka strona jest generowana przez kod HTML przedstawiony na listingu 10.1.

Rysunek 10.1.
*Pasek nawigacyjny
na stronie WWW*

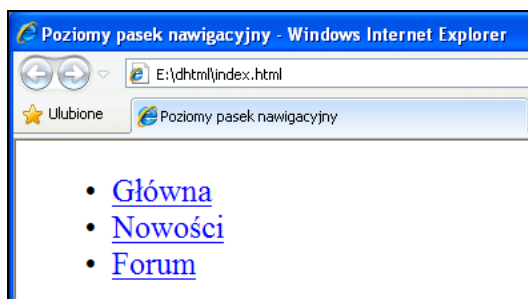


Listing 10.1. Kod HTML paska nawigacyjnego

```
<body>
  <div id="menuDiv">
    <ul id="menu">
      <li><a href="glowna.html">Główna</a></li>
      <li><a href="nowosci.html">Nowości</a></li>
      <li><a href="forum.html">Forum</a></li>
    </ul>
  </div>
</body>
```

W sekcji `<body>` znajduje się warstwa `menuDiv`, a w niej zwykła lista nienumerowana utworzona za pomocą znaczników `` i ``. Każdy element tej listy to zwyczajny odnośnik wygenerowany przez znacznik `<a>` z atrybutem `href` wskazującym daną podstronę serwisu (czy też dowolny inny adres internetowy, nie ma tu żadnych ograniczeń). Oczywiście wczytanie takiego kodu do przeglądarki spowodowałoby pojawienie się na stronie zwykłej listy wypunktowanej zaprezentowanej na rysunku 10.2. Aby zamienić ją na pasek nawigacyjny, trzeba zastosować zbiór reguł CSS. Zostały przedstawione na listingu 10.2.

Rysunek 10.2.
*Widok witryny bez
nadanych reguł CSS*

**Listing 10.2.** Style CSS dla skryptu 87.

```
<style type="text/css">
  #menu{
    font-weight:bold;
    padding:0px;
    margin:0px;
  }
  #menu li{
    background-color:#A0A0A0;
    float:left;
    list-style-type:none;
    padding-right:2ex;
    cursor:pointer;
    width:12ex;
    height:3ex;
  }
  #menu li a{
    text-decoration:none;
    color:black;
    display:block;
    text-align:center;
  }
```

```
margin-top:2px;
}
#menu li:hover{
background-color:#E0E0E0;
}
</style>
```

Pierwsza reguła, z selektorem `#menu`, dotyczy warstwy `divMenu`. Usuwa margines wewnętrzny (`padding` — wypełnienie) i zewnętrzny (`margin`) oraz ustala pogrubienie czcionki (`font-weight:bold`;). Te definicje można dowolnie zmieniać, a także dodawać inne, stosownie do bieżących potrzeb.

Dużo ważniejsza jest druga reguła (z selektorem `#menu li`), która dotyczy elementów `li` zawartych w warstwie `menu`, a więc elementów listy wypunktowanej. Definicja `float:left`; ustala lewy przepływ tych elementów, zatem ustawią się one w jednej poziomej linii, z kolei definicja `list-style-type:none`; usuwa punktory. Dzięki temu uzyskamy poziomą belkę z menu. Pozostałe składniki selektora `#menu li` mają znaczenie czysto wizualne. Ustalana jest szerokość (`width`), wysokość (`height`), kolor tła (`background-color`), prawy margines (`padding-right`) oraz kształt kursora (`cursor`).

Trzeci z selektorów dotyczy odnośników umieszczonych w elementach listy. Usunięte zostało standardowe pokreślenie odsyłaczy (`text-decoration:none`;), kolor zmieniono na czarny (`color:black`;), a sposób wyświetlania został ustalony na blokowy (`display:block`;). Został również wycentrowany tekst (`text-align:center`;), oraz określony górny margines (`margin-top:2px`;).

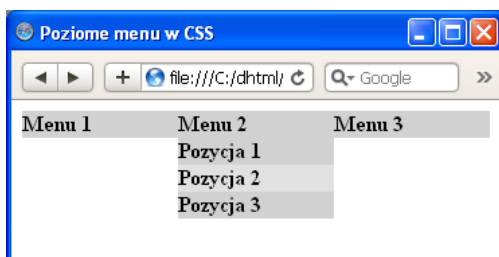
Ostatni selektor (`#menu li:hover`) korzysta z pseudoklasy `hover` i powoduje, że po najechaniu kursorem myszy na dany element będzie on wyróżniany przez zmianę koloru tła (`background-color:#E0E0E0`;). Ten efekt nie będzie dostępny w przeglądarce Internet Explorer w wersji 6.

Skrypt 88. [C][F][E8][O][S] Klasyczne menu poziome z użyciem CSS

Na stronie WWW można zamieścić klasyczne menu poziome, podobne do menu stosowanych w zwykłych aplikacjach — będzie miało postać przedstawioną na rysunku 10.3. Można je wykonać wyłącznie za pomocą HTML i CSS — nie trzeba używać JavaScriptu. Takie właśnie zadanie realizuje skrypt 88. (menu poziome z wykorzystaniem JavaScriptu zostanie pokazane w przykładzie 89.). Zaletą takiego rozwiązania jest prawidłowe działanie efektu nawet przy wyłączonej obsłudze skryptów w przeglądarce. Wadą jest konieczność stosowania zwykłych odnośników. Wybranie pozycji menu będzie powodowało wyłącznie wczytanie podstrony serwisu — nie będzie można wykonać żadnych innych działań, które byłyby możliwe przy użyciu JavaScriptu.

Rysunek 10.3.

Poziome menu
zbudowane
za pomocą CSS



Budowę menu należy zacząć od kodu HTML. Będzie się składało ze zwykłych list nieuporządkowanych. Treść sekcji `<body>` została przedstawiona na listingu 10.3.

Listing 10.3. Treść sekcji `<body>` skryptu 88.

```
<body>
  <div id="menuDiv">
    <ul id="menu">
      <li>Menu 1
        <ul>
          <li><a href="url">Pozycja 1</a></li>
          <li><a href="url">Pozycja 2</a></li>
          <li><a href="url">Pozycja 3</a></li>
        </ul>
      </li>
      <li>Menu 2
        <ul>
          <li><a href="url">Pozycja 1</a></li>
          <li><a href="url">Pozycja 2</a></li>
          <li><a href="url">Pozycja 3</a></li>
        </ul>
      </li>
      <li>Menu 3
        <ul>
          <li><a href="url">Pozycja 1</a></li>
          <li><a href="url">Pozycja 2</a></li>
          <li><a href="url">Pozycja 3</a></li>
        </ul>
      </li>
    </ul>
  </div>
</body>
```

W kodzie strony została umieszczona warstwa `menuDiv`, a w niej utworzona za pomocą znacznika `` lista nienumerowana o identyfikatorze `menu`. Lista składa się z trzech głównym pozycji zdefiniowanych za pomocą znaczników ``: `Menu 1`, `Menu 2` i `Menu 3`. To będą główne pozycje menu wyświetlane na poziomej belce. Wewnątrz każdej głównej pozycji znajduje się kolejna lista nienumerowana zawierająca podpozycje przypisane danemu menu. Każda taka podpozycja (zdefiniowana również za pomocą znacznika ``) zawiera klasyczny odnośnik wskazujący adres wybranej podstrony (ogólniej: dowolny adres internetowy). Na listingu miejsca wystąpienia adresów podstron zostały oznaczone ciągami `url`.

Tak przygotowany kod po wczytaniu do przeglądarki spowoduje pojawianie się typowych list zagnieżdżonych. Witryna przyjmie więc postać przedstawioną na rysunku 10.4. Aby zamienić listy na klasyczne menu poziome, należy użyć stylów CSS zaprezentowanych na listingu 10.4.

Rysunek 10.4.
*Postać menu bez
użycia stylów CSS*



Listing 10.4. *Style CSS dla menu ze skryptu 88.*

```
<style type="text/css">
#menu{
    font-weight:bold;
    padding:0px;
    margin:0px;
}
#menu li{
    background-color:#D0D0D0;
    float:left;
    list-style-type:none;
    padding-right:2ex;
    cursor:pointer;
    width:12ex;
}
#menu li li{
    float:none;
}
#menu li li a{
    text-decoration:none;
    color:black;
}
#menu li ul{
    display:none;
    position:absolute;
    padding:0px;
}
```

```
#menu li:hover ul{
    display:block;
}
#menu li li:hover{
    background-color:#E0E0E0;
}
</style>
```

Pierwszy selektor (`#menu`) dotyczy całej listy z głównymi pozycjami. Usuwane są marginesy oraz ustawiana pogrubiona czcionka. Drugi selektor (`#menu li`) dotyczy elementów `li` zawartych w liście o identyfikatorze `menu`, czyli elementów głównych (Menu 1, Menu 2, Menu 3). Muszą być ustawione obok siebie w poziomie i nie mogą zawierać punktów. Jest to osiągane za pomocą reguł `float:left;` i `list-style-type:none;`. Pozostałe reguły mają znaczenie dekoracyjne. Określany jest kolor tła (`background-color:#D0D0D0;`), wypełnienie z prawej strony (`padding-right:2ex;`), kształt kursora (`cursor:pointer;`) i długość elementu (`width:12ex;`).

Trzeci selektor (`#menu li li`) dotyczy elementów list zagnieżdżonych (elementy `` zawarte w elementach `` w elemencie o identyfikatorze `menu`), czyli pozycji przypisanych danym menu głównym (na listingu 10.3 i rysunkach są to: Pozycja 1, Pozycja 2, Pozycja 3 itd.). Mają być one wyświetlane jedna pod drugą, konieczne jest więc wyłączenie przepływów włączonych w regule z selektorem `#menu li`. Stąd definicja `float:none;`.

Kolejny selektor (`#menu li li a`) określa zachowanie odnośników zawartych w pozycjach menu. Wyłączane jest standardowe podkreślenie tekstów odnośników (`text-decoration:none;`), a kolor tekstu jest definiowany jako czarny (`color:black;`).

Bardzo ważny jest selektor `#menu li ul`. Dotyczy on całych list zagnieżdżonych (zawierających pozycje menu głównych). Standardowo nie powinny być wyświetlane, muszą się pojawiać dopiero wtedy, gdy nad pozycją główną pojawi się kursor myszy. Dlatego występuje tu cecha `display` o wartości `none`. Kiedy jednak są wyświetlane, muszą się znajdować dokładnie pod pozycjami głównymi, do których zostały przypisane. Osiągamy to dzięki pozycjonowaniu bezwzględnemu (`position:absolute;`) oraz usunięciu wypełnień (`padding:0px;`).

Skoro listy wewnętrzne domyślnie nie są wyświetlane, a mają się pojawiać po wskazaniu kursorem myszy pozycji z listy zewnętrznej, konieczne było skonstruowanie selektora korzystającego z pseudoklas `hover`. Ma on postać: `#menu li:hover ul` i zawarta jest w nim reguła `display:block;`. Cała reguła oznacza: gdy kursor myszy znajdzie się nad elementem `li` zawartym w elemencie (liście) o identyfikatorze `menu`, zmień na blokowy sposób wyświetlania zawartego niżej w hierarchii elementu typu `ul` (listy wewnętrznej). Dzięki temu pojawiają się pozycje przypisane menu głównym.

Ostatni selektor (`#menu li li:hover`) umożliwia wyróżnianie aktualnie wskazywanego elementu listy wewnętrznej — zmienia kolor tła na `#E0E0E0` (odcień szarości).

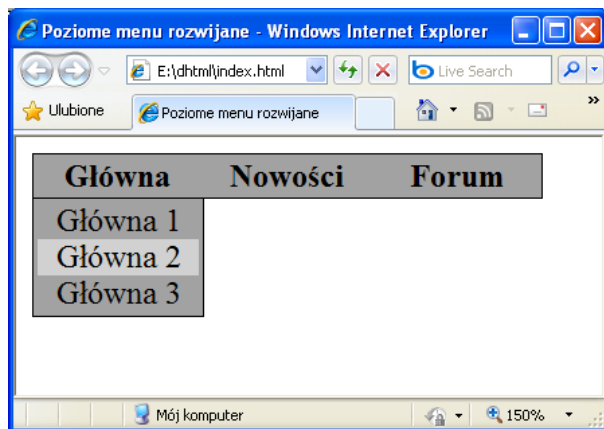
Skrypt 89.

Menu poziome z użyciem JavaScriptu

[C][E][F][O][S]

Skrypt 89. realizuje efekt klasycznego menu poziomego (z pozycjami rozwijanymi w dół) przy użyciu JavaScriptu. Takie podejście pozwala na wykonywanie dowolnych czynności po wybraniu danej pozycji menu, a także w dużej mierze uniezależnia od występujących w przeglądarkach różnic interpretacyjnych dotyczących stylów CSS — wymaga jednak, aby była włączona obsługa skryptów. Kod części HTML będzie miał postać przedstawioną na listingu 10.5. Menu będzie zaś wyglądało tak, jak zaprezentowano to na rysunku 10.5.

Rysunek 10.5.
*Menu obsługiwane
za pomocą JavaScriptu*



Listing 10.5. *Część HTML skryptu 89.*

```
<body>
  <div id="menubar">
    <div id="menubar1" class="menubar"
      onmouseover="showMenuItem('menuitems1', this);">Główna</div>
    <div id="menubar2" class="menubar"
      onmouseover="showMenuItem('menuitems2', this);">Nowości</div>
    <div id="menubar3" class="menubar"
      onmouseover="showMenuItem('menuitems3', this);">Forum</div>
  </div>
  <div id="menu">
    <div id="menuitems1" class="menuitems">
      <span onclick="akcja(this);">Główna 1</span>
      <span onclick="akcja(this);">Główna 2</span>
      <span onclick="akcja(this);">Główna 3</span>
    </div>
    <div id="menuitems2" class="menuitems">
      <span onclick="akcja(this);">Nowości 1</span>
```

```
<span onclick="akcja(this);">Nowości 2</span>
</div>
<div id="menuitems3" class="menuitems">
  <span onclick="akcja(this);">Forum 1</span>
</div>
</div>
</body>
```

Menu składa się z dwóch głównych warstw: menubar i menus. Warstwa menubar to poziomy pasek zawierający pozycje menu głównego — jest stale widoczna na stronie. Poszczególne pozycje są generowane za pomocą kolejnych warstw. Każda z nich ma atrybut `class` określający ich wspólną klasę CSS, a także atrybut `onmouseover`. Dzięki niemu wskazanie kursorem myszy danej pozycji spowoduje wywołanie funkcji `showMenuItem` — otrzyma ona w postaci argumentów nazwę menu, które ma być wyświetlone, oraz wskazanie do bieżącej pozycji. Innymi słowy, po najechaniu kursorem myszy na daną pozycję główną zostanie wywołana funkcja wyświetlająca odpowiadające jej menu podrzędne.

Warstwa menus zawiera zestaw warstw odpowiadających poszczególnym rozwijanym menu. Każda z nich ma nadany unikatowy identyfikator (jest używany przy wywołaniach wspomnianej wyżej funkcji `showMenuItem`), a także atrybut `class` określający wspólną dla wszystkich tych warstw klasę CSS (`menuitems`). Poszczególne pozycje każdego menu są realizowane za pomocą znaczników `` (można też użyć innych elementów, np. warstw czy też odpowiednio ostylowanych¹ wykazów). Każdy z nich ma atrybut `onclick` powodujący, że po kliknięciu zostanie wywołana funkcja `akcja` i zostanie jej przekazany odnośnik do klikniętego elementu. Dzięki temu pozycje menu będą reagowały na kliknięcia.

Aby odpowiednio ułożyć elementy menu na stronie, trzeba zastosować odpowiednie reguły CSS. Zostały przedstawione na listingu 10.6.

Listing 10.6. *Style CSS dla skryptu 89.*

```
<style type="text/css">
#menubar{
  background-color:#A0A0A0;
  border:1px solid black;
  position:absolute;
  z-index:2;
}
.menubar{
  float:left;
  width:10ex;
  font-weight:bold;
  text-align:center;
  padding:2px;
  cursor:pointer;
}
```

¹ „Ostylowanych”, czyli takich, którym przypisano style CSS.

```
.menuitems{
  position:absolute;
  background-color: #A0A0A0;
  border:1px solid black;
  text-align:center;
  width:10ex;
  margin:0px;
  padding:2px;
  display:none;
  z-index:1;
}
.menuitems span{
  display:block;
  cursor:pointer;
}
.menuitems span:hover{
  background-color: #D0D0D0;
}
</style>
```

Pierwsza reguła (selektor `#menubar`) dotyczy warstwy `menubar` realizującej poziomy pasek menu. Ważne są tu pozycjonowanie bezwzględne (`position:absolute`) oraz właściwość `z-index`. Chodzi o to, aby pasek znajdował się nad warstwami z pozycjami menu (warstwami o klasach `menuitems`). Dzięki temu przykryje wystające (w niektórych przeglądarkach) elementy warstw. Dlatego też w tym selektorze cecha `z-index` ma wartość 2, a w klasie `menuitems` — 1. Pozostałe dwie cechy to określenie obramowania (`border:1px solid black;`) oraz koloru tła (`background-color:#A0A0A0;`).

Reguła z selektorem `.menubar` dotyczy głównych pozycji wyświetlanych na pasku menu. Najważniejsze jest tu ustalenie lewego przepływu (`float:left;`), dzięki czemu pozycje ustawią się w poziomie jedna za drugą. Pozostałe cechy mają znaczenie bardziej dekoracyjne. Ustalane są: pogrubienie czcionki (`font-weight:bold;`), centrowanie tekstu (`text-align:center;`) i kształt kursora (`cursor:pointer;`), a także szerokość (długość) warstw (`width:10ex;`) i wypełnienie (`padding:2px;`). Ważne jest, aby te dwie ostatnie cechy były takie same jak w selektorze dotyczącym klasy `menuitems` (da to najlepszy efekt wizualny).

Kolejna reguła (selektor `.menuitems`) dotyczy klasy `menuitems`, czyli boksów (warstw) zawierających pozycje podrzędne przypisane danym pozycjom głównym. Najważniejsze cechy to: `position` o wartości `absolute` (określenie sposobu pozycjonowania, co pozwoli ustalić położenie warstwy za pomocą JavaScriptu), `display` o wartości `none` (wyłączenie wyświetlania warstwy) i `z-index` o wartości 1 (ustalenie kolejności wyświetlania — pod paskiem głównym). Pozostałe cechy dotyczą wyglądu: kolor tła, obramowanie, wyrównywanie tekstu, długość, wypełnienie. Można je zmieniać dowolnie, pamiętając, aby długość i wypełnienie były takie same jak w głównym pozycjach menu.

Reguła z selektorem `.menuitems span` dotyczy poszczególnych pozycji menu, które są realizowane za pomocą znaczników `span`. Ustalany jest kształt kursora (wskaźnik — `pointer`) oraz zmieniany sposób wyświetlania (na blokowy — `block`). Są to jednak czynności czysto kosmetyczne.

Ostatnia reguła (selektor `.menuitems span:hover`) pozwala na wyróżnianie aktywnych (wskazywanych przez kursor) pozycji menu (bloków `span`). Kolor tła jest wtedy zmieniany na jasnoszary (`#D0D0D0`). Za aktywację reguły, gdy kursor myszy znajdzie się nad elementem, odpowiada pseudoklasa `hover` (efekt nie będzie widoczny w Internet Explorerze w wersji 6.).

Na listingu 10.7 została przedstawiona pierwsza część skryptu obsługującego menu.

Listing 10.7. *Pierwsza część skryptu obsługującego menu*

```
<script type="text/javascript">
  var activeMenu = null;
  function showMenuItem(menuId, menuBarItem)
  {
    if(!menuBarItem) return;

    xOff = 0;
    yOff = menuBarItem.offsetHeight/1;

    while(menuBarItem.offsetParent){
      xOff += menuBarItem.offsetLeft/1;
      yOff += menuBarItem.offsetTop/1;
      menuBarItem = menuBarItem.offsetParent;
    }
    closeMenu();
    activeMenu = document.getElementById(menuId);
    if(!activeMenu) return;
    activeMenu.style.top = yOff + "px";
    activeMenu.style.left = xOff + "px";
    activeMenu.style.display = "block";
  }
  //dalsza część skryptu
```

Na początku kodu znajduje się deklaracja zmiennej globalnej `activeMenu`, która będzie przechowywała odniesienie do aktualnie wyświetlanego menu (warstwy z menu), a za nią — definicja funkcji `showMenuItem`. Funkcja przyjmuje dwa argumenty `menuId` i `menuBarItem`. Pierwszy to identyfikator warstwy z menu, która ma być wyświetlona, drugi — odniesienie do głównej pozycji menu, która została kliknięta (czyli tej, pod którą ma się pojawić menu).

Na początku sprawdzane jest, czy argument `menuBarItem` jest niepusty — tylko wtedy są wykonywane dalsze czynności (jeśli jest pusty, wykonywanie kodu jest przerywane). Podstawą jest obliczenie współrzędnych, w których ma się pojawić warstwa z menu (warstwa o identyfikatorze wskazanym przez `menuId`). Odbyna się to na takiej samej zasadzie, jaka została przedstawiona w skrypcie 62. z rozdziału 7. (sumowane są pozycje `left` i `top` wszystkich sąsiadujących elementów). Jedyną różnicą jest ustalenie początkowej pozycji poziomej. Ponieważ menu ma się pojawić dokładnie pod elementem wskazywanym `activeMenu`, a nie przy jego prawym dolnym rogu, początkową wartością `xOff` jest 0 (a nie `menuBarItem.offsetWidth`).

Po dokonaniu sumowań zmienna `xOff` będzie zawierała pozycję poziomą (wartość właściwości `left`), a zmienna `yOff` — pozycję pionową (wartość właściwości `top`). Nastę-

puje wtedy zamknięcie bieżącego menu (wywołanie funkcji `closeMenu()`) oraz przypisanie zmiennej `activeMenu` odwołania do elementu o identyfikatorze określonym przez argument `menuId` (jeśli pobranie odwołania się nie powiedzie, wykonywanie kodu jest kończone za pomocą instrukcji `return`). Zmienna `activeMenu` będzie zatem zawierała odwołanie do warstwy, która ma się pojawić na ekranie. Ustalane jest wtedy położenie warstwy (wartość zmiennej `yPos` jest przypisywana właściwości `top` obiektu `style`, a wartość zmiennej `xPos` — właściwości `left`) oraz jest ona wyświetlana (właściwości `display` jest przypisywana wartość `block`).

Do prawidłowego działania kodu potrzebne są jeszcze dwie funkcje: zamykająca menu i wykonująca żądaną akcję po wskazaniu danej pozycji menu. Obie zostały przedstawione na listingu 10.8.

Listing 10.8. *Dalsza część kodu skryptu 89.*

```
//początek skryptu z listingu 10.7
function closeMenu()
{
    if(activeMenu){
        activeMenu.style.display = "none";
        activeMenu = null;
    }
}
function akcja(obj)
{
    alert("Kliknięto pozycję '" + obj.innerHTML + "'.");
}
document.onclick = closeMenu;
</script>
```

Funkcja `closeMenu` ma zamknąć aktywne menu, czyli to, które jest aktualnie wyświetlane. Menu jest widoczne na ekranie, jeśli zmienna `activeMenu` jest niepusta. Jest to badane za pomocą instrukcji warunkowej `if`. Jeżeli zatem `activeMenu` zawiera odniesienie do warstwy z menu, najpierw zmieniana jest wartość właściwości `display` obiektu `style` — otrzymuje wartość `none`, a następnie `activeMenu` otrzymuje wartość `null` (co oznacza, że żadne menu nie jest aktualnie wyświetlane).

Funkcja `akcja` jest wykonywana po wybraniu dowolnej pozycji z menu. Otrzymuje w postaci argumentu odwołanie do tej pozycji. To, jakie zadania będzie wykonywać, zależy od danego projektu strony. W tym przykładzie wykonywana jest po prostu instrukcja wyświetlająca w nowym oknie dialogowym nazwę danej pozycji (wartość właściwości `innerHTML` obiektu otrzymanego jako argument).

Należy zwrócić uwagę na ostatnią instrukcję kodu. Jest to przypisanie funkcji `closeMenu` właściwości `onclick` obiektu `document`. Oznacza to, że każde kliknięcie w obszarze dokumentu będzie powodowało wywołanie tej funkcji. Tym samym, jeśli aktualnie będzie otwarte menu, zostanie zamknięte. Dzięki temu nie będzie pozostawało na stałe na ekranie.

Skrypt 90.

Wysuwane menu poziome

[C][E][F][O][S]

Menu poziome może być wysuwane z efektem animacji. Kody HTML i CSS można z powodzeniem zapożyczyć z przykładu 89., uzupełniając jedynie regułę CSS dotyczącą klasy `menuitems` (reguła z selektorem `.menuitems`) o definicję cechy `overflow`. Powinna mieć wartość `hidden` (`overflow:hidden;`), tak by nie była wyświetlana zawartość wykraczająca poza bieżące rozmiary warstw z menu. Kod skryptu obsługującego efekt będzie musiał być rozbudowany w stosunku do przykładu 89. Pierwsza część została przedstawiona na listingu 10.9.

Listing 10.9. Początek skryptu tworzącego animację wysuwanego menu

```
<script type="text/javascript">
  var activeMenu = null;
  var timeout = 20;
  var krokY = 3;
  var timerId = null;
  var offsetHeight = 0;

  function showMenuItem(menuId, menuBarItem)
  {
    // tutaj treść funkcji z listingu 10.7

    offsetHeight = activeMenu.offsetHeight;
    activeMenu.style.height = "0px";
    show();
  }
  // dalsza część skryptu
```

Idea rozwijania menu została zapożyczona ze skryptu 60., choć realizacja techniczna jest nieco inna. Występują dwie główne różnice. Po pierwsze, menu rozwijane jest tylko w pionie, nie ma więc potrzeby obsługi kierunku poziomego. Po drugie, zamiast wyliczać automatycznie liczbę kroków animacji, został zastosowany ogranicznik maksymalnej wysokości menu. Liczba kroków wynikać będzie zatem wyłącznie z wysokości menu oraz stałej wartości `krokY` (określającej liczbę pikseli, o jaką ma się zwiększyć wysokość menu w jednym kroku animacji). Takie rozwijanie spowoduje, że wszystkie menu, niezależnie od ich wielkości, będą rozwijane z taką samą szybkością.

Na początku kodu znajdują się definicje niezbędnych zmiennych globalnych. Zmienna `activeMenu` pełni taką samą rolę jak w przypadku skryptu 89., natomiast zmienne `timeout` (czas pomiędzy kolejnymi krokami), `krokY` (wartość, o którą zostanie zwiększona wysokość w każdym kroku) i `timerId` (identyfikator timera), taką samą rolę jak w przypadku skryptu 60. Dodatkowa zmienna `offsetHeight` będzie natomiast przechowywała maksymalną wysokość danego menu w pikselach.

Zadanie funkcji `showMenuItem` jest takie samo jak w przykładzie 89. — ma wyświetlić menu. Identyczna jest też początkowa część kodu. Ponieważ jednak tym razem menu

ma być wyświetlane stopniowo, na końcu dodanych zostało kilka instrukcji. Najpierw wysokość warstwy menu (`activeMenu.offsetHeight`) jest pobierana i zapisywana w zmiennej `offsetHeight`. Następnie właściwości `height` obiektu `style` przypisywany jest ciąg `0px`, co powoduje zmianę stylu i ustalenie wysokości warstwy na 0 pikseli (nie będzie przez to widoczna na ekranie). Wywoływana jest także wykonująca animację funkcja `show`.

Treść tej funkcji wraz z pozostałą częścią skryptu została przedstawiona na listingu 10.10.

Listing 10.10. *Pozostała część kodu skryptu*

```
//początek kodu z listingu 10.9
function show()
{
    if(!activeMenu) return;

    var height = krokY + parseInt(activeMenu.style.height);
    activeMenu.style.height = height + "px";

    if(offsetHeight > activeMenu.offsetHeight){
        timerId = setTimeout("show("+licznik+");", timeout);
    }
}

function closeMenu()
{
    if(timerId){
        clearTimeout(timerId);
        timerId = null;
    }
    if(activeMenu){
        activeMenu.style.height = "";
        activeMenu.style.display = "none";
        activeMenu = null;
    }
}

function akcja(obj)
{
    alert("Kliknięto pozycję '" + obj.innerHTML + "'.");
}

document.onclick = closeMenu;
</script>
```

Funkcja `show` wykonuje animację, czyli stopniowo zwiększa wysokość warstwy zapisanej w globalnej zmiennej `activeMenu`). Najpierw sprawdza, czy zmienna `activeMenu` jest niepusta. Jeśli tak jest, tworzona jest zmienna pomocnicza `height`, w której jest zapisywana wartość wynikająca z dodania do bieżącej wysokości warstwy `activeMenu` wartości zapisanej w `krokY`. Otrzymany wynik jest przypisywany właściwości `height` obiektu `style` warstwy `activeMenu`, co powoduje zwiększenie wysokości menu.

Następnie badane jest, czy wartość zapisana w `offsetHeight`, czyli maksymalna wysokość menu, jest większa niż wartość zapisana w `activeMenu.offsetHeight`, czyli bieżąca wysokość menu. Jeśli tak jest, oznacza to, że menu nie osiągnęło pełnej wysokości

i należy wykonać kolejny krok animacji. Za pomocą metody `setInterval` jest zatem ustawiany timer powodujący kolejne wywołanie funkcji `show` po czasie wskazywanym przez zmienną `timeout`.

Zadaniem funkcji `closeMenu` jest zamknięcie menu oraz wyłączenie timera, o ile jest aktywny. A zatem jeżeli zmienna `timerId` jest niepusta (różna od `null`), jest wywoływana metoda `clearTimeout`, a zmiennej `timerId` jest przypisywana wartość `null`. Podobnie, jeżeli zmienna `activeMenu` jest niepusta (na ekranie jest widoczne menu), wyświetlanie menu jest wyłączane przez przypisanie ciągu `none` właściwości `display`, a zmiennej `activeMenu` jest przypisywana wartość `null`. Ważną operacją jest również usunięcie wartości przypisanej właściwości `height` obiektu `style`:

```
activeMenu.style.height = "";
```

To powoduje, że warstwa z menu odzyska swoje standardowe wymiary (pozwoli to na poprawną animację przy kolejnym wyświetleniu menu).

Ostania z zaprezentowanych funkcji — akcja — wykonuje dokładnie takie samo zadanie, jak było to w przypadku skryptu 89.

Skrypt 91. [C][E][F][O][S] Menu z efektem przenikania

Skrypt 91. realizuje efekt menu poziomego, które pojawia się stopniowo na ekranie po wskazaniu danej pozycji kursorem myszy. Jest to osiągane przez odpowiednią manipulację przezroczystością warstwy z menu. Cały kod to połączenie pomysłów i rozwiązań technicznych ze skryptów 89., 90. i 61. Kody HTML i CSS będą takie same jak w przypadku skryptów 89. i 90. (listingi 10.5 i 10.6). Skrypt JavaScript realizujący funkcje menu będzie pochodził z przykładu 89. (listingi 10.7 i 10.8). Funkcje przenikania zostaną zrealizowane na bazie kodu z listingów 7.7 i 7.8.

Ostateczne procedury JavaScriptu będą miały postać przedstawioną na listingu 10.11.

Listing 10.11. *Skrypt realizujący funkcję przenikania menu*

```
<script type="text/javascript">
  var activeMenu = null;

  var timeout = 30;
  var krok = 0.05;
  var bieżącyKrok = 0;
  var timerId = null;

  function showMenuItem(menuId, menuBarItem)
  {
    //początek funkcji z listingu 10.7

    bieżącyKrok = 0;
    activeMenu.style.opacity = "0.0";
```

```
        activeMenu.style.filter = "alpha(opacity=0)";
        show();
    }
    function show()
    {
        if(!activeMenu) return;
        bieżącyKrok += krok;
        if(bieżącyKrok > 1) bieżącyKrok = 1;

        activeMenu.style.opacity = bieżącyKrok.toFixed(2);
        activeMenu.style.filter = "alpha(opacity:" +
            (bieżącyKrok * 100).toFixed(0) + ")";
        if(bieżącyKrok != 0 && bieżącyKrok != 1)
            timerId = setTimeout("show();", timeout);
        else timerId = null;
    }

    //dalsza część skryptu z listingu 10.8
</script>
```

Zmienna globalna `activeMenu` pełni taką samą rolę jak w przypadku skryptów 88. i 90., a zmienne `timeout` (czas pomiędzy kolejnymi fazami animacji), `krok` (stopień zmiany przezroczystości w każdym kroku animacji), `bieżącyKrok` (bieżąca wartość atrybutu określającego przezroczystość warstwy), `timerId` (identyfikator timera obsługującego animację) — taką samą rolę jak w skrypcie 61.

Początek funkcji `showMenuItem` jest taki sam jak w skrypcie 89. Na końcu pojawiły się jednak instrukcje pozwalające na wywołanie efektu przenikania. Ustalana jest początkowa wartość zmiennej `bieżącyKrok` — 0 oznacza pełną przezroczystość warstwy. Kolejne dwie instrukcje modyfikują początkowy styl warstwy. Zmieniane są właściwości `opacity` (dla przeglądarek innych niż Internet Explorer) i `filter` (dla przeglądarki Internet Explorer) obiektu `style`. Na końcu kodu wywoływana jest funkcja `show` rozpoczynająca animację.

Funkcja `show` wykonuje operacje na warstwie wskazywanej przez `actionMenu`, najpierw sprawdza więc, czy ta zmienna jest niepusta. Następnie zmienia stan zmiennej `bieżącyKrok`, dodając do niej wartość zapisaną w `krok`. Po wykonaniu tej operacji niezbędne jest stwierdzenie, czy nie nastąpiło przekroczenie zakresu, czyli czy `bieżącyKrok` jest nie większa niż jeden. Jeżeli jest większa od 1, następuje odpowiednia korekta. Po dokonaniu weryfikacji są modyfikowane właściwości obiektu `style`.

Właściwości `opacity` przypisywana jest wartość zapisana w `bieżącyKrok`. Aby uniknąć przypisania nieprawidłowych danych wynikających z niedokładności reprezentacji liczb rzeczywistych, za pomocą metody `toFixed` wartość jest zaokrąglana do dwóch miejsc po przecinku. Podobna sytuacja ma miejsce w przypadku właściwości `filter`. Tu dodatkowo wartość zapisana w `bieżącyKrok` jest mnożona przez 100 (w Internet Explorerze całkowita przezroczystość to 0, a brak przezroczystości to 100). Ponieważ uzyskana wartość ma być całkowita, używane jest wywołanie `toFixed(0)`.

Po dokonaniu przypisań badane jest, czy proces zmiany przezroczystości ma być kontynuowany. Będzie tak wtedy, gdy `bieżącyKrok` jest różne od 1. W takiej sytuacji

wykonywane jest ponowne wywołanie metody `setTimeout`. W przeciwnym przypadku zmiennej `timerId` (przechowującej identyfikator timera) jest przypisywana wartość `null`.

Dalsza część skryptu, czyli funkcje `closeMenu` i akcja oraz instrukcja zmieniająca właściwość `onclick` obiektu `document`, są takie same jak w przykładzie 89. Z funkcji `closeMenu` należy jedynie usunąć wiersz modyfikujący właściwość `height` (instrukcja: `activeMenu.style.height = ""`).

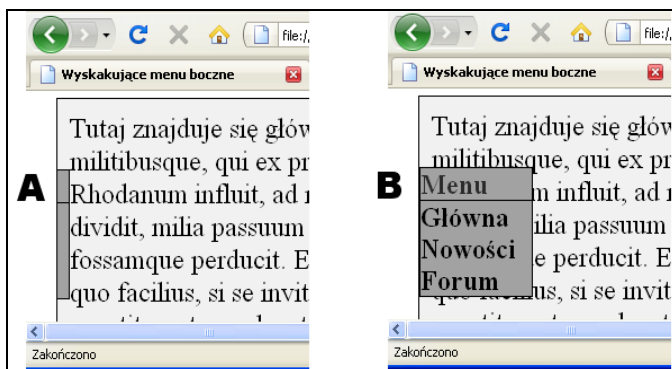
Skrypt 92. Menu wysuwane z boku

[C][E][F][O][S]

Menu może być ukryte przy boku wybranego elementu witryny (bądź okna przeglądarki). Wtedy stale dostępna jest jedynie niewielka część wskazująca obszar aktywny, który nie przesłania właściwej treści strony (zostało to zilustrowane na rysunku 10.6A). Pełne menu pojawia się po wskazaniu aktywnego obszaru kursorem myszy (strona przyjmuje wtedy widok zaprezentowany na rysunku 10.6B) i można wtedy wybrać dowolną z jego pozycji. W taki właśnie sposób działa skrypt 92. Część HTML została przedstawiona na listingu 10.12.

Rysunek 10.6.

Ukryte i odkryte menu boczne



Listing 10.12. Część HTML skryptu 92.

```
<body>
  <div id="mainDiv">
    Tutaj znajduje się główna treść.
    <div id="menu" onmouseover="showhide(this, 'show');"
      onmouseout="showhide(this, 'hide');">
      <div id="menuTitle">Menu</div>
      <div class="menuitem" onclick="menuclicked(this);">Główna</div>
      <div class="menuitem" onclick="menuclicked(this);">Nowości</div>
      <div class="menuitem" onclick="menuclicked(this);">Forum</div>
    </div>
  </div>
</body>
```

W sekcji <body> została umieszczona warstwa mainDiv, która może zawierać dowolną treść. Przy jej końcu zostały zdefiniowane kolejne warstwy tworzące menu. Dzięki temu menu będzie pozycjonowane względem warstwy mainDiv. Menu składa się z warstwy głównej o identyfikatorze menu oraz warstw wewnętrznych odpowiadających tytułowi i poszczególnym pozycjom. Pierwsza warstwa wewnętrzna ma identyfikator menuTitle i zawiera tytuł menu, a pozostałe (zawierające atrybut class o wartości menuItemem) określają pozycje umożliwiające wykonanie dowolnie zdefiniowanych akcji (np. przeniesienie na kolejne podstrony serwisu).

Warstwa menu ma przypisane atrybuty onmouseover i onmouseout, będzie więc reagowała na ruchy myszy. Gdy kursor znajdzie się nad warstwą, zostanie wywołana funkcja showhide z drugim argumentem równym show — co oznacza żądanie wyświetlenia menu. Gdy kursor opuści obszar warstwy, zostanie wywołana ta sama funkcja, ale z drugim argumentem równym hide — co oznacza żądanie schowania menu. Każda z pozycji (warstwy z klasą menuItemem) ma natomiast przypisany atrybut onclick, który powoduje, że po kliknięciu danej pozycji zostanie wywołana funkcja menuclicked i zostanie jej przekazany argument wskazujący bieżącą pozycję (warstwę — this).

Do kodu HTML należy dodać style CSS, które spowodują odpowiednie ułożenie warstw. Style zostały przedstawione na listingu 10.13. Uwzględniono tylko reguły istotne dla prawidłowego działania efektu. Kwestie czysto dekoracyjne zostały pomięte, gdyż można je dobierać dowolnie.

Listing 10.13. *Style CSS dla skryptu 92.*

```
<style type="text/css">
  #mainDiv{
    overflow:hidden;
    position:relative;
    /* dalsze definicje */
  }
  #menu{
    position:absolute;
    width:9ex;
    left:-60px;
    top:40px;
    /* dalsze definicje */
  }
  #menuTitle{
    border-bottom: 1px solid black;
  }
  .menuItemem{
    cursor:pointer;
  }
  .menuItemem:hover{
    background-color:#E0E0E0;
  }
</style>
```

Dla warstwy mainDiv zostały zdefiniowane cechy: overflow o wartości hidden oraz position o wartości relative. Dzięki pierwszej fragment menu znajdujący się poza

obszarem warstwy nie będzie widoczny. Dzięki drugiej menu będzie pozycjonowane względem warstwy, a nie elementów nadrzędnych (można również użyć pozycjonowania bezwzględnego).

Dla warstwy menu ważne jest pozycjonowanie bezwzględne (`position: absolute;`) — pozycja będzie ustalana względem elementu nadrzędnego, czyli warstwy `mainDiv`², a także ustalenie ujemnej lewej pozycji (`left: -60px;`). Dzięki temu będzie przesunięta poza obręb warstwy `mainDiv`. Wielkość przesunięcia jest związana z szerokością warstwy określoną przez cechę `width` (`width: 9ex;`) i należy ją dobrać do konkretnych warunków. Ustalono także położenie pionowe (`top: 40px;`).

Warstwa `menuTitle` otrzymała obramowanie dolne (`border-bottom: 1px solid black;`), dzięki temu tytuł menu będzie oddzielony poziomą linią od pozycji menu. Pozycje menu (klasa `menuItem`) otrzymały kursor w postaci wskaźnika (`cursor: pointer;`). Każdy element klasy `menuItem` będzie też wyróżniany po najechaniu na niego kursorem myszy (selektor `.menuItem: hover`). Zmieni się wtedy kolor tła (`background-color: #E0E0E0;`) — ten efekt będzie niedostępny w przeglądarce Internet Explorer 6.

Ostatnim elementem przykładu jest skrypt obsługujący menu. Został przedstawiony na listingu 10.14.

Listing 10.14. *Skrypt obsługujący wyskakujące menu boczne*

```
<script type="text/javascript">
  var posActive = 0;
  var posInactive = -60;
  function showhide(menu, action)
  {
    if(!menu) return;
    if(action == 'show')
      menu.style.left = posActive + "px";
    else
      menu.style.left = posInactive + "px";
  }
  function menuclicked(menu)
  {
    alert("Kliknięto pozycję '" + menu.innerHTML + "'.");
  }
</script>
```

Na początku znajdują się dwie zmienne globalne określające pozycję menu w stanie zwiniętym (`posInactive`) i rozwiniętym (`posActive`). Zmienna `posActive` powinna mieć taką samą wartość (w pikselach) jak wartość atrybutu `left` w regule CSS z selektorem `#menu` (w tym przypadku `-60`), natomiast `posInactive` równe 0 oznacza, że rozwinięte menu będzie się znajdowało tuż przy lewym brzegu warstwy `mainDiv` (warstwy, względem której jest pozycjonowane).

² Chyba że warstwa `mainDiv` będzie miała pozycjonowanie statyczne. Wtedy pozycja będzie ustalana względem najbliższego elementu nadrzędnego o pozycjonowaniu innym niż statyczne. Nie dotyczy to jednak omawianego przykładu.

Za obsługę wyświetlania menu odpowiada funkcja `showhide`. Przyjmuje dwa argumenty. Pierwszy — `menu` — to wskazanie na obiekt menu (inaczej: obiekt menu), drugi — `action` — określa, jak akcja ma być wykonana. Przyjęto założenie, że menu będzie wyświetlane, gdy `action` będzie równe `show`, a chowane — w każdym innym przypadku.

Pokazanie pełnego menu odbywa się przez przypisanie wartości zapisanej w zmiennej `posActive` uzupełnionej o ciąg `px` właściwości `left` obiektu `style`. Jest to więc przesunięcie warstwy z menu w prawo. Schowanie menu to przypisanie wartości zapisanej w `posInactive` tej samej właściwości, czyli zwyczajne przesunięcie warstwy w lewo.

Funkcja `menuclicked` jest wykonywana po kliknięciu danej pozycji menu. Otrzymuje w postaci argumentu wskazanie klikniętego menu, co pozwala na rozpoznanie danej pozycji. Kod funkcji należy uzupełnić wedle konkretnych potrzeb. W omawianym przypadku jedynym działaniem jest wyświetlenie nazwy klikniętej pozycji (a dokładniej wewnętrznej treści warstwy tworzącej pozycję).

Skrypt 93. Przełączane menu z niezależnymi pozycjami

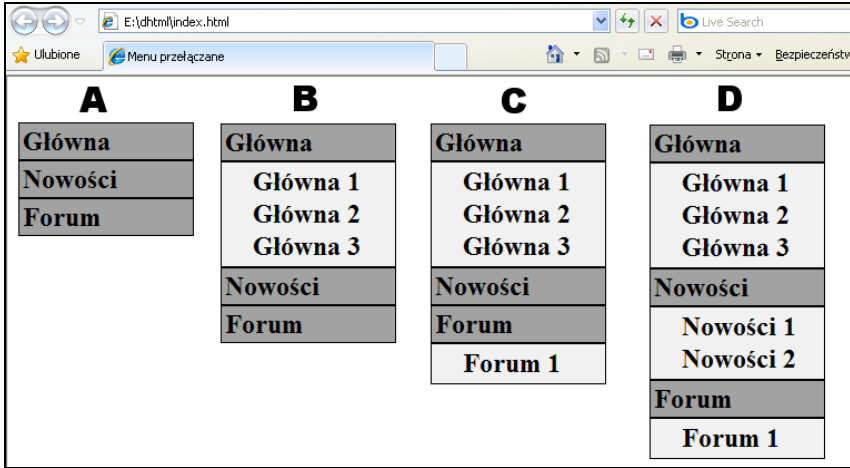
[C][E][F][O][S]

Skrypt 93. realizuje efekt przełączanego menu, czyli takiego, w którym pierwotnie widoczne są tylko nagłówki głównych pozycji. Dana pozycja jest natomiast rozwijana bądź zwijana po kliknięciu przez użytkownika. Tak więc każde menu może być w stanie widocznym lub schowanym. W tym przykładzie będą trzy główne pozycje niezależne od siebie. To znaczy menu będzie mogło być całkowicie schowane (rysunek 10.7A), rozwinięta będzie mogła być tylko jedna pozycja (rysunek 10.7B), dowolne dwie (rysunek 10.7C) czy też wszystkie naraz (rysunek 10.7D).

Kod HTML został przedstawiony na listingu 10.15.

Listing 10.15. Kod HTML skryptu 93.

```
<body>
  <div id="menuDiv">
    <div id="menu1" class="mainmenus" onclick="showhideMenu('menuitems1');">
      Główna
    </div>
    <div id="menuitems1" class="menuitems">
      <div onclick="menuclicked(this);">Główna 1</div>
      <div onclick="menuclicked(this);">Główna 2</div>
      <div onclick="menuclicked(this);">Główna 3</div>
    </div>
    <div id="menu2" class="mainmenus" onclick="showhideMenu('menuitems2');">
      Nowości
    </div>
    <div id="menuitems2" class="menuitems">
      <div onclick="menuclicked(this);">Nowości 1</div>
```



Rysunek 10.7. Różne stopnie rozwinięcia menu

```

        <div onclick="menuclicked(this);">Nowości 2</div>
    </div>
    <div id="menu3" class="mainmenu" onclick="showhideMenu('menuitems3');">
        Forum
    </div>
    <div id="menuitems3" class="menuitems">
        <div onclick="menuclicked(this);">Forum 1</div>
    </div>
</div>
</body>

```

W sekcji `<body>` została umieszczona warstwa `menuDiv`, w której znalazły się wszystkie elementy menu. Składa się ono z występujących kolejno po sobie warstw nagłówkowych (tych, które będą stale widoczne na ekranie) i warstw zawierających pozycje przypisane warstwom głównym (tych, które będą się pojawiać po kliknięciu danego nagłówka). Warstwom nagłówkowym przypisano klasę CSS `mainmenu`, a warstwom z pozycjami — klasę `menuitems`. Każdy z tych elementów ma także unikatowy identyfikator. Wszystkie elementy są ułożone jeden pod drugim.

Warstwy wyświetlane po kliknięciu nagłówków zawierają konkretne pozycje menu. Te pozycje zostały utworzone również za pomocą warstw, z których każda ma przypisany atrybut `onclick` powodujący wywołanie funkcji `menuclicked`. Do funkcji w postaci argumentu jest przekazywane wskazanie do klikniętej pozycji (`this`). Warstwy nagłówkowe również muszą reagować na kliknięcia, dlatego też każdej przypisano atrybut `onclick` zawierający wywołanie funkcji `showhideMenu`. Funkcja ta otrzymuje w postaci argumentu nazwę (identyfikator) menu, które ma się pojawić na ekranie w wyniku kliknięcia.

Do takiego kodu najlepiej dodać style CSS określające wygląd poszczególnych menu. Nie jest to jednak istotne dla omawianego efektu, dlatego też nie będą prezentowane. Warto jedynie zadbać, aby warstwy z klasami `mainmenu` i `menuitems` miały taką samą szerokość.

Na listingu 10.16 został przedstawiony kod JavaScript powodujący włączanie i wyłączanie poszczególnych menu.

Listing 10.16. Kod JavaScript obsługujący skrypt 93.

```
<script type="text/javascript">
  function showhideMenu(id)
  {
    var menu = document.getElementById(id);
    if(!menu) return;
    if(menu.active){
      menu.style.display = "none";
      menu.active = false;
    }
    else{
      menu.style.display = "block";
      menu.active = true;
    }
  }
  function menuclicked(menu)
  {
    alert("Kliknięto pozycję '" + menu.innerHTML + "'");
  }
</script>
```

Funkcja `showhideMenu` otrzymuje w postaci argumentu identyfikator menu, które ma być schowane lub pokazane na ekranie. Najpierw za pomocą metody `getElementById` pobiera element strony o tym identyfikatorze i zapisuje go w pomocniczej zmiennej `menu`. Jeśli po wykonaniu tej operacji zmienna będzie pusta (dokładniej: będzie zawierała wartość `undefined`), będzie to oznaczało, że nie ma takiego elementu strony. Wtedy działanie kodu jest kończone.

Jeśli jednak menu (warstwa z menu) istnieje, trzeba stwierdzić, w jakim jest stanie (aktywnym czy nieaktywnym). W tym celu badany jest stan właściwości `active` danej warstwy. Jeśli jest ona równa `true` (`if(menu.active){}`), oznacza to, że menu jest wyświetlane i trzeba je zamknąć. Zatem właściwość `display` obiektu `style` otrzymuje wartość `none`, a właściwość `active` — wartość `false`. W przeciwnym przypadku (`active` równa `false`) menu nie jest wyświetlane, trzeba je więc pokazać na ekranie (`else{}`). Wtedy właściwość `display` obiektu `style` przypisywana jest wartość `block` (blokowy sposób wyświetlania), a właściwość `active` — wartość `true`.

Oczywiście typowa warstwa z menu standardowo nie zawiera właściwości o nazwie `active`. To jednak nie jest przeszkodą. Przy pierwszym wywołaniu funkcji `showhideMenu` z identyfikatorem danej warstwy na pewno nie jest ona (warstwa) wyświetlana (początkowo wszystkie menu są „zwinięte”), nie ma też właściwości `active`. To oznacza, że warunek `if(menu.active)` będzie fałszywy (wyrażenie `menu.active` będzie miało wartość `undefined`, która zostanie przekształcona na `false`) i zostanie wykonany blok `else`. Tym samym właściwość `active` danej warstwy zostanie utworzona i zostanie jej przypisana wartość `true`. A zatem przy każdym kolejnym wywołaniu właściwość ta będzie już obecna i będzie zawierała prawidłową wartość.

Funkcja `menuclicked` ma takie samo znaczenie jak w przypadku skryptu 92.

Skrypt 94. [C][E][F][O][S]

Menu przełączane wykluczające

W skrypcie 93. zostało przedstawione menu typu przełączanego, w którym każda z pozycji głównych mogła być w dwóch stanach: zwiniętym i rozwinętym. Przy tym wszystkie pozycje główne były niezależne od siebie. Odmianą tego typu menu jest menu przełączane wykluczające, w którym w jednym momencie może być rozwinięta tylko jedna pozycja główna. Taki właśnie efekt został zrealizowany w skrypcie 94. Kod HTML może pozostać prawie taki sam jak w przykładzie 93. (listing 10.15), a style CSS można dopasować dowolnie. Nieco inaczej będzie jedynie wyglądało wywołanie funkcji `showhideMenu`, bowiem będzie przyjmowała dwa argumenty. Pierwszy będzie określał główną warstwę zawierającą wszystkie menu, a drugi — menu, które ma zostać rozwinięte (wyświetlone). W związku z tym definicje głównych pozycji będą miały teraz następującą postać:

```
<div id="menu1" class="mainmenu"
      onclick="showhideMenu('menuDiv', 'menuitems1');">
```

Funkcja `showHideMenu` przyjmie zaś postać przedstawioną na listingu 10.17.

Listing 10.17. Treść funkcji `showhideMenu`

```
<script type="text/javascript">
  function showhideMenu(menuDiv, id)
  {
    var menu = document.getElementById(id);
    var menuDiv = document.getElementById(menuDiv);
    if(!menu || !menuDiv) return;
    var divs = menuDiv.getElementsByTagName('div');
    for(var i = 0; i < divs.length; i++){
      if(divs[i].className == 'menuitems' && divs[i] != menu){
        divs[i].style.display = "none";
        divs[i].active = false;
      }
    }
    if(menu.active){
      menu.style.display = "none";
      menu.active = false;
    }
    else{
      menu.style.display = "block";
      menu.active = true;
    }
  }
</script>
```

Argument `menuDiv` wskazuje główną warstwę zawierającą wszystkie składowe menu, a `id` — identyfikator warstwy do wyświetlenia. Kod funkcji rozpoczyna się od pobrania odwołań do tych elementów witryny, zapisania ich w zmiennych `menuDiv` i `menu`

oraz sprawdzenia, czy te operacje zakończyły się sukcesem. Jeżeli uzyskanie odwołań nie powiedzie się, wykonywanie kodu jest kończone za pomocą instrukcji `return`.

Kolejny krok to uzyskanie odwołań do wszystkich warstw zapisanych na warstwie wskazywanej przez `menuDiv`. Ta czynność jest wykonywana za pomocą metody `getElementsByTagName`:

```
var divs = menuDiv.getElementsByTagName('div');
```

Po wykonaniu powyższej instrukcji zmienna `divs` będzie zawierała tablicę z poszukiwanymi warstwami. Ponieważ chcemy uzyskać menu wykluczające, najpierw trzeba przejrzeć wszystkie warstwy z pozycjami menu. Jeśli któraś jest włączona, należy ją wyłączyć, a następnie włączyć bądź wyłączyć (w zależności od jej bieżącego stanu) warstwę wskazywaną przez zmienną `menu`.

W pętli `for` przeglądane są zatem elementy warstwy `menuDiv`. Jeżeli dana warstwa zawiera pozycje menu, czyli należy do klasy `menuitems` (`divs[i].className == 'menu-items'`) i (&&) jednocześnie nie jest to warstwa bieżąca, czyli ta zapisana w menu (`divs[i] != menu`), wyłączane jest jej wyświetlanie:

```
divs[i].style.display = "none";
```

a właściwości `active` jest przypisywana wartość `false` oznaczająca, że warstwa nie jest wyświetlana:

```
divs[i].active = false;
```

Takie postępowanie oznacza, że po zakończeniu pętli zostaną wyłączone wszystkie warstwy zawierające pozycje menu poza warstwą bieżącą, która nie zmieni swojego stanu.

Dalsze działania są więc identyczne jak w przykładzie 93., tzn. zmieniany jest stan warstwy bieżącej (warstwy wskazywanej przez zmienną `menu`). Jeżeli była nieaktywna (właściwość `active` równa `false` lub brak właściwości `active`), stanie się aktywna (pojawi się na ekranie), a jeśli była aktywna, stanie się nieaktywna (przestanie być wyświetlana). W ten sposób zostało uzyskane przełączane menu o pozycjach wzajemnie wykluczających.

Skrypt 95. [C][E][F][O][S] Przełączane menu z animacją

Przełączane menu, takie jak przedstawiono w skrypcie 93., można uzupełnić o efekt animacji. Kliknięcie danej pozycji głównej będzie wtedy powodowało stopniowe rozwinięcie lub zwinięcie przypisanych jej pozycji. Przy tym animacja każdego menu powinna być niezależna od wszystkich pozostałych. Takie właśnie zadanie jest realizowane przez skrypt 95. Kod HTML pozostanie taki sam jak w skrypcie 93. (listing 10.15). Niezbędna będzie natomiast wymiana skryptu. Jego pierwsza część została przedstawiona na listingu 10.18.

Listing 10.18. Początek kodu skryptu 95.

```
<script type="text/javascript">
  var timeout = 15;
  function showhideMenu(id)
  {
    var menu = document.getElementById(id);
    if(!menu) return;
    if(!menu.maxHeight){
      menu.style.display = 'block';
      menu.maxHeight = menu.offsetHeight;
      menu.style.height = "0px";
    }
    if(menu.active){
      menu.active = false;
      menu.step = -3;
    }
    else{
      menu.style.display = "block";
      menu.active = true;
      menu.step = 3;
    }
    if(menu.timerId){
      clearInterval(menu.timerId);
    }
    menu.timerId = setInterval("move('"+id+"')", timeout);
  }
  //dalsza część skryptu
```

Znajdująca się na początku kodu globalna zmienna `timeout` określa w milisekundach szybkość animacji — czas między kolejnymi jej fazami (krokami). Funkcja `showhideMenu` przyjmuje jeden argument — `id` — zawierający identyfikator warstwy z menu, które ma się pojawić lub zniknąć (rozwinąć lub zwinąć). Odwołanie do tego elementu strony jest pobierane za pomocą metody `getElementById` i zapisywane w zmiennej `menu`.

Następnie badane jest to, czy w obiekcie warstwy istnieje właściwość `maxHeight` (`if(!menu.maxHeight){}`). Ma ona zawierać maksymalną wysokość, jaką może osiągnąć warstwa. Będzie to potrzebne do realizacji efektu animacji. Standardowo takiej właściwości nie ma, przy pierwszym odwołaniu trzeba więc ją utworzyć. Odbywa się to przez chwilowe włączenie wyświetlania menu:

```
menu.style.display = 'block';
```

utworzenie pola `maxHeight` przez przypisanie mu wartości właściwości `offsetHeight` (wysokość warstwy w pikselach):

```
menu.maxHeight = menu.offsetHeight;
```

oraz natychmiastowe zmniejszenie wysokości warstwy do zera (bowiem jeśli nie ma właściwości `maxHeight`, oznacza to, że menu jest w stanie nieaktywnym):

```
menu.style.height = "0px";
```

Po określeniu wartości `maxHeight`, o ile było to konieczne, następuje stwierdzenie, czy menu ma być rozwijane, czy też zwijane. Odbywa się to na takiej samej zasadzie jak

w przypadku skryptu 93. Jeżeli istnieje właściwość `menu.active` i ma wartość `true` (`if(menu.active){}`), oznacza to, że menu aktualnie jest wyświetlane i ma zostać zwinięte. Wtedy właściwość `menu.active` otrzymuje wartość `false`, a właściwość `menu.step` — wartość ujemną (w tym przypadku `-3`). Pole `menu.step` będzie bowiem określało liczbę pikseli, o jaką ma się zmienić wysokość warstwy (wartość ujemna oznacza zatem zwijanie menu).

Jeżeli jednak właściwość `menu.active` ma wartość `false` bądź nie jest obecna, oznacza to, że menu nie jest wyświetlane i ma zostać rozwinięte. Wtedy właściwość `display` obiektu `style` warstwy z menu otrzymuje wartość `block`, właściwość `menu.active` — wartość `true`, a właściwość `menu.step` wartość dodatnią (w tym przypadku `3` — dodatnia wartość `menu.step` oznacza rozwijanie menu).

Kolejna instrukcja to sprawdzenie, czy w warstwie istnieje niestandardowa właściwość `timerId` i czy jest różna od `null`. Jeśli tak jest, oznacza to, że aktualnie jest aktywny timer i w związku z tym trwa proces animacji (proces zwijania lub rozwijania menu). Trzeba go więc zatrzymać. Jest to wykonywane przez wywołanie metody `clearInterval`.

Ostatnia instrukcja to ustawienie timera wywołującego funkcję `move` w interwałach określonych przez zmienną `timeout`. Identyfikator timera jest zapisywany we właściwości `timerId` warstwy (jeśli jeszcze takiej właściwości nie ma, zostanie w tym miejscu utworzona). Oznacza to, że obiekt warstwy przechowuje identyfikator obsługującego ją timera. Funkcja `move` zajmuje się animacją warstwy (menu) o identyfikatorze przekazanym w postaci argumentu. Jej treść została przedstawiona na listingu 10.19.

Listing 10.19. Treść funkcji *move*

```
function move(id)
{
    var menu = document.getElementById(id);
    if(!menu) return;
    var height = menu.step + parseInt(menu.style.height);
    if(height <= 0){
        height = 0;
        menu.style.display = "none";
    }
    if(height >= menu.maxHeight) height = menu.maxHeight;
    menu.style.height = height + "px";
    if(height == 0 || height == menu.maxHeight){
        clearInterval(menu.timerId);
        menu.timerId = null;
    }
}
```

Otrzymany przez funkcję argument `id` wskazuje warstwę podlegającą animacji. Odwołanie do warstwy jest zatem pobierane i zapisywane w zmiennej `menu`. Dalsze czynności są wykonywane tylko wtedy, gdy ta operacja zakończyła się sukcesem (wartość zmiennej `menu` jest różna od `undefined` — nie jest pusta). Funkcja nie rozróżnia tego, czy wykonywane jest rozwijanie czy zwijanie warstwy — nie ma takiej potrzeby. Po prostu odczytuje bieżącą wysokość warstwy (`menu.style.height`), dodaje ją do wartości właściwości `menu.step`, a wynik zapisuje w zmiennej pomocniczej `height`. Tym

samym wysokość warstwy będzie zwiększana, gdy `menu.step` jest dodatnie (menu jest wtedy rozwijane), a zmniejszana, gdy `menu.step` jest ujemne (menu jest wtedy zwijane). Wartość `menu.step` została z kolei ustalona z funkcji `showhideMenu`.

Po ustaleniu wartości zmiennej `height` trzeba sprawdzić, czy nie przekracza dopuszczalnych granic. Przekroczenie nastąpi, gdy wartość będzie mniejsza od 0 lub większa od `maxHeight`. Jeżeli zatem `height` jest mniejsza od zera, oznacza to koniec zwijania warstwy. Jest wtedy wyłączane wyświetlanie (`menu.style.display = "none";`), a `height` otrzymuje wartość 0. Jeśli `height` jest większa od `maxHeight`, oznacza to, że warstwa (menu) została w pełni rozwinięta, wtedy `height` otrzymuje wartość zapisaną w `maxHeight`.

Po dokonaniu opisanych korekt wartość zapisana w `height` jest uzupełniana o ciąg `px` i przypisywana właściwości `height` obiektu `style` warstwy `menu`. Następnie badane jest to, czy powinna się zakończyć animacja. Będzie tak wtedy, gdy `height` jest równe 0 lub równe `maxHeight`. W takiej sytuacji wyłączany jest timer o identyfikatorze zapisanym w `menu.timerId`, a właściwości `menu.timerId` jest przypisywana wartość `null`. Jest to sygnałem dla skryptu, że timer nie jest aktywny.

Skrypt 96. [C][E][F][O][S]

Przesuwany boks menu (menu ustawiane przez użytkownika)

Menu zazwyczaj znajduje się w stałym miejscu wyznaczonym przez twórcę witryny. Tak jednak być nie musi. Bez problemów można utworzyć boks z menu, który będzie mógł być przesuwany po całej witrynie, zatem użytkownik będzie miał możliwość samodzielnego ustawienia najdogodniejszej pozycji. Menu należy zatem umieścić na warstwie i przygotować procedury pozwalające na jej przesuwanie za pomocą ruchów myszy. Najlepiej po prostu użyć funkcji przedstawionych i dokładnie omówionych w opisie do skryptu 62. z rozdziału 7. Kod HTML przyjąłby zatem postać przedstawioną na listingu 10.20.

Listing 10.20. *Kod HTML skryptu 96.*

```
<body>
  <div id="menuDiv" class="menu">
    <div class="menuheader" onmousedown="dragDiv('menuDiv');">Menu</div>
    <div onclick="menuclicked(this);" class="menuitem">Pozycja 1</div>
    <div onclick="menuclicked(this);" class="menuitem">Pozycja 2</div>
    <div onclick="menuclicked(this);" class="menuitem">Pozycja 3</div>
  </div>
  <div id="mainMenu">
    Treść strony
  </div>
</body>
```

W sekcji <body> znajduje się warstwa o identyfikatorze menuDiv, w której zawarte zostały poszczególne pozycje menu, a także warstwa mainDiv przechowująca pozostałą część strony (oczywiście warstw z treścią może być dowolnie więcej). Menu jest złożone z serii kolejnych warstw. Pierwsza tworzy nagłówek, czyli tytuł menu, a pozostałe odpowiadają poszczególnym pozycjom. Warstwa menu ma przypisaną klasę CSS menu, warstwa z pozycją nagłówkową — klasę menuheader, a warstwy tworzące pozycje — klasę menuitem. To pozwala na swobodne kształtowanie wystroju. Kod CSS nie jest jednak istotny dla tego przykładu i można go dobrać dowolnie.

Ważna jest obsługa zdarzeń dotyczących warstw z pozycjami. Warstwa nagłówkowa (menuheader) ma atrybut onmousedown, w którym zawarto wywołanie funkcji dragDiv. Funkcja będzie więc wywoływana, gdy nad pozycją nagłówkową zostanie wciśnięty przycisk myszy. Argument funkcji wskazuje identyfikator warstwy z menu (divMenu). Oznacza to zatem, że po naciśnięciu przycisku myszy nad warstwą z tytułem menu rozpocznie się proces przesuwania całego menu.

Z kolei każda warstwa tworząca pozycje menu (warstwy z klasą CSS menuitem) ma atrybut onclick, w którym zawarto wywołanie funkcji menuclicked z argumentem wskazującym na klikniętą pozycję. Dzięki temu menu będą reagowały na kliknięcia. Została tu użyta ta sama technika co w m.in. skrypcie 92.

Ostatnim składnikiem jest kod skryptu obsługujący przesuwane menu. Najlepiej skorzystać z pomysłu przedstawionego w przykładzie 62. Trzeba jedynie dostosować kod do obecnej sytuacji, w której element inicjujący przesuwanie (warstwa z pozycją nagłówkową) nie jest elementem przesuwanym, ale warstwą zawartą w przesuwanej warstwie (menuDiv). Trzeba więc dopisać kod funkcji dragDiv. Skrypt przyjmie zatem postać przedstawioną na listingu 10.21.

Listing 10.21. *Kod skryptu obsługującego przesuwalne menu*

```
<script type="text/javascript">
  // tutaj pełna treść skryptu 62.

  // tutaj treść funkcji menuclicked z listingu 10.14

  function dragDiv(id)
  {
    var div = document.getElementById(id);
    if(div) startDrag(div);
  }
</script>
```

Funkcja dragDiv otrzymuje w postaci argumentu identyfikator warstwy, która ma być przesuwana, i wywołuje funkcję startDrag (z przykładu 62.), która jako argument przyjmuje obiekt warstwy do przesuwania. Jedynym zadaniem jest więc użycie metody getElementById obiektu document do pobrania obiektu wskazywanego przez id i po sprawdzeniu, że operacja ta zakończyła się sukcesem, wywołanie funkcji startDrag.

Skrypt 97.

Menu kontekstowe

[C][E][F][S]

Za pomocą JavaScriptu można wyposażyć stronę w menu kontekstowe wywoływane za pomocą prawego przycisku myszy. Takie właśnie zadanie jest realizowane przez skrypt 97. Kliknięcie w dowolnym miejscu dokumentu prawym przyciskiem myszy spowoduje pojawienie się w tej lokalizacji menu. Wybranie dowolnej pozycji będzie przenosiło na wskazaną podstronę serwisu (lub pod dowolny adres internetowy), natomiast ponowne kliknięcie poza obszarem pozycji spowoduje zamknięcie menu.

Kod HTML przyjmie postać zaprezentowaną na listingu 10.22.

Listing 10.22. *Kod HTML skryptu 97.*

```
<body>
  <div id="mainDiv">
    Treść strony.
  </div>
  <div id="menuDiv" class="menu">
    <div class="menuheader">Menu</div>
    <div class="menuitem"><a href="href">Pozycja 1</a></div>
    <div class="menuitem"><a href="href">Pozycja 2</a></div>
    <div class="menuitem"><a href="href">Pozycja 3</a></div>
  </div>
</body>
```

W sekcji `<body>` znalazły się dwie warstwy: przechowująca główną treść strony — `mainDiv`, oraz zawierająca menu — `menuDiv`. W warstwie z menu znajduje się seria kolejnych warstw (elementów definiowanych za pomocą znacznika `<div>`) odpowiadających poszczególnym pozycjom. Istnieją dwa typy pozycji. Pierwszy to pozycja nagłówkowa (z klasą CSS `menuheader`) z tytułem menu. Drugi typ to pozycje zwykłe (z klasą CSS `menuitem`), z których każda zawiera odnośnik (element zdefiniowany za pomocą znacznika `<a>`). Zatem kliknięcie każdej pozycji będzie powodowało przeniesienie na stronę (podstronę) o adresie zawartym w atrybucie `href` znacznika `<a>`.

Tak utworzone menu jest obsługiwane przez skrypt zaprezentowany na listingu 10.23.

Listing 10.23. *Skrypt obsługujący menu kontekstowe*

```
<script type="text/javascript">
  document.oncontextmenu = showMenu;
  document.onclick = hideMenu;

  function mousePos(evt)
  {
    //treść funkcji mousePos
  }

  function showMenu(evt)
  {
    var div = document.getElementById('menuDiv');
```



```
if(!div) return;

var xy = mousePos(evt);
div.style.left = xy.x + "px";
div.style.top = xy.y + "px";
div.style.display = 'block';

return false;
}

function hideMenu()
{
    var div = document.getElementById('menuDiv');
    if(div) div.style.display = 'none';
}
</script>
```

Pierwsze dwie instrukcje modyfikują właściwości `oncontextmenu` i `onclick` obiektu `document`. Dzięki temu zdarzenia `contextmenu` i `click` otrzymają nowe procedury obsługi. W pierwszym przypadku jest to funkcja `showMenu`, a w drugim — `hideMenu`. Zatem gdy w obszarze dokumentu powstanie zdarzenie polegające na wywołaniu menu kontekstowego (w typowym przypadku jest to kliknięcie prawym przyciskiem myszy, ale może to być również wciśnięcie odpowiedniego klawisza na klawiaturze), zostanie wywołana funkcja `showMenu`, a przy zwykłym kliknięciu — funkcja `hideMenu`.

Ponieważ funkcja `showMenu` stała się procedurą obsługi zdarzenia `contextMenu`, jako argument (`evt`) otrzymuje obiekt zdarzenia (nie dotyczy to przeglądarki Internet Explorer). Pierwszą czynnością jest pobranie odwołania do warstwy z menu (warstwy o identyfikatorze `menuDiv`), zapisanie go w zmiennej `div` i sprawdzenie, czy zmienna ta jest pusta. Jeśli jest pusta, oznacza to, że w kodzie HTML nie ma warstwy z menu, nie ma więc też czego wyświetlać. W takiej sytuacji wykonywanie kodu jest przerywane za pomocą instrukcji `return`.

W kolejnej instrukcji wywoływana jest funkcja `mousePos` zwracająca współrzędne aktualnego położenia kursora myszy. Pobrane dane są zapisywane w zmiennej pomocniczej `xy`. Treść funkcji `mousePos` można zapożyczyć bezpośrednio ze skryptu 62. Odczytane współrzędne (właściwości `x` i `y` obiektu `xy`) są używane do ustalenia położenia warstwy. Współrzędna `x` jest przypisywana (po uzupełnieniu o ciąg `px`) właściwości `left` obiektu `style` warstwy `div` (warstwy z menu), a współrzędna `y` — właściwości `top`. Po ustaleniu położenia warstwa jest wyświetlana poprzez przypisanie wartości `block` właściwości `display`.

Funkcja `showMenu` zwraca wartość `false`, co zapobiega wyświetleniu standardowego menu kontekstowego przeglądarki.

Zadaniem funkcji `hideMenu` jest schowanie warstwy z menu. Kod jest o wiele prostszy, gdyż w tym przypadku nie ma konieczności ustalania współrzędnych. Odwołanie do warstwy jest więc pobierane za pomocą metody `getElementById` obiektu `document`, a warstwa jest chowana dzięki przypisaniu ciągu `none` właściwości `display` obiektu `style`.

Rozdział 11.

Powiązane opcje i menu hierarchiczne

Ostatni rozdział książki poświęcony jest tematowi powiązanych opcji i menu hierarchicznych. Zawiera jedynie cztery przykłady, jednak składające się ze stosunkowo dużej ilości kodu. Zostanie przedstawione menu w postaci drzewiastej, poziome menu rozwijane hierarchicznie (takie, w którym podpozycje zawierają własne podpozycje), powiązane listy rozwijane umożliwiające wybór „łańcucha” opcji, a także powiązane pola wyboru.

Skrypt 98. Boczne drzewo menu

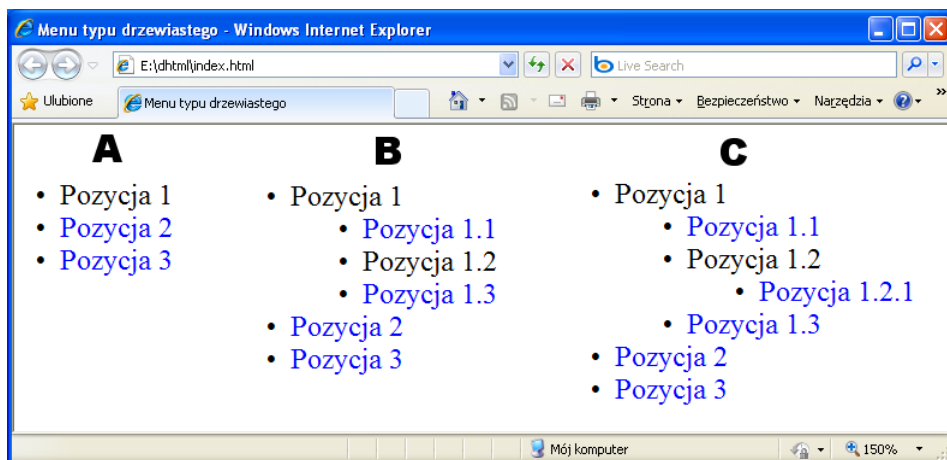
[C][E][F][O][S]

Menu może tworzyć strukturę hierarchiczną, w której każda pozycja może mieć dowolną liczbę podpozycji. Powstaje w ten sposób wielopoziomowe menu zagnieżdżone. Stosunkowo często spotyka się strukturę drzewiastą umieszczoną z boku strony pozwalającą na dostęp do wielu powiązanych ze sobą opcji. Taki efekt będzie realizowany przez skrypt 98. Menu w postaci zwiniętej będzie miało postać przedstawioną na rysunku 11.1A. Część pozycji będzie miało swoje własne podpozycje, które będzie można dowolnie rozwijać i związać (rysunki 11.1B i C).

Tworzenie tego typu menu nie jest skomplikowane. Wystarczy użyć zwyczajnych zagnieżdżonych list wypunktowanych i dodać do nich obsługę zdarzeń. Kod HTML będzie miał więc postać zaprezentowaną na listingu 11.1.

Listing 11.1. *Kod HTML skryptu 98.*

```
<body onload="initMenu('menuDiv', 'firstUL');">
  <div id="menuDiv">
    <ul id="firstUL">
      <li>Pozycja 1
```



Rysunek 11.1. Różne stopnie rozwinięcia menu drzewiastego

```
<ul>
  <li><a href="url">Pozycja 1.1</a></li>
  <li>Pozycja 1.2
    <ul>
      <li><a href="url">Pozycja 1.2.1</a></li>
    </ul>
  </li>
  <li><a href="url">Pozycja 1.3</a></li>
</ul>
</li>
<li><a href="url">Pozycja 2</a></li>
<li><a href="url">Pozycja 3</a></li>
</ul>
</div>
</body>
```

W sekcji `<body>` została umieszczona warstwa menu `div`, a w niej zestaw zagnieżdżonych list zdefiniowanych za pomocą znaczników `` i ``. Lista z głównymi pozycjami otrzymała identyfikator `firstUL`. W pozycjach, które nie mają podpozycji, zostały umieszczone zwyczajne odnośniki. Oczywiście w praktycznej realizacji ciągi `url` należy zamienić na odpowiednie adresy. Taka struktura oznacza trzy główne pozycje, z których pierwsza ma trzy podpozycje (1.1, 1.2 i 1.3), a druga podpozycja drugiej podpozycji ma kolejną podpozycję (1.2.1). Jest to więc struktura z rysunku 11.1C.

Znaczniki `` nie zawierają żadnych atrybutów związanych z obsługą zdarzeń, gdyż przy dużej ich liczbie taki sposób obsługi byłby niewygodny. Muszą jednak reagować na kliknięcia powodujące zwijanie i rozwijanie odpowiednich gałęzi drzewa menu, dlatego też odpowiednie procedury obsługi zdarzenia `click` zostaną dodane przez funkcję `initMenu` przypisaną jako procedura obsługi zdarzenia `load` sekcji `<body>`. Ta funkcja zostanie zatem wykonana tuż po załadowaniu strony do przeglądarki. Oprócz nadania obsługi zdarzeń zwinie też wszystkie pozycje, tak aby w początkowym stanie były widoczne tylko te główne.

Funkcja przyjmuje dwa argumenty. Pierwszy określa warstwę zawierającą całe menu, a drugi — identyfikator pierwszej listy (listy z głównymi pozycjami). Treść funkcji została przedstawiona na listingu 11.2.

Listing 11.2. *Treść funkcji `initMenu`*

```
function initMenu(id, ul)
{
    var div = document.getElementById(id);
    var firstUL = document.getElementById(ul);
    if(!div || !firstUL) return;
    var uls = firstUL.getElementsByTagName('ul');
    for(var i = 0; i < uls.length; i++){
        uls[i].style.display = 'none';
    }
    var lis = div.getElementsByTagName('li');
    for(var i = 0; i < lis.length; i++){
        lis[i].onclick = menuItemClick;
    }
}
```

Argument `id` wskazuje identyfikator warstwy z menu, a `ul` — identyfikator pierwszej listy. Wymienione elementy są pobierane oraz zapisywane w zmiennych `div` i `firstUL`. Następnie badane jest, czy te operacje zakończyły się sukcesem. Jeśli tak, za pomocą metody `getElementsByTagName` pobierane są wszystkie elementy typu `ul` (utworzone w kodzie HTML za pomocą znaczników ``) zawarte poniżej (niżej w hierarchii, wewnątrz) listy reprezentowanej przez `firstUL`. Pobrane elementy są zapisywane w tablicy `uls`. Są to listy wewnętrzne, które powinny być zwinięte, dlatego wyłączane jest ich wyświetlanie, co odbywa się przez przypisanie wartości `none` właściwości `display` (zamiast tej procedury można w kodzie HTML też definiować odpowiednie reguły CSS).

W drugiej części funkcji za pomocą metody `getElementsByTagName` pobierane są wszystkie elementy `li` (zdefiniowane w kodzie HTML za pomocą znaczników ``) zawarte w warstwie menu. Ponieważ muszą reagować na kliknięcia, właściwości `onclick` każdego uzyskanego elementu jest przypisywana funkcja `menuItemClick`. Tym samym kliknięcie dowolnej pozycji menu będzie powodowało wywołanie tej funkcji. Jej treść została przedstawiona na listingu 11.3.

Listing 11.3. *Treść funkcji `menuItemClick`*

```
function menuItemClick(evt)
{
    for(var i = 0; i < this.childNodes.length; i++){
        if(this.childNodes[i].nodeName.toLowerCase() == 'ul'){
            if(this.childNodes[i].style.display == 'none')
                this.childNodes[i].style.display = 'block';
            else
                this.childNodes[i].style.display = 'none';
        }
    }
    evt = evt ? evt : window.event;
```

```

    if(evt.stopPropagation) evt.stopPropagation();
    evt.cancelBubble = true;
}

```

Funkcja `menuItemClick` jest wywoływana w odpowiedzi na kliknięcie danej pozycji menu (odwołanie do tej pozycji uzyskamy dzięki wskazaniu `this`). Jej zadaniem jest rozwinięcie, jeśli były zwinięte, lub zwinięcie, jeśli były rozwinięte, wszystkich pozycji bezpośrednio podrzędnych (o ile takie istnieją). Bezpośrednio podrzędnych, to znaczy takich, które są zawarte bezpośrednio w danej pozycji (np. dla pozycji 1 bezpośrednio podrzędne są 1.1, 1.2 i 1.3, ale już nie 1.2.1). Te operacje są wykonywane w pętli `for`. Przebiega ona w drzewie DOM przez wszystkie węzły potomne (`childNodes`) węzła `this` (czyli bieżącej pozycji). Jeżeli mamy do czynienia z węzłem typu `ul` (czyli listą podrzędną, potomną — `this.childNodes[i].nodeName.toLowerCase() == 'ul'`), badane jest, czy jest ona wyświetlana (`else`), czy też nie (`if(this.childNodes[i].style.display == 'none')`). Jeżeli jest wyświetlana, ma zostać wyłączona (`this.childNodes[i].style.display = 'none';`), a jeśli nie jest wyświetlana, ma zostać włączona (`this.childNodes[i].style.display = 'block';`).

Po zakończeniu pętli konieczne jest przerwanie propagacji obiektu zdarzenia (inaczej funkcja będzie wykonywana dla wszystkich elementów nadrzędnych do bieżącego, co spowoduje nieprawidłowe zachowanie menu). W przeglądarkach z rodziny Internet Explorer obiekt zdarzenia znajduje się we właściwości `event` obiektu `window`, a w pozostałych — jest przekazywany jako argument funkcji obsługującej zdarzenie. Stąd instrukcja:

```
evt = evt ? evt : window.event;
```

ujednolicająca zapis.

Przerwanie propagacji zdarzeń odbywa się przez wywołanie metody `stopPropagation` (w przeglądarkach typu Firefox, Chrome itp.) lub przypisanie wartości `true` właściwości `cancelBubble` (Internet Explorer). Ponieważ Internet Explorer nie zawiera metody `stopPropagation` przed jej wywołaniem, za pomocą instrukcji `if` następuje upewnienie się, że jest ona obecna.

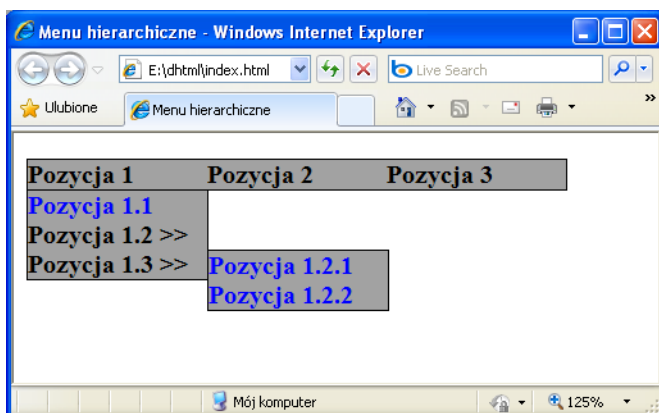
Skrypt 99.

[C][E8][F][O][S]

Menu rozwijane z podpozycjami

Pozycje poziomego menu rozwijanego mogą zawierać swoje własne podpozycje. Menu miałoby wtedy postać przedstawioną na rysunku 11.2. Pozycje zwykłe, bez elementów potomnych, będą zawierały klasyczne odnośniki — pozostałe będą rozwijane i zwiżane za pomocą kliknięć. Część HTML skryptu będzie miała strukturę analogiczną do przedstawionej w przykładzie 98. Menu będzie się zatem składało z zagnieżdżonych list. Początek kodu HTML został przedstawiony na listingu 11.4.

Rysunek 11.2.
*Hierarchiczne
 menu poziome*



Listing 11.4. *Początek kodu HTML dla skryptu 99.*

```
<body onload="initMenu('menuDiv', 'firstUL');">
  <div id="menuDiv">
    <ul id="firstUL">
      <li>Pozycja 1
        <ul>
          <li><a href="url">Pozycja 1.1</a></li>
          <li>Pozycja 1.2 >>
            <ul>
              <li><a href="url">Pozycja 1.2.1</a></li>
              <li><a href="url">Pozycja 1.2.2</a></li>
            </ul>
          </li>
          <li>Pozycja 1.3 >>
            <ul>
              <li><a href="url">Pozycja 1.3.1</a></li>
              <li><a href="url">Pozycja 1.3.2</a></li>
            </ul>
          </li>
        </ul>
      </li>
      <li>Pozycja 2
        <!-- dalsze pozycje menu -->
      </li>
    </ul>
  </div>
</body>
```

Przedstawiona struktura składa się z serii znaczników `` i `` tworzących zwykłe listy zagnieżdżone. Pozycje generowane za pomocą elementów `` zawierają albo odnośnik wygenerowany dzięki znacznikowi `<a>`, albo tekst i listę zagnieżdżoną powstałą dzięki znacznikowi ``. Lista z głównymi pozycjami ma identyfikator `firstUL`, a warstwa z menu — identyfikator `menuDiv`. Znacznik `<body>` ma atrybut `onload` zawierający wywołanie funkcji `initMenu`. Jej znaczenie jest takie samo jak w przykładzie 98., nie będzie więc ponownie prezentowana.

Ponieważ chcemy uzyskać menu poziome rozwijane, niezbędne jest użycie stylów CSS wpływających na położenie elementów. Zostały przedstawione na listingu 11.5.

Uwzględniono tylko reguły mające bezpośredni wpływ na omawiany efekt. Elementy dekoracyjne zostały pominięte — należy je dobrać wedle własnych potrzeb.

Listing 11.5. *Podstawowe style CSS dla skryptu 99.*

```
<style type="text/css">
  li{
    cursor:pointer;
    width:14ex;
  }
  ul{
    list-style-type:none;
  }
  #firstUL li{
    float:left;
  }
  #firstUL li li{
    float:none;
  }
  #firstUL li ul{
    position:absolute;
    padding:0px;
  }
  #firstUL li ul ul{
    left:14ex;
  }
</style>
```

Mamy tu reguły z następującymi selektorami:

- ♦ `li` — dotyczy wszystkich pozycji; ustalany jest kształt kursora (`pointer`) oraz szerokość pozycji (`14ex`);
- ♦ `ul` — dotyczy wszystkich list; usuwany jest wyróżnik (`punkt`or — `list-style-type:none`);
- ♦ `#firstUL li` — dotyczy pozycji głównych; ustalany jest lewy przepływ (`float:left;`), dzięki czemu ustawią się w poziomie jedna za drugą;
- ♦ `#firstUL li li` — dotyczy pozostałych pozycji; usuwany jest lewy przepływ (`float:none;`), dzięki czemu będą wyświetlane jedna pod drugą;
- ♦ `#firstUL li ul` — dotyczy list zagnieżdżonych w pozycjach (czyli podpozycji); ustalane jest pozycjonowanie bezwzględne oraz usuwane jest wypełnienie, dzięki czemu będą wyświetlane pod pozycjami głównymi.
- ♦ `#firstUL li ul ul` — dotyczy list zagnieżdżonych w listach; przesuwane są o szerokość jednej pozycji w prawo (lewy brzeg przesunięty o `14ex` — `left:14ex;`), dzięki czemu będą wyświetlane z prawej strony pozycji nadrzędnych.

Skrypt obsługujący menu będzie oparty na przykładzie 98., choć niezbędne będą dodatkowe funkcje. Pierwsza część kodu została przedstawiona na listingu 11.6.

Listing 11.6. *Skrypt obsługujący hierarchiczne menu*

```
<script type="text/javascript">
  //tutaj treść funkcji initMenu

  function menuItemClick(evt)
  {
    closeAllExcept(this);
    for(var i = 0; i < this.childNodes.length; i++){
      if(this.childNodes[i].nodeName.toLowerCase() == 'ul'){
        if(this.childNodes[i].style.display == 'none'){
          this.childNodes[i].style.display = 'block';
          closeAllDown(this.childNodes[i]);
        }
      }
    }
    //dalsza część funkcji menuItemClick
  }
  //dalsza część skryptu
```

Funkcja `initMenu` pozostała w takiej samej postaci jak w przykładzie 98. (listing 11.2), dlatego też nie została ponownie zaprezentowana. Treść funkcji `menuItemClick` również jest bardzo podobna, została jednak uzupełniona o dodatkowy kod. Ogólne zadanie tej funkcji jest takie samo jak w skrypcie 98. — chodzi o zwiniecie lub rozwinięcie klikniętej pozycji (o ile zawiera pozycje podrzędne). Nie można jednak dopuścić, aby jednocześnie były rozwinięte dwa menu podrzędne. Jeżeli zatem została kliknięta pewna pozycja, to muszą zostać zamknięte (zwinięte) wszystkie inne podpozycje z tego samego poziomu. To zadanie wykonuje funkcja `closeAllExcept`. Wywołanie:

```
closeAllExcept(this);
```

należy rozumieć jako: zwiń wszystkie rozwinięte menu z aktualnego poziomu z wyjątkiem menu bieżącego (`this`).

Druga zmiana znajduje się w pętli `for`. Po wykryciu, że dana pozycja ma być rozwinięta (`if(this.childNodes[i].style.display == 'none'){}),` nie wystarczy jej tylko wyświetlić, dodatkowo trzeba zamknąć wszystkie pozycje podrzędne (o ile takie istnieją). Tym zajmuje się funkcja `closeAllDown`. Wywołanie:

```
closeAllDown(this.childNodes[i]);
```

oznacza: zamknij wszystkie pozycje podrzędne w stosunku do pozycji wskazywanej przez `this.childNodes[i]`.

Treść funkcji `closeAllExcept` i `closeAllDown` została przedstawiona na listingu 11.7.

Listing 11.7. *Dalsza część kodu skryptu 99.*

```
//początek kodu z listingu 11.6
function closeAllDown(e1)
{
  if(!e1) return;
  var uls = e1.getElementsByTagName('ul');
  for(var i = 0; i < uls.length; i++)
    uls[i].style.display = 'none';
}
```

```

    }
    function closeAllExcept(menu)
    {
        if(!menu || !menu.parentNode) return;
        var nodes = menu.parentNode.childNodes;
        for(var i = 0; i < nodes.length; i++){
            if(nodes[i].nodeName.toLowerCase() == 'li' && nodes[i] != menu){
                closeAllDown(nodes[i]);
            }
        }
    }
}
</script>

```

Funkcja `closeAllDown` ma zamknąć wszystkie menu podrzędne elementu przekazanego w postaci argumentu `el`. Po sprawdzeniu, że wartość elementu nie jest pusta, następuje pobranie zawartych w `el` elementów typu `ul` (list zdefiniowanych w kodzie HTML za pomocą znacznika ``). W tym celu jest używana metoda `getElementsByTagName`:

```
var uls = el.getElementsByTagName('ul');
```

Po wykonaniu tej instrukcji zmienna `uls` będzie zawierała tablicę z odwołaniami do pobranych elementów. Zawartość tablicy jest przeglądana w pętli typu `for`. W każdym elemencie modyfikowana jest właściwość `display` obiektu `style` — to powoduje wyłączenie wyświetlania pobranych list.

Funkcja `closeAllExcept` otrzymuje w postaci argumentu `menu` obiekt klikniętego menu, a jej zadaniem jest wyłączenie wyświetlania wszystkich menu znajdujących się na tym samym poziomie co przekazane w postaci argumentu. Najpierw sprawdza, czy obiekt `menu` jest niepusty oraz czy istnieje obiekt nadrzędny (`menu.parentNode`). Jeśli tak jest, w zmiennej pomocniczej `nodes` zapisywane jest odwołanie do wszystkich elementów potomnych elementu nadrzędnego w stosunku do elementu przekazanego jako argument:

```
nodes = menu.parentNode.childNodes;
```

Ma to na celu wyłącznie skrócenie zapisu dalszych instrukcji.

W pętli `for` przeglądane są wszystkie elementy zapisane w `nodes`. Jeżeli bieżący element jest typu `li` (`nodes[i].nodeName.toLowerCase() == 'li'`) i jednocześnie nie jest to element kliknięty (`nodes[i] != menu`), wyłączane są wszystkie przypisane mu menu podrzędne. Odbywa się to za pomocą opisanej wyżej funkcji `closeAllDown`.

Skrypt 100.

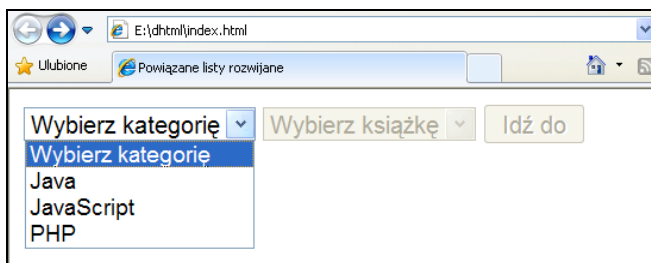
[C][E][F][O][S]

Powiązane listy rozwijane

Na stronach WWW często spotyka się powiązane listy rozwijane pozwalające na dokonywanie wyboru pewnej opcji w kilku kolejnych krokach. Najpierw wybierana jest kategoria ogólna, a później coraz bardziej szczegółowe. Przykładem jest wybór marek

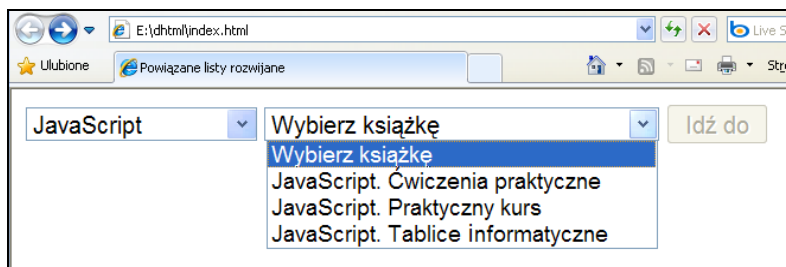
i modeli samochodów czy też kategorii tematycznych i tytułów książek. Sposób realizacji takiego zadania został zobrazowany w skrypcie 100. Po załadowaniu witryny do przeglądarki w dokumencie pojawiają się dwie listy i przycisk. Tylko pierwsza lista będzie aktywna i umożliwi wybór kategorii tematycznej książek. Ten etap został zilustrowany na rysunku 11.3.

Rysunek 11.3.
*Rozwinięta pierwsza
lista rozwijana*



Dopiero po dokonaniu wyboru kategorii uaktywni się druga lista, zawierająca zestaw tytułów książek. Ten etap został zilustrowany na rysunku 11.4. Wybranie danego tytułu spowoduje aktywację przycisku *Idź do*. Kliknięcie go umożliwi załadowanie do przeglądarki witryny odpowiadającej danej książce. Przycisk będzie aktywny tylko wtedy, gdy na drugiej liście będzie zaznaczona pozycja z tytułem książki.

Rysunek 11.4.
*Wybranie kategorii
pozwoliło
na wyświetlenie
listy książek*



Na listingu 11.8 został przedstawiony kod HTML witryny działającej w opisany sposób.

Listing 11.8. *Kod HTML skryptu 100.*

```
<body onload="generujListę1();">
  <div id="listyDiv">
    <select id="lista1" size="1" onchange="lista1Change(this);">
      <option value="">Wybierz kategorię</option>
    </select>
    <select id="lista2" size="1" onchange="lista2Change(this);"
      disabled="disabled">
      <option value="">Wybierz książkę</option>
    </select>
    <input type="button" id="btnGo" value="Idź do" onclick="btnGoClick();"
      disabled="disabled" />
  </div>
</body>
```

W sekcji <body> została umieszczona warstwa `listyDiv`, a w niej dwie listy (`lista1` i `lista2`) oraz przycisk (`btnGo`). Listy zostały utworzone za pomocą typowych znaczników <select>. Każda z nich otrzymała też atrybut `onchange`. Dzięki temu po zmianie aktywnej pozycji pierwszej listy zostanie wywołana funkcja `lista1Change` (jako argument otrzyma obiekt listy), a po zmianie aktywnej pozycji drugiej listy — funkcja `lista2Change` (jako argument otrzyma również obiekt listy). Druga lista została domyślnie wyłączona (atrybut `disabled`).

Przycisk został zrealizowany za pomocą znacznika <input> z atrybutem `type` ustawionym na `text`. Podobnie jak druga z list, domyślnie jest wyłączony (atrybut `disabled`). Ponieważ został użyty atrybut `onclick`, przycisk będzie reagował na kliknięcia — zostanie wywołana funkcja `btnGoClick`.

Obie listy zawierają po jednej pozycji. Te pozycje nie są istotne dla działania skryptu, bowiem właściwa zawartość będzie generowana przez kod JavaScript. Zostały jednak umieszczone ze względu na zachowanie zgodności ze standardami HTML (znacznik <select> nie powinien być pusty). Pierwsza lista zostanie wypełniona danymi tuż po załadowaniu witryny do przeglądarki. Odpowiada za to funkcja `generujListę1()`, której wywołanie zostało umieszczone jako treść atrybutu `onload` sekcji <body>.

Skrypt obsługujący stronę będzie się składał z trzech głównych części. Pierwsza, zawierająca definicję struktury danych, została przedstawiona na listingu 11.9.

Listing 11.9. *Pierwsza część skryptu 100.*

```
<script type="text/javascript">
var baseURL = "http://helion.pl/ksiazki/";
var dane1 = new Array(
    "Java", "JavaScript", "PHP"
);
var dane2 = new Array(
    new Array(
        {t:"Java. Ćwiczenia praktyczne",u:"cwjav2.htm"},
        {t:"Java. Praktyczny kurs",u:"pkjav2.htm"},
        {t:"Java. Tablice informatyczne",u:"tijav2.htm"}
    ),
    new Array(
        {t:"JavaScript. Ćwiczenia praktyczne",u:"cwjas2.htm"},
        {t:"JavaScript. Praktyczny kurs",u:"jscpk.htm"},
        {t:"JavaScript. Tablice informatyczne",u:"tijjs.htm"}
    ),
    new Array(
        {t:"PHP. 101 praktycznych skryptów",u:"php102.htm"},
        {t:"PHP5. Praktyczny kurs",u:"php5pk.htm"},
        {t:"PHP. Tablice informatyczne",u:"tiphp2.htm"}
    )
);
//dalsza część skryptu
```

Ponieważ wszystkie adresy WWW, do których będą prowadziły poszczególne opcje, będą się znajdowały w jednej lokalizacji, została użyta zmienna pomocnicza `baseURL`,

wskazująca początek każdego z adresów. W przypadku stosowania odnośników z różnych domen należy tę zmienną pominąć i podawać pełne adresy bezpośrednio w opcjach w tablicy dane2.

Lista kategorii, czyli pozycje pierwszej listy wyboru, została zapisana w tablicy dane1. Jest to najprostsza możliwa konstrukcja, kolejne komórki tablicy zawierają ciągi znaków z nazwami kolejnych kategorii. Bardziej złożona jest konstrukcja tablicy dane2 zawierającej wszystkie dane, jakie mogą się pojawić na drugiej liście wyboru. Ponieważ na pierwszej liście znajdują się trzy kategorie, w drugiej tablicy muszą się znaleźć trzy różne zestawy opcji — po jednym zestawie dla każdej kategorii.

Dlatego też elementami drugiej tablicy są trzy kolejne tablice. Każda z nich zawiera serię obiektów przechowujących dane dotyczące poszczególnych pozycji. Każdy obiekt ma dwa pola: `t` i `u`. Pierwsze określa nazwę książki należącej do danej kategorii, a drugie — adres WWW przypisany danej opcji (a dokładniej część adresu, która ma być dodana do wartości zapisanej w `baseURL`). A zatem w tablicy dane1 mamy np. pierwszą pozycję (o indeksie 0) Java — tej pozycji odpowiada pierwsza pozycja (również o indeksie 0) w tablicy dane2, która zawiera zestaw obiektów odpowiadających książkom o Javie. Dane z obu tablic zostały zatem logicznie powiązane. Potrzebne są więc funkcje wypełniające listy wyboru. Zostały przedstawione na listingu 11.10.

Listing 11.10. *Funkcje generujące listy wyboru*

```
function generujListę1()
{
    var lista = document.getElementById('lista1');
    if(!dane1 || !lista) return;
    lista.length = 0;
    lista[0] = new Option("Wybierz kategorię", "");
    for(var i = 0; i < dane1.length; i++){
        lista[lista.length] = new Option(dane1[i], i);
    }
}
function generujListę2(id)
{
    var lista = document.getElementById('lista2');
    if(!dane2 || !lista) return;
    lista.length = 0;
    lista[0] = new Option("Wybierz książkę", "");

    if(id < 1 || id > dane2.length) return;
    id -= 1;
    for(var i = 0; i < dane2[id].length; i++){
        lista[lista.length] = new Option(dane2[id][i].t, dane2[id][i].u);
    }
}
```

Zadaniem funkcji `generujListę1`, wykonywanej tuż po załadowaniu strony do przeglądarki, jest wypełnienie danymi pierwszej listy wyboru (o identyfikatorze `lista1`). Odwołanie do listy jest pobierane za pomocą metody `getElementById` obiektu `document` i zapisywane w zmiennej `lista`. Następnie sprawdzane jest, czy zmienne `dane1` i `lista` są niepuste. Tylko wtedy są wykonywane dalsze czynności. Jeżeli brakuje tablicy `dane1`

lub nie ma w kodzie HTML listy o identyfikatorze `lista1`, wykonywanie kodu jest przerywane za pomocą instrukcji `return`.

Przed wypełnieniem listy danymi jest ona czyszczona, co odbywa się przez przypisanie właściwości `length` wartości 0. To powoduje usunięcie wszystkich istniejących pozycji. Następnie tworzona jest pierwsza pozycja (o indeksie 0) z napisem *Wybierz kategorię*:

```
lista[0] = new Option("Wybierz kategorię", "");
```

Używany jest tu konstruktor obiektu typu `Option`, któremu w postaci argumentów przekazywana jest nazwa pozycji oraz jej wartość. W tym przypadku wartością jest pusty ciąg znaków.

Dalsze pozycje tworzone są w pętli `for`, która przebiega całą tablicę `dane1`. Pozycje są tworzone i dodawane do listy w jednej instrukcji:

```
lista[lista.length] = new Option(dane1[i], i);
```

Ten zapis oznacza: zapisz w liście `lista`, w pozycji wskazywanej przez właściwość `lista.length`, nową opcję o tytule pobranym z tablicy `dane1` spod indeksu `i` i nadaj jej wartość wskazywaną przez indeks `i`. Tym samym w omawianym przypadku w indeksie 0 uzyskamy pozycję o tytule *Java* i wartości 0, w indeksie 1 — pozycję o tytule *JavaScript* o wartości 1, a w indeksie 2 — pozycję o tytule *PHP* o wartości 2.

Zadaniem funkcji `generujListę2` jest wypełnienie drugiej listy rozwijanej. Otrzymuje argument `id` wskazujący indeks wybranej pozycji pierwszej listy, a tym samym indeks tablicy `dane2` zawierający właściwy zestaw elementów. Początek kodu (pierwsze cztery instrukcje) jest analogiczny do przedstawionego przy omawianiu funkcji `generujListę1`, zmieniły się jedynie obiekty i identyfikatory, na których wykonywane są operacje. Zamiast `lista1` i `dane1` są to `lista2` i `dane2`.

Kolejne instrukcje są już zupełnie inne. Najpierw następuje weryfikacja argumentu `id`. Jego wartość musi się zawierać w przedziale od 1 (indeks 0 oznaczałby, że w pierwszej liście aktywna jest pozycja *Wybierz kategorię*) do długości tablicy `dane2` (`dane2.length`). Jeżeli przekazana wartość przekracza ten zakres, wykonywanie kodu jest kończone za pomocą instrukcji `return`. Jeśli zakres jest prawidłowy, od wartości zapisanej w `id` jest odejmowane 1. To pozwala na dopasowanie indeksu `id` do indeksów w tablicy `dane2` — po tej operacji w `id` będzie zapisany indeks właściwej tablicy wewnętrznej znajdującej się w tablicy `dane2`.

Lista jest wypełniana danymi w pętli `for`, która przebiega całą tablicę określoną jako `dane2[id]`. Konstrukcja tworząca pojedynczą opcję jest analogiczna do tej z funkcji `generujListę1`, choć ma nieco bardziej złożoną postać:

```
lista[lista.length] = new Option(dane2[id][i].t, dane2[id][i].u);
```

`dane2[id]` to tablica wewnętrzna w tablicy `dane2` zawierająca listę obiektów z tytułami książek i adresami WWW. `dane2[id][i]` to aktualnie przetwarzany obiekt. Pole `t` określa tytuł książki, więc `dane2[id][i].t` to tytuł bieżącej pozycji listy. Pole `u` określa adres WWW, więc `dane2[id][i].u` to wartość bieżącej pozycji listy.

Do pełnej funkcjonalności skryptu brakuje jeszcze procedur obsługi zdarzeń. Zostały przedstawione na listingu 11.11.

Listing 11.11. *Funkcje obsługujące zdarzenia*

```
function lista1Change(lista)
{
    var lista2 = document.getElementById('lista2');
    var btnGo = document.getElementById('btnGo');
    if(!lista || !lista2 || !btnGo) return;
    generujListę2(lista.selectedIndex);

    if(lista.selectedIndex == 0) lista2.disabled = true;
    else lista2.disabled = false;

    if(lista2.selectedIndex < 1) btnGo.disabled = true;
    else btnGo.disabled = false;
}
function lista2Change(lista)
{
    var btnGo = document.getElementById('btnGo');
    if(!lista || !btnGo) return;
    if(lista.selectedIndex == 0) btnGo.disabled = true;
    else btnGo.disabled = false;
}
function btnGoClick()
{
    var lista2 = document.getElementById('lista2');
    if(lista2[lista2.selectedIndex].value)
        document.location = baseUrl + lista2[lista2.selectedIndex].value;
}
```

Funkcja `lista1Change` jest wykonywana wtedy, gdy zmieni się aktywna pozycja pierwszej listy (`lista1`). Obiekt listy jest przekazywany w postaci argumentu po to, by nie pobierać go ponownie w treści funkcji. Ponieważ będą dodatkowo potrzebne odwołania do drugiej listy i do przycisku, są pobierane za pomocą metody `getElementById` i zapisywane w zmiennych `lista2` i `btnGo`. Następnie sprawdzane jest, czy istnieją wszystkie wskazane obiekty i jeśli istnieją, wywoływana jest funkcja `generujListę2` wypełniająca treścią drugą z list. Przekazywany argument wskazuje indeks aktywnej pozycji listy `lista1` (`lista.selectedIndex`).

Kolejne instrukcje sterują włączaniem i wyłączeniem elementów interfejsu. Jeżeli aktywną pozycją pierwszej listy jest ta o indeksie 0 (czyli pozycja *Wybierz kategorię* — `if(lista.selectedIndex == 0)`), należy wyłączyć drugą listę (`lista2.disabled = true;`). W przeciwnym przypadku druga lista powinna być włączona (`lista2.disabled = false;`). Z kolei jeśli indeks zaznaczonej pozycji w drugiej liście jest mniejszy niż 1 (czyli jeśli jest to 0 lub -1 — `if(lista2.selectedIndex < 1)`), należy wyłączyć przycisk *Idź do* (`btnGo.disabled = true;`), a w przeciwnym przypadku przycisk powinien być włączony (`btnGo.disabled = false;`).

Funkcja `lista2Change` jest wykonywana po zmianie aktywnej pozycji na drugiej liście (`lista2`). Otrzymuje w postaci argumentu obiekt listy, dzięki czemu nie musi być pobierany wewnątrz funkcji. Pobierane jest natomiast odwołanie do przycisku *Idź do*

(btnGo), niezbędne jest bowiem ustalenie, czy ma być włączony czy wyłączony. Przycisk powinien być nieaktywny, gdy indeks aktywnej pozycji listy (lista.selected-Index) jest mniejszy niż 1 (czyli równy 0 bądź -1). Za pomocą instrukcji if jest zatem badany stan właściwości selectedIndex i na tej podstawie przycisk jest włączany (btnGo.disabled = false;) bądź wyłączany (btnGo.disabled = true;).

Funkcja btnGoClick jest wykonywana po kliknięciu przycisku i ma za zadanie wczytać stronę odpowiadającą wybranej pozycji z listy lista2. Pobiera więc odwołanie do listy i zapisuje je w pomocniczej zmiennej lista2. Następnie bada, czy wartość wybranej (aktywnej) pozycji nie jest pusta (pozycje niemające przypisanych adresów zawierały puste ciągi znaków — w tym przypadku chodzi o pozycję o tytule *Wybierz książkę*). Jeśli nie jest pusta, oznacza to, że zawiera fragment adresu URL, który trzeba dodać do tego zapisanego w zmiennej baseUrl. Tak skonstruowany adres WWW jest przypisywany właściwości location obiektu document, co powoduje wczytanie nowej strony do przeglądarki.

Skrypt 101.

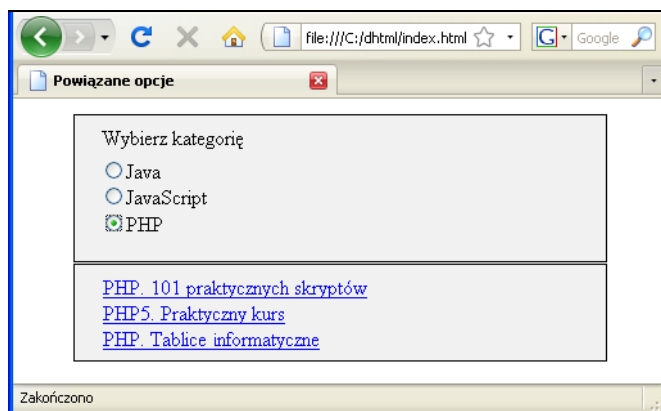
[C][E][F][O][S]

Powiązane opcje wyboru

Skrypt 101. jest ilustracją tego, jak na podstawie zgromadzonych danych budować pola wyboru pozwalające na wybór pewnej opcji, powodującej dynamiczne utworzenie kolejnej treści. Przygotowane zostaną kategorie tematyczne, z których powstaną pola wyboru typu radio. Wskazanie dowolnej opcji spowoduje wyświetlenie listy odnośników przypisanej do danej kategorii. Witryna będzie zatem miała postać przedstawioną na rysunku 11.5. Kod HTML będzie tym razem wyjątkowo prosty. Został przedstawiony na listingu 11.12.

Rysunek 11.5.

Lista wyboru kategorii



Listing 11.12. *Kod HTML skryptu 101.*

```
<body onload="generujPola('polaDiv');">
  <div id="polaDiv">
    </div>
```



```
<div id="daneDiv">
</div>
</body>
```

W sekcji `<body>` zostały umieszczone dwie zwyczajne warstwy: `polaNdiv` i `daneDiv`. Na pierwszej znajdują się pola wyboru umożliwiające wybór kategorii, a na drugiej będą wyświetlane listy odnośników przypisanych danej kategorii. Obie warstwy są początkowo puste, gdyż zawartość będzie generowana dynamicznie. Za utworzenie listy kategorii będzie odpowiedzialna funkcja `generujPola`, której wywołanie zostało przypisane właściwości `onload` znacznika `<body>`. To oznacza, że ta funkcja zostanie wykonana tuż po załadowaniu strony. Kod funkcji wraz z pozostałą częścią skryptu został zaprezentowany na listingu 11.13. Ponieważ zostało przyjęte, że struktura danych pozostanie taka sama jak w przykładzie 100., początkowy fragment kodu został pominięty (definicje tablic `dane1` i `dane2` oraz zmiennej `baseURL`).

Listing 11.13. *Kod JavaScript skryptu 101.*

```
<script type="text/javascript">
//tutaj definicje danych
function generujPola(id)
{
    var div = document.getElementById(id);
    if(!div || !dane1) return;
    var str = "<fieldset><legend>Wybierz kategorię<\legend>";
    for(var i = 0; i < dane1.length; i++){
        str += "<input type='radio' name='kategoria' value='" + i + "' ";
        str += "onclick='pokażKsiążki(" + i + ")' \/>";
        str += dane1[i] + "<br \/>";
    }
    str += "<\fieldset>";
    div.innerHTML = str;
}
function pokażKsiążki(index)
{
    var div = document.getElementById('daneDiv');
    if(!div || !dane2 || index < 0 || index >= dane2.length) return;

    var str = "";
    for(var i = 0; i < dane2[index].length; i++){
        str += "<a href='" + baseURL + dane2[index][i].u + "'>";
        str += dane2[index][i].t + "<\a><br \/>";
    }
    div.innerHTML = str;
}
</script>
```

Funkcja `generujPola` przyjmuje argument określający identyfikator warstwy, na której mają się pojawić pola wyboru z nazwami kategorii. Odwołanie do tej warstwy pobierane jest za pomocą metody `getElementById` i zapisywane w zmiennej `div`. Następnie sprawdzane jest, czy operacja udała się (zmienna `div` jest niepusta) oraz czy istnieje tablica `dane1`. Jeśli dane są prawidłowe, są wykonywane dalsze czynności.

Zmienna `str` będzie przechowywała kod HTML warstwy `polaDiv`, czyli pola wyboru. Kod tworzony jest w pętli typu `for`, która odczytuje dane z tablicy `dane1` i wstawia je w odpowiednie miejsca. Ogólna struktura tego kodu jest następująca:

```
<fieldset>
  <legend>Wybierz kategorię</legend>
  <input type='radio' name='kategoria' value='nr_kategorii '
        onclick='pokażKsiążki(nr_kategorii)' />nazwa_kategorii<br >
  <!-- dalsze definicje pól wyboru -->
</fieldset>
```

Jest to zatem zgrupowany za pomocą znacznika `<fieldset>` zestaw elementów typu `<input>` z atrybutem `type` ustawionym na `radio`. Każdy taki znacznik (pole wyboru) ma przypisany atrybut `onclick`, dzięki czemu będzie reagował na kliknięcia. Będzie wtedy wywoływana funkcja `pokażKsiążki`, której w postaci argumentu zostanie przekazany numer kategorii. Po zakończeniu pętli wartość zmiennej `str` (czyli kod HTML warstwy) jest przypisywany właściwości `innerHTML` obiektu `div`, co powoduje, że pola pojawiają się na witrynie.

Funkcja `pokażKsiążki` ma za zadanie utworzyć i wyświetlić na warstwie `daneDiv` listę odnośników do książek z danej kategorii. Numer kategorii jest przekazywany w postaci argumentu `index`. Pierwsza instrukcja pobiera odwołanie do warstwy `daneDiv` i zapisuje je w zmiennej `div`. Następnie za pomocą instrukcji `if` ze złożonym wyrażeniem warunkowym weryfikowana jest poprawność danych. Badane jest, czy są niepuste zmienne `div` i `dane2` (tablica z danymi) oraz czy indeks jest nie mniejszy od zera i nie większy od długości tablicy `dane2` pomniejszonej o 1. Dalsze czynności są wykonywane tylko wtedy, gdy weryfikacja zakończy się sukcesem.

Na podstawie danych pobranych z tablicy `dane2` w pętli `for` tworzona jest lista odnośników generowanych za pomocą znaczników `<a>`. Budowany kod HTML jest cyklicznie dopisywany do zmiennej `str`, której zawartość po zakończeniu pętli jest zapisywana we właściwości `innerHTML` warstwy `daneDiv` (zmienna `div`). Struktura pojedynczego odnośnika jest następująca:

```
<a href='adres'>tytuł</a>
```

Bieżący adres jest uzyskiwany dzięki dodawaniu `baseURL + dane2[index][i].u`, a tytuł — dzięki odwołaniu `dane2[index][i].t`. Odbywa się to na tej samej zasadzie co w przypadku przykładu 100.

Skorowidz

:hover, 247
<a>, 194
<fieldset>, 105, 213, 290
<form>, 109
<head>, 95
<iframe>, 40, 42
, 23, 204, 237
<input>, 13, 48, 109, 126, 150, 195, 284
<label>, 13
<legend>, 105
, 61, 246, 248
<link>, 96, 99
<option>, 197
<p>, 195
<script>, 231
<select>, 40, 96, 197, 199
, 42, 116, 239
<style>, 14, 84
<table>, 27, 76
<td>, 27, 76
<textarea>, 128
<tr>, 27, 76
, 61, 246, 248

A

action, 109
adres
 e-mail, 146
 URL, 41
 WWW, 196
AJAX, 56
aktywny element formularza, 129
animowany napis, 164
appendChild(), 99, 188
ataki brute-force, 54
automatyczne podświetlanie wszystkich wystąpień
 poszukiwanego ciągu znaków, 118
automatyczne przenoszenie kursora między
 elementami formularza, 120
automatyczne przewijanie treści strony, 24

B

background-color, 27, 142
backgroundImage, 222
backgroundPosition, 222, 223
belka nawigacyjna, 245
blokada klawiszy, 121
blokada wpisywania w formularzu wybranych
 znaków, 121
blokowanie zaznaczenia fragmentu strony, 59
blur, 130
boczne drzewo menu, 275
boksy reklamowe, 171, 184, 185
 efekt przejścia, 190
 efekt skrolowania, 186
 zmienna treść, 184
border, 142, 253
border-bottom, 262
border-top, 41
button, 13

C

charAt(), 166, 170
checkbox, 13
chowanie warstw, 36
class, 23, 81
clearInterval(), 159
clearTimeout(), 137
click, 13, 24, 195, 273
clientX, 182
clientY, 182
close(), 18
closeModal(), 137
closeTooltip(), 172
color, 157, 240, 247
confirm(), 18, 135
contextmenu, 273
cookies, 56
createElement(), 99, 188
CSS, 26, 95

cursor, 27, 262
czas, 65
czasowy pasek postępu, 144
czcionki, 101

D

dane JSON, 238
data, 65
Date, 58, 65, 66
DHTML, 7
directories, 12
display, 253
display:block, 45
display:none, 36, 86
document.onmousemove, 181
document.onmouseup, 181
document.title, 140
document.write(), 62
dodatkowy opis odnośnika, 199
dodawanie strony do zakładek, 60
dostęp do ukrytej treści, 50
dostęp do zmiennych globalnych, 170
drag & drop, 83, 180
dynamiczna weryfikacja danych w trakcie ich wprowadzania, 124
dynamiczna zmiana stylu strony, 95
dynamiczne podpowiedzi, 171, 174

E

efekty
 lupa, 222
 obraz pływający po witrynie, 226
 przejście, 186, 190
 przełączane menu, 263
 przenikanie, 190, 258
element strony jako odnośnik, 195
e-mail, 146
event, 278

F

filter, 179
filter:alpha, 23
float, 52, 247, 253
focus, 130
fokus, 130
font-size, 101
font-style, 101
font-weight, 240, 247
for, 81
form1, 110
formatowanie elementów witryny, 76

formatowanie formularza, 49
formaty danych, 111
formularz logowania, 48, 54
formularze, 107
 aktywny element, 129
 automatyczne przenoszenie kursora między elementami, 120
 blokada klawiszy, 121
 blokada wpisywania wybranych znaków, 121
 dynamiczna weryfikacja danych w trakcie ich wprowadzania, 124
 elementy HTML, 109
 fokus, 130
 ograniczenie liczby znaków wpisywanych do rozszerzonego pola tekstowego, 128
 pole tekstowe, 109
 pole tekstowe automatycznie zmieniające wielkość, 126
 przycisk submit, 109
 rozszerzone pole tekstowe, 128
 sprawdzanie danych, 107
 walidacja, 107
 wartości pól, 110
 weryfikacja z uwzględnieniem formatu danych, 111
 wpisywanie danych, 120
 wyróżnianie aktywnego elementu, 129
 wyróżnianie błędnych pól, 111
funkcja skrótu, 54
funkcjonalność przeciagnij i upuść, 180

G

galeria obrazów z podpisami, 237
generowanie komórek tabeli, 88
generowanie komórki nagłówkowej, 87
getBrowserType(), 29, 32, 33
getCookie(), 58
getElementById(), 16, 37, 40, 60, 66, 159, 235, 265, 268
getElementsByTagName(), 188, 202, 206, 232, 267
getHours(), 66, 90
getMinutes(), 66
getSeconds(), 66

H

hasło zabezpieczające witrynę, 53
height, 12, 37, 142, 220
href, 50, 99, 194
HTML, 7, 8

I

id, 217
 identyfikacja przeglądarki, 29
 Image, 206, 219
 indexOf(), 242, 243
 innerHeight, 220
 innerHTML, 66, 114, 162, 185, 188
 innerWidth, 220
 interfejs do przewijania treści strony
 (przez kliknięcia), 21
 isNaN(), 17, 239

J

JavaScript, 7, 16
 JSON, 238

K

kalendarz, 78
 kalendarz przesuwalny (drag & drop), 83
 kalendarz typu pop-up pozwalający
 na wskazanie daty, 84
 karty, 43
 keyCode, 123
 keydown, 121, 122
 keypress, 121
 keyup, 120, 121, 125
 klasy CSS, 81
 klasyczne menu poziome, 247
 klawisze, 123
 kod pocztowy, 122
 kolor płynący po napisie, 160
 kolory, 159
 komórki nagłówkowe, 87
 kompozycja stylu z wybranymi elementami, 104
 kursor, 262

L

left, 12, 184
 length, 129
 liczba odwiedzin, 56
 list-style-type, 247
 listy, 61
 nienumerowane, 246
 rozwijane, 197
 litery pojawiające się pojedynczo, 168
 load, 58, 158, 220
 location, 12, 55, 198
 location.href, 33, 196, 198
 logo stale widoczne w wybranym miejscu
 strony, 223

logowanie użytkowników, 48
 lupa, 220

Ł

ładowanie obrazów z paskiem postępu (I), 230
 ładowanie obrazów z paskiem postępu (II), 234

M

marginesy, 247
 margin-left, 52
 Math, 37
 Math.abs(), 78
 Math.floor(), 70, 170
 Math.random(), 170
 max(), 37
 maxHeight, 268
 maxLength, 120, 127
 menu, 245
 boczne drzewo menu, 275
 efekt przenikania, 258
 przełączane menu wykluczające, 266
 przełączane menu z animacją, 267
 przełączane menu z niezależnymi pozycjami, 263
 przesuwany boks menu, 270
 ustawianie przez użytkownika, 270
 menu hierarchiczne, 275
 boczne drzewo menu, 275
 rozwijane menu z podpozycjami, 278
 menu kontekstowe, 272
 menu poziome
 CSS, 247
 JavaScript, 251
 wysuwane menu, 256
 menu przełączane wykluczające, 266
 menu przełączane z animacją, 267
 menu rozwijane z podpozycjami, 278
 menu wysuwane z boku, 260
 menubar, 12
 modalne okno dialogowe, 33
 modyfikacja
 pasek stanu, 139
 pasek tytułowy, 138
 mousedown, 60
 mousemove, 211
 mouseout, 24, 27, 172, 195, 196, 203
 mouseover, 24, 27, 172, 195, 196, 203
 mouseup, 211

N

NaN, 17
 napis na sinusoidzie, 164
 napis płynnie zmieniający kolor, 157

napis pływający w dowolnym miejscu witryny, 153
 napis pływający w polu tekstowym, 149
 navigator, 29
 null, 16

O

obliczanie liczby dni między podanymi datami, 75
 obliczanie pozostałej liczby dni, 74
 obrazy, 203

- automatyczna zamiana obrazów, 205
- efekt obrazu pływającego po witrynie, 226
- galeria obrazów z podpisami, 237
- logo stale widoczne w wybranym miejscu strony, 223
- lupa, 220
- ładowanie z paskiem postępu (I), 230
- ładowanie z paskiem postępu (II), 234
- pokaz slajdów, 212
- powiększanie fragmentów obrazu, 220
- przesuwanie obrazu po stronie, 207
- rozmiary, 219
- skalowanie za pomocą myszy, 210
- wyszukiwanie obrazów po opisie, 240
- wyświetlanie na nowej warstwie
 - przykrywającej zawartość strony, 216
- wyświetlanie w nowym oknie, 218
- zmiana obrazu po najechaniu myszą, 203
- zmiana rozmiarów z podaniem nowych wartości, 208

obsługa formularzy, 107
 obsługa zdarzeń, 20
 odliczanie czasu, 39

- czas do zadanej daty, 74
- podany czas, 70

odmierzanie czasu, 68
 odnośniki, 193

- decydowanie miejsca otwierania, 201
- dodatkowy opis, 199
- element strony jako odnośnik, 195
- potwierdzenie, 193
- symulacja odnośników, 195
- wybór z listy rozwijanej (automatyczny), 198
- wybór z listy rozwijanej (manualny), 196

ograniczenie liczby znaków wpisywanych do rozszerzonego pola tekstowego, 128
 okna, 11

- otwarcie okna o zadanych parametrach, 11
- parametry, 12
- przesuwanie za pomocą myszy, 180
- właściwości określające wygląd, 12
- zamykanie, 18

okna dialogowe, 18
 modalne, 33

okno potwierdzania z odliczaniem, 135
 onblur, 130
 onclick, 13, 147, 196, 197, 271
 onfocus, 130
 onload, 205, 221
 onmousedown, 181, 183
 onmouseout, 27, 200, 204, 261
 onmouseover, 27, 200, 204, 252, 261
 opacity, 23, 179, 226, 259
 open(), 11, 17, 219
 opis, 174
 opis przesuwany za pomocą myszy, 180
 Option, 286
 otwieranie okna, 219

- okno o zadanych parametrach, 11

overflow, 174, 186, 256

P

padding, 247
 pageX, 182
 pageY, 182
 parametry okna, 12
 parentNode, 282
 parseInt(), 17, 73, 129, 142, 239
 pasek nawigacyjny, 245
 pasek postępu, 230

- czasowy, 143
- zdarzeniowy, 140

pasek stanu, 139, 196
 pasek tytułowy, 138
 płynna zmiana koloru, 157
 pływające napisy, 149
 podpowiedzi, 171
 podświetlanie komórki tabeli lub innego elementu witryny

- CSS, 26
- JavaScript, 27

podświetlanie wszystkich wystąpień poszukiwanego ciągu znaków, 118
 pojawiająca się podpowiedź, 178
 pokaz slajdów, 212
 pole tekstowe, 109

- pływający napis, 149

pole tekstowe automatycznie zmieniające wielkość, 126
 pole wyboru, 17
 position, 173, 224, 253
 position:absolute, 36, 186, 262
 potwierdzanie operacji przez użytkownika, 133
 powiązane listy rozwijane, 282
 powiązane opcje, 275
 powiązane opcje wyboru, 288

powiększanie
 fragment obrazu, 220
 tekst, 99
 poziome menu rozwijane, 278
 pozycjonowanie bezwzględne, 173, 186
 printBrowserType(), 30
 procedury obsługi zdarzeń, 20
 prompt(), 133
 przeciągnij i upuść, 180
 przeglądarki, 8
 przejście, 190
 przełączane menu wykluczające, 266
 przełączane menu z animacją, 267
 przełączane menu z niezależnymi pozycjami, 263
 przenikanie, 190
 menu, 258
 przesuwalny kalendarz, 83
 przesuwanie obrazu po stronie, 207
 przesuwanie tekstu, 153
 przesuwany boks menu, 270
 przeszukiwanie tekstu, 114
 przewijanie
 tekst, 150
 treść strony, 21
 przezroczystość, 23, 179
 przypisanie wybranemu elementowi stylu
 wprowadzonego przez użytkownika, 102

R

ramki, 40, 41
 random(), 170
 RegExp, 117
 reklamy, 184, 185
 rel, 99
 removeChild(), 217
 replace(), 115, 117
 resizable, 12
 rgb(), 159
 rozmiary obrazu, 219
 rozpoznanie typu przeglądarki, 29
 rozszerzone pole tekstowe, 128
 ograniczenie wpisywanej liczby znaków, 128

S

scroll(), 25
 scrollbars, 12
 scrollBy(), 21
 selectedIndex, 197
 selectstart, 60
 send(), 56
 setAttribute(), 99
 setCookie(), 58

setInterval(), 69, 73, 145, 152
 setTimeout(), 39, 66, 73
 SHA1, 54, 55
 showModal(), 137
 showTooltip(), 172
 skalowanie obrazu za pomocą myszy, 210
 skrolowanie, 149, 186
 splash screen, 38
 split(), 72
 sprawdzanie poprawności danych, 107
 src, 204, 217
 stan pól wyboru, 17
 status, 12, 140
 stoper, 68
 strona powitalna, 38
 strona tytułowa, 38
 strona zależna od typu przeglądarki, 32
 style, 157
 style CSS, 26, 95
 czcionki, 101
 dynamiczna zmiana stylu strony, 95
 font-size, 101
 font-style, 101
 kompozycja stylu z wybranych elementów, 104
 pasek nawigacyjny, 245
 powiększanie tekstu, 99
 przypisanie wybranemu elementowi stylu
 wprowadzonego przez użytkownika, 102
 zależność od przeglądarki, 98
 zmniejszanie tekstu, 99
 submit, 109
 substring(), 129
 switch...case, 81
 symulacja kart, 43
 kod JavaScript, 46
 style CSS, 45
 symulacja odnośników, 195
 symulacja pisania na klawiaturze, 168
 system logowania, 53
 system menu, 245

T

tabele, 76
 target, 202
 text, 13
 text-align, 52, 240, 247, 253
 text-decoration, 247
 this, 181
 this.value, 125
 timer, 73, 159
 title, 138
 toGMTString(), 58
 toLowerCase(), 30, 242, 243, 278

toolbar, 12
top, 12, 184
treść, 11
 przedstawianie w symulowanych kartach, 43
 zależność od daty, 91
 zależność od godziny (pory dnia), 89
 zmiana o określonej godzinie, 92
treść wyświetlana w zagnieżdżonym oknie
 temat wybierany kliknięciem, 41
 wybór tematu z listy, 39
tworzenie elementów HTML, 99
type, 99

U

ukrywanie elementów, 36
undefined, 16
URL, 41
userAgent, 29

V

value, 17

W

walidacja formularzy, 107
warstwa przesuwana za pomocą myszy, 180
warstwy, 36
wartości losowe, 170
wersje strony
 zależność od dnia tygodnia, 91
 zależność od pory dnia, 90
weryfikacja danych, 126
 adres e-mail, 146
 z uwzględnieniem formatu danych
 i wyróżnianiem błędnych pól, 111
weryfikacja kodu dostępu, 55
which, 123
wiadomości, 184
width, 12, 37, 220
window.event, 278
window.open(), 17
window.status, 140, 196
wpisywanie danych w formularzu, 120
wskazywanie daty, 84
wybór daty, 85
wybór odnośnika z listy rozwijanej
 automatyczny, 198
 manualny, 196
wyłączanie timera, 159
wyrażenia regularne, 113, 115
 adres e-mail, 147

wyrównywanie
 elementy, 109
 tekst, 52
wyróżnianie
 aktywny element formularza, 129
 wskazane komórki tabeli, 26
wyskakująca odpowiedź, 171
wysuwana odpowiedź, 174
wysuwane menu poziome, 256
wyszukiwanie
 fraz w tekście strony, 114
 obrazy po opisie, 240
wyświetlanie
 obrazu na nowej warstwie przykrywającej
 zawartość strony, 216
 obraz w nowym oknie, 218
 reklamy, 185
 warstwy, 36

X

XHTML, 8
XMLHttpRequest, 56

Z

zagnieżdżone ramki, 40
zamykanie okna przeglądarki, 18
zautomatyzowana zamiana obrazów, 205
zdarzenia, 20, 278
 blur, 130
 click, 24, 273
 contextmenu, 273
 focus, 130
 keydown, 121, 122
 keypress, 121
 keyup, 120, 121, 125
 load, 58, 220
 mousedown, 60
 mouseout, 24, 27, 172, 203
 mouseover, 24, 27, 172, 203
 onmouseout, 27
 onmouseover, 27
 selectstart, 60
zegar cyfrowy, 65
z-index, 33, 36, 253
zliczanie liczby odwiedzin, 56
zmiana
 obraz po najechaniu myszą, 203
 rozmiar obrazu z podaniem nowych wartości, 208
 rozmiar pola tekstowego, 127
 styl komórek tabeli, 28
 style CSS, 96
zmiennne globalne, 170
zmniejszanie tekstu, 99

PROGRAM PARTNERSKI

GRUPY WYDAWNICZEJ HELION



1. ZAREJESTRUJ SIĘ
2. PREZENTUJ KSIĄŻKI
3. ZBIERAJ PROWIZJĘ

Zmień swoją stronę WWW
w działający bankomat!

Dowiedz się więcej i dołącz już dzisiaj!

<http://program-partnerski.helion.pl>

GRUPA WYDAWNICZA

 **Helion SA**