helion.pl

Idź do

- Spis treści
- Przykładowy rozdział

Katalog książek

- Katalog online
- Zamów drukowany katalog

Twój koszyk

- Dodaj do koszyka

Cennik i informacje

- Zamów informacje o nowościach
- Zamów cennik

Czytelnia

- Fragmenty książek online

Ajax, JavaScript i PHP. Intensywny trening



Autor: Phil Ballard, Michael Moncur
Tłumaczenie: Andrzej Grażyński
ISBN: 978-83-246-2072-2
Tytuł oryginału: Sams Teach Yourself Ajax, JavaScript, and PHP All in One (Sams Teach Yourself)
Format: 170x230, stron: 432
Zawiera CD-ROM

Naucz się łączyć największe zalety języków oraz technik programowania i twórz interaktywne strony internetowe

- Jak unikać typowych błędów i sprawnie rozwiązywać problemy programistyczne?
- Jak połączyć zalety HTML, XML i PHP dla uzyskania pożądanych efektów?
- Jak tworzyć aplikacje wyposażone w aktywny interfejs użytkownika?

Statyczne strony WWW to dziś już przeszłość. Powszechnie dostępne narzędzia dają programistom prawie nieograniczone możliwości w zakresie tworzenia interaktywnych witryn internetowych, wzbogaconych o najróżniejsze efekty wizualne, animacje oraz wbudowane narzędzia pomocnicze. Największą popularność zdobyły sobie narzędzia z kategorii open source – z powodu ich minimalnego kosztu oraz niezwykle dużych zasobów, dostępnych za pośrednictwem Internetu. Z tej książki dowiesz się, jak tworzyć bogate i interaktywne strony WWW, łącząc rozmaite techniki i korzystając z różnych języków.

Książka „Ajax, JavaScript i PHP. Intensywny trening" poprowadzi Cię krok po kroku po podstawowych zasadach programowania w językach JavaScript, PHP i HTML oraz technologiach programowania. Dzięki temu podręcznikowi dowiesz się m.in., jak wykorzystywać dostępne biblioteki ajaksowe do implementowania i ulepszania podstawowych mechanizmów aplikacji. Szybko nauczysz się tworzyć interaktywne strony WWW, zarówno za pomocą technologii serwerowych, jak i technik oraz narzędzi umiejscowionych po stronie klienckiej, a także ich kombinacji.

- Tworzenie stron WWW w języku HTML
- Stylizacja stron za pomocą arkuszy CSS
- Tworzenie skryptów w języku JavaScript
- Wbudowanie skryptu w stronę WWW
- Obiektowy model dokumentu (DOM)
- Obiekty String
- Instrukcje warunkowe i pętle
- Funkcje wbudowane i biblioteki
- Konstruowanie aplikacji ajaksowych
- Zmienne w języku PHP
- Funkcje liczbowe, łańcuchy i tabele
- Kontrola przepływu sterowania
- Usługi webowe oraz protokoły REST i SOAP

Łącz, kompiluj, dobieraj – niech Twoje strony WWW zrobią wrażenie!

Kontakt

Helion SA
ul. Kościuszki 1c
44-100 Gliwice
tel. 032 230 98 63
e-mail: helion@helion.pl
© Helion 1991-2008

Spis treści

Wstęp **13**

Część I Dla przypomnienia...

Rozdział 1. Funkcjonowanie sieci WWW **21**

Powstanie internetu ... 21
Sieć WWW ... 22
Podstawy protokołu HTTP ... 27
Żądania i odpowiedzi protokołu HTTP ... 28
Formularze HTML ... 31
Podsumowanie ... 34

Rozdział 2. Tworzenie stron WWW w języku HTML i ich stylizacja za pomocą arkuszy CSS **35**

Podstawy języka HTML ... 35
Elementy strony WWW tworzonej w języku HTML ... 38
Bardziej skomplikowana strona WWW ... 44
Użyteczne znaczniki HTML ... 45
Definiowanie własnego stylu dokumentu ... 46
Definiowanie reguł stylistycznych ... 48
Stylizacja elementów grupowanych w klasy ... 48
Wiązanie reguł stylistycznych z dokumentami ... 50
Stylizowanie tekstu ... 53
Kreślenie linii poziomych ... 58
Podsumowanie ... 60

Rozdział 3. Anatomia aplikacji ajaksowej **61**

Zapotrzebowanie na Ajax ... 61
Ajax i interakcja ... 64
Z czego składa się Ajax? ... 65
Jak to wszystko razem funkcjonuje? ... 68
Podsumowanie ... 69

Część II Wprowadzenie do programowania w języku JavaScript

Rozdział 4. Tworzenie prostych skryptów w języku JavaScript 73
Narzędzia do tworzenia skryptów ... 73
Wyświetlanie bieżącego czasu za pomocą JavaScriptu ... 74
Zaczynamy tworzenie skryptu ... 75
Dodawanie instrukcji JavaScriptu ... 75
Wyświetlanie informacji ... 77
Wbudowywanie skryptu w stronę WWW ... 77
Testowanie skryptu ... 78
Reguły składniowe języka JavaScript ... 88
Komentarze ... 90
Sprawdzone i zalecane praktyki programistyczne ... 91
Podsumowanie ... 93

Rozdział 5. Obiektowy model dokumentu (DOM) 95
Obiekty ... 95
Podstawy modelu DOM ... 97
Programowanie obsługi dokumentów ... 99
Dostęp do historii przeglądanych stron ... 102
Obiekt location ... 103
Podsumowanie ... 106

Rozdział 6. Zmienne, łańcuchy i tablice JavaScriptu 107
Wykorzystywanie zmiennych ... 107
Operatory i wyrażenia ... 111
Typy danych JavaScriptu ... 113
Konwersje typów danych ... 114
Obiekty String ... 115
Wykorzystywanie podłańcuchów ... 119
Tablice liczbowe ... 123
Tablice łańcuchów ... 124
Sortowanie tablic liczbowych ... 126
Podsumowanie ... 129

Rozdział 7. Funkcje i obiekty JavaScriptu 131
Wykorzystywanie funkcji ... 131
Obiekty ... 137

Obiekty jako narzędzia upraszczające kodowanie ... 139
Rozszerzanie definicji obiektów wbudowanych ... 142
Podsumowanie ... 146

Rozdział 8. Instrukcje warunkowe i pętle 149

Instrukcja if ... 150
Wyrażenia uwarunkowane ... 154
Testowanie wielu warunków ... 155
Badanie wielu warunków za pomocą instrukcji switch ... 158
Pętle for ... 160
Pętla while ... 162
Pętla do...while ... 163
Wykorzystywanie pętli ... 163
Iterowanie po zestawie właściwości obiektu ... 165
Podsumowanie ... 170

Rozdział 9. Funkcje wbudowane i biblioteki 171

Obiekt Math ... 171
Przykład zastosowania — generowanie liczb pseudolosowych ... 173
Instrukcja wiążąca — słowo kluczowe with ... 176
Obsługa daty i czasu ... 177
Wykorzystywanie bibliotek niezależnych producentów ... 180
Inne biblioteki ... 182
Podsumowanie ... 182

Część III Wprowadzenie do Ajaksa

Rozdział 10. Serce Ajaksa — obiekt XMLHTTPRequest 185

Czym jest obiekt XMLHTTPRequest? ... 185
Tworzenie instancji obiektu XMLHTTPRequest ... 186
Podsumowanie ... 192

Rozdział 11. Komunikacja z serwerem 193

Wysyłanie żądania do serwera ... 193
Monitorowanie statusu żądania ... 198
Funkcja zwrotna ... 199
Podsumowanie ... 202

Rozdział 12. Przetwarzanie otrzymanych danych 203
Właściwości responseText i responseXML 203
Parsowanie właściwości responseXML 207
Sprzężenie zwrotne 208
Podsumowanie 211
Rozdział 13. Pierwsza aplikacja ajaksowa 213
Konstruowanie aplikacji ajaksowych 213
Dokument HTML 214
Skrypt 215
Wszystko razem... 219
Podsumowanie 223

Część IV Programowanie skryptów serwera w języku PHP

Rozdział 14. Poznajemy PHP 227
Podstawy PHP 227
Pierwszy skrypt w PHP 230
Podsumowanie 235
Rozdział 15. Zmienne 237
Zmienne w języku PHP 237
Typy danych 239
Liczby 242
Typy danych liczbowych 244
Funkcje liczbowe 244
Łańcuchy 246
Formatowanie łańcuchów 248
Funkcje łańcuchowe 250
Tablice 252
Funkcje tablicowe 255
Obliczenia na datach 256
Formatowanie dat 258
Podsumowanie 261
Rozdział 16. Kontrola przepływu sterowania 263
Instrukcje warunkowe 263
Pętle 270
Podsumowanie 272

Rozdział 17. Funkcje 273
Wykorzystywanie funkcji ... 273
Argumenty wywołania i zwracana wartość ... 275
Biblioteki funkcji ... 279
Podsumowanie ... 281

Rozdział 18. Klasy 283
Programowanie zorientowane obiektowo na gruncie PHP ... 283
Czym jest klasa? ... 284
Tworzenie i wykorzystywanie obiektów ... 285
Podsumowanie ... 290

Część V Zaawansowane technologie ajaksowe

Rozdział 19. Wykorzystywanie właściwości responseText 293
responseText — tylko tekst i aż tekst ... 293
Podsumowanie ... 300

Rozdział 20. AHAH — asynchroniczny HTML i HTTP 301
Co to jest AHAH? ... 301
Biblioteki dla AHAH ... 302
Wykorzystywanie biblioteki myAHAHlib.js ... 305
Podsumowanie ... 312

Rozdział 21. Wykorzystywanie właściwości responseXML 313
Ajax to także „x” ... 313
Właściwość responseXML ... 314
Przykład zastosowania — czytnik RSS nagłówków wiadomości ... 317
Podsumowanie ... 326

Rozdział 22. Usługi webowe oraz protokoły REST i SOAP 327
Koncepcja usługi webowej ... 327
REST — Representational State Transfer ... 328
Przykład usługi REST — implementacja ... 330
Podstawy protokołu ... 336
SOAP i aplikacje ajaksowe ... 338
SOAP i REST — porównanie ... 339
Podsumowanie ... 340

Rozdział 23. Biblioteka JavaScriptu dla aplikacji ajaksowych 341
Biblioteka dla aplikacji ajaksowych 341
Dla przypomnienia — myAHAHlib.js 342
Implementowanie biblioteki 343
Korzystanie z biblioteki 347
Podsumowanie 351

Rozdział 24. Pułapki Ajaksa 353
Nawigowanie po historii stron 354
Zakładki i hiperłącza 355
Komunikacja z użytkownikiem 356
Miękkie lądowanie 356
Widoczność dla wyszukiwarek 357
Wyróżnianie elementów aktywnych 358
Ajax — niekoniecznie dobry na wszystko 358
Bezpieczeństwo 359
Kodowanie międzyplatformowe 360
Ajax nie poprawi złego projektu 360
Kilka pułapek programistycznych 361
Podsumowanie 362

Część VI Narzędzia i zasoby Ajaksa

Rozdział 25. Biblioteka prototype.js 365
prototype.js — pierwsze spotkanie 365
Obiekt Ajax — otoczka obiektu XMLHTTPRequest 368
Przykład zastosowania — czytnik notowań giełdowych 371
Podsumowanie 374

Rozdział 26. Rico 375
Biblioteka Rico 375
Budowanie interfejsu użytkownika 380
Podsumowanie 385

Rozdział 27. Script.aculo.us 387
Pobieranie biblioteki 387
Dołączanie plików 388
Wywoływanie efektów specjalnych 388
Tworzenie skryptu 388
Podsumowanie 391

Rozdział 28. XOAD **393**
O bibliotece XOAD ... 393
XOAD_HTML ... 397
Zaawansowane programowanie z użyciem biblioteki XOAD ... 400
Podsumowanie ... 401

Część VII Dodatki

A JavaScript, PHP i Ajax w internecie **405**

B Słownik **409**

Skorowidz **415**

Część II

Wprowadzenie do programowania w języku JavaScript

Rozdział 4. Tworzenie prostych skryptów w języku JavaScript **73**
Rozdział 5. Obiektowy model dokumentu (DOM) **95**
Rozdział 6. Zmienne, łańcuchy i tablice JavaScriptu **107**
Rozdział 7. Funkcje i obiekty JavaScriptu **131**
Rozdział 8. Instrukcje warunkowe i pętle **149**
Rozdział 9. Funkcje wbudowane i biblioteki **171**

Rozdział 4

Tworzenie prostych skryptów w języku JavaScript

W rozdziale:

- **narzędzia do tworzenia skryptów,**
- **wyświetlanie aktualnego czasu za pomocą JavaScriptu,**
- **rozpoczynanie tworzenia skryptu,**
- **dodawanie instrukcji JavaScriptu,**
- **wyświetlanie informacji w ramach skryptu,**
- **integrowanie skryptu ze stroną WWW,**
- **testowanie skryptu,**
- **reguły składniowe języka JavaScript,**
- **znaczenie i wykorzystywanie komentarzy,**
- **użyteczne wskazówki dla programistów.**

Jak już wspominaliśmy, język JavaScript jest jednym ze środków tworzenia skryptów po stronie klienta WWW. Instrukcje JavaScriptu mogą być wbudowywane bezpośrednio w kod strony WWW i wykonywane podczas wyświetlania tej strony.

W niniejszym rozdziale przedstawimy tworzenie prostego skryptu oraz jego edytowanie i testowanie za pomocą przeglądarki. Przy okazji omówimy podstawowe zadania związane z tworzeniem i wykorzystywaniem skryptów.

Narzędzia do tworzenia skryptów

W przeciwieństwie do wielu języków programowania, tworzenie skryptów w języku JavaScript nie wymaga specjalnego oprogramowania — wystarczające okazują się narzędzia oferowane przez system operacyjny.

Edytory tekstu

Pierwszym narzędziem niezbędnym do tworzenia skryptów jest *edytor tekstu*. Ponieważ kod skryptu jest de facto „kawałkiem" tekstu, zazwyczaj wbudowywanym w kod dokumentów HTML, wystarczającym do tego celu może być każdy edytor zdolny do tworzenia „czystych" plików ASCII, na przykład popularny windowsowy Notatnik. „Przydatny" nie oznacza jednak „najbardziej odpowiedni" — nasze uwagi z rozdziału 2. na temat edytorów programisty stosują się w całej pełni do JavaScriptu (a także do języka PHP, którym zajmiemy się w dalszej części książki).

Na CD-ROM-ie

Na CD-ROM-ie dołączonym do niniejszej książki znajduje się edytor *jEdit* w wersjach dla Javy, Macintosha i Windows. Edytor ten znakomicie nadaje się do tworzenia kodu w języku JavaScript.

Przeglądarki

Dwie następne rzeczy, jakie potrzebne są do posługiwania się JavaScriptem, to przeglądarka WWW i komputer, na którym można ją uruchomić. Zalecanymi przez nas przeglądarkami są Firefox i Internet Explorer (możliwie w najnowszych wersjach), które można pobrać ze stron, odpowiednio: *http://www.mozilla-europe.org/pl/firefox/* i *http://www.microsoft.com*.

Jako absolutne minimum wymagamy jednej z przeglądarek: Firefox 1.0, Netscape 7 lub Internet Explorera 6; zalecamy jednak testowanie skryptów w środowisku jednej z ostatnich wersji Firefoksa i Internet Explorera, jako używanych najczęściej.

Uwaga

Do opublikowania stworzonego skryptu w internecie potrzebujemy jeszcze jednej rzeczy — dostępu do serwera WWW. Większość skryptów, które prezentujemy w niniejszej książce, można jednak uruchomić bezpośrednio z dysku w lokalnym komputerze.

Wyświetlanie bieżącego czasu za pomocą JavaScriptu

Najczęściej prezentowane przykłady wykorzystywania JavaScriptu związane są z wyświetlaniem daty i czasu. Ponieważ skrypty JavaScriptu wykonywane są w środowisku przeglądarki WWW, wyświetlane są data i czas obowiązujące

w strefie czasowej użytkownika, jednakże nietrudno przeliczyć je (oczywiście z pomocą JavaScriptu) na czas „uniwersalny" (UTC).

Uwaga

Czas uniwersalny (UTC — *Universal Time, Coordinated*) to wzorcowy czas mierzony za pomocą zegarów atomowych, wzorowany na czasie GMT (*Greenwich Mean Time*) obowiązującym w strefie południka zerowego, przechodzącego przez przedmieście Londynu, Greenwich.

W charakterze przykładu wprowadzającego Czytelnika w programowanie w języku JavaScript zaprezentujemy wyświetlanie daty i czasu jako elementu strony WWW.

Zaczynamy tworzenie skryptu

Podobnie jak wszystkie inne skrypty, nasz przykładowy skrypt ograniczony będzie znacznikami `<script>...</script>`.

Ostrzeżenie

Między znacznikami `<script>...</script>` mogą pojawić się wyłącznie poprawne instrukcje JavaScriptu. W przypadku napotkania konstrukcji nie będącej poprawną instrukcją skryptu przeglądarka wyświetla odpowiedni komunikat.

Inicjując nowy plik w ulubionym edytorze tekstu, napiszmy więc pierwszy wiersz kodu:

```
<script LANGUAGE="JavaScript" type="text/javascript"> </script>
```

Ponieważ nie będziemy wykorzystywać żadnych nowych cech JavaScriptu 1.1 i wersji późniejszych, nie określiliśmy konkretnej wersji w znaczniku `<script>`. Nasz skrypt powinien być dzięki temu poprawnie interpretowany nawet przez przeglądarki tak archaiczne, jak Netscape 2.0 czy Internet Explorer 3.0.

Dodawanie instrukcji JavaScriptu

Zadaniem naszego skryptu jest określenie i wyświetlenie bieżącej daty i czasu — lokalnego i uniwersalnego. Na szczęście większość potrzebnych do tego mechanizmów — w szczególności konwersja między formatami dat — wbudowane są w interpreter JavaScriptu.

Zmienne jako magazyny danych

Aby wyświetlić bieżącą datę i czas, musimy najpierw niezbędną do tego informację, po jej odczytaniu, przechować pod postacią *zmiennej*. O zmiennych będziemy szerzej pisać w rozdziale 6., w tej chwili potraktujemy je jako media do przechowywania wartości — liczb, tekstu czy (jak w obecnym przykładzie) daty i czasu.

Dodajmy więc do naszego kodu pierwszą instrukcję; zwróćmy uwagę na wielkość liter — w języku JavaScript jest ona istotna w poleceniach i nazwach zmiennych.

```
now = new Date();
```

Wskutek wykonania powyższej instrukcji utworzona zostanie nowa zmienna o nazwie `now`, a zmiennej tej przypisana zostanie wartość reprezentująca bieżące wskazanie daty i czasu. Wskazanie to udostępniane jest przez wbudowany obiekt JavaScriptu `Date`, oferujący wiele wygodnych mechanizmów do obsługi daty i czasu. Powrócimy do niego w rozdziale 9.

Uwaga
Uwaga

Zwróć uwagę na średnik, stanowiący dla interpretera informację, że skończyła się instrukcja. Średniki są w języku JavaScript opcjonalne, jednakże ich stosowanie pozwala uniknąć pewnych pospolitych błędów. Dla przejrzystości będziemy je więc stosować konsekwentnie we wszystkich przykładach niniejszej książki.

Obliczanie żądanych wielkości

Zmienne JavaScriptu wewnętrznie przechowują wskazanie daty i czasu jako liczbę milisekund, które upłynęły od północy rozpoczynającej dzień 1 stycznia 1970 roku. Tę mało wygodną w praktyce wartość można jednak łatwo przekonwertować na kilka bardziej użytecznych formatów — co wykonywane jest wewnętrznie przez JavaScript i w szczegóły czego nie będziemy tutaj wnikać.

Dodajmy w związku z tym do naszego skryptu dwie poniższe instrukcje:

```
localtime = now.toString();
utctime = now.toGMTString();
```

W wyniku ich wykonania utworzone zostaną dwie kolejne zmienne — `localtime` i `utctime` — zawierające tekstowe postaci czasu, odpowiednio: lokalnego i uniwersalnego, odpowiadającego wartości przechowywanej w zmiennej `now`.

Uwaga
Uwaga

Zmienne `localtime` i `utctime` przechowują tekst (na przykład „Wed, 12 Nov 2008 20:23:21 UTC") i w terminologii programistycznej nazywane są *łańcuchami* (ang. *strings*). Łańcuchami tekstowymi zajmiemy się bardziej szczegółowo w rozdziale 6.

Wyświetlanie informacji

Zawartość przechowywana w zmiennych `localtime` i `utctime` nie byłaby wiele warta, gdyby użytkownik nie mógł zobaczyć jej w zrozumiałej dla siebie postaci. JavaScript oferuje wiele środków służących wyświetlaniu informacji — jednym z najprostszych jest instrukcja `document.write`.

Za pomocą tej instrukcji można wyświetlać tekst, liczby i w ogóle wszelką zawartość zmiennych. Ponieważ skrypt stanowi część strony WWW, owo wyświetlenie dokonywać się będzie w ramach tejże strony. Aby je zrealizować, dodajmy następujące instrukcje bezpośrednio przez zamykającym znacznikiem `</script>`:

```
document.write("<b>Czas lokalny:</b> " + localtime + "<br>");
document.write("<b>Czas uniwersalny:</b> " + utctime);
```

W rezultacie ujrzymy na ekranie wartość czasu lokalnego i uniwersalnego, wraz z odpowiednim komentarzem.

Zwróćmy uwagę na znaczniki HTML — w szczególności na znacznik `<b>` — pojawiające się w tekście przeznaczonym do wyświetlenia. Ponieważ, jak wspominaliśmy, JavaScript wyświetli ów tekst w ramach strony WWW, przeto musi mieć ona postać zgodną z językiem HTML. I tak, znacznik `<br>` w pierwszej instrukcji spowoduje, że druga instrukcja wyświetlać będzie zawartość od nowego wiersza, zaś znaczniki `<b>` i `</b>` spowodują pogrubienie tekstu, który ograniczają.

Uwaga

Zwróćmy uwagę na operatory „+” w prezentowanych instrukcjach. Realizują one „dodawanie” do siebie fragmentów tekstu, czyli składanie (konkatenowanie) kilku łańcuchów w pojedynczy łańcuch przeznaczony do wyświetlenia. Gdyby operator „+” pojawił się między liczbami, oznaczałby ich *arytmetyczne* dodawanie.

Wbudowywanie skryptu w stronę WWW

Skompletowaliśmy już skrypt obliczający żądane wartości i wyświetlający je w czytelnej formie. Jego treść widoczna jest na listingu 4.1.

Listing 4.1. Skrypt wyświetlający bieżącą datę i czas

```
<script language="JavaScript" type="text/javascript">
now = new Date();
```

```
localtime = now.toString();
utctime = now.toGMTString();
document.write("<b>Czas lokalny:</b> " + localtime + "<BR>");
document.write("<b>Czas uniwersalny:</b> " + utctime);
</script>
```

Aby zrobić z powyższego skryptu użytek, musimy go teraz wbudować w kod strony WWW. Najprościej można to zrobić, używając jedynie niezbędnych znaczników <html>, <head> i <body> (wraz z ich zamykającymi odpowiednikami) i (dla elegancji) nagłówka <h1>. Otrzymany w ten sposób kod prostej strony WWW widoczny jest na listingu 4.2.

Listing 4.2. Prosta strona WWW, zawierająca skrypt wyświetlający bieżącą datę i czas

```
<html>
<head><title>Wyświetlanie daty i czasu</title></head>
<body>
<h1>Bieżąca data i czas</h1>
<p>
<script language="JavaScript" type="text/javascript">
now = new Date();
localtime = now.toString();
utctime = now.toGMTString();
document.write("<b>Czas lokalny:</b> " + localtime + "<BR>");
document.write("<b>Czas uniwersalny:</b> " + utctime);
</script>
</p>
</body>
</html>
```

Tak skonstruowany kod możemy teraz zapisać w pliku z rozszerzeniem *.htm* lub *.html*.

Uwaga

> Notatnik — jak i kilka innych edytorów tekstu dostępnych w systemie Windows — opatruje tworzone pliki domyślnym rozszerzeniem *.txt*, należy więc w ich przypadku podać rozszerzenie (*.htm* lub *.html*) w sposób jawny.

Testowanie skryptu

Aby przetestować działanie utworzonego skryptu, należy uruchomić przeglądarkę WWW — Internet Explorer, Firefox, Operę lub inną wybraną przez użytkownika — i załadować do niej plik zawierający kod widoczny na listingu 4.2.

Klikając bezpośrednio plik z rozszerzeniem *.htm* lub *.html* reprezentowany w oknie Eksploratora Windows, spowodujemy wczytanie jego zawartości do (uruchomionej automatycznie) przeglądarki domyślnej. Można też załadować plik do uruchomionej już przeglądarki, wybierając odpowiednią opcję z jej menu głównego (na przykład *Plik/Otwórz*).

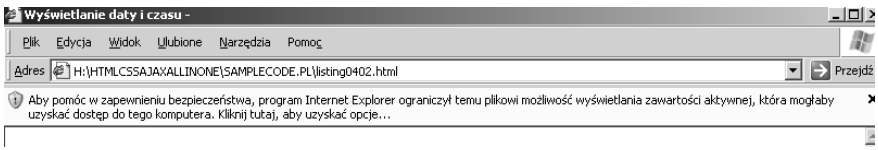
Jeżeli przy tworzeniu skryptu nie popełniliśmy błędów, powinniśmy ujrzeć stronę podobną do widocznej na rysunku 4.1, oczywiście z aktualnym wskazaniem czasu.



RYSUNEK 4.1. Wyświetlenie bieżącej daty i czasu w oknie Firefoksa

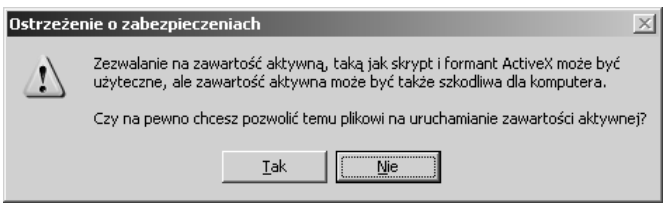
Ostrzeżenie

W przypadku Internet Explorera w wersji 6.0 lub nowszej możemy spotkać się z odmową wykonania skryptu, zależnie od obowiązujących ustawień zabezpieczenia — zamiast oczekiwanej zawartości ujrzymy wówczas ostrzegawczy komunikat:



Internet Explorer domyślnie zezwala bowiem na wykonywanie skryptów zawartych w dokumentach pobieranych ze zdalnych serwerów, lecz blokuje taką możliwość w odniesieniu do plików lokalnych.

Klikając ów komunikat, spowodujemy wyświetlenie menu kontekstowego, z którego należy wybrać opcję *Zezwalaj na zablokowaną zawartość…* po czym w wyświetlonym oknie komunikatu kliknąć przycisk *Tak*.



W efekcie powinniśmy ujrzeć oczekiwaną stronę.

Modyfikowanie skryptu

W zasadzie otrzymaliśmy to, czego oczekiwaliśmy, mimo jednak nienagannej merytorycznie treści jej forma wydaje się mało elegancka, chociażby w porównaniu z tym, co zobaczyć możemy na zwykłym zegarku. Rozbudujmy zatem nasz skrypt tak, by godziny, minuty i sekundy wyświetlane były osobno, oddzielone od siebie dwukropkami. Będą nam do tego potrzebne trzy dodatkowe zmienne: `hours`, `mins` i `secs`, przechowujące wartości, kolejno: godzin, minut i sekund, oraz mechanizm pozwalający wyodrębnić te wartości z bieżącego wskazania czasu. Ponownie JavaScript oferuje nam wszystko, co niezbędne:

```
hours = now.getHours();
mins = now.getMinutes();
secs = now.getSeconds();
```

Jak łatwo się domyślić, metody `getHours()`, `getMinutes()` i `getSeconds()`, traktując zawartość zmiennej `now` jako wskazanie czasu, wyodrębniają z niej żądane komponenty.

„Zegarowe" wyświetlenie bieżącego czasu ujmiemy — dla należytej wyrazistości — w ramy nagłówka `<h1>` i oczywiście nie zapomnimy o rozdzielających dwukropkach:

```
document.write("<h1>");
document.write(hours + ":" + mins + ":" + secs);
document.write("</h1>");
```

W pierwszej z powyższych instrukcji generowany jest znacznik otwierający wspomniany nagłówek. W drugiej poszczególne składniki wskazania czasu — godziny, minuty, sekundy i rozdzielające dwukropki — konkatenowane są do postaci pojedynczego łańcucha; trzecia instrukcja generuje zamykający znacznik `</h1>`.

Uzupełniając aktualną postać skryptu w opisany sposób, nadamy mu nową postać widoczną na listingu 4.3.

Listing 4.3. Wyświetlenie bieżącej daty i czasu w formie tekstowej i zegarowej

```
<html>
<head><title>Wyświetlanie daty i czasu</title></head>
<body>
<h1>Bieżąca data i czas</h1>
<p>
<script language="JavaScript">
now = new Date();
```

```
localtime = now.toString();
utctime = now.toGMTString();
document.write("<b>Czas lokalny:</b> " + localtime + "<BR>");
document.write("<b>Czas uniwersalny:</b> " + utctime);
hours = now.getHours();
mins = now.getMinutes();
secs = now.getSeconds();
document.write("<h1>");
document.write(hours + ":" + mins + ":" + secs);
document.write("</h1>");
</script>
</p>
</body>
</html>
```

Po zapisaniu powyższej zawartości w pliku z rozszerzeniem *.htm* lub *.html* i załadowaniu tego pliku do przeglądarki (w sposób uprzednio opisany) ujrzymy stronę podobną do tej z rysunku 4.2. Odświeżając tę stronę (np. za pomocą klawisza *F5*), powodować będziemy każdorazowo wykonanie skryptu i tym samym wyświetlenie aktualnego czasu[1].



RYSUNEK 4.2. Zmodyfikowane wyświetlanie daty i czasu, tym razem w oknie Internet Explorera

Uwaga

Mimo wszystko naszemu skryptowi brakuje jeszcze trochę do pełnej elegancji, głownie za sprawą ułomnego formatowania liczb — jednocyfrowe godziny i (lub) minuty *nie* są poprzedzane zerem. Ponadto zegar działa w trybie 24-godzinnym, z czego mogą być niezadowoleni użytkownicy preferujący wskazania 12-godzinne. W rozdziale 9. pokażemy, jak poradzić sobie z tymi problemami.

[1] W niektórych przeglądarkach, na przykład w Operze, możemy zażądać periodycznego odświeżania strony w sposób automatyczny (co sekundę lub co minutę), otrzymując namiastkę działającego zegarka — *przyp. tłum.*

Błędy w skrypcie

Każdy programista doskonale wie — bo przekonał się osobiście — że programowanie nieuchronnie związane jest z popełnianiem błędów. Nawet w przypadku tak prostych skryptów jak dotychczas prezentowane można się zwyczajnie pomylić podczas wpisywania instrukcji.

Aby zobaczyć, jak zachowa się przeglądarka podczas napotkania błędu w wykonywanym skrypcie, zmodyfikujmy jedną z instrukcji widocznych na listingu 4.3, celowo usuwając nawias zamykający parametr wywołania metody `document.write`:

```
document.write("<h1>";
```

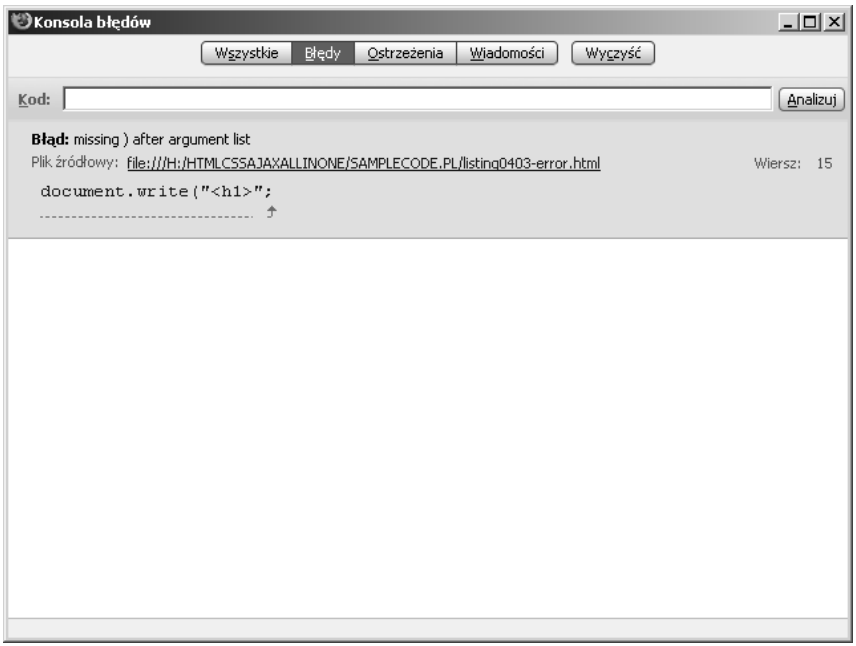
Zapiszmy tak zmodyfikowany kod do pliku i wczytajmy ów plik do przeglądarki. Zależnie od konkretnej przeglądarki, może wydarzyć się jedna z dwóch rzeczy: wyświetlony zostanie komunikat o błędzie albo wykonanie błędnego skryptu zostanie przerwane.

Jeśli wyświetlony zostanie komunikat o błędzie, będziemy na dobrej drodze do jego poprawienia, w tym przypadku do uzupełnienia brakującego nawiasu. Jeżeli jednak wspomniany komunikat nie pojawi się, będziemy zmuszeni samodzielnie znaleźć źródło błędu. W tym celu:

- W Firefoksie lub Netscape musimy wpisać **`javascript:`** w pole adresowe, w wyniku czego uruchomiona zostanie konsola JavaScriptu (możemy także z menu *Narzędzia* wybrać opcję *Konsola błędów*). Na rysunku 4.3 widoczny jest komunikat o brakującym nawiasie wyświetlany w oknie konsoli.
- W Internet Explorerze musimy natomiast wybrać z menu *Narzędzia* opcję *Opcje internetowe...*, w wyświetlonym oknie dialogowym przejść na stronę *Zaawansowane* i w liście opcji usunąć zaznaczenie opcji *Wyłącz debugowanie skryptu (Internet Explorer)* oraz zaznaczyć opcję *Wyświetl powiadomienie o każdym błędzie skryptu*.

Uwaga

Zwróćmy uwagę na pole *Kod* w oknie konsoli błędów na rysunku 4.3. Możemy wpisać weń dowolną instrukcję JavaScriptu i kliknąć przycisk *Analizuj*, w wyniku czego wspomniana instrukcja poddana zostanie natychmiastowemu wykonaniu analizie przez interpreter JavaScriptu. Daje to doskonałą okazję do przetestowania rozmaitych możliwości oferowanych przez JavaScript.



RYSUNEK 4.3. Komunikat o błędzie JavaScriptu wyświetlany w oknie konsoli

Komunikat w oknie konsoli na rysunku 4.3 jednoznacznie wskazuje przyczynę błędu — brakujący nawias — nie jest to jednak reguła i należy być świadomym, że (zwłaszcza w przypadku rozbudowanych skryptów i błędów o bardziej subtelnym charakterze) miejsce, w którym błąd jest sygnalizowany, może znajdować się daleko od miejsca skrywającego faktyczną przyczynę tego błędu.

Podczas gdy Internet Explorer reaguje *natychmiastowym* wyświetleniem okna dialogowego w przypadku wystąpienia błędu w wykonywanym skrypcie, konsola błędów Firefoksa wyświetla listę napotkanych błędów, dając tym samym okazję do zidentyfikowania „podejrzanych" instrukcji; co więcej, konsola ta daje możliwość natychmiastowej analizy dowolnej instrukcji (o czym wspominaliśmy przed chwilą). Stąd praktyczny wniosek, by zainstalować Firefox nawet na komputerze, którego użytkownicy posługują się Internet Explorerem — przyda się jako narzędzie do testowania i debugowania JavaScriptu.

Instrukcje

Instrukcje stanowią podstawowe jednostki programu w większości języków programowania, także w JavaScripcie. Ogólnie rzecz biorąc, instrukcja jest sekcją kodu, odpowiedzialną za wykonanie pewnej jednostkowej akcji. Przykładowo, każda z poniższych instrukcji powoduje wyodrębnienie

pojedynczego komponentu (godzin, minut lub sekund) ze wskazania czasu przechowywanego w zmiennej now:

```
hours = now.getHours();
mins = now.getMinutes();
secs = now.getSeconds();
```

Chociaż każda instrukcja JavaScriptu zapisywana jest w pojedynczym wierszu, nie jest to bezwzględnie wymagane (przynajmniej przez składnię języka) — można rozciągnąć pojedynczą instrukcję na kilka wierszy lub „upakować" kilka instrukcji w jednym.

Zwyczajowo *średnik* pełni rolę ogranicznika instrukcji, nie jest jednak wymagany, jeśli po instrukcji występuje znak nowego wiersza. Jeśli upakowujemy kilka instrukcji w jednym wierszu, muszą być one bezwzględnie rozdzielone średnikami.

Wiązanie funkcji z zadaniami

W naszym przykładowym skrypcie wielokrotnie wykorzystywaliśmy instrukcje zawierające fragmenty ujęte w nawiasy, na przykład

```
document.write("Testowanie.");
```

Taka instrukcja stanowi przykład wywołania *funkcji*. Każda funkcja dostarcza prostego sposobu na wykonanie pewnego zadania, na przykład wyświetlenia tekstu na stronie WWW. JavaScript oferuje użytkownikowi szeroką gamę „gotowych", czyli *wbudowanych* funkcji, czego przykłady przytoczyliśmy (i przytaczać będziemy jeszcze niejednokrotnie) w niniejszej książce.

Wywołanie funkcji wiąże się z przekazaniem *parametru* (wyrażenia występującego w nawiasach, poprzedzonych nazwą funkcji) oraz ze *zwróceniem wartości*, która przypisywana jest (jakiejś) zmiennej. Przykładowo, poniższa instrukcja wyświetla użytkownikowi zaproszenie do wprowadzenia pewnego tekstu, który następnie przypisany zostaje zmiennej o nazwie text:

```
text = prompt("Wprowadź jakiś tekst:");
```

Użytkownik może także definiować własne funkcje, co okazuje się użyteczne z co najmniej dwóch powodów. Po pierwsze, funkcje stanowią naturalny sposób *modularyzacji* kodu, czyli jego logicznego podziału na dobrze określone jednostki funkcjonalne, wskutek czego ów kod staje się czytelniejszy i bardziej zrozumiały. Po drugie — i ważniejsze — raz zaprogramowana, w postaci funkcji,

realizacja określonego zadania może być wielokrotnie wykorzystywana w kodzie programu przez wielokrotne wywołania tej funkcji. Dzięki temu użytkownik zostaje uwolniony od konstruowania identycznych fragmentów kodu.

> Definiowaniem funkcji, ich wywoływaniem i programowaniem zwracanych wartości zajmiemy się dokładniej w rozdziale 7.
>
> **Uwaga**

Zmienne

Zmienne stanowią rodzaj kontenerów do przechowywania wartości rozmaitych typów: tekstu, liczb, dat i innych danych. Wykonanie poniższej instrukcji powoduje utworzenie zmiennej o nazwie `fred` i przypisanie jej wartości liczbowej 27:

```
var fred = 27;
```

Zmiennymi JavaScriptu zajmiemy się dokładniej w rozdziale 6.

Instrukcje warunkowe

Mimo iż procedury zdarzeniowe (*event handlers*) odpowiedzialne są za prawidłowe reagowanie skryptu na poszczególne zdarzenia, użytkownik często sam musi badać spełnienie pewnych warunków, na przykład poprawności wprowadzonego przez użytkownika adresu e-mail.

W JavaScripcie temu celowi służą *instrukcje warunkowe* (*conditional statements*), których przykładem jest instrukcja `if`, na przykład taka:

```
if (count==1) alert ("Licznik osiągnął wartość 1.");
```

W powyższej instrukcji następuje porównanie wartości zmiennej `count` z liczbą 1 i jeśli okażą się one równe, wyświetlany jest stosowny komunikat. Podobne instrukcje warunkowe spotykać będziemy w większości skryptów prezentowanych w niniejszej książce.

> Instrukcje warunkowe opiszemy dokładniej w rozdziale 8., poświęconym kontroli przepływu sterowania w skryptach JavaScriptu.
>
> **Uwaga**

Pętle

Innym użytecznym mechanizmem JavaScriptu — obecnym zresztą w większości języków programowania — są *pętle* (*loops*), grupujące bloki instrukcji

wykonywane w sposób powtarzalny. Wykonanie poniższej pętli spowoduje dziesięciokrotne wyświetlenie alertu:

```
for (i=1; i<=10; i++) {
   alert("Tak, to jest kolejny alert!");
}
```

Instrukcja `for` to tylko jeden z przykładów pętli. Generalnie pętle przeznaczone są do realizacji tego, w czym komputery okazują się najlepsze — konsekwentnego, kontrolowanego powtarzania tych samych czynności. W typowych skryptach można napotkać wiele pętli realizujących zadania znacznie mniej banalne niż pętla prezentowana powyżej.

Uwaga

Pętle opiszemy dokładniej w rozdziale 8.

Obsługa zdarzeń

Nie wszystkie skrypty wbudowywane są w kod strony WWW przez umieszczenie ich między ogranicznikami `<script>...</script>`. Skrypt może bowiem pełnić także rolę *procedury obsługi zdarzenia* (*event handler*). Mimo skomplikowanej nazwy, jego rola jest oczywista — jest nią właściwe reagowanie na określone typy zdarzeń zachodzących w systemie.

Sama koncepcja zdarzenia nie jest obca nikomu w codziennym życiu, niekoniecznie mającym związek z komputerami. Zdarzenia mogą być oczekiwane, związane z kalendarzem, jak „Wizyta u dentysty" czy „Imieniny Karola", bądź nieoczekiwane — czego przykładem może być awaria kanalizacji, pożar, niespodziewana wizyta krewnych czy kontrola z urzędu skarbowego.

Większości zdarzeń — oczekiwanych lub nie — potrafimy stawić czoło, uruchamiając przygotowany a priori scenariusz, na przykład „Gdy nadejdą imieniny Karola, wyślij mu prezent" albo „Gdy zobaczysz (usłyszysz), że zbliżają się Twoi krewni, wyłącz wszystkie światła i udawaj, że nie ma Cię w domu".

Na gruncie JavaScriptu obsługa zdarzeń odbywa się na podobnej zasadzie — każda z procedur zdarzeniowych określa zachowanie się przeglądarki w przypadku wystąpienia określonego zdarzenia. Co prawda zdarzenia tego rodzaju — kliknięcie prawym przyciskiem myszy lub zakończenie wczytywania strony — są może mniej ekscytujące niż te zachodzące w codziennym życiu, niemniej jednak ich obsługa jest niezbędnym elementem funkcjonowania współczesnych przeglądarek i jednocześnie integralnym elementem JavaScriptu.

Większość zdarzeń, o których mowa, wywoływanych jest przez działania użytkownika (czego przykładem może być naciśnięcie klawisza czy kliknięcie), wskutek których przeglądarka porzuca na chwilę „normalną" pracę i przechodzi do wykonywania właściwej procedury zdarzeniowej. Niektóre zdarzenia — jak zakończenie wczytywania strony z serwera WWW — nie są inicjowane przez użytkownika w sposób bezpośredni.

Każda procedura zdarzeniowa skojarzona jest z określonym obiektem strony WWW i specyfikowana jest w znaczniku HTML otwierającym ów obiekt. Przykładowo, obrazki i hiperłącza tekstowe powiązane są ze zdarzeniem `onMouseOver` zachodzącym wówczas, gdy wskaźnik myszy pojawi się nad obiektem. Zdarzeniu temu można przypisać procedurę obsługi w następujący sposób:

```
<img src="button.gif" onMuseOver="highlight();">
```

Jak widać, scenariusz obsługi zdarzenia specyfikowany jest jako wartość atrybutu, którego nazwą jest nazwa zdarzenia. Jeżeli obsługa ta zaprogramowana jest w postaci funkcji, wspomniany scenariusz sprowadza się do wywołania tejże (jak w powyższym przykładzie) — to kolejna przesłanka przemawiająca za stosowaniem funkcji.

W punkcie „Przekonaj się sam…" przy końcu niniejszego rozdziału znajdzie Czytelnik konkretny przykład działającej procedury zdarzeniowej.

Kolejność wykonywania skryptów

Wnikliwy Czytelnik zorientował się już zapewne, że w daną stronę WWW może być wbudowanych wiele różnych skryptów — tych ograniczonych znacznikami `<script>…</script>`, tych pochodzących z zewnętrznych plików i tych pełniących rolę procedur zdarzeniowych. Interesujące staje się w tym momencie pytanie o to, *w jakiej kolejności* skrypty te będą wykonywane. Ta kwestia na szczęście rozwiązana została zgodnie z pewnymi logicznymi zasadami, mianowicie:

- skrypty typu `<script>`, należące do sekcji `<head>` — niezależnie od tego, czy wbudowane w kod, czy też importowane z zewnętrznych plików — wykonywane są w pierwszej kolejności. Ponieważ nie mogą one produkować zawartości wyświetlanej na stronie, stanowią idealne miejsce do definiowania funkcji wykorzystywanych w innych skryptach;
- skrypty typu `<script>` znajdujące się w ciele dokumentu (sekcji `<body>`) wykonywane są w drugiej kolejności, podczas wczytywania i wyświetlania

strony, w kolejności identycznej z kolejnością ich zdefiniowania w kodzie tej strony;

- skrypty pełniące rolę procedur zdarzeniowych uruchamiane są w momencie występowania odnośnych zdarzeń; przykładowo, skrypt obsługujący zdarzenie `onLoad` wykonywany jest w momencie rozpoczęcia wczytywania strony. Jako że sekcja `<head>` zostaje wczytana przed wygenerowaniem jakiegokolwiek zdarzenia, funkcje zdefiniowane w skryptach znajdujących się w tej sekcji mogą być wykorzystywane w procedurach zdarzeniowych.

Reguły składniowe języka JavaScript

JavaScript jest językiem stosunkowo nieskomplikowanym, co nie zmienia faktu, że zdefiniowany jest on na bazie ścisłych reguł syntaktycznych. Do reguł tych odwoływać się będziemy nieustannie w całej niniejszej książce, obecnie ograniczymy się jednak do przedstawienia tych najważniejszych, których przestrzeganie pozwoli uniknąć wielu powszechnie popełnianych błędów.

Wielkość liter

Niemal we wszystkich elementach JavaScriptu wielkość liter *ma* znaczenie — wielkie litery odróżniane są od swych małych odpowiedników. Należy zatem pamiętać o kilku zasadach wynikających bezpośrednio z tego faktu:

- słowa kluczowe JavaScriptu, jak `for` czy `if`, zawsze pisane są w całości małymi literami;
- nazwy obiektów wbudowanych, jak `Math` czy `Date`, są *kapitalizowane* — rozpoczynane są wielką literą, reszta pisana jest w całości małymi;
- nazwy obiektów DOM zwykle pisane są małymi literami, lecz w nazwach *metod* tych obiektów można mieszać litery małe z wielkimi, najczęściej w celu rozróżnienia fraz słowotwórczych[2] (z wyjątkiem pierwszej), jak w przypadku `toLowerCase` czy `getElementById`.

W przypadku wątpliwości związanych ze stosowaniem małych i wielkich liter najlepiej wzorować się na poprawnych rozwiązaniach, na przykład skryptach prezentowanych w niniejszej książce. Większość błędów wynikających z użycia

[2] Nazywa się to potocznie *notacją wielbłądzią* (ang. *camel notation*) — *przyp. tłum.*

litery niewłaściwej wielkości kwitowanych jest przez przeglądarki stosownymi komunikatami.

Nazwy zmiennych, obiektów i funkcji

Nadając nazwy definiowanym zmiennym, obiektom i funkcjom, można używać liter — małych i wielkich — cyfr i znaku podkreślenia. Każda z tych nazw może zaczynać się literą lub znakiem podkreślenia.

Wybór konkretnej konwencji w zakresie stosowania małych i wielkich liter jest sprawą użytkownika, który nie powinien jednak zapominać o tym, że JavaScript jest wrażliwy na wielkość liter: `score`, `Score` i `SCORE` to z punktu widzenia JavaScriptu trzy *różne* nazwy. Odwołując się do zdefiniowanej nazwy, należy zatem czynić to z wiernym zachowaniem wielkości liter.

Słowa zarezerwowane

Dowolność w zakresie wyboru nazw definiowanych przez użytkownika ograniczona jest regułą zabraniającą używania w tym celu *słów zarezerwowanych*. Należą do nich m.in. słowa kluczowe tworzące osnowę składniową języka (jak `if` czy `for`), nazwy obiektów DOM (jak `window` i `document`) oraz nazwy obiektów wbudowanych (na przykład `Math` i `Date`).

Spacjowanie

Znaki niewidoczne — spacje, tabulatory, znaki końca wiersza — zwane przez programistów „białymi znakami" (ang. *whitespaces*) są z punktu widzenia składni JavaScriptu nieistotne. Można zatem dowolnie umieszczać białe znaki między kolejnymi elementami składniowymi[3], łamać instrukcje między kilka wierszy itp. Stwarza to doskonałą okazję do nadawania tworzonym skryptom pożądanej czytelności.

---

[3] Nie są obojętne spacje występujące wewnątrz stałych tekstowych. O ile w instrukcji

```
text = prompt("Wprowadź jakiś tekst:");
```

spacje i tabulatory występujące przed otwierającym cudzysłowem (i po zamykającym cudzysłowie) są nieistotne, to umieszczone między cudzysłowami zostaną literalnie wypisane na ekranie — napis „Wprowadź jakiś tekst" różni się przecież od napisu „ Wprowadź jakiś tekst " — *przyp. tłum.*

Komentarze

Komentarze umożliwiają umieszczanie w kodzie skryptu tekstu ignorowanego w zupełności przez interpreter, lecz zawierającego informację istotną dla człowieka analizującego kod; są więc komentarze naturalnym środkiem przydawania tworzonym skryptom cech *samodokumentacji*. Jak pokazało wieloletnie doświadczenie, nieskomentowany kod może być niezrozumiały nawet dla jego autorów, którzy powracają do niego po kilku miesiącach niewidzenia.

W języku JavaScript komentarzem jest każdy wiersz rozpoczynający się od pary ukośników („//”), na przykład

```
// To jest komentarz
```

Komentarz rozpoczynający się od pary ukośników można także umieszczać po instrukcji, zwykle w celu wyjaśnienia jej sensu, na przykład

```
a = a + 1;  // Zwiększamy licznik w celu uwzględnienia początkowego
            // ogranicznika
```

W JavaScripcie można także używać komentarzy w stylu zapożyczonym z języka C. Komentarz taki rozpoczyna się sekwencją „/*” , kończy sekwencją „*/” i może zajmować wiele wierszy, na przykład

```
/*
  Poniższa funkcja realizuje pojedynczy krok permutacji
  metodą przestawiania sąsiadujących elementów. Jest ona
  efektywniejsza od metody obrotów elementarnych, w której
  pojedynczy krok może wymagać liczby przestawień proporcjonalnej
  do kwadratu długości obracanej porcji.
*/
```

Wielowierszowe komentarze wykorzystywane bywają zwykle do „wykomentowywania”, czyli tymczasowego ukrywania przed interpreterem tych fragmentów kodu, które nie są kompletne lub też których działanie zakłócałoby poprawną pracę pozostałego kodu.

Ostrzeżenie
Ostrzeżenie

Ponieważ opisywane komentarze są częścią JavaScriptu, nie języka HTML, dopuszczalne są jedynie między znacznikami `<script>...</script>` oraz w zewnętrznych, dołączanych plikach skryptów.

Sprawdzone i zalecane praktyki programistyczne

Oprócz formalnych reguł, bezwzględnie stanowiących o tym, co w JavaScripcie poprawne, a co nie, do kanonu zasad programisty warto wpisać także poniższe zalecenia, których przestrzeganie — choć formalnie niewymagane — pozwoli zaoszczędzić wielu (niepotrzebnych) kłopotów zarówno twórcy skryptu, jak i tym, którym przyjdzie w przyszłości ów skrypt wykorzystywać lub modyfikować.

- **Nie żałuj komentarzy**. O tym, jak trudno czyta się nieskomentowany kod, może przekonać się nie tylko przypadkowy jego czytelnik, lecz także autor, któremu po kilku miesiącach czy latach przychodzi poprawiać lub unowocześniać własny kod. Treść komentarzy, jako zupełnie ignorowana przez interpreter, nie podlega żadnym regułom, programista może więc bez skrępowania opisywać swoje intencje wobec poszczególnych fragmentów skryptu, czy nawet poszczególnych instrukcji.
- **Umieszczaj średniki na końcu instrukcji, a każdą instrukcję zapisuj w osobnym wierszu.** Przyczyni się to do łatwiejszego debugowania skryptu.
- **Jeżeli to możliwe, umieszczaj skrypty w oddzielnych plikach.** Oddzielenie JavaScriptu od kodu HTML nie tylko ułatwi debugowanie, lecz także umożliwi wielokrotne wykorzystywanie raz stworzonego skryptu.
- **Unikaj konstrukcji specyficznych dla konkretnych przeglądarek.** Niektóre z mechanizmów JavaScriptu honorowane są tylko przez niektóre przeglądarki. Należy konsekwentnie unikać stosowania takich mechanizmów, chyba że ich użycie okaże się bezwzględnie konieczne.
- **Potraktuj JavaScript jako narzędzie drugorzędne.** Nie wykorzystuj JavaScriptu do wykonywania wszelkich zadań związanych z wyświetlaniem strony WWW, w szczególności do zadań tak podstawowych, jak np. realizacja hiperłączy. Nie zapominaj, że użytkownik może zablokować obsługę JavaScriptu przez przeglądarkę i wtedy funkcjonalność strony ograniczona zostanie do elementów zrealizowanych przy użyciu czystego HTML-u. Zadbaj więc o to, by elementów takich było jak najwięcej, a JavaScript wykorzystaj tylko do uzupełnienia ich repertuaru (filozofia taka nazywa się *progresywnym ulepszaniem*).

Przekonaj się sam...

Implementowanie obsługi zdarzeń

Na zakończenie rozdziału zaprezentujemy prosty przykład obsługi zdarzenia — pokażemy, jak doprowadzić do wystąpienia zdarzenia i jak zaprogramować w języku JavaScript jego obsługę. Na listingu 4.4 pokazujemy kod ilustrującej to przykładowej strony WWW.

Listing 4.4. Dokument HTML z zaprogramowaną obsługą zdarzenia

```
<html>
<head>
<title>Przykład obsługi zdarzenia</title>
</head>
<body>
<h1>Przykład obsługi zdarzenia</h1>
<p>
<a href="http://helion.pl/"
onClick="alert('O! Wystąpiło zdarzenie!');">Kliknij to hiperłącze</a>
w celu przetestowania obsługi zdarzenia.
</p>
</body>
</html>
```

Zgodnie z tym, co wcześniej pisaliśmy, scenariusz obsługi zdarzenia, które chcemy obsłużyć — `onClick` — jest wartością atrybutu o tejże nazwie, w elemencie `<a>`, którego kliknięcie ma wspomniane zdarzenie wygenerować:

```
onClick="alert('O! Wystąpiło zdarzenie!');
```

W scenariuszu obsługi zdarzenia wykorzystaliśmy wbudowaną funkcję `alert()`, powodującą wyświetlenie komunikatu o treści podanej jako parametr. W skryptach mniej banalnych niż zaprezentowany obsługa zdarzeń odbywa się zwykle w ramach funkcji definiowanych przez użytkowników. Na rysunku 4.4 widzimy okno ze wspomnianym komunikatem, wyświetlane wskutek kliknięcia jedynego widocznego hiperłącza.

W następnym rozdziale zaprezentujemy kolejne przykłady podobnej obsługi zdarzeń.



RYSUNEK 4.4. Komunikat wyświetlany w ramach obsługi zdarzenia onClick generowanego w wyniku kliknięcia hiperłącza

Podsumowanie

W zakończonym właśnie rozdziale przedstawiliśmy prosty program w języku JavaScript i przetestowaliśmy jego działanie, wbudowując go w stronę WWW i wyświetlając ją w przeglądarce. Zwróciliśmy też uwagę na niezbędne narzędzia, jakimi dysponować musi twórca skryptów: edytor tekstu i przeglądarkę WWW. Pokazaliśmy ponadto, co stanie się, gdy interpreter napotka błąd w wykonywanym kodzie skryptu.

W procesie tworzenia przykładowego skryptu wykorzystaliśmy kilka podstawowych elementów JavaScriptu — zmienne, instrukcję `document.write` i kilka funkcji związanych z datą i czasem.

Opisaliśmy kilka najważniejszych elementów składniowych JavaScriptu: funkcji, obiektów, procedur zdarzeniowych, instrukcji warunkowych i pętli. Zwróciliśmy także uwagę na duże znaczenie komentarzy dla przejrzystości kodu, jego czytelności i łatwości w utrzymaniu. Na zakończenie zaprezentowaliśmy przykład prostej obsługi zdarzenia.

W następnym rozdziale zajmiemy się modelem obiektu-dokumentu (DOM — *Document Object Model*) i opiszemy sposoby dostępu do poszczególnych elementów strony za pomocą obiektów wpisujących się w ten model.