helion.pl

IDŹ DO
PRZYKŁADOWY ROZDZIAŁ
SPIS TREŚCI

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

JavaScript dla każdego



Autorzy: Andrew Watt, Jonathan Watt, Jinjer Simon, Jim O'Donnell
Tłumaczenie: Rafał Hładyn, Radosław Meryk
ISBN: 83-7197-777-8
Tytuł oryginału: STY JavaScript in 21 Days
Format: B5, stron: 680
Przykłady na ftp: 134 kB

Poznanie języka JavaScript to dla webmastera-amatora ważny krok na drodze do profesjonalizmu. Ten prosty język programowania pozwala wzbogacić strony internetowe o elementy interaktywne. JavaScript ma wiele zastosowań: umożliwia wizualne uatrakcyjnienie strony WWW poprzez umieszczenie na niej efektów graficznych (takich jak podmiana obrazka w momencie, gdy użytkownik wskaże go myszką), pozwala na dokonywanie weryfikacji danych wprowadzanych do formularzy, a nawet na pisanie całych aplikacji, np. do obsługi prostych sklepów internetowych.

Książka „JavaScript dla każdego" to podręcznik języka JavaScript przeznaczony dla osób nie będących programistami. Jeśli chcesz dobrze opanować ten język i nauczyć się pisać własne skrypty (a nie tylko wykorzystywać gotowe fragmenty kodu znalezione w sieci), dobrze zrobisz sięgając po tę książkę. Przedstawia ona zarówno podstawy JavaScriptu, jak i jego bardziej zaawansowane aspekty. Znajdziesz w niej również rozdziały poświęcone najnowocześniejszym technologiom, takim jak SVG (grafika wektorowa na stronach WWW) czy JavaScript 1.5.

Dzięki tej książce:

- Wzbogacisz swoje strony internetowe o elementy interaktywne, poznasz DHTML
- Nauczysz się języka JavaScript w stopniu pozwalającym na tworzenie własnych programów
- Dowiesz się, jak pisać skrypty działające w różnych przeglądarkach
- Nauczysz się łączyć technologię SVG z JavaScriptem

Wydawnictwo Helion
ul. Chopina 6
44-100 Gliwice
tel. (32)230-98-63
e-mail: helion@helion.pl

Spis treści

O Autorach **15**

Wstęp **17**

Część I **21**

Rozdział 1. Podstawy **23**

Przedstawiamy język JavaScript 24
Co to jest JavaScript? 24
Do czego służy JavaScript? 26
JavaScript w kontekście 26
Potrzeba istnienia czegoś więcej niż HTML 26
Po stronie serwera czy po stronie klienta? 27
Po stronie serwera 27
Po stronie klienta 28
Realia 29
Wybór właściwej technologii 29
JavaScript, Jscript i ECMAScript 32
JavaScript 32
JScript 32
ECMAScript 33
Zaczynamy 34
Wybór edytora tekstowego i przeglądarki WWW 34
Podstawowe sprawy do zapamiętania 36
Gdzie umieszczamy kod JavaScript 37

Wielokrotne wykorzystywanie kodu ... 45
Dokumentowanie kodu ... 46
Słowa zarezerwowane ... 47
Typy danych ... 48
Liczby ... 49
Boolean ... 50
Ciągi znaków ... 51
Przydatne narzędzia ... 53
Funkcja alert() ... 53
Funkcja confirm() ... 54
Funkcja prompt() ... 55
Podsumowanie ... 56
Warsztat ... 56
Pytania i odpowiedzi ... 56
Quiz ... 57
Odpowiedzi do quizu ... 57
Ćwiczenia ... 57

Rozdział 2. Praca z danymi ... 59
Proste przechowywanie danych ... 59
Zmienne ... 60
Stałe ... 64
Operatory ... 65
Podstawowe wiadomości o operatorach ... 66
Operatory w języku JavaScript ... 68
Operatory arytmetyczne ... 68
Operatory porównań ... 70
Operator warunkowy ... 75
Operatory logiczne ... 77
Operatory łańcuchowe ... 79
Operatory przypisania ... 80
Operatory bitowe ... 82
Priorytety operatorów ... 83
Podsumowanie ... 85
Warsztat ... 85
Pytania i odpowiedzi ... 85
Quiz ... 86
Odpowiedzi do quizu ... 86
Ćwiczenia ... 86

Rozdział 3. Funkcje i instrukcje ... 87
Funkcje użytkownika ... 87
Czym są funkcje? ... 88
Tworzenie funkcji ... 88
Wykorzystanie funkcji ... 90
Predefiniowane funkcje globalne ... 97
Kodowanie i dekodowanie adresów URI ... 99
Zamiana ciągów znaków na kod ... 100
Funkcje arytmetyczne ... 101
Czym są instrukcje? ... 107
Instrukcje sterujące ... 108
Instrukcje warunkowe ... 109
Instrukcje pętli ... 115
Instrukcja while ... 116

Instrukcja do while ... 117
Instrukcja for ... 118
Instrukcja for in ... 121
Podsumowanie ... 121
Warsztat ... 121
Pytania i odpowiedzi ... 121
Quiz ... 122
Odpowiedzi do quizu ... 122
Ćwiczenie ... 122

Rozdział 4. JavaScript to język obiektowy ... 123
Podstawowe informacje o obiektach ... 123
Czym są obiekty? ... 124
Obiekty języka JavaScript po stronie klienta ... 125
Obiekt window ... 126
Notacja z kropką ... 126
Zmienna czy właściwość? Funkcja czy metoda? ... 127
Obiekty potomne obiektu window ... 132
Obiekt document ... 133
Właściwości ... 133
Metody ... 135
Obiekty potomne ... 139
Obiekt body ... 142
Obiekty środowiskowe ... 143
Obiekt location ... 143
Ładowanie nowej strony ... 144
Dostęp do adresu URL ... 146
Obiekt history ... 148
Obiekt navigator ... 150
Obiekt screen ... 151
Obiekty rdzenia języka JavaScript ... 151
Eksploracja obiektu ... 154
Podsumowanie ... 158
Warsztat ... 159
Pytania i odpowiedzi ... 159
Quiz ... 159
Odpowiedzi na quiz ... 159
Ćwiczenia ... 159

Rozdział 5. Wprowadzenie do tablic ... 161
Czym są tablice? ... 161
Tworzenie tablic ... 163
Uzyskiwanie dostępu do indywidualnych elementów tablicy ... 164
Tworzenie pustej tablicy i wypełnianie jej danymi ... 164
Literały tablicowe ... 166
Wykorzystywanie tablic ... 167
Zastosowanie tablic równoległych ... 170
Właściwości obiektu Array ... 172
Metody obiektu Array ... 176
Metoda toString() ... 177
Metoda toLocaleString() ... 178
Metoda join() ... 178
Dodawanie elementów do tablicy i usuwanie ich z tablicy ... 179
Metoda reverse() ... 185
Metoda toSource() ... 188

Metoda valueOf() ... 189
Tablice asocjacyjne ... 189
Zapisywanie obiektów w elementach tablic ... 192
Podsumowanie ... 194
Warsztat ... 195
Pytania i odpowiedzi ... 195
Quiz ... 195
Odpowiedzi do quizu ... 195
Ćwiczenie ... 195

Rozdział 6. Formularze HTML i obiekt String ... 197

Pobieranie danych z formularzy HTML ... 197
Do czego służy język JavaScript w formularzach? ... 200
Dostęp do elementów formularzy z poziomu języka JavaScript ... 200
Właściwości elementu <form> ... 201
Metody obiektu Form ... 205
Elementy formularzy ... 209
Kolekcja elements ... 210
Właściwości wspólne dla kilku elementów formularza ... 212
Elementy typu text ... 214
Element textarea ... 216
Pola wyboru ... 218
Przełączniki ... 220
Listy wyboru ... 222
Przyciski ... 225
Obiekt String ... 225
Właściwości obiektu String ... 227
Metody obiektu String ... 227
Sprawdzanie danych wprowadzanych przez użytkownika ... 232
Podsumowanie ... 236
Warsztat ... 236
Pytania i odpowiedzi ... 236
Quiz ... 237
Odpowiedzi do quizu ... 237
Ćwiczenia ... 237

Rozdział 7. Liczby i obliczenia ... 239

Obiekt Number ... 239
Liczby w JavaScript ... 240
Właściwości obiektu Number ... 244
Metody obiektu Number ... 247
Metoda valueOf() ... 254
Obiekt Math ... 255
Predefiniowane właściwości ... 256
Metody obiektu Math ... 258
Tworzenie własnych funkcji obiektu Math ... 265
Obliczanie wartości silnia z liczby ... 265
Obliczanie pierwiastka stopnia n ... 266
Logarytm o podstawie N ... 267
Podsumowanie ... 267
Warsztat ... 268
Pytania i odpowiedzi ... 268
Quiz ... 268
Odpowiedzi do quizu ... 268
Ćwiczenia ... 268

Część II ... 269

Rozdział 8. Przeglądarki ... 271
Różne przeglądarki, różne wersje JavaScript ... 271
Historia języka JavaScript ... 272
Standardy ECMA ... 273
Tworzenie skryptów zgodnych z wieloma przeglądarkami ... 274
Przeglądarki, które nie obsługują JavaScript ... 275
Przeglądarki z wyłączoną obsługą JavaScript ... 276
Pobieranie informacji o przeglądarce ... 277
Sprawdzanie obsługi określonych obiektów w przeglądarce ... 282
Model W3C DOM ... 283
Dostęp do węzłów ... 284
Podsumowanie ... 286
Warsztat ... 287
Pytania i odpowiedzi ... 287
Quiz ... 288
Odpowiedzi do quizu ... 288
Ćwiczenia ... 289

Rozdział 9. Funkcje daty i czasu ... 291
Daty w języku JavaScript ... 291
Obiekt Date ... 292
Utworzenie obiektu Date dla określonej daty ... 294
Formatowanie dat ... 295
Przekształcanie liczbowych wartości dnia i miesiąca na nazwy ... 297
Kompozycje wartości dat ... 298
Formatowanie czasu ... 300
Zmiana formatu czasu na AM/PM ... 302
Strefy czasowe ... 304
Konwersja formatów daty i czasu ... 305
Obliczenia z datami ... 306
Konwersja pomiędzy ciągami znaków a datami ... 308
Podsumowanie ... 308
Warsztat ... 309
Pytania i odpowiedzi ... 309
Quiz ... 309
Odpowiedzi do quizu ... 310
Ćwiczenia ... 310

Rozdział 10. Zdarzenia i ich obsługa ... 311
Zdarzenia ... 311
Tworzenie procedury obsługi zdarzenia ... 312
Obsługa zdarzenia w postaci instrukcji JavaScript ... 313
Obsługa zdarzeń za pomocą wywołania funkcji JavaScript ... 314
Przechwytywanie zdarzeń bezpośrednio w kodzie JavaScript ... 315
Rodzaje zdarzeń ... 316
W jaki sposób obsługiwać zdarzenia? ... 318
Monitorowanie zmian w formularzach ... 318
Wykonywanie działań ze zdarzeniami związanymi z klawiaturą ... 321
Monitorowanie aktywności obiektu ... 323
Monitorowanie zdarzeń obiektów Window oraz Document ... 325
Przechwytywanie działań wykonywanych myszą ... 325
Podsumowanie ... 331
Warsztat ... 331

Pytania i odpowiedzi ... 331
Quiz ... 332
Odpowiedzi do quizu ... 332
Ćwiczenia ... 333

Rozdział 11. Dynamiczny HTML ... 335
Czym jest DHTML? ... 335
Zastosowanie arkuszy stylów kaskadowych ... 336
Definiowanie stylów ... 337
Tworzenie reguł CSS ... 341
Tworzenie klas ... 345
Warstwy ... 348
Zmiana atrybutów elementu HTML ... 351
Przesuwanie elementów ... 353
Podsumowanie ... 357
Warsztat ... 357
Pytania i odpowiedzi ... 357
Quiz ... 358
Odpowiedzi do quizu ... 358
Ćwiczenia ... 358

Rozdział 12. Okna i ramki ... 359
Czym są okna i ramki? ... 359
Właściwości i metody obiektu Window ... 360
Określanie adresu wyświetlanej w oknie strony ... 365
Działania dotyczące historii stron wyświetlanych w przeglądarce ... 368
Działania dotyczące paska stanu ... 369
Zastosowanie obiektu Screen ... 371
Zastosowanie ramek ... 373
Adresy stron wyświetlanych w ramkach ... 377
Uniemożliwianie wyświetlania w ramkach ... 379
Wymuszanie wyświetlania ramek ... 380
Okna podręczne ... 382
Podsumowanie ... 384
Warsztat ... 384
Pytania i odpowiedzi ... 385
Quiz ... 385
Odpowiedzi do quizu ... 386
Ćwiczenia ... 386

Rozdział 13. Wyrażenia regularne ... 387
Dlaczego wyrażenia regularne są tak potrzebne ... 387
Czym są wyrażenia regularne? ... 389
Dopasowanie sekwencji znaków tekstowych ... 390
Proste przykłady wzorców ... 392
Możliwości dopasowania pojedynczej litery ... 392
Klasy znaków ... 395
Występowanie znaków we wzorcu określoną ilość razy ... 396
Wzorce uwzględniające zmienne ilości znaków ... 399
Przegląd wyrażeń regularnych ... 404
Definiowanie wyrażeń regularnych ... 404
Tworzenie obiektu przez przypisanie ... 404
Tworzenie wyrażeń regularnych przy pomocy konstruktora ... 405
Znaki specjalne ... 407
Kwantyfikatory ... 411
Metody obiektu RegExp ... 412

Metoda exec() ... 413
Metoda test() ... 414
Właściwości obiektu RegExp ... 414
Określanie zasięgu działania wyrażenia regularnego przy pomocy właściwości global ... 414
Rozróżnianie wielkich i małych liter w wyrażeniach regularnych przy pomocy właściwości ignoreCase ... 415
Podsumowanie ... 415
Warsztat ... 416
Pytania i odpowiedzi ... 416
Quiz ... 416
Odpowiedzi do quizu ... 416
Ćwiczenia ... 417

Rozdział 14. Zaawansowane zarządzanie tablicami ... 419
Metody obiektu Array ... 419
Metoda concat() ... 420
Metoda slice() ... 427
Metoda splice() ... 429
Metoda sort() ... 434
Tablice wielowymiarowe ... 444
Podsumowanie ... 450
Warsztat ... 450
Pytania i odpowiedzi ... 450
Quiz ... 451
Odpowiedzi do quizu ... 451
Ćwiczenia ... 451

Część III ... 453

Rozdział 15. Debuggowanie i obsługa błędów ... 455
Zapobieganie błędom oraz ich klasyfikacja ... 456
Wstawianie komentarzy ... 456
Śledzenie zmiennych ... 456
Typy błędów w programowaniu w języku JavaScript ... 457
Znajdowanie błędów składni ... 458
Błędy składni w przeglądarce Netscape Navigator ... 458
Błędy składni w przeglądarce Internet Explorer ... 459
Debuggowanie błędów występujących podczas ładowania strony ... 460
Stosowanie metody document.writeln() ... 462
Stosowanie różnorodnych danych wejściowych ... 465
Debuggowanie błędów I: zdarzenia dyskretne ... 467
Zastosowanie metody alert() ... 469
Zastosowanie elementów formularzy HTML ... 472
Debuggowanie błędów II: zdarzenia ciągłe ... 475
Zaawansowane techniki debuggowania ... 478
Tworzenie interpretera JavaScript działającego w trakcie uruchomienia strony ... 479
Wyświetlanie informacji debuggowania w nowym oknie przeglądarki ... 481
Debuggery JavaScript ... 481
Podsumowanie ... 483
Warsztaty ... 483
Pytania i Odpowiedzi ... 483
Quiz ... 483
Odpowiedzi do quizu ... 484
Ćwiczenie ... 484

Rozdział 16. Pliki cookie: trwałe przechowywanie danych ... 485
Przechowywanie stanu ... 485
Pliki cookie: wstęp ... 487
Zalety cookie ... 487
Ograniczenia plików cookie ... 488
Wady plików cookie ... 488
Mity dotyczące plików cookie ... 488
W jaki sposób używać plików cookie ... 489
Odczytywanie wartości z plików cookie ... 489
Ustawianie wartości cookie ... 490
Przykład cookie ... 495
Dokąd zmierza technologia cookie ... 507
Gdzie można znaleźć dodatkowe informacje o cookie ... 508
Inne możliwości przechowywania stanu ... 508
Wysyłanie informacji przy użyciu łańcucha zapytań ... 509
Sposób używania ukrytych zmiennych w formularzach ... 511
Podsumowanie ... 512
Warsztat ... 512
Pytania i Odpowiedzi ... 512
Quiz ... 512
Odpowiedzi do quizu ... 512
Ćwiczenie ... 512

Rozdział 17. Poufność i bezpieczeństwo ... 513
Bezpieczeństwo danych osób odwiedzających ... 513
Ograniczenia plików ... 514
Cookies ... 515
Dostęp do okna przeglądarki ... 516
Ograniczenia zasobów komputera ... 516
Bezpieczeństwo danych administratorów WWW ... 517
Polityka tego samego pochodzenia ... 519
Wpływ polityki bezpieczeństwa na JavaScript ... 520
Ochrona hasłem ... 520
Tworzenie podpisanych skryptów w przeglądarce Netscape Navigator ... 523
Podsumowanie ... 525
Warsztat ... 526
Pytania i Odpowiedzi ... 526
Quiz ... 527
Odpowiedzi do quizu ... 527
Ćwiczenia ... 527

Rozdział 18. Rozszerzenia i aplety ... 529
Różnica pomiędzy apletem a rozszerzeniem ... 530
Wykrywanie zainstalowanych wtyczek ... 531
W jaki sposób przeglądarki zarządzają rozszerzeniami ... 532
Sprawdzanie rozszerzeń ... 534
Sprawdzanie, czy przeglądarka obsługuje dany rodzaj plików ... 537
Praca z obiektami rozszerzeń ... 539
Praca z apletami ... 540
Podstawy technologii Java ... 541
Wywołanie apletu Javy ... 542
Komunikacja z apletami Javy ... 542
Podsumowanie ... 543
Warsztat ... 544
Pytania i odpowiedzi ... 544

Quiz ... 545
Odpowiedzi do quizu ... 545
Ćwiczenia ... 545

Rozdział 19. Tworzenie własnych obiektów ... 547
Czym jest obiekt użytkownika? ... 547
Łączenie różnych elementów danych w jeden obiekt ... 548
Tworzenie metod i właściwości obiektu użytkownika ... 549
Tworzenie obiektów za pomocą konstruktora ... 549
Wstawianie właściwości do obiektu ... 550
Tworzenie egzemplarza obiektu użytkownika ... 551
Dostęp do właściwości obiektu użytkownika ... 553
Zmiana wartości właściwości obiektu ... 553
Tworzenie metod obiektu ... 555
Łączenie obiektów ... 558
Podsumowanie ... 560
Warsztat ... 560
Pytania i odpowiedzi ... 560
Quiz ... 561
Odpowiedzi do quizu ... 561
Ćwiczenia ... 562

Rozdział 20. JavaScript i handel elektroniczny ... 563
Zawartość serwisu e-commerce — wymagania ... 563
Struktura sklepu internetowego ... 564
Katalog internetowy ... 566
Tworzenie własnej bazy danych ... 566
Projektowanie katalogu ... 568
Kod sklepu księgarni internetowej ... 570
Tworzenie pliku books.js ... 572
Wirtualny koszyk ... 574
Rozwiązywanie problemu liczb zmiennoprzecinkowych ... 576
Usuwanie zamówień ... 577
Zbieranie informacji o kliencie ... 578
Kod strony wirtualnego koszyka ... 580
Kwestie warte rozważenia ... 584
Podsumowanie ... 584
Warsztat ... 584
Pytania i odpowiedzi ... 584
Quiz ... 585
Odpowiedzi do quizu ... 585
Ćwiczenia ... 586

Rozdział 21. Technologia SVG ... 587
Przegląd technologii SVG ... 587
Dlaczego SVG? ... 588
Dlaczego SVG w połączeniu z JavaScript? ... 589
Podstawowe narzędzia SVG ... 589
Przykład kodu SVG ... 591
Model obiektowy dokumentu SVG (SVG DOM) ... 593
Przeglądarka modelu obiektowego dokumentu Batik ... 593
Podstawy SVG DOM ... 595
JavaScript w dokumentach SVG ... 595
Element <script> w dokumencie SVG ... 596
Określanie języka skryptowego ... 596
Szkielet dokumentu SVG z kodem JavaScript ... 597

Wstawianie tytułu w rysunku SVG ... 598
Tworzenie prostych kształtów przy pomocy kodu JavaScript ... 600
Wstawianie tekstu za pomocą kodu JavaScript ... 602
Tworzenie płynnych animacji przy pomocy kodu JavaScript ... 603
Implementacja zdarzeń ... 605
Zaawansowane animacje w języku JavaScript ... 610
Wzajemne oddziaływanie modeli obiektowych dokumentów HTML i SVG ... 615
Gdzie można znaleźć informacje o SVG? ... 616
Podsumowanie ... 617
Warsztat ... 617
Pytania i odpowiedzi ... 617
Quiz ... 617
Odpowiedzi do quizu ... 618
Ćwiczenia ... 618

Dodatki ... 619

Dodatek A Nowe cechy JavaScript 1.5 ... 621

Dodatek B Kody kolorów ... 625

Dodatek C Spis funkcji ... 631

Dodatek D Zasoby sieciowe ... 647

Dodatek E Krótka historia języka JavaScript ... 653

Skorowidz ... 659

Rozdział 1.
Podstawy

JavaScript to język skryptowy, który umożliwia wprowadzanie elementów interaktywnych na stronach WWW zapisanych w języku HTML lub XHTML. Język XHTML — Rozszerzony Hipertekstowy Język Znaczników (ang. *Extended Hypertext Markup Language*) — odpowiada językowi HTML 4 zapisanemu w składni języka XML. JavaScript idealnie nadaje się do wprowadzania na strony WWW dodatkowych właściwości oraz elementów interaktywnych. Posługiwanie się jedynie czystym językiem HTML (lub XHTML) nie pozwala na uzyskanie takich efektów.

W niniejszej książce zaprezentujemy najbardziej użyteczne właściwości języka JavaScript. Czasami, niektóre z tematów poruszanych w tej publikacji mogą się nam wydać nieco abstrakcyjne. Zapamiętajmy na początek, że język JavaScript daje największe możliwości w przypadku, gdy wykorzystujemy kilka jego elementów równocześnie. Zatem, aby zrozumieć opis określonego zagadnienia, powinniśmy najpierw poznać kilka właściwości języka JavaScript i wykonać przykłady krok po kroku. Wraz z coraz głębszym poznawaniem omawianych tu zagadnień upewnimy się co do własnych zdolności łączenia właściwości języka JavaScript i będziemy mogli tworzyć skrypty, które na razie będą po prostu działały, a kiedy nabierzemy wprawy, także skrypty wykonujące dokładnie takie działania, jakich oczekujemy. Po przeanalizowaniu przykładów, po zakończeniu lektury książki i przestudiowaniu języka JavaScript staniemy się kompetentnymi autorami skryptów w tym języku, dzięki czemu będziemy mogli wykorzystać jego możliwości do ożywienia naszych stron WWW.

W tym rozdziale przedstawiono zarówno zastosowania, jak i podstawową strukturę języka JavaScript. Informacje te dadzą nam solidne podstawy, na których oprzemy naszą wiedzę na temat interesującego nas języka podczas lektury kolejnych rozdziałów tej książki.

W tym rozdziale interesować nas będą następujące zagadnienia:

- Czym jest JavaScript?
- Podstawy składni języka JavaScript, z uwzględnieniem typów danych oraz podstawowych funkcji.

Przedstawiamy język JavaScript

Podstawowymi pytaniami, jakie zadajemy sobie przystępując do nauki czegoś nowego, są: „co to jest?" oraz „do czego to służy?". Spróbujmy zatem odpowiedzieć na nie w odniesieniu do języka JavaScript.

Co to jest JavaScript?

Przytaczając powszechnie powtarzaną definicję, możemy powiedzieć, że JavaScript jest wieloplatformowym, obiektowym językiem skryptowym, najpowszechniej wykorzystywanym w sieci. Osobom rozpoczynającym dopiero swą przygodę z programowaniem taka definicja niewiele wyjaśni, przyjrzyjmy się zatem poszczególnym jej częściom.

Język wieloplatformowy

W terminologii komputerowej pojęcie *platforma* odnosi się do systemu operacyjnego. Obecnie w komputerach biurowych powszechnie wykorzystuje się takie systemy operacyjne, jak Windows (różne odmiany), Linux czy MacOS. Mówiąc, że język JavaScript jest wieloplatformowy, mamy na myśli fakt, że kod w nim zapisany będzie w większości przypadków z powodzeniem działać w różnych systemach operacyjnych, dając w nich te same wyniki.

Wieloplatformowa natura JavaScript jest bardzo ważnym aspektem tego języka. Obecnie w komputerach wykorzystywanych do łączenia się z Internetem stosuje się wiele różnych systemów operacyjnych. Bez właściwości języka JavaScript, która polega na spójnym działaniu na różnych platformach, napisanie skryptów JavaScript oferujących ulepszenia na stronach WWW dla wykorzystujących różne platformy użytkowników Internetu, byłoby znacznie trudniejsze.

Jednym z podstawowych celów twórców języka JavaScript było opracowanie wieloplatformowego języka skryptowego. Możliwość pracy w różnych systemach znacznie przyczyniła się do sukcesu tego języka. Bez tej cechy, język JavaScript nie przemawiałby tak wyraziście do projektantów stron WWW, którzy zazwyczaj chcą docierać do jak najszerszego grona użytkowników.

Język obiektowy

Prawdopodobnie do tej pory najczęściej korzystaliśmy z języka HTML — co oznacza, że wiemy, iż jest to język znaczników. W językach tego typu znaczniki wykorzystywane są do otaczania fragmentów tekstu. Znaczniki informują przeglądarkę, w jaki sposób taki tekst ma być interpretowany.

Język JavaScript jest inny. Wymaga, aby struktura danych, np. informacja na stronie WWW, już istniała. Ponadto, dane zapisane na stronie traktowane są jako obiekty o hierarchicznej strukturze. Jeżeli termin „obiekty o hierarchicznej strukturze" niewiele nam mówi, nie ma powodu do niepokoju — powrócimy do tego tematu w dalszej części

książki. Cały rozdział 4. — „JavaScript to język obiektowy" — poświęcony jest omówieniu zagadnienia obiektowego charakteru języka JavaScript, a także opisowi obiektów, które będą wykorzystane w kolejnych rozdziałach. Na razie wystarczy nam informacja, że obiekty umożliwiają łatwiejszą organizację stron WWW oraz wykonywanie działań z tymi stronami.

Język skryptowy

JavaScript jest także językiem skryptowym. Warto zauważyć, że istnieją znaczące różnice pomiędzy językami skryptowymi, a niezależnymi językami programowania (np. C++ lub Visual Basic). Do podstawowych różnic można zaliczyć tę, że języki skryptowe są interpretowane i zazwyczaj wymagają one zapisania znacznie mniejszej ilości kodu.

Jeżeli język jest interpretowany, oznacza to po prostu, że wpisywany przez nas kod nie musi być przed wykorzystaniem kompilowany do postaci binarnej. Zamiast tego, języki skryptowe posługują się instrukcjami, które poddawane są analizie za pomocą interpretera. Interpreter analizuje kod za każdym razem, kiedy go uruchamiamy.

Ważną różnicą jest także to, że języki skryptowe działają wewnątrz innego programu lub aplikacji, np. przeglądarki WWW. Typowy skrypt JavaScript umieszczony po stronie klienta zawarty jest wewnątrz strony WWW zapisanej w języku HTML lub XHTML. Kod zapisany w takich językach, jak C++ czy Java może być wykonywany niezależnie (w przypadku języka Java możliwe jest również wykonywanie programów w obrębie stron WWW — taki program określa się wówczas mianem apletu).

Interpreter języka JavaScript to oprogramowanie wbudowane w przeglądarkę WWW. Jego zadaniem jest pobieranie kodu JavaScript i postępowanie, krok po kroku, według instrukcji wchodzących w skład kodu.

Język JavaScript jest obsługiwany przez wszystkie główne przeglądarki w wersjach 3.0 i nowszych.



Skrypty JavaScript mogą być uruchamiane po stronie klienta — wewnątrz przeglądarki WWW lub na serwerze WWW — wewnątrz aplikacji serwera WWW. W tej książce będziemy omawiać jedynie skrypty uruchamiane po stronie klienta.

Ważną, wyróżniającą języki skryptowe cechą, którą zauważymy podczas ich stosowania jest to, że np. podczas pisania skryptów JavaScript zapisujemy znacznie mniej kodu niż w przypadku pisania niezależnego programu. Dzieje się tak dlatego, że przeglądarka WWW zawiera wiele użytecznych funkcji obsługujących język JavaScript.

Skrypty są łatwiejsze do napisania, ale są wykonywane nieco wolniej niż skompilowany kod. Zaletą języków skryptowych jest to, że — w porównaniu z niektórymi językami niezależnymi — często łatwiej je napisać oraz nie wymagają skomplikowanych i drogich narzędzi.

Do czego służy JavaScript?

JavaScript ma wiele zastosowań. Podstawowym jest ulepszanie stron WWW. Język ten umożliwia programowanie stron WWW w taki sposób, że można wprowadzać do statycznych stron efekt ruchu oraz różnorodne elementy interaktywne. Uzyskanie takich efektów nie jest możliwe, jeśli posługujemy się jedynie językiem HTML. Za pomocą JavaScript można także zaimplementować obsługę błędów dla danych wprowadzanych w formularzach HTML.

JavaScript w kontekście

Zanim naprawdę rozpoczniemy studiowanie języka JavaScript, przyjrzyjmy się zagadnieniom zapotrzebowania na jego istnienie oraz miejsca, jakie zajmuje na tle innych popularnych technologii WWW.

Potrzeba istnienia czegoś więcej niż HTML

W ciągu ostatniej dekady przeciętny użytkownik komputerów był świadkiem olbrzymiego wzrostu jakości i funkcjonalności wykorzystywanych programów. Postępy w rozwoju sprzętu i oprogramowania zaowocowały znaczącą poprawą zarówno funkcji, jak i wyglądu programów. Użytkownicy komputerów przyzwyczaili się do programów kolorowych, dynamicznych i wymagających aktywności. Coraz rzadziej akceptują informacje podawane w nieciekawej, statycznej formie. Wprowadzenie skryptów JavaScript na stronach WWW czyni strony WWW bardziej interaktywnymi niż było to możliwe za pomocą funkcji oferowanych w technologiach dostępnych po stronie serwera.

Podobnie jest w przypadku Internetu. Chociaż w początkowym zamyśle strony WWW miały prezentować statyczny tekst, bez elementów graficznych, to jednak Internet od momentu swego powstania znacznie się zmienił. Rozwój języka HTML i wprowadzenie arkuszy kaskadowych stylów (CSS) stanowiły milowe kroki na drodze do wprowadzenia kolorów i grafiki na stronach WWW. Dzięki tym technologiom strony WWW stały się bardziej estetyczne. Pomimo to czysty HTML w dalszym ciągu jest formatem statycznym. W zakresie interaktywności z użytkownikiem może on zaoferować co najwyżej wykorzystanie hiperłączy lub pewnych elementów formularzy, które użytkownicy mogą wypełniać. Jednak nawet wówczas, bez dodatkowych technologii, za pomocą czystego języka HTML, nie można wykonać takich użytecznych czynności, jak choćby sprawdzenie poprawności wprowadzanych informacji.

Konkurencja pomiędzy rosnącą liczbą ośrodków WWW wywiera coraz większą presję na projektantów stron WWW. Muszą oni szukać sposobów na przyciągnięcie uwagi użytkowników do ich ośrodków oraz sprawienie, aby chętnie do nich wracali. Ta presja doprowadziła do rozwoju różnorodnych, niekiedy ekscytujących technologii, które mają na celu wprowadzenie prostych, ale użytecznych usprawnień oferowanych przez język HTML. Niektóre z nowych technologii mają na celu poprawę komfortu użytkowników dzięki tworzeniu dynamicznych i interaktywnych stron WWW, natomiast inne — wprowadzenie funkcji użytecznych w biznesie oraz w świadczeniu innych usług.

Język JavaScript jest jedną z podstawowych dostępnych technologii ulepszania stron WWW. Jego specyfika polega na możliwości ożywienia statycznych stron HTML.

Po stronie serwera czy po stronie klienta?

Ogólnie rzecz biorąc, technologie ulepszania stron WWW można podzielić na dwie kategorie: technologie po stronie serwera oraz technologie po stronie klienta. „Po stronie klienta" oraz „po stronie serwera" to określenia miejsca działania technologii, czyli tego, jaki komputer wykorzystywany jest do wykonywania działania.

Nie jest jeszcze wystarczająco jasne, o co chodzi? Otóż, w każdej sieci komputerowej, niezależnie od tego, czy jest to sieć Internet czy też sieć wewnętrzna firmy, możemy wyróżnić dwa typy komputerów: klient i serwer. Zadaniem serwera jest przechowywanie dokumentów (w naszym przypadku stron WWW) i wysyłanie ich do innych komputerów, które żądają dostępu do nich. Komputer wysyłający żądanie do serwera (żądający pliku przechowywanego na serwerze) określa się mianem *klient*. A zatem określenie „technologia po stronie serwera" oznacza po prostu, że do uruchamiania programów i przetwarzania danych wykorzystywany jest serwer. Podobnie, w technologii „po stronie klienta" dane przetwarzane są w komputerze klienckim, w większości przypadków w przeglądarce WWW lub w powiązaniu z oprogramowaniem przeglądarki. Jest to bardzo istotny podział, ponieważ wpływa on na sposób działania określonej technologii.

Po stronie serwera

Przykładami znanych technologii „po stronie serwera" są między innymi: *CGI* (*Common Gateway Interface*), *ASP* (*Active Server Pages*) czy też *JSP* (*Java Server Pages*). Dość często podczas korzystania z ośrodka WWW w obrębie adresu *URL* (*Uniform Resource Locator*) możemy odnaleźć ciąg „cgi" lub nazwy plików kończące się rozszerzeniami *.asp* lub *.jsp*. Prawdopodobnie korzystamy z technologii tych dość regularnie, do wykonywania różnych czynności. Na przykład, z przetwarzania po stronie serwera korzystamy za każdym razem, kiedy w wyszukiwarce zatwierdzamy frazę do poszukiwania. W przypadku wyszukiwarek, do pobierania kryteriów wyszukiwania wykorzystywany jest formularz HTML, a następnie kryteria te wysyłane są do serwera w celu przetworzenia. Po wykonaniu zadań przez programy lub przez skrypty na serwerze, wyniki są przekształcane na format HTML i zwracane przez serwer WWW do przeglądarki.

Problemem, z jakim spotykamy się w przypadku technologii po stronie serwera, jest czas uzyskiwania wyników. Jego przyczyny są dwojakie. Po pierwsze, w Internecie występuje problem określany jako „zatwierdź i czekaj". Dotarcie danych do serwera i ich przetworzenie zajmuje pewien czas; podobnie potrzebny jest pewien czas, aby dane zostały przesłane z powrotem. Może to być szczególnie uciążliwe dla tych użytkowników, którzy łączą się z Internetem wykorzystując łącze telefoniczne.

Innym powodem możliwych opóźnień jest fakt, że w technologiach przetwarzania po stronie serwera następuje obciążanie serwera zadaniami przetwarzania. Nie stanowi to zbyt dużego problemu, jeżeli serwer w danym czasie obsługuje tylko ograniczoną liczbę żądań. Jednak każdy komputer ma limitowane możliwości przetwarzania. W przypadku

zajętych serwerów, przetwarzanie wielu tysięcy żądań w ciągu godziny i obsługiwanie dziesiątek, jeśli nie setek żądań jednocześnie, może znacząco spowolnić działanie, a w niektórych przypadkach spowodować nawet zatrzymanie serwera. Jeżeli często korzystamy z Internetu, to z całą pewnością spotkaliśmy się z komunikatem o błędzie, który informuje, że serwer WWW nie odpowiada i sugeruje ponowienie próby w późniejszym czasie. Jest bardzo prawdopodobne, że w czasie, kiedy wysyłaliśmy żądanie do serwera, nastąpiło przekroczenie jego możliwości przetwarzania i nie był on w stanie przyjąć dalszych żądań.

Oczekiwanie na ładowanie stron spowodowane czasem przesyłania w sieci oraz czasem przetwarzania na serwerze — zjawisko występujące w technologiach po stronie serwera — jest zjawiskiem niepożądanym. Powoduje ono, że wykonywanie skryptów po stronie serwera dla niektórych aplikacji (jak np. DHTML — dynamiczny HTML) po prostu się nie opłaca. Cały proces będzie bowiem zbyt powolny, a czas reakcji serwera niewystarczająco krótki, aby można było zapewnić interakcję pomiędzy użytkownikiem, a stroną WWW w czasie rzeczywistym. Na marginesie, warto zaznaczyć, że terminem „dynamiczny HTML" określa się różnorodne kombinacje języka HTML, JavaScript oraz technologii CSS, które umożliwiają wprowadzanie na stronach WWW interaktywności i animacji.

Kiedy decydujemy się na wybór przetwarzania po stronie serwera, powinniśmy wnikliwie rozważyć wszystkie za i przeciw takiego rozwiązania. Cel, jakim jest zwiększenie interaktywności i zastosowanie nowych rozwiązań, nie powinien być negowany przez zbyt długi czas oczekiwania.

Po stronie klienta

Kiedy używamy technologii po stronie serwera, komputer-klient (a często także użytkownik) bezczynnie oczekuje na załadowanie strony, podczas gdy po drugiej stronie serwer ledwo nadąża obsługiwać wszystkie żądania. Oczywistym rozwiązaniem tego problemu jest wykonywanie przynajmniej części zadań przetwarzania na komputerze-kliencie.

Pierwszą korzyścią, jaką uzyskamy po przesunięciu pewnej części obciążenia na stronę klienta jest zmniejszenie konieczności częstego ładowania strony. Ma to wpływ na zmniejszenie czasu oczekiwania, ponieważ dane nie muszą tak często przebywać swojej drogi w sieci. Dla przykładu, jeżeli sprawdzenie poprawności danych wprowadzanych w formularzu HTML nastąpi po stronie klienta, z wykorzystaniem skryptów JavaScript, unikniemy całkowicie opóźnień sieciowych — przynajmniej do czasu, kiedy dane będą sprawdzone i przygotowane do przesłania na serwer w celu dalszego przetwarzania.

Ważną zaletą wykorzystywania skryptów po stronie klienta jest uzyskanie możliwości programowania w obrębie samej strony WWW. W efekcie, zyskujemy możliwość tworzenia dynamicznych stron WWW odpowiadających na działania użytkowników, którzy przeglądają stronę i wykonują na niej jakieś działania.

Oprócz bezpośredniej korzyści, jaką jest zmniejszenie obciążenia serwera, które jest związane z koniecznością kilkakrotnego ładowania strony, istotne jest także to, że serwer może odpowiadać na żądania znacznie szybciej, ponieważ nie musi wielokrotnie

wykonywać tych samych skryptów. W konsekwencji, żądania nie czekają przed wykonaniem w kolejkach lub czas oczekiwania jest znacznie krótszy. Poprzez jego skrócenie możemy w znaczący sposób poprawić komfort użytkowników przeglądających strony.

Realia

Oczywiście, w rzeczywistości komputer-klient nie może wykonywać całego przetwarzania. We wszystkich usługach wykorzystujących technologię WWW konieczne jest przesyłanie danych na serwer w celu przechowywania lub przetwarzania. Dla przykładu, złożenie zamówienia w ośrodku handlu elektronicznego nie będzie możliwe, jeżeli zamówienie nigdy nie opuści naszego komputera. Z drugiej strony, wykorzystywanie zasobów serwera (jego mocy obliczeniowej), powodujące nieuniknione koszty czasowe związane z przesyłaniem danych na serwer, jest bezzasadne, jeżeli zadanie można równie dobrze, a czasem nawet lepiej, wykonać po stronie klienta.

W praktyce wykorzystuje się zarówno technologie po stronie serwera, jak i po stronie klienta. Obydwa rodzaje technologii są istotne dla działania nowoczesnych stron WWW.

Kiedy zadanie wymaga przesłania informacji na serwer, na przykład w celu przechowywania ich na serwerze, należy przesłać te informacje. Jeżeli jednak jest to możliwe, to przetwarzanie danych i sprawdzanie ich poprawności powinno być przeprowadzane po stronie klienta, za pomocą skryptów. Najlepsze efekty uzyskuje się poprzez równomierne rozłożenie obciążenia pomiędzy serwer a komputery klienckie.

Wybór właściwej technologii

Istnieją inne technologie wykonywane po stronie klienta (jak choćby Flash), które służą głównie do poprawienia wyglądu strony oraz zwiększenia komfortu użytkownika. Zazwyczaj same przeglądarki nie zawierają wbudowanej obsługi technologii Flash, z tego prostego względu, że sensowne jest obsługiwanie jedynie ograniczonej liczby formatów danych. Aby korzystać z tej technologii, trzeba zainstalować odpowiednie rozszerzenie, tzw. „wtyczkę" (ang. *plug-in*). Większość przeglądarek posiada wbudowaną obsługę języka JavaScript, ale dodanie wszystkich dodatkowych funkcji spowodowałoby zwiększenie rozmiarów przeglądarek (a i tak ich rozmiary są już pokaźne).

Dla rozwiązania problemu rozmiarów przeglądarek opracowano programowe rozszerzenia, które jako pierwsze pojawiły się w przeglądarce Netscape Navigator w wersji 2. Dzięki umożliwieniu wyboru potrzebnych użytkownikowi rozszerzeń, można było częściowo rozwiązać problem rozmiarów przeglądarek. Wykorzystując rozszerzenia można było bez problemów umieszczać w dokumentach dane różnych formatów, co z kolei pozwalało uniknąć konieczności ładowania aplikacji obsługującej określony format. W celu ożywienia treści strony mogły zawierać animację, a także dźwięk lub sekwencje wideo, które można było pobrać i odtworzyć lub przesłać jako strumień.

Wadą technologii WWW wykorzystujących programowe rozszerzenia jest fakt, że nie każdy użytkownik przeglądający stronę WWW ma zainstalowane odpowiednie biblioteki (rozszerzenia). W przypadku technologii Flash nie stanowi to zbyt wielkiego problemu,

ponieważ takie rozszerzenie jest ogólnie dostępne. W przypadku nowszego formatu skalowalnej grafiki wektorowej — SVG (format ten omówimy w rozdziale 21. — „Technologia SVG) — znacznie mniej użytkowników będzie posiadać odpowiednie rozszerzenie, dlatego należy zapewnić alternatywne rozwiązanie przeglądania danych dla tych użytkowników, którzy nie mogą przeglądać formatu SVG. Można na przykład umieścić łącze pozwalające pobrać odpowiednie oprogramowanie, ale niektórzy użytkownicy, z różnych powodów, mogą nie chcieć z niego korzystać. W przypadku JavaScript, technologia ta jest od kilku lat wbudowaną funkcją większości głównych przeglądarek WWW.

Skrypty JavaScript są zazwyczaj bardzo zwięzłe, chociaż możemy opracować tak długie i złożone skrypty, jak tylko chcemy. Pamiętajmy jednak, że pliki dużych rozmiarów — co jest bardzo częste w przypadku technologii Flash — mogą zniechęcać wielu użytkowników, szczególnie w przypadku, gdy do połączenia z Internetem wykorzystują połączenia modemowe. Starajmy się więc, aby nasze skrypty JavaScript były jak najbardziej zwięzłe.

W niektórych przypadkach programista będzie potrzebował większych możliwości po stronie klienta niż te, które zapewnia język HTML oraz inne popularne technologie WWW. Pomocny okazał się język Java. Aplet jest niewielką aplikacją Javy (stąd ta nazwa), opracowany specjalnie z myślą o wykorzystaniu w środowisku przeglądarki WWW. Java to kompletny język programowania, działający na wielu platformach. Za jego pomocą można wykonywać tak złożone operacje, jak choćby usuwanie lub nadpisywanie plików, dlatego też aplety mają ograniczone możliwości (po to, aby nie były wykorzystywane do stwarzania zagrożeń bezpieczeństwa). Java stanowi logiczny wybór w tych środowiskach (jak niektóre środowiska intranetowe), w przypadku których jesteśmy pewni, iż przeglądarki wszystkich użytkowników mają możliwość wykonywania aplikacji Java. Wykorzystanie języka Java w Internecie może jednak stwarzać pewien kłopot, gdyż nie wszyscy użytkownicy włączają obsługę apletów Java w swoich przeglądarkach.

Aplety Java można bezproblemowo integrować ze stronami WWW. Na przykład powszechnie wykorzystuje się je do implementacji klientów chat uruchamianych w przeglądarkach WWW, z tego względu, że w języku Java istnieje możliwość utrzymywania połączenia z serwerem poprzez nasłuchiwanie nowych komunikatów, co nie jest możliwe w przypadku większości technologii skryptowych. Właściwość wykorzystania strumieni w języku Java powoduje, że jest to powszechnie wykorzystywana technologia w ośrodkach nieustannie uaktualnianych, np. w ośrodkach prezentujących wiadomości lub ceny akcji.

Wadą apletów Java, podobnie jak rozszerzeń programowych, jest to, że ich załadowanie może zajmować więcej czasu niż użytkownicy zechcą czekać. Java nie jest obsługiwana przez wszystkie przeglądarki, a niektórzy użytkownicy mogą również wyłączyć jej obsługę. W przypadku języka Java występuje także problem złożoności, szczególnie dla tych projektantów, którzy nie znają innych języków programowania. Java to kompletny język programowania, który rządzi się swoimi własnymi prawami. Z tego powodu wielu programistów wykonujących swoją pracę okazjonalnie i dorywczo rezygnuje z wykorzystywania tego języka. Doskonałym rozwiązaniem jest zatem język posiadający możliwości Javy, ale mający prostszą składnię i stawiający programistom mniejsze wymagania.

W 1995 r. Brendon Eich, zainspirowany projektem firmy Apple — HyperCard, rozpoczął pracę nad takim właśnie językiem. Początkowo firma Netscape wydała język pod nazwą LiveScript, który w zamyśle miał działać zarówno po stronie serwera, jak i po stronie klienta. Na rynku przeglądarek dominował wówczas Netscape. Szybko zdobywał zwolenników także język LiveScript, dzięki możliwości odciążenia serwera i przekazania części obciążenia do klienta.

Na początku 1996 roku, wraz z pojawieniem się przeglądarki Netscape w wersji 2, nazwę języka LiveScript zmieniono na JavaScript. Niestety, skojarzenie z językiem Java spowodowało — i w dalszym ciągu powoduje — wiele nieporozumień wśród początkujących programistów WWW. Otóż JavaScript to nie Java! Nie jest nawet okrojoną wersją tego języka. Pomimo podobieństw syntaktycznych i strukturalnych istnieją także znaczące różnice pomiędzy tymi dwoma językami.

Język Java został opracowany jako kompletny język programowania. Za jego pomocą można tworzyć oddzielny graficzny interfejs użytkownika (GUI). W przypadku języka JavaScript jest inaczej. Został on opracowany z myślą o działaniu w środowisku przeglądarki WWW razem z istniejącymi elementami HTML.

W języku JavaScript istnieje wprawdzie wiele elementów, których trzeba się nauczyć, ale mimo to łatwiej go opanować niż pełny język programowania. JavaScript można produktywnie wykorzystywać niemal od razu. Kod napisany w tym języku może być tak prosty lub tak złożony, jak sobie tego życzymy. W niektórych zastosowaniach możemy wykorzystywać JavaScript do wykonywania takich prostych zadań, jak umieszczenie kursora w pierwszym elemencie formularza, po to, aby użytkownik mógł rozpocząć wprowadzanie danych od razu po załadowaniu strony. Można również wykorzystać ten język do stworzenia złożonego menu nawigacyjnego wykorzystującego grafikę. Kiedy po przestudiowaniu tej książki nabędziemy umiejętności programowania w języku JavaScript, będziemy mogli sami decydować o tym, w jaki sposób chcemy wykorzystać ten język.

Jak już powiedzieliśmy, język JavaScript można wykorzystać po stronie serwera; ma jednak także kilka innych zastosowań. Najpopularniejsze wykorzystanie tego języka polega na użyciu go po stronie klienta na stronach HTML. Kod JavaScript jest pobierany wraz ze stroną HTML i uruchamiany w komputerze-kliencie (w komputerze, za pomocą którego użytkownik przegląda stronę WWW). To właśnie dzięki temu zastosowaniu można przekształcić statyczny kod HTML oraz statyczną grafikę w elementy interaktywnie współdziałające z użytkownikiem. Właśnie dzięki temu język JavaScript stał się tym, czym jest, czyli technologią powszechnie wykorzystywaną na stronach WWW.

Dzięki technologii JavaScript możemy przemieszczać elementy na stronie WWW oraz odpowiadać na różnorodne działania użytkowników.

JavaScript nie jest technologią odpowiednią do zastosowania w każdym projekcie WWW. Stanowi jednak naturalny wybór do wykonywania zadań sprawdzania poprawności danych w formularzach, rozpoznawania działań wykonywanych przez użytkownika (nawet takich, jak wykrywanie pozycji kursora myszy na ekranie), wykonywania operacji na stronie WWW oraz przetwarzania danych, których nie trzeba przesyłać na serwer.

Język JavaScript nie posiada żadnych wbudowanych możliwości graficznych, ale technologię tę coraz częściej wykorzystuje się do skryptów graficznych formatów wektorowych, jak Flash lub SVG. Zastosowanie języka JavaScript do obsługi formatu SVG zostanie przedstawione w rozdziale 21. — „Technologia SVG".

JavaScript, Jscript i ECMAScript

Język JavaScript ma kilka odmian, zatem jeżeli dopiero zaczynamy uczyć się tego języka, niektóre pojęcia mogą nas wprowadzać w błąd. Bardziej zaawansowani programiści często będą odnosić się do języków JavaScript, Jscript oraz ECMAScript. Spróbujmy zatem określić, czym dokładnie są te języki skryptowe i jakie są pomiędzy nimi różnice?

JavaScript

Jak wspomniano wcześniej, JavaScript jest nazwą wybraną przez firmę Netscape, twórcę technologii JavaScript, do określania pierwotnie zastrzeżonego języka skryptowego. W rzeczywistości, nazwa JavaScript była własnością firmy Netscape. Wersja 1.0 języka JavaScript pojawiła się w przeglądarce Netscape 2.0, natomiast Netscape 3 wyposażono w język JavaScript w wersji 1.1.

JScript

Nie potrzeba było wiele czasu, aby firma Microsoft doceniła potencjał języka JavaScript i wyposażyła przeglądarkę Microsoft Internet Explorer w wersji 3 w swoją implementację tego języka. Implementację tę określono nazwą JScript, ponieważ wykorzystanie nazwy JavaScript naruszyłoby prawa własności firmy Netscape do tej technologii. Jednakże, pomimo iż język Jscript wykorzystuje własny interpreter oraz w pewnym stopniu różni się od JavaScript, to jednak oba te języki są do siebie podobne. Z punktu widzenia programisty, ktoś, kto nauczył się jednego z tych języków, w dużej mierze zna też drugi z nich. Chociaż w pewnym momencie firma Microsoft preferowała wykorzystywanie języka VBScript zamiast JavaScript, to jednak ten drugi pozostał głównym językiem wykorzystywanym przez programistów do pisania skryptów po stronie klienta. Częściowo wynikało to z tego, że początkowo przeglądarka Netscape była bardziej popularna, ale jeszcze ważniejszym powodem był fakt, że język JavaScript był dostępny w większej liczbie przeglądarek niż JScript. Obecnie rzadko spotyka się skrypty VBScript na stronach WWW, z tego prostego względu, że język ten obsługują wyłącznie przeglądarki firmy Microsoft. Może nieco dziwić to, że wraz ze zmniejszeniem popularności języka JavaScript do tworzenia skryptów po stronie klienta, wzrosła popularność języka VBScript w technologii skryptów po stronie serwera — ASP.

Po stronie klienta, pomiędzy językami JavaScript a JScript istniało tak dużo różnic, że pisanie skryptów przeznaczonych do wykorzystania na wielu platformach nie było zbyt łatwe. Specyfikację JavaScript przekazano do Europejskiego Stowarzyszenia Producentów Komputerów (ECMA) w celu stworzenia „standardu". Tak powstał język ECMAScript.

ECMAScript

Wraz z pojawieniem się 4. wersji przeglądarek Netscape Navigator oraz Internet Explorer, w obu spróbowano zaimplementować standardową wersję języka ECMAScript. Wersja zaimplementowana w przeglądarce Internet Explorer okazała się bliższa standardu niż implementacji firmy Netscape i fakt ten, jako jeden z kilku czynników, przyczynił się być może do ekspansji przeglądarki Microsoftu, która zdominowała rynek przeglądarek WWW.

Europejskie Stowarzyszenie Producentów Komputerów *ECMA* (*European Computer Manufacturer Association*) jest międzynarodowym stowarzyszeniem przemysłowym. Organizacja ta opracowała kilka standardów branży komputerowej, z których większość można pobrać za darmo ze strony *http://www.ecma.ch*. ECMAScript jest nazwą, która została zaadoptowana do standardu ECMA-262, opracowanego po tym, jak firma Netscape zaproponowała język JavaScript w wersji 1.1 jako standard. ECMAScript to de facto międzynarodowy standard języka JavaScript. W czasie powstawania tej książki opracowano trzecią wersję tego standardu, która odpowiadała językowi JavaScript w wersji 1.5 oraz JScript w wersji 5.5.

Należy zdać sobie sprawę z tego, że ECMAScript stanowi standard dla rdzenia języka JavaScript. Jako rdzeń rozumiemy te właściwości języka, które istnieją niezależnie od środowiska, w którym wykorzystujemy JavaScript (przeglądarka WWW, PDF itp.). Obejmuje to takie właściwości, jak funkcje obliczeń arytmetycznych oraz obsługi dat. Te elementy języka nie wchodzą w interakcje oraz nie zależą od przeglądarki.

Elementy umożliwiające komunikację języka JavaScript ze środowiskiem hosta są przedmiotem innego standardu, który określany jest mianem *W3C DOM* (*World Wide Web Consortium Document Object Model*, czyli Obiektowy Model Dokumentu Konsorcjum WWW). Standard ten jest istotny dla języka HTML, dlatego też mogliśmy się z nim spotkać już wcześniej. Oprócz opisu metody zapisu dokumentów HTML, standard ten określa, w jaki sposób można uzyskać dostęp do elementów dokumentów HTML z poziomu języka JavaScript i jak wykonywać na nich operacje. Ten aspekt języka omówiono w dalszej części tej książki.

W praktyce ECMAScript nie jest językiem, którego powinni się uczyć programiści piszący skrypty JavaScript. Jest on przeznaczony dla projektantów przeglądarek, którzy powinni zadbać, aby ich implementacja JavaScript zawierała te same funkcje, co implementacje w innych przeglądarkach.

Język ECMAScript stanowi istotny krok na drodze do większej jednolitości języka JavaScript. W JavaScript zawsze znajdą się elementy, które nie będą częścią specyfikacji ECMAScript, częściowo po to, aby zapewnić wsteczną zgodność ze starszymi przeglądarkami. Język JavaScript jest zgodny z ECMAScript, a ponadto zawiera dodatkowe właściwości.

Sporo miejsca zajęło jak dotąd opisanie tła języka JavaScript; teraz przejdziemy do omówienia sposobu wykorzystania samego języka.

Zaczynamy

Aby stać się autorem skryptów JavaScript, potrzebujemy dwóch narzędzi: edytora tekstowego oraz przeglądarki WWW.

Wybór edytora tekstowego i przeglądarki WWW

Zapewne do tworzenia stron WWW przyzwyczailiśmy się wykorzystywać edytory typu *WYSIWYG* (*What You See Is What You Get*, czyli „otrzymasz to, co widzisz"), jak: Adobe GoLive, Microsoft FrontPage lub NetObjects Fusion. Jednak, aby tworzyć skrypty, będziemy musieli przyzwyczaić się do ręcznego wpisywania kodu, chociaż takie edytory, jak Adobe GoLive oraz Macromedia DreamWeaver zawierają wbudowane mechanizmy służące do tworzenia skryptów JavaScript, z których można korzystać bez konieczności rozumienia kodu generowanego przez program. Zazwyczaj edytory WYSIWYG generują miłe dla oka efekty, ale jeżeli nie rozumiemy języka JavaScript, który leży u podstaw tych efektów, nie będziemy w stanie ich zmodyfikować, a tym samym uzyskać dokładnie takich wyników, jakich oczekujemy.

Opisane w tej książce przykłady umieszczono dla wygody na dołączonej do niej płycie CD-ROM. Jeżeli jednak wpiszemy kod samodzielnie, zapamiętamy o wiele więcej. Zachęcamy zatem, aby to robić. Nawet te sytuacje, w których popełnimy błędy przy wpisywaniu kodu, pomogą nam poznać informacje potrzebne do tego, aby stać się wydajnym autorem skryptów.

Bez względu na to, na jakie rozwiązanie się zdecydujemy, jeśli chcemy napisać własne skrypty, będzie nam potrzebne narzędzie do ich wpisywania, czyli edytor tekstów.

Kiedy zapisujemy tekst w procesorze[1] tekstu, jest on zapisywany w pliku, który zawiera wiele kodów formatowania. Przeglądarki nie potrafią obsługiwać takich kodów. Do zapisania skryptu możemy zatem wykorzystać procesor tekstu, ale powinniśmy pamiętać, aby zapisać plik w formacie zwykłego tekstu (ASCII). Często jest z tym więcej problemów niż jest to warte. Z drugiej strony, edytor tekstowy zapisuje tekst w formacie ASCII, bez dodawania zastrzeżonych kodów, które są niezrozumiałe dla przeglądarki WWW. Posługując się edytorem tekstu można również w stosunkowo prosty sposób zapisać pliki z rozszerzeniem *.htm* lub *.html*.



W tej książce będziemy konsekwentnie wykorzystywać rozszerzenie *.htm*. Przeglądarki na 32-bitowych platformach Windows traktują pliki z rozszerzeniem *.htm* tak samo, jak pliki z rozszerzeniem *.html*. Na niektórych platformach jednak dozwolone jest wykorzystanie jedynie trzyliterowych rozszerzeń. Jeżeli wykorzystujemy taką platformę, wówczas będziemy mogli posługiwać się jedynie rozszerzeniem *.htm*.

Użytkownicy Windows z pewnością mają zainstalowaną jakąś wersję Notatnika. Aby go uruchomić, należy kliknąć ikonę *Notatnik* w menu *Start*, *Programy*, *Akcesoria*. Notatnik

[1] Przez „procesor" rozumie się tutaj złożony edytor tekstowy, taki jak Microsoft Word, zaś „edytorem" nazywany jest zwykle prosty edytor w rodzaju systemowego Notatnika Windows — *przyp. tłum.*

będzie odpowiednim narzędziem do tworzenia większości skryptów JavaScript, ale jeżeli pliki osiągną zbyt duże rozmiary, możemy skorzystać z edytora WordPad lub pobrać bardziej zaawansowany edytor tekstu np. TextPad. Dla użytkowników systemu MacOS wystarczy na początek edytor SimpleText, ale w miarę tworzenia dłuższych i bardziej skomplikowanych skryptów może przydać się edytor posiadający więcej funkcji, jak choćby BBEdit.

Większość zrzutów ekranów w tej książce wykonano posługując się Notatnikiem oraz przeglądarką Netscape 6 dla platformy Windows. Czytelnicy mogą pracować w innym systemie operacyjnym, np. MacOS lub jednej z odmian systemu UNIX, zatem wyniki uzyskane na ekranie mogą się nieco różnić od pokazanych w książce. Jeżeli uzyskamy nieco inny wynik, nie powinniśmy się tym zbytnio przejmować. Różne przeglądarki na różnych platformach wyświetlają elementy w nieco odmienny sposób, ale dane pozostają te same.

Po utworzeniu skryptów należy je oczywiście przetestować, aby upewnić się, że działają oraz działają tak, jak tego oczekujemy. Fakt, iż Czytelnicy interesują się nauką JavaScript świadczy o tym, że z pewnością potrafią posługiwać się przeglądarkami WWW, np. Microsoft Internet Explorer lub Netscape Navigator. O ile tylko nasza przeglądarka potrafi obsługiwać skrypty JavaScript i ma włączoną obsługę tego języka, na tym etapie wybór konkretnej przeglądarki nie ma znaczenia. Zarówno przeglądarka Netscape w wersji 6, jak i dowolna z ostatnich wersji przeglądarki Internet Explorer pozwoli na przeglądanie wyników działania naszych skryptów.

Podczas nauki języka JavaScript dobrze jest mieć dostęp do kilku przeglądarek WWW, aby móc przetestować kod we wszystkich tych przeglądarkach. Podczas przeglądania skryptów w wielu przeglądarkach dobrze jest znać różnice w wyglądzie stron WWW oraz w sposobie działania elementów interaktywnych. Wiedząc o tych różnicach, będziemy zdawać sobie sprawę z tego, co widzą użytkownicy, którzy odwiedzają nasze strony WWW.

Oto lista niektórych dostępnych obecnie przeglądarek WWW wraz z adresami URL, pod którymi możemy je pobrać, jeżeli jeszcze nie zainstalowaliśmy ich w naszym komputerze:

- Internet Explorer. Najnowsza wersja tej przeglądarki to 6.0. Można ją pobrać pod adresem *http://www.microsoft.com/windows/ie/*. Jeżeli dysponujemy wcześniejszą wersją przeglądarki Internet Explorer i chcemy ją uaktualnić, powinniśmy otworzyć przeglądarkę Internet Explorer, przejść do menu *Narzędzia* i wybrać polecenie *Windows Update*, co spowoduje zainicjowanie procesu uaktualniania naszej przeglądarki. Jeżeli nigdy wcześniej tego nie robiliśmy, bądźmy przygotowani na to, że może to trochę potrwać, gdyż może się zdarzyć, że przed pobraniem nowej wersji przeglądarki Internet Explorer będziemy zmuszeni pobrać kilka „kluczowych" aktualizacji.
- Netscape Navigator 4.7 można pobrać pod adresem *http://home.netscape.com/browsers*. Ze strony tej można także uzyskać dostęp do szeregu przeglądarek firmy Netscape.
- Przeglądarkę Netscape Navigator 6.2 można znaleźć pod adresem *http://home.netscape.com/browsers/*.

- Przeglądarkę Mozilla można pobrać pod adresem *http://mozilla.org/releases/*.
- Opera 6 jest dostępna pod adresem *http://www.opera.com/download/*.



Aby korzystać z przeglądarki, nie musimy być połączeni z Internetem. Strony WWW zapisane w naszym komputerze można ładować i przeglądać lokalnie.

Podstawowe sprawy do zapamiętania

Przedstawimy teraz pewne podstawowe zasady, o jakich należy pamiętać podczas tworzenia skryptów w języku JavaScript. Z czasem będziemy je stosować niemal odruchowo, ale zanim to nastąpi, powinniśmy przestrzegać ich świadomie w trakcie pisania skryptów.

Rozróżnianie wielkich i małych liter

W języku JavaScript wielkość liter ma znaczenie. Stosując zatem wielkie i małe litery w tworzonych nazwach zmiennych lub stałych języka JavaScript, należy być bardzo uważnym i zachować konsekwencję. Bardzo popularny błąd, jaki przytrafia się początkującym programistom, polega na nadaniu nazwy zmiennej lub funkcji, a następnie ponownym jej zapisaniu, ale z różną wielkością liter dla jednego lub więcej znaków. W krótkim skrypcie taki błąd jest łatwy do zauważenia, ale w dłuższych skryptach będzie bardzo trudno wykryć, skąd on pochodzi. Tak więc, jeżeli nadamy jakiemuś elementowi nazwę „Element", musimy za każdym razem odwoływać się do niego wykorzystując tę samą wielkość liter. Jeżeli tego nie zrobimy, wystąpi błąd lub uzyskamy nieoczekiwane wyniki.

Średniki

Dobrą praktyką jest zwyczajowe kończenie każdej instrukcji JavaScript średnikiem. Nie stanowi to wymogu; w rzeczywistości jest potrzebne tylko wtedy, gdy chcemy umieścić co najmniej dwie instrukcje JavaScript w jednym wierszu. Często tak się zdarza w czasie tworzenia konstrukcji pętli `FOR`, którą opisano w dalszych rozdziałach.

Znaki końca wiersza

Dzięki znakom końca wiersza nasz kod stanie się bardziej czytelny. Im dłuższy kod, tym bardziej istotna jest jego czytelność.

Odstępy

Znaki odstępów (spacje, tabulatory, znaki końca wiersza) nie są widoczne na ekranie, lecz tworzą odstępy w kodzie. Oprócz tych spacji, które są potrzebne do oddzielenia poszczególnych elementów kodu JavaScript, interpreter języka ignoruje nadmiarowe odstępy. Tak więc kod:

```
x = y + z
```

można zapisać jako:

```
x    =    y    +    z
```

Chociaż interpreter języka JavaScript dopuszcza wpisywanie kodu w ten sposób, to jednak zapis taki powoduje, że dla większości użytkowników czytanie i rozumienie kodu staje się trudniejsze. Zamysłem, który kryje się za ignorowaniem nadmiarowych odstępów w języku JavaScript jest umożliwienie wpisywania kodu w taki sposób, aby można było łatwo wydzielić główne elementy struktur kodu. Dzięki zastosowaniu odstępów podczas tworzenia struktury naszego kodu, stanie się on dla nas bardziej czytelny. Na przykład, jeżeli mamy fragment kodu zapisany pomiędzy nawiasami klamrowymi:

```
{kod
kod
kod}
```

to możemy wykorzystać odstępy, aby kod ten stał się bardziej czytelny:

```
{
   kod
   kod
   kod
}
```

Dzięki temu, że nawiasy okrągłe — otwierający i zamykający — są zapisane w oddzielnych wierszach, w obu przypadkach na początku wiersza, możemy z łatwością wskazać, gdzie taki blok kodu się zaczyna, a gdzie kończy. W trakcie przeglądania dużego fragmentu kodu JavaScript możemy łatwo stwierdzić, jaka instrukcja należy do jakiego bloku.

Gdzie umieszczamy kod JavaScript

JavaScript, podobnie jak HTML, jest zwykłym tekstem, który można wpisywać bezpośrednio w edytorze tekstowym, takim jak np. Notatnik. Jednak, aby interpreter języka JavaScript poprawnie rozpoznawał i właściwie przetwarzał nasz kod, musimy umieścić go w miejscu, w którym interpreter języka przeglądarki WWW się go spodziewa.

Istnieją trzy miejsca, gdzie możemy umieścić skrypty JavaScript na naszych stronach HTML:

- w bloku skryptu wpisywanym wewnątrz kodu HTML,
- w oddzielnym pliku zawierającym kod JavaScript,
- w obrębie otwierającego znacznika wielu elementów HTML.

Bloki skryptów

Wpisywanie kodu JavaScript w dokumencie HTML za pomocą bloków skryptów jest najprostszym sposobem wstawiania kodu JavaScript. Co to jest blok skryptów?

Mówiąc najprościej, blok skryptów to fragment kodu JavaScript otoczony parą znaczników HTML `<script>` — otwierającego znacznika `<script>`, poprzedzającego kod JavaScript oraz zamykającego znacznika `</script>`, który następuje po zakończeniu

kodu. Znacznika <script> można użyć wewnątrz kodu HTML w celu oznaczenia obszaru kodu JavaScript dokładnie w taki sposób, w jaki możemy wykorzystać znacznik <p> do oznaczenia akapitu lub <form> do oznaczenia formularza. Przykład wykorzystania znacznika <script> pokazano na wydruku 1.1.

źródło **Wydruk 1.1.** *Szkielet kodu JavaScript (skeleton.htm)*

```
<html>
<head>
<title>Tytuł dokumentu</title>

<script>

// KOD JAVASCRIPT ZNAJDZIE SIĘ
// TUTAJ, POMIĘDZY OTWIERAJĄCYM,
// A ZAMYKAJĄCYM ZNACZNIKIEM
// SCRIPT.

</script>

</head>
<body>

Tu będzie treść
naszej strony.

</body>
</html>
```

Blok skryptów można umieścić niemal w każdym miejscu pomiędzy otwierającym a zamykającym znacznikiem HTML naszej strony WWW. Jeżeli jednak umieszczenie bloku skryptów w konkretnej części strony HTML nie jest konieczne, najlepiej umieścić blok skryptów pomiędzy otwierającym, a zamykającym znacznikiem <head> oraz po tytule i wszystkich metaznacznikach, które wykorzystujemy. Zapewnia to, że skrypty będą w pełni załadowane, zanim strona pojawi się w przeglądarce. Pozwala to na uniknięcie wielu błędów, jakie mogą się pojawić, kiedy użytkownik próbuje wywołać skrypt, który nie został jeszcze załadowany.

Spójrzmy na efekt wyświetlenia strony w zależności od miejsca, w którym umieściliśmy skrypt. W tym celu w wybranym edytorze tekstów wpiszemy kod pokazany na wydruku 1.2.

źródło **Wydruk 1.2.** *Tworzenie — za pomocą skryptu JavaScript — okienka z ostrzeżeniem (firstAlert.htm)*

```
<html>
<head>

<script>

alert("Czy widzisz nagłówek strony?");

</script>
```

```
</head>
<body>

<h1>Nagłówek strony</h1>

</body>
</html>
```

wynik Ten kod spowoduje wyświetlenie wyniku pokazanego na rysunku 1.1.



Rysunek 1.1. *Ostrzeżenie wyświetlane przez blok skryptów JavaScript umieszczony w nagłówku strony*

Spróbujmy teraz przemieścić blok skryptów za nagłówek strony, tak jak na wydruku 1.3.

źródło **Wydruk 1.3.** *Okno ostrzeżenia utworzone przez JavaScript w obrębie treści HTML (secondAlert.htm)*

```
<html>
<head>

</head>
<body>

<h1>Nagłówek strony</h1>

<script>

alert("Czy widzisz nagłówek strony?");

</script>

</body>
</html>
```

wynik Powinniśmy teraz uzyskać taki wynik, jak na rysunku 1.2.



Rysunek 1.2. *Ostrzeżenie wyświetlane przez blok skryptów JavaScript umieszczony w treści HTML*

analiza Zwróćmy uwagę na różnicę spowodowaną umieszczeniem bloku skryptów w innym miejscu. Kiedy blok skryptów znajdował się w nagłówku dokumentu, okno ostrzeżenia wyświetlane było przed załadowaniem tekstu „Nagłówek strony" do przeglądarki. Poprzez przemieszczenie bloku skryptów do treści dokumentu HTML, poniżej nagłówków `<h1>`, tekst „Nagłówek strony" został załadowany w przeglądarce, zanim wyświetlone zostało okno ostrzeżenia skryptu JavaScript.



Wiersze dokumentu HTML są ładowane od góry do dołu i każdy wiersz ładowany jest od lewej do prawej. Z tego powodu instrukcje znajdujące się wcześniej w bloku kodu będą też wcześniej załadowane.



Pamiętajmy, że znaczniki `<script>` są parą znaczników HTML lub XHTML; należy zatem pamiętać o umieszczeniu zamykającego znacznika `</script>`.

Zewnętrzne pliki skryptów JavaScript

Kod JavaScript, który zazwyczaj umieszcza się pomiędzy znacznikami `<script>`, można także zapisać w oddzielnym pliku tekstowym. Pomimo że jest to plik tekstowy, należy unikać nadawania mu rozszerzenia *.txt*. Aby taki plik mógł być poprawnie rozpoznawany przez interpreter, powinien mieć rozszerzenie *.js*. Jeśli tylko zewnętrzny plik skryptu ma rozszerzenie *.js*, może być włączony do strony HTML i zawarty w nim kod będzie wykonany z identycznym skutkiem, jak gdyby był wpisany w treści strony.

W celu włączenia skryptu JavaScript, należy dodać atrybut `src` w obrębie otwierającego znacznika `<script>`.

Otwierający znacznik `<script>` będzie miał wtedy następującą postać:

```
<script src="MojPierwszyZewnetrznySkrypt.js">
```

Pomiędzy otwierającym a zamykającym znacznikiem nie należy wprowadzać żadnego tekstu, zatem najlepiej wprowadzić zamykający znacznik </script> bezpośrednio za znacznikiem otwierającym, w następujący sposób:

```
<script src="MojPierwszyZewnetrznySkrypt.js"></script>
```



W języku XHTML istnieje opcja pozwalająca na użycie pustego znacznika script. Ma on postać <script />. Niestety, w niektórych przeglądarkach użycie tego znacznika powoduje błąd, zatem bezpieczniej jest użyć pary znaczników <script></script>.

Przykład przedstawiono na wydruku 1.4. Pokazane nań łącze nie prowadzi do rzeczywistego pliku. Zaprezentowany kod ma jedynie na celu zademonstrowanie składni.



Wydruk 1.4. *Strona HTML, która korzysta z zewnętrznego pliku zawierającego skrypt JavaScript (external.htm)*

```
<html>
<head>
<title>Tytuł dokumentu</title>

<script src="sciezka/do/pliku/nazwapliku.js"></script>

</head>
<body>

Tutaj znajdzie się
zawartość strony.

</body>
</html>
```

Większość kodu zaprezentowanego w tej książce będzie przedstawiona w blokach skryptów, ponieważ w ten sposób łatwiej uczyć nowego tematu. Jednak w praktyce, kiedy już osiągniemy poziom pozwalający na przystąpienie do tworzenia rzeczywistego kodu, przekonamy się, że istnieją powody, dla których warto zapisywać skrypty w plikach zewnętrznych. Powody te omówione zostaną już wkrótce, w sekcji „Wielokrotne wykorzystywanie kodu"; na razie przedstawimy kilka nowych informacji dotyczących zastosowania znaczników <script>.

W obrębie początkowego znacznika HTML

Umieszczenie kodu JavaScipt w obrębie otwierającego znacznika HTML to przypadek specjalny. Jedynymi konstrukcjami języka JavaScript, które wykorzystujemy w ten sposób, są procedury obsługi zdarzeń. Procedury obsługi zdarzeń to zagadnienie dość zaawansowane. Zaprezentowano je w rozdziale 10. — „Zdarzenia i ich obsługa". Na razie zignorujemy po prostu ten aspekt, do czasu, kiedy lepiej poznamy język JavaScript.

Określenie języka skryptowego

Obecnie domyślnym językiem skryptowym dla wszystkich przeglądarek jest JavaScript, tak więc przeglądarki HTML będą poprawnie interpretować kod bez konieczności informowania, że napisano go przy użyciu JavaScriptu. Istnieje jednak powód, dla którego

warto zdefiniować używany przez nas język. Otóż domyślny język skryptowy może się w przyszłości zmienić. W ciągu najbliższych kilku lat mogą też nastąpić poważne zmiany w przeglądarkach WWW. Coraz częściej wykorzystuje się język *XML* (Rozszerzalny Język Znaczników — ang. *Extensible Markup Language*), rośnie także popularność wykorzystywania przeglądarek przenośnych. Na razie jednak nic nie wskazuje na to, aby w którejkolwiek z przeglądarek miał się zmienić domyślny język skryptowy.

Aby zabezpieczyć się przed skutkami ewentualnej zmiany domyślnego języka skryptowego, przynajmniej w niektórych przeglądarkach, powinniśmy nabrać nawyku wprowadzania w otwierającym znaczniku `<script>` dwóch atrybutów: `language` oraz `type`.

Do niedawna atrybut `language` był zatwierdzonym sposobem informowania przeglądarki, że skrypt napisano w języku JavaScript. Chociaż tak już nie jest, to jest to jedyna metoda wykonania tej czynności dla przeglądarek w wersji 4 i starszych. Zatem dla zachowania zgodności z poprzednimi wersjami, w dalszym ciągu konieczne jest wykorzystanie atrybutu `language` w otwierającym znaczniku `<script>`, za każdym razem, kiedy rozpoczynamy wpisywanie skryptu.

Atrybut `language` wpisuje się w taki sam sposób, jak inne atrybuty HTML, np. atrybut `bgColor`. W przypadku języka JavaScript należy określić wartość tego atrybutu jako `"javascript"`.

W tym przypadku znaki *J* oraz *S* nazwy JavaScript nie są zapisane wielkimi literami. Dzięki temu będziemy mogli wykorzystać nasze skrypty w dokumentach XHTML. Język XHTML, podobnie jak JavaScript, rozróżnia wielkie i małe litery i z tego powodu ciąg `"javascript"` (wszystkie litery małe) jest jedyną formą, której można użyć w dokumencie XHTML.



Nazwa XHTML pochodzi od angielskiej nazwy eXtensible HyperText Markup Language (Rozszerzalny Hipertekstowy Język Znaczników). Język XHTML w wersji 1.0 odpowiada językowi HTML w wersji 4.0, która została zapisana za pomocą składni języka XML, a nie SGML. W składni SGML zapisane są wszystkie wersje języka HTML.

Przy założeniu, że zastosowaliśmy same małe litery, otwierający znacznik `script` powinien przyjąć następującą postać:

```
<script language="javascript">
```

lub — w przypadku wykorzystania pliku zewnętrznego:

```
<script src="sciezka/do/pliku/nazwapliku.js" language="javascript"
```

Nowe wersje przeglądarek, np. Microsoft Internet Explorer w wersji 5 i nowszych oraz Netscape Navigator w wersji 6 i nowszych, będą jeszcze przez jakiś czas obsługiwały atrybut `language`, ale wydaje się prawdopodobne, że z upływem czasu atrybut ten zniknie.

Atrybut type

W specyfikacji języków HTML 4.0 oraz XHTML 1.0 wewnątrz otwierającego znacznika `script` występuje dodatkowy atrybut — `type`. Powodem wprowadzenia tego atrybutu jest ustanowienie standardu co do sposobu opisu typu zawartości plików, do których

można uzyskać dostęp w sieci. Typy plików określane są terminem **typ medium** (ang. *media type*); każdy z nich posiada własny, standardowy sposób opisu. Jeżeli typ medium to `"text/javascript"`, wówczas przeglądarka WWW lub inna aplikacja WWW będzie „wiedzieć", że ma do czynienia z kodem JavaScript. Poprzez dostęp do atrybutu `type` można sprawdzić, w jaki sposób można obsłużyć zawartość pliku.

Jeżeli korzystamy z języka JavaScript, atrybut `type` dla znacznika `script` przyjmuje wartość `"text/javascript"`.

Aby uzyskać maksymalną zgodność pomiędzy określonymi przeglądarkami, najlepiej wykorzystać zarówno atrybut `language`, jak i `type`. Dzięki temu zapewnimy nie tylko to, że nasz stary skrypt będzie można wykorzystać w starych przeglądarkach, ale również to, że w przyszłości nie będzie trzeba powracać do kodu i nie zaistnieje konieczność wprowadzania atrybutu `type`.

Jeżeli powrócimy do przykładowego kodu bloku skryptów, to będzie on miał teraz następującą postać:

```
<script language="javascript" type="text/javascript">
// KOD JAVASCRIPT ZNAJDZIE SIĘ
// TUTAJ, POMIĘDZY OTWIERAJĄCYM,
// A ZAMYKAJĄCYM ZNACZNIKIEM
// SCRIPT.
</script>
```

Stare przeglądarki

Aby uzyskać zgodność z bardzo starymi przeglądarkami, powinniśmy wprowadzić dodatkowy, ostatni element. Jest on szczególnie przydatny, jeżeli wykorzystujemy bloki skryptów, które mogą być przeglądane za pomocą starych przeglądarek. Powinniśmy dodać komentarze do kodu JavaScript po to, aby zabezpieczyć się przed wyświetlaniem kodu na stronie WWW przez bardzo stare przeglądarki, które nie potrafią rozpoznać kodu JavaScript. Niemal na pewno wyświetlanie kodu na stronie nie jest naszym zamiarem.

Problem ten wiąże się z tym, że naprawdę stare przeglądarki nie obsługują znacznika `<script>`. Przyjrzyjmy się pokrótce sytuacji, w której przeglądarka nie potrafi zinterpretować znacznika. Wpiszemy kod HTML pokazany na wydruku 1.5 i po zapisaniu go w pliku zobaczymy, w jaki sposób jest wyświetlany w przeglądarce.

źródło **Wydruk 1.5.** *Co się dzieje, jeżeli przeglądarka nie rozpoznaje znacznika (nonExistentTag.htm)*

```
<html>
<head>
<title>Tytuł dokumentu</title>

<nonExistentTag>

WPISZMY W TYM MIEJSCU DUŻO TEKSTU. NIE MA ZNACZENIA CO TO JEST, BYLE TYLKO BYŁO
TEGO DUŻO.

JAK ZAUWAŻYLIŚMY, TEN TEKST ZNAJDUJE SIĘ W NAGŁÓWKU DOKUMENTU. NIE CHCEMY CHYBA,
ABY TEN TEKST NAPRAWDĘ POJAWIŁ SIĘ NA STRONIE. ZGADZA SIĘ?

</nonExistentTag>
```

```
</head>
<body>

<h1>Tytuł dokumentu</h1>

<p>Może w tym miejscu dopiszemy co nieco.</p>

</body>
</html>
```

wynik Jeżeli spróbujemy uruchomić ten skrypt, na ekranie pojawi się obraz przedstawiony na rysunku 1.3.



Rysunek 1.3. *Zawartość nierozpoznanych znaczników pojawi się na stronie WWW*

analiza Jak widać, przeglądarka wyświetliła tekst, który był wpisany pomiędzy nierozpoznanym znacznikiem otwierającym i znacznikiem zamykającym. To samo zdarzy się w starszych przeglądarkach, które nie rozpoznają znacznika `<script>`. Znacznik ten będzie w nich zignorowany, a przeglądarka wyświetli tekst skryptu. Nawet jeżeli skrypt nie uruchomi się, może to spowodować bałagan na stronie, która w innym przypadku wyglądałaby dobrze.

Na szczęście istnieje drobna sztuczka, która wykorzystuje komentarze HTML. Możemy ją użyć po to, aby upewnić się, że tekst skryptu nie będzie wyświetlony na stronie. Spójrzmy na kod przedstawiony na wydruku 1.6.

źródło **Wydruk 1.6.** *Wykorzystanie komentarzy HTML w celu uniknięcia błędów w starszych typach przeglądarek (hideScript.htm)*

```
<html>
<head>
<title>Ukrywanie skryptów za pomocą komentarzy.</title>
<script language="javascript" type="text/javascript">
```

```
<!--

Tu będzie kod JavaScript
...

//-->

</script>

</head>
<body>

Tu będzie treść strony...

</body>
</html>
```

analiza Jeżeli przyjrzymy się temu kodowi dokładnie, zauważymy, że na wydruku użyto dwóch komentarzy HTML, które zagnieżdżono wewnątrz znaczników `script`. Oznacza to, że w przypadku, gdyby znaczniki `<script>` zostały zignorowane (w przeglądarkach, które takich znaczników nie rozpoznają), cały kod znajdujący się pomiędzy komentarzami będzie ukryty, ponieważ przeglądarka nie wyświetla treści umieszczonej w komentarzu HTML. Jest to dosyć sprytne rozwiązanie, ale zastanówmy się, co się stanie z kodem JavaScript w przypadku, gdy będzie on interpretowany przez nowsze przeglądarki, które rozpoznają znaczniki `<script>`. Czy wprowadzone komentarze HTML mogą spowodować błędy, jeżeli zostaną użyte wewnątrz znaczników `<script>`? Otóż nie; interpreter JavaScript wie, że nie powinien interpretować otwierającego komentarza HTML oraz dwóch znaków ukośnika, aż do napotkania znacznika zamykającego komentarz (wkrótce dowiemy się, dlaczego).

Zatem, jeżeli umieścimy komentarze HTML pomiędzy znacznikami `<script>` oraz `</script>`, to nasz kod będzie działał normalnie i jednocześnie zabezpieczymy się przed ewentualnym wyświetleniem kodu na stronie.

Wielokrotne wykorzystywanie kodu

Zanim przeanalizujemy przykład kodu JavaScript, przyjrzyjmy się powodom, dla których warto zapisywać kod JavaScript w zewnętrznych plikach *.js*, które potem włączamy do strony.

Dla niewielkich fragmentów kodu, które są typowe dla osobistych stron WWW, prawdopodobnie lepiej jest zapisywać kod JavaScript w samej treści strony. Jeśli fragmenty kodu są krótkie, przechowywanie wszystkiego w jednym pliku powoduje, że odszukanie kodu i pielęgnacja skryptu są łatwiejsze.

Jeżeli jednak istnieje prawdopodobieństwo, że ten sam fragment kodu może być wykorzystany nie tylko w jednym dokumencie, lepiej zapisać kod JavaScript w pliku zewnętrznym.

Pierwszą korzyścią płynącą z umieszczenia kodu na zewnątrz jest fakt, że w sytuacji, kiedy trzeba zmodyfikować skrypt (jest niemal pewne, że w pewnym momencie taka potrzeba nastąpi), nie będzie konieczne, by wprowadzać wiele razy te same zmiany na każdej stronie, która korzysta ze skryptu. Dzięki temu zyskujemy wiele czasu, który pochłonęłoby nam otwieranie każdego pliku, wklejanie zmian, zapisywanie, a następnie kopiowanie każdego pliku na serwer. Ponadto, nie musimy martwić się o to, aby zmiany na każdej ze stron zostały wykonane dokładnie tak samo.

Druga zaleta nie jest tak oczywista. Niektóre przeglądarki na czas trwania sesji z przeglądarką buforują zewnętrzne pliki *.js*, zatem poprzez umieszczenie kodu w zewnętrznym pliku *.js*. powodujemy, że odwiedzający nasze strony goście muszą pobrać pliki *.js* tylko raz, na czas trwania sesji. Tak więc, na przykład, jeżeli mamy w katalogu kilka plików wykorzystujących menu nawigacyjne, napisane w języku JavaScript, to pobieramy kod tylko raz i mamy możliwość jego wielokrotnego wykorzystania na kilku stronach. Ponieważ większość użytkowników Internetu w dalszym ciągu do nawiązania połączenia wykorzystuje modemy, skrócenie czasu pobierania stron jest bardzo istotne.

Dokumentowanie kodu

Kiedy zaczniemy tworzyć bardziej zaawansowane aplikacje JavaScript, wówczas zauważymy, że kod stopniowo będzie stawał się coraz bardziej rozbudowany i złożony.

Jeżeli nie będziemy dokumentować naszego kodu, zrozumienie go przez kogoś innego, bez analizowania wiersz po wierszu sposobu jego działania, będzie bardzo trudne. Nawet sam autor kodu może mieć trudności w jego zrozumieniu, kiedy po jakimś czasie potrzebne stanie się wprowadzenie zmian. Wpisywanie komentarzy w najważniejszych miejscach kodu może okazać się bardzo pomocne dla lepszego zrozumienia kodu i może uchronić nas przed frustracją oraz stratą czasu.

W języku JavaScript istnieją dwie metody wprowadzania komentarzy w kodzie; pozwalają one na dokumentowanie czynności wykonywanych przez kod. Pierwsza metoda polega na utworzeniu komentarza będącego pojedynczym wierszem, natomiast druga umożliwia utworzenie komentarzy składających się z wielu wierszy.

Aby utworzyć komentarz obejmujący pojedynczy wiersz, wystarczy wprowadzić dwa kolejne ukośniki (//). Interpreter języka JavaScript zignoruje wszystkie znaki, które następują po dwóch znakach ukośnika, aż do końca wiersza. Ukośniki zapowiadające komentarz można umieścić na początku wiersza lub w dowolnym jego miejscu, po fragmencie kodu. Oto przykłady zastosowania tego sposobu wprowadzania komentarzy:

```
//Ten cały wiersz jest komentarzem
var dynMenuWidth = 10 // Ten komentarz wprowadzono po fragmencie kodu
```

Czasami, dla odpowiedniego skomentowania fragmentu kodu, potrzeba więcej niż tylko jednego wiersza lub nawet dwóch wierszy. Moglibyśmy oczywiście wprowadzać dwa ukośniki na początku każdego wiersza komentarza, ale jest to trochę uciążliwe. W języku JavaScript istnieje inny, nieco mniej kłopotliwy sposób oznaczania dłuższego komentarza.

Aby rozpocząć komentarz, który obejmuje kilka wierszy, wprowadzamy znak ukośnika, po którym następuje gwiazdka (/*). W takim przypadku komentarz nie kończy się wraz z końcem wiersza. Interpreter języka JavaScript traktuje wszystko, co następuje po tym symbolu jako komentarz, aż do napotkania znaku gwiazdki, po którym występuje ukośnik (*/). Oto omawiana metoda zaznaczania komentarza:

```
/* Cały tekst znajdujący
się w takim bloku komentarza
jak ten będzie zignorowany. */
```



Sposób ten jest również bardzo użyteczny w sytuacjach, w których występuje potrzeba czasowego wyłączenia dużego fragmentu kodu. Jest to o wiele łatwiejsze niż wprowadzanie i późniejsze usuwanie dwóch ukośników rozpoczynających każdy wiersz.



Należy unikać zagnieżdżania komentarzy. Jeżeli mamy komentarze takie jak następujący:

```
/*
x=1;
y=2; /* ten komentarz zamknie komentarz zewnętrzny */
z=3;
*/
```

to symbol */ zamykający drugi komentarz spowoduje zamknięcie również pierwszego komentarza. Zatem, kiedy interpreter języka JavaScript napotka symbol */ w ostatnim wierszu kodu, wystąpi błąd.

Dla łatwiejszego rozpoznania początku i końca komentarza możemy wprowadzić na jego początku i na końcu wiersz składający się w całości z jednego znaku, np. myślnika. O ile wiersze te znajdują się w bloku komentarza, interpreter języka JavaScript zignoruje je.

```
/*
-----------------------------------------

    Cały tekst znajdujący
    się w takim bloku komentarza, a także
    wiersze powyżej i poniżej będą zignorowane

-----------------------------------------
*/
```

Jakkolwiek nie powinniśmy skąpić komentarzy, to jednak powinniśmy się starać, aby były one zwięzłe. Pamiętajmy, że są one pobierane wraz z pozostałą częścią skryptu za każdym razem, kiedy użytkownik żąda dowolnej z naszych stron.

Słowa zarezerwowane

W języku JavaScript występuje zbiór słów, które mają specjalne znaczenie dla interpretera języka. Oto te słowa:

```
abstract
boolean
break
byte
case
catch
char
class
const
continue
debugger
default
delete
do
double
else
enum
export
extends
false
final
finally
float
for
function
goto
if
implements
import
in
instanceof
int
interface
long
native
new
null
package
private
protected
public
return
short
static
super
switch
synchronized
this
throw
throws
transient
true
try
typeof
var
void
volatile
while
with
```

Na razie nie martwmy się tym, że nie wiemy, w jaki sposób korzystać z wymienionych tu słów kluczowych. Wraz z poznawaniem kolejnych rozdziałów, poznamy także te z nich, które będą nam potrzebne. Przedstawiając listę tych słów na początku książki chcemy poinformować Czytelników, że niektóre słowa są zastrzeżone i nie można ich wykorzystywać w kodzie w inny sposób niż zgodny z rolą, jaką pełnią w języku JavaScript.



W języku JavaScript wielkość liter ma znaczenie. Teoretycznie, można by zatem zapisać jeden ze znaków nazwy zastrzeżonej wielką literą po to, by wykorzystać słowo zastrzeżone do nadania nazwy innemu elementowi. Jest to jednak bardzo zła praktyka, która prowadzi do potencjalnych błędów.

Typy danych

W ostatniej części tego rozdziału opisano różne typy danych występujących w JavaScript; dowiemy się z niej także, kiedy będzie potrzebna konwersja typów danych.

Dane, czyli inaczej informacje wykorzystywane przez komputery, mogą mieć różnorodne postaci. Natychmiast przychodzą nam na myśl liczby, daty oraz tekst. W zależności od języka programowania wykorzystywanego do przetwarzania, dane zawierające różne rodzaje informacji mogą być traktowane w odmienny sposób. Ważne jest, aby wiedzieć, w jaki sposób używany przez nas język programowania interpretuje dane poszczególnych typów.

W niektórych językach programowania wykorzystuje się ścisłą kontrolę typów. Oznacza to, że kiedy w czasie pisania programu wprowadzamy określony element, to najpierw musimy zadeklarować jego typ. Dane muszą być potem traktowane zgodnie ze ściśle określonymi zasadami, które dotyczą tego konkretnego typu danych. Jeżeli złamiemy te zasady, wystąpi błąd. Na przykład, jeżeli na końcu instrukcji spróbujemy wpisać liczbę,

a korzystamy z języka programowania, który stosuje ścisłą kontrolę typów, komputer nie będzie potrafił określić, jaką czynność ma wykonać. Większość konwencjonalnych języków programowania, takich jak C lub C++, to języki o ścisłej kontroli typów.

Język JavaScript określić można jako język o dynamicznej kontroli typów. Ponadto JavaScript jest językiem o słabej kontroli typów. Jak można się domyślić, podczas wprowadzania elementu w skrypcie JavaScript nie musimy deklarować jego typu. Inną cechą języków o dynamicznej kontroli typów danych jest fakt, iż można w nich zmieniać typy w czasie wykonywania (od momentu uruchomienia skryptu do jego zakończenia).

Nie oznacza to jednak, że możemy zignorować typy danych. Po prostu JavaScript jest bardziej elastyczny od niektórych innych języków. Jeżeli nie będziemy dokładnie wiedzieć, w jaki sposób dane interpretowane są w języku JavaScript, to w dalszym ciągu możemy uzyskiwać nieoczekiwane wyniki. Na przykład zapis

```
10 + 0
```

będzie zinterpretowany przez JavaScript jako liczba `10`, natomiast

```
10 + "0"
```

interpreter języka JavaScript przekształci na ciąg `"100"`. Z dalszej części tego rozdziału dowiemy się, dlaczego tak się dzieje.

Liczby

W kompletnych językach programowania, takich jak Java czy C++, w zależności od rozmiaru oraz charakterystycznych cech, istnieje kilka typów danych reprezentujących liczby. Program przydziela określoną ilość pamięci, zależną od typu danych.

W języku JavaScript położono nacisk na łatwość pisania kodu. Z tego powodu różne rodzaje liczb nie są rozróżniane. Dzięki temu jedno ze źródeł możliwych błędów kodowania zostało wykluczone.

W języku JavaScript liczby mogą być interpretowane jako całkowite (np. liczba 10) lub zmiennoprzecinkowe (liczby z częścią ułamkową, np. 1,55). Z cała pewnością niezwykle wygodne jest to, że w języku JavaScript liczby w większości traktowane są jako zmiennoprzecinkowe, zatem nie musimy martwić się typami liczbowymi. Ponieważ aplikacje napisane w JavaScript są zazwyczaj niewielkie, różnica w wydajności jest niezauważalna.

Język JavaScript obsługuje dodatnie i ujemne liczby w zakresie od -2^{1024} do 2^{1024} (co w przybliżeniu odpowiada zakresowi od -10^{307} do 10^{307}).

Oprócz liczb dziesiętnych (o podstawie 10), w języku JavaScript obsługiwane są także liczby ósemkowe (o podstawie 8) oraz szesnastkowe (o podstawie 16). Liczby te mogą przydać się na przykład do obsługi kolorów obiektów, z tego względu, iż w językach HTML oraz XHTML kolory powszechnie przedstawia się w postaci szesnastkowej.

Jeżeli nie określimy jawnie, że wprowadzana liczba jest liczbą ósemkową lub szesnastkową, to język JavaScript będzie ją interpretował jako liczbę dziesiętną. Aby oznaczyć liczbę jako ósemkową, wystarczy rozpocząć ją od cyfry 0. Aby oznaczyć liczbę szesnastkową,

należy rozpocząć liczbę od cyfry 0 oraz znaku "x". Zwróćmy uwagę na to, że pomimo iż można wprowadzać do przetwarzania liczby ósemkowe i szesnastkowe, to w dalszym ciągu uzyskany wynik będzie miał postać liczby dziesiętnej. Na przykład następujący kod:

```
alert(10+10);
```

zgodnie z naszymi oczekiwaniami wyświetli liczbę 20. Natomiast kod:

```
alert(010+010);
```

wyświetli liczbę 16. Dzieje się tak dlatego, że ósemkowa liczba 10 odpowiada dziesiętnej liczbie 8, a 8+8 daje wynik 16 (pamiętajmy, że uzyskany wynik jest w postaci dziesiętnej). Z tych samych powodów kolejny kod, w którym dodajemy szesnastkową liczbę 10 do szesnastkowej liczby 10, spowoduje wyświetlenie w wyniku liczby 32.

```
alert(0x10+0x10);
```



Za każdym razem, kiedy pobieramy od użytkownika liczby dziesiętne, musimy pamiętać o obcinaniu wiodących zer. W innym przypadku liczby te będą interpretowane jako ósemkowe.

Istnieją trzy specjalne wartości liczbowe. Są to:

```
Infinity
-Infinity
NaN
```

Wynik równy nieskończoności dodatniej (`Infinity`) lub ujemnej (`-Infinity`) możemy uzyskać w dwóch przypadkach: jeżeli liczba przekroczy maksymalną wartość obsługiwaną przez JavaScript lub jeżeli spróbujemy podzielić liczbę przez zero. W przypadku dzielenia przez zero, wynik nieskończoność (`Infinity`) uzyskamy, jeżeli dzielnik był liczbą dodatnią, natomiast minus nieskończoność (`-Infinity`), jeżeli dzielnik był liczbą ujemną.

Wartość `NaN` jest skrótem od *Not a Number* — „to nie jest liczba". Wartość tę uzyskamy w przypadku, gdy wykonamy niewłaściwe działanie z liczbą, takie jak podzielenie jej przez ciąg znaków. Na przykład, kiedy spróbujemy podzielić `100` przez ciąg `"kangur"`:

```
100/"kangur";
```

w wyniku tego działania uzyskamy `NaN`.

Boolean

W czasie pisania skryptów często będziemy podejmować decyzje co do tego, czy jakieś działanie ma być wykonywane, czy też nie. Załóżmy na przykład, że mamy stronę WWW, na której wypełniamy formularz online, służący do wysłania wiadomości e-mail. Zanim użytkownik prześle formularz, chcemy sprawdzić, czy wprowadzony adres e-mail jest poprawny. Na razie nie przejmujmy się skryptem służącym do wykonania tego sprawdzenia (opracujemy go w dalszej części książki). Wynikiem testu będzie wartość „tak" — jest to poprawny adres e-mail" i wtedy wiadomość będzie wysłana lub „nie" — nie jest to poprawny adres e-mail"; wówczas poprosimy o ponowne wprowadzenie poprawnego adresu.

W takim przypadku poszukujemy wartości `true` lub `false`. Sytuacja ta zdarza się w programowaniu tak często, że ze względu na nią wymyślono typ danych Boolean. Dane typu Boolean mogą przyjmować jedną z dwóch wartości: `true` lub `false`. Nie ma takich wartości, jak: „chyba", „być może" czy „prawdopodobnie". JavaScript podejmuje decyzje na podstawie wartości „tak" lub „nie", na zasadach logiki boolean (zero - jedynkowej).

Ciągi znaków

Bardziej oczywistym typem danych, który będziemy wykorzystywać w naszych skryptach są dane typu `string` (ciągi znaków). Sam kod JavaScript jest zapisany jako tekst, zatem bez mechanizmów służących do rozróżnienia, które teksty są kodem, a które ciągami znaków, mogłoby dojść do niejednoznacznych sytuacji. W celu rozwiązania tego problemu, do oznaczenia, że wiersz lub ciąg znaków nie jest kodem, stosuje się cudzysłowy lub apostrofy. Dzięki temu interpreter języka JavaScript traktuje tekst otoczony cudzysłowami lub apostrofami jako ciąg znaków, a nie jako kod.

Dane, które mają być interpretowane jako ciąg powiązanych ze sobą znaków określa się mianem *ciąg znaków* albo *łańcuch znaków*. W łańcuchach znaków możemy zapisywać nie tylko litery alfabetu, ale także inne znaki. Wystarczy ująć je w cudzysłów bądź apostrof.

W języku JavaScript do oznaczenia ciągów znaków możemy użyć zarówno cudzysłowów, jak i apostrofów. Zatem ciągi `"Jestem ciągiem znaków"` oraz `'Ja też jestem ciągiem znaków'`, to równoznaczne sposoby oznaczania danych typu `string`. Musimy jednak zwrócić uwagę na to, aby nie dopuścić do sytuacji, w której początek ciągu znaków oznaczamy jednym rodzajem znaku, np. apostrofem, a koniec innym, np. cudzysłowem. Jeżeli zapiszemy `"Jestem ciągiem znaków'` lub `'Jestem ciągiem znaków"`, to takie zapisy spowodują powstanie błędu. Zapamiętajmy więc, że do oznaczenia początku i końca ciągu znaków musimy użyć tego samego znaku (apostrofu lub cudzysłowu).

Możemy pomyśleć, że skoro do oznaczenia początku i końca ciągu znaków można użyć zarówno apostrofu, jak i cudzysłowu, zatem nie powinno mieć znaczenia, czy rozpoczynamy ciąg znaków jednym z nich, a kończymy innym. Jeżeli jednak chwilę się nad tym zastanowimy, dojdziemy do wniosku, że w takim przypadku niemożliwe byłoby umieszczenie w ciągu znaku cudzysłowu lub apostrofu. Tymczasem jest to możliwe, ponieważ, jeżeli rozpoczniemy ciąg znaków jednym ze znaków, interpreter języka JavaScript oczekuje na znak tego samego typu i dopiero kiedy go napotka, stwierdza, że dotarł do końca ciągu znaków, a wszystko za tym znakiem, to kod. W takiej konfiguracji, jeżeli zamierzamy umieścić w ciągu znaków jeden ze znaków — apostrof lub cudzysłów — to do rozpoczęcia i zakończenia ciągu musimy użyć tego z nich, który nie został użyty do wydzielenia ciągu.

Prawdopodobnie dość szybko zdarzy się sytuacja, w której będziemy musieli umieścić w ciągu znaków zarówno cudzysłów, jak i apostrof. Zazwyczaj jest to potrzebne do samego kodowania, ale na razie zademonstrujemy tę sytuację dla przykładu, zakładając, że musimy utworzyć ciąg znaków z następującego zdania:

```
Jan krzyknął "Hej, podaj mi płytę Mc'Cartneya!".
```

Gdybyśmy otoczyli ciąg znakami cudzysłowu:

```
"Jan krzyknął "Hej, podaj mi płytę Mc'Cartneya!"."
```

wówczas dla interpretera języka JavaScript oznaczałoby to, że ciągiem znaków jest jedynie tekst *"Jan krzyknął "*. Następujący po ciągu tekst *Hej...* dla interpretera nic nie znaczy, zatem powstałby błąd. To samo zdarzy się, jeśli otoczymy ciąg apostrofami. Kiedy interpreter JavaScript dojdzie do ciągu *Mc'*, wystąpi błąd, ponieważ apostrof w ciągu *Mc'Cartneya* zostanie zinterpretowany jako apostrof kończący ciąg znaków.

W języku JavaScript do rozwiązania tego problemu służy znak lewego ukośnika (\). Jeżeli w JavaScript użyjemy tego znaku, oznacza to, że następny wprowadzony po nim znak jest znakiem specjalnym. W przypadku apostrofu lub cudzysłowu, poprzedzenie ich znakiem lewego ukośnika oznacza po prostu, że nie oznaczają one końca ciągu znaków. Możemy na przykład zapisać:

```
'To jest płyta Mc\'Cartneya'
```

lub jeżeli chcemy umieścić w ciągu pięć cudzysłowów, możemy napisać:

```
"\"\"\"\"\""
```

W przypadku użycia apostrofów, zapisalibyśmy:

```
'\'\'\'\'\''
```

Jeżeli znak jest poprzedzony lewym ukośnikiem, mówimy o nim, że jest to znak sterujący. Istnieje kilka innych znaków sterujących o specjalnym znaczeniu. Oto one:

`\b`	znak backspace
`\f`	wysunięcie strony
`\n`	nowy wiersz
`\r`	powrót karetki
`\t`	tabulacja
`\'`	apostrof
`\"`	cudzysłów
`\\`	lewy ukośnik
`\xNN`	znak według kodowania Latin-1 (*x* to po prostu znak *x*, natomiast *NN* oznacza liczbę szesnastkową)
`\uNNNN`	znak według kodowania Unicode (*u* to po prostu znak *u*, natomiast *NNNN* oznacza liczbę szesnastkową)

Przypuśćmy, że w ciągu znaków chcemy umieścić znak ©. Dla zestawu znaków Latin-1 oraz Unicode zapiszemy odpowiednio:

```
"Copyright \xA9 2002 Sams Publishing"
"Copyright \u00A9 2002 Sams Publishing"
```

Obydwa ciągi znaków będą przekształcone do następującej postaci:

```
Copyright © 2002 Sams Publishing
```

Jak zapewne zauważyliśmy, zarówno w zestawie znaków Latin-1, jak i Unicode, znak © ma w zasadzie ten sam kod; jedyną różnicą są dwa zera występujące w drugim przypadku na początku kodu. Istotnie, w zestawach znaków Unicode i Latin-1 kody znaków są takie same, ale dwa wiodące zera umożliwiają zakodowanie wielu dodatkowych znaków. Aby zapoznać się z pełnym zestawem znaków Unicode, możemy odwiedzić serwis WWW Unicode, mieszczący się pod adresem *http://www.unicode.org*.



Chociaż zestaw znaków Unicode daje nam możliwość obsługi znacznie większej liczby znaków, to jednak trzeba pamiętać, że ten zestaw znaków nie jest obsługiwany przez przeglądarki w wersji 4 i starsze. W rzeczywistości niektórych znaków nie ma nawet w najnowszych przeglądarkach.

Przydatne narzędzia

W kilku kolejnych rozdziałach nauczymy się wykorzystywać narzędzia JavaScript, które umożliwiają wyświetlanie wyników, podejmowanie decyzji przez użytkowników oraz pobieranie od nich danych. Właściwa nazwa tych narzędzi to „funkcje"; nie przejmujmy się, jeśli nie wiemy jeszcze, czym są funkcje. Więcej miejsca poświęcono na ich omówienie w rozdziale 3. — „Funkcje i instrukcje".

Funkcja alert()

Funkcję `alert()` wykorzystaliśmy już w tym rozdziale kilkakrotnie, w celu wyświetlenia okna z ostrzeżeniem. Prawdopodobnie niejednokrotnie zetknęliśmy się z małymi okienkami, które wyświetlają się wiele razy na stronach WWW, aby nas przed czymś ostrzec lub powiadomić o konsekwencjach działania.

Aby wyświetlić okno ostrzeżenia na naszej stronie WWW, powinniśmy skorzystać z funkcji języka JavaScript `alert()`. Przy korzystaniu z tej funkcji, musimy jedynie wprowadzić komunikat, który ma być wyświetlony. Interpreter JavaScript wykona wszystkie czynności związane z utworzeniem okna na drugim planie. Oto przykład:

```
alert("Cześć wszystkim!");
```

Wywołanie funkcji `alert()` można umieścić w dowolnym miejscu bloku skryptów; spowoduje ona wyświetlenia okna z ostrzeżeniem.

Wykorzystując funkcję `alert()` należy pamiętać o dwóch sprawach. Po pierwsze, komunikat, który ma być wyświetlony, musi być otoczony parą nawiasów. Ponadto, ponieważ komunikat jest ciągiem znaków, musi być ujęty w cudzysłowy lub apostrofy. Jeżeli nasz komunikat to po prostu liczba, możemy umieścić go bez apostrofu czy cudzysłowu (ale w dalszym ciągu w nawiasach). Oto przykład:

```
alert(55);
```

W następnych kilku rozdziałach będziemy dosyć często wykorzystywać funkcję `alert()` po to, by zaprezentować wyniki naszych skryptów. Na przykład, aby sprawdzić, jak JavaScript zinterpretuje sumę `5+5`, możemy zapisać:

```
alert(5+5);
```

W rezultacie na ekranie pojawi się okno ostrzeżenia, wyświetlające liczbę `10`. Dzięki temu będziemy mieli pewność, że interpreter JavaScript zinterpretuje wartość sumy `5+5` jako `10`.

Funkcja confirm()

Funkcja `confirm()` jest nieco bardziej zaawansowana od funkcji `alert()`. Nie poprzestaje na wyświetleniu informacji, lecz pozwala na dokonanie wyboru spomiędzy dwóch opcji. Pewnie i w tym przypadku mieliśmy już okazję spotkać się z niewielkimi okienkami wyświetlanymi przez tę funkcję na stronach WWW. Okno takie jest podobne do tego, które wyświetla funkcja `alert()`, ale oprócz komunikatu i przycisku `OK` występuje w nim jeszcze przycisk `Cancel`.

Na razie nie będziemy zbyt szczegółowo omawiać funkcji `confirm()`, ponieważ przydaje się ona jedynie w połączeniu z instrukcjami sterującymi, które pozwalają na podejmowanie decyzji. Mówiąc prościej, przycisk, który użytkownik wciska, określa jedną z dwóch wartości zwracanych do skryptu. Te dwie wartości to `true` lub `false`, z którymi już się zetknęliśmy. Można je uważać za komputerowe odpowiedniki odpowiedzi „tak" oraz „nie". Wartość przesłana przez użytkownika umożliwia skryptowi określenie, czy należy wykonywać określone działanie, czy też nie.

Funkcję `confirm()` wywołuje się w podobny sposób, jak funkcję `alert()`. Także i w tym przypadku można ją umieścić w bloku skryptów. Oto przykład:

źródło

```
confirm("Czy jesteś pewien, że chcesz wyzerować formularz?");
```

wynik Wywołanie funkcji `confirm()` w tej postaci spowoduje wyświetlenie okna pokazanego na rysunku 1.4.



Rysunek 1.4. *Nasze pierwsze okno confirm()*

Aby zobaczyć wartości `true` lub `false` zwracane przez funkcję `confirm()`, możemy umieścić jej wywołanie wewnątrz funkcji `alert()`. Wartość zwrócona przez funkcję `confirm()` będzie wyświetlona w oknie ostrzeżenia. Oto kod służący do wykonania takiego sprawdzenia:

```
alert(confirm("Czy jesteś pewien?"));
```

Część wiersza, która będzie poddana przetwarzaniu jako pierwsza, to funkcja `confirm()`. Po wyświetleniu okna `confirm()` i kliknięciu przycisku `OK` lub `Anuluj`, zostanie zwrócona

wartość `true` lub `false` i będzie ona umieszczona jako parametr funkcji `alert()`. Ta z kolei spowoduje wyświetlenie okna z odpowiednią wartością. Wypróbujmy tę instrukcję samodzielnie.

W kolejnych rozdziałach zobaczymy, w jaki sposób wykorzystać wartość zwracaną przez funkcję `confirm()` do podejmowania decyzji w naszych skryptach.

Funkcja prompt()

Funkcja `prompt()` jest ostatnią spośród trzech funkcji, które zostaną teraz omówione. Jej działanie polega na wyświetleniu prośby o wprowadzenie tekstu, który będzie wykorzystany przez skrypt. Na przykład może pojawić się pytanie o nazwisko, po czym skrypt umieści to nazwisko w kodzie HTML strony w celu dokonania personalizacji.

Funkcja `prompt()` jest nieco bardziej skomplikowana od dwóch poprzednich funkcji, ale także jej obsługa jest dość oczywista. Wymaga podania jako parametrów dwóch ciągów znaków. Pierwszy z tych ciągów jest komunikatem, który ma być wyświetlony w oknie funkcji `prompt()`, a drugi służy do domyślnego wypełnienia obszaru edycyjnego, w którym wprowadzamy nasze dane (domyślne wypełnianie działa tylko w przeglądarce Internet Explorer). Te dwa ciągi znaków oddziela się przecinkiem. Pokazano to w następującym wierszu kodu:

źródło

```
prompt("Proszę podać nazwisko.", "Wpisz je tutaj.");
```

wynik Instrukcja ta spowoduje wyświetlenie okna pokazanego na rysunku 1.5.

Rysunek 1.5. *Nasze pierwsze okno prompt()*



analiza Tekst wprowadzany przez użytkownika jest zwracany do skryptu w taki sam sposób, jak wartości `true` lub `false` w przypadku funkcji `confirm()`.

Należy zwrócić uwagę, że jeżeli nie chcemy wypełnić pola edycji w oknie `prompt()`, to nie pomijamy drugiego parametru funkcji. W takim przypadku należy wykorzystać ciąg pusty, który stanowią następujące bezpośrednio po sobie cudzysłowy lub apostrofy.

```
prompt("Proszę podać nazwisko.","");
```

Pokazane tu funkcje pozwolą na podejmowanie trzech działań, które będą nam potrzebne w następnych kilku rozdziałach, kiedy będziemy omawiać podstawy języka JavaScript. Dzięki oknom wyświetlanym przez funkcję `alert()` możemy poznać wyniki działania fragmentów kodu, za pomocą okien `confirm()` możemy podejmować decyzje, a okna `prompt()` pozwalają na wprowadzanie danych z zewnątrz do naszych skryptów.

Podsumowanie

W tym rozdziale Czytelnik zapoznał się wstępnie z podstawowymi informacjami dotyczącymi języka JavaScript; omówiono tu także miejsce tego języka wśród innych technologii WWW. Dowiedzieliśmy się, w jaki sposób, dzięki dążeniu projektantów WWW do interakcji z użytkownikami, doszło do powstania wielu nowych technologii, oraz przekonaliśmy się, że dzięki wieloplatformowej naturze języka JavaScript nadaje się on idealnie do wykorzystania w sieci. Po tym, jak nauczyliśmy się, w jaki sposób wstawiać kod JavaScript na naszych stronach przy zastosowaniu bloku skryptów lub zewnętrznych plików, poznaliśmy także kilka podstawowych właściwości języka — na przykład wykorzystanie odstępów po to, aby nasz kod był bardziej czytelny. Zapoznaliśmy się także z bardziej ogólnymi tematami; poznaliśmy na przykład pięć typów danych obsługiwanych w języku JavaScript: liczby, wartości logiczne, ciągi znaków, wartości puste (`null`) i niezdefiniowane.

Poznaliśmy także kilka podstawowych konstrukcji języka JavaScript oraz zapoznaliśmy się z przykładami prostego kodu.

Warsztat

W podrozdziałach zatytułowanych „Warsztat" umieszczono quizy i ćwiczenia, które będą stanowiły podsumowanie każdego rozdziału. Zadamy w nich kilka pytań, które mogliby zadać nasi Czytelnicy oraz udzielimy odpowiedzi na nie. Przedstawimy także ćwiczenia, które pozwolą na samodzielne studiowanie materiału zaprezentowanego w rozdziale.

Pytania i odpowiedzi

P. Napisałem taki kod:

```
Alert("Cześć! Uczę się języka JavaScript");
```

ale to nie działa. Dlaczego?

O. JavaScript jest językiem, w którym małe litery są odróżniane od wielkich. Aby kod zaczął działać, powinniśmy zapisać:

```
alert("Cześć! Uczę się języka JavaScript");
```

P. Jak mogę zapytać użytkownika o to, gdzie mieszka?

O. Można w tym celu wykorzystać okno monitu, które wykorzystuje funkcję `prompt()` języka JavaScript. Możemy na przykład napisać:

```
prompt("Gdzie mieszkasz?","Tutaj wpisz swoje miejsce zamieszkania.");
```

Quiz

1. Elementy języka JavaScript: `alert()`, `confirm()` oraz `prompt()` to funkcje, czy instrukcje?
2. Czy następujący element : "Witajcie w świecie" to typ danych, czy ciąg znaków?
3. Co oznacza `NaN`? Czy oznacza „to nie jest liczba” (*not a number*), czy też „to prawie liczba” (*nearly a number*)?

Odpowiedzi do quizu

1. Elementy `alert()`, `confirm()` oraz `prompt()` to funkcje języka JavaScript.
2. "Witajcie w Świecie" jest ciągiem znaków. W języku JavaScript ciągi znaków ujmuje się w apostrofy lub cudzysłowy.
3. Ciąg `NaN` oznacza „to nie jest liczba” (*not a number*). Wartość tę funkcja zwraca na przykład wtedy, gdy podejmiemy próbę wykonania niewłaściwej operacji matematycznej.

Ćwiczenia

1. Utwórz okno ostrzeżenia (`alert()`), które wyświetla informację: „To jest podręcznik JavaScript dla każdego”.
2. Utwórz okno potwierdzenia (`confirm()`), które wyświetla pytanie do użytkownika: „Czy podoba Ci się nauka języka JavaScript?.
3. Utwórz okno zapytania (`prompt()`), które wyświetla pytanie do użytkownika „Jak Ci na imię?”, a następnie w miejscu przeznaczonym na wpisanie imienia wyświetla tekst „Tutaj wpisz swoje imię”.