helion.pl
KSIĘGARNIA INTERNETOWA · WYDAWNICTWO

IDŹ DO
PRZYKŁADOWY ROZDZIAŁ
SPIS TREŚCI

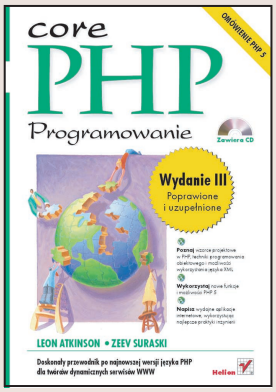
KATALOG KSIĄŻEK
KATALOG ONLINE
ZAMÓW DRUKOWANY KATALOG

TWÓJ KOSZYK
DODAJ DO KOSZYKA

CENNIK I INFORMACJE
ZAMÓW INFORMACJE O NOWOŚCIACH
ZAMÓW CENNIK

CZYTELNIA
FRAGMENTY KSIĄŻEK ONLINE

PHP. Programowanie. Wydanie III



Autorzy: Leon Atkinson, Zeev Suraski
Tłumaczenie: Jarosław Dobrzański
ISBN: 83-7361-355-2
Tytuł oryginału: Core PHP Programming, 3 Edition
Format: B5, stron: 952

Książka „PHP. Programowanie. Wydanie III" to kolejne wydanie praktycznego przewodnika dla programistów stron internetowych. Jeden z najlepszych programistów PHP — Leon Atkinson, uczy wszystkiego, co potrzebujesz, by stworzyć dobrą i szybką aplikację sieciową. Dokładnie opisuje składnię PHP i kluczowe elementy języka. Atkinson przedstawia także najważniejsze funkcje PHP, w tym funkcje wejścia-wyjścia, przetwarzania danych, matematyczne, daty, czasu, konfiguracji, funkcje umożliwiające współpracę z bazami danych, funkcje graficzne i sieciowe. Prezentuje on również działanie PHP na przykładach realizujących sortowanie, przeszukiwanie, analizę łańcuchów i inne zadania.

Książka składa się z czterech części:

1. wstępu do programowania,
2. przewodnika po wszystkich funkcjach,
3. przeglądu typowych problemów programistycznych
4. części umożliwiającej zastosowanie zdobytej wiedzy przy tworzeniu witryn.

Pierwsza część zajmuje się kwestiami dotyczącymi wszystkich języków programowania: jak wygląda skrypt PHP, jak sterować przebiegiem programu i jak zarządzać danymi.
Część druga organizuje funkcje według ich zastosowania i zawiera przykłady ich zastosowania. PHP udostępnia bardzo dużo funkcji, dlatego część ta jest najobszerniejsza.
Część trzecia zajmuje się rozwiązywaniem typowych problemów programistycznych, takich jak sortowanie czy generowanie grafiki.
Ostatnia część udziela porad dotyczących tworzenia całych witryn sieciowych za pomocą PHP.

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

Spis treści

Słowo wstępne 9

Przedmowa 11

Część I Programowanie w PHP 13

Rozdział 1. Wprowadzenie do PHP 15

1.1. Historia PHP 16
1.2. Co sprawia, że PHP jest lepszy od innych języków? 18
1.3. Interfejsy do systemów zewnętrznych 20
1.4. Jak PHP współpracuje z serwerem sieciowym? 20
1.5. Wymagania sprzętowe i programowe 21
1.6. Jak wygląda skrypt PHP? 25
1.7. Przechowywanie danych 27
1.8. Odbieranie informacji od użytkownika 29
1.9. Wybieranie pomiędzy alternatywami 31
1.10. Powtarzanie sekwencji kodu 32

Rozdział 2. Zmienne, operatory i wyrażenia 35

2.1. Spojrzenie ogólne 35
2.2. Typy danych 37
2.3. Zmienne 40
2.4. Stałe 45
2.5. Operatory 45
2.6. Budowanie wyrażeń 57

Rozdział 3. Instrukcje sterujące 61

3.1. Instrukcja if 61
3.2. Operator ? 64
3.3. Instrukcja switch 64
3.4. Pętle 66
3.5. Instrukcje exit, die i return 72
3.6. Wyjątki 73
3.7. Instrukcja Declare 74

Rozdział 4. Funkcje 77

4.1. Deklarowanie funkcji 77
4.2. Instrukcja return 78
4.3. Zakres 79
4.4. Zmienne statyczne 81
4.5. Argumenty 82
4.6. Rekurencja 85
4.7. Dynamiczne wywołania funkcji 86

Rozdział 5. Tablice ... 87

5.1. Tablice jednowymiarowe ... 87
5.2. Indeksowanie tablic ... 88
5.3. Inicjalizacja tablic ... 89
5.4. Tablice wielowymiarowe ... 90
5.5. Rzutowanie tablic ... 91
5.6. Operator + ... 92
5.7. Odwołania do tablic z wnętrza łańcucha ... 93

Rozdział 6. Klasy i obiekty ... 95

6.1. Programowanie obiektowe ... 96
6.2. Model obiektowy w PHP 5 ... 97
6.3. Definiowanie klasy ... 98
6.4. Konstruktory i destruktory ... 100
6.5. Klonowanie ... 102
6.6. Dostęp do metod i właściwości ... 103
6.7. Statyczne składniki klas ... 106
6.8. Typy dostępności ... 107
6.9. Wiązanie ... 111
6.10. Metody i klasy abstrakcyjne ... 114
6.11. Przeciążanie z poziomu użytkownika ... 117
6.12. Automatyczne ładowanie klas ... 118
6.13. Serializacja obiektów ... 119
6.14. Przestrzenie nazw ... 120
6.15. Ewolucja modułu Zend ... 122

Rozdział 7. Operacje wejścia-wyjścia i dostęp do dysku ... 129

7.1. Połączenia HTTP ... 130
7.2. Wysyłanie treści do przeglądarki ... 131
7.3. Buforowanie treści ... 132
7.4. Zmienne środowiskowe ... 133
7.5. Pobieranie danych z formularzy ... 133
7.6. Przesyłanie tablic w formularzach ... 134
7.7. Cookies ... 135
7.8. Pobieranie plików od użytkownika ... 136
7.9. Zapis do plików i ich odczytywanie ... 138
7.10. Sesje ... 140
7.11. Funkcje include i require ... 142
7.12. Nie ufaj danym użytkownika ... 144

Część II Funkcje PHP ... 145

Rozdział 8. Komunikacja z przeglądarką ... 147

8.1. Zmienne generowane przez moduł PHP ... 147
8.2. Stałe generowane przez moduł PHP ... 152
8.3. Przesyłanie tekstu do przeglądarki ... 156
8.4. Buforowanie wyjścia ... 159
8.5. Obsługa sesji ... 162
8.6. Nagłówki HTTP ... 169

Rozdział 9. System operacyjny 173
9.1. Pliki 173
9.2. Pliki skompresowane 217
9.3. Direct I/O 224
9.4. Diagnostyka 227
9.5. POSIX 252
9.6. Polecenia interpretera 257
9.7. Sterowanie procesami 262

Rozdział 10. Funkcje sieciowe 267
10.1. Ogólne funkcje sieciowe 267
10.2. Gniazda 274
10.3. FTP 289
10.4. Curl 300
10.5. SNMP 311

Rozdział 11. Funkcje przetwarzania danych 315
11.1. Typy danych, stałe i zmienne 315
11.2. Tablice 326
11.3. Obiekty i klasy 357
11.4. Funkcje definiowane przez użytkownika 361

Rozdział 12. Kodowanie i dekodowanie 367
12.1. Łańcuchy 367
12.2. Porównywanie łańcuchów 376
12.3. Kodowanie i dekodowanie 378
12.4. Kompresja 401
12.5. Szyfrowanie 403
12.6. Mieszanie 411
12.7. Sprawdzanie pisowni 416
12.8. Wyrażenia regularne 420
12.9. Kodowanie zestawów znaków 427

Rozdział 13. Funkcje matematyczne 437
13.1. Operacje matematyczne 437
13.2. Liczby losowe 447
13.3. Liczby dowolnej precyzji 449

Rozdział 14. Funkcje daty i czasu 453
14.1. Data i czas 453
14.2. Niestandardowe kalendarze 462

Rozdział 15. Konfiguracja PHP 467
15.1. Dyrektywy konfiguracyjne 467
15.2. Konfiguracja 499

Rozdział 16. Funkcje graficzne 509
16.1. Analizowanie obrazów 510
16.2. Tworzenie obrazków 513

Rozdział 17. Bazy danych ... 557
17.1. Abstrakcyjna baza danych typu DBM ... 558
17.2. DBX ... 562
17.3. LDAP ... 566
17.4. MySQL ... 578
17.5. ODBC ... 591
17.6. Oracle ... 606
17.7. Postgres ... 620
17.8. Sybase ... 637

Rozdział 18. Warstwy obiektowe ... 647
18.1. COM ... 647
18.2. CORBA ... 652
18.3. Java ... 654

Rozdział 19. Inne funkcje ... 657
19.1. Apache ... 657
19.2. IMAP ... 660
19.3. MnoGoSearch ... 681
19.4. OpenSSL ... 686
19.5. Komunikaty systemu System V ... 696
19.6. Semafory systemu System V ... 700
19.7. Pamięć wspólna systemu System V ... 702

Rozdział 20. XML ... 707
20.1. DOM XML ... 709
20.2. Expat XML ... 722
20.3. WDDX ... 733

Część III Algorytmy ... 737

Rozdział 21. Sortowanie, wyszukiwanie i liczby losowe ... 739
21.1. Sortowanie ... 740
21.2. Wbudowane funkcje sortujące ... 740
21.3. Sortowanie z funkcją porównującą ... 744
21.4. Wyszukiwanie ... 746
21.5. Indeksowanie ... 748
21.6. Liczby losowe ... 749
21.7. Identyfikatory losowe ... 751
21.8. Losowanie banera reklamowego ... 752

Rozdział 22. Analiza składni i łańcuchów ... 755
22.1. Podział łańcuchów na elementy ... 755
22.2. Wyrażenia regularne ... 757
22.3. Definiowanie wyrażeń regularnych ... 758
22.4. Stosowanie wyrażeń regularnych w skryptach PHP ... 759

Rozdział 23. Integracja z bazami danych ... 767
23.1. Tworzenie tabel HTML z rezultatami zapytań SQL ... 767
23.2. Śledzenie odwiedzających za pomocą identyfikatorów sesji ... 772
23.3. Przechowywanie danych w bazie ... 780
23.4. Warstwy abstrakcyjne baz danych ... 786

Rozdział 24. Sieć ... 787
24.1. Uwierzytelnianie w HTTP ... 787
24.2. Sterowanie buforem przeglądarki ... 790
24.3. Ustawianie typu dokumentu ... 791
24.4. E-mail z załącznikami ... 792
24.5. Wiadomości pocztowe HTML ... 795
24.6. Weryfikacja adresu skrzynki pocztowej ... 798

Rozdział 25. Generowanie grafiki ... 803
25.1. Przyciski dynamiczne ... 803
25.2. Generowanie grafiki „w locie” ... 808
25.3. Wykresy słupkowe ... 808
25.4. Wykresy kołowe ... 811
25.5. Rozciąganie pojedynczych pikseli ... 813

Część IV Inżynieria oprogramowania ... 815

Rozdział 26. Integracja z HTML-em ... 817
26.1. Umieszczanie fragmentów kodu PHP w dokumencie HTML ... 817
26.2. Używanie PHP do generowania całych dokumentów HTML ... 823
26.3. Separowanie HTML-a od PHP ... 824
26.4. Generowanie kodu HTML za pomocą PHP ... 826

Rozdział 27. Projektowanie ... 829
27.1. Tworzenie specyfikacji wymagań ... 830
27.2. Tworzenie dokumentów projektowych ... 833
27.3. Zarządzanie zmianami ... 834
27.4. Modularyzacja za pomocą include ... 839
27.5. FreeEnergy ... 840
27.6. Szablony ... 842
27.7. Szkielety aplikacji ... 846
27.8. PEAR ... 847
27.9. Adresy przyjazne wyszukiwarkom ... 848

Rozdział 28. Efektywność i diagnostyka ... 851
28.1. Optymalizacja ... 852
28.2. Mierzenie wydajności ... 853
28.3. Optymalizacja najwolniej wykonywanych fragmentów ... 857
28.4. Kiedy przechowywać treść w bazie ... 859
28.5. Strategie diagnostyczne ... 859
28.6. Symulowanie połączeń HTTP ... 860
28.7. Buforowanie treści strony ... 861
28.8. Kompresja generowanej treści ... 862
28.9. Unikanie eval ... 863
28.10. Unikanie dynamicznego ładowania rozszerzeń ... 865
28.11. Zwiększanie szybkości realizacji zapytań MySQL ... 866
28.12. Optymalizacja sesji zapisujących dane na dysku ... 867
28.13. Unikanie przekazywania argumentów przez odwołania (czyli dlaczego nie ufać instynktowi) ... 868
28.14. Unikanie konkatenacji dużych łańcuchów ... 870
28.15. Unikanie umieszczania dużych plików na serwerze Apache z uaktywnionym PHP ... 871

28.16. Rola trwałych połączeń z bazą danych 871
28.17. Unikanie w miarę możliwości korzystania z exec, operatorów ` i system 872
28.18. Zastosowanie php.ini-recommended 873
28.19. Stosowanie wyrażeń regularnych tylko tam, gdzie są niezbędne 873
28.20. Optymalizacja pętli 873
28.21. Konfiguracja serwera IIS 874

Rozdział 29. Wzorce projektowe 875

29.1. Definicja wzorców 875
29.2. Singleton 877
29.3. Fabryka 880
29.4. Obserwator 882
29.5. Strategia 885

Dodatki 889

Dodatek A Kody z ukośnikiem 891

Dodatek B Kody ASCII 893

Dodatek C Operatory 899

Dodatek D Znaczniki PHP 901

Dodatek E Konfiguracja PHP w czasie kompilacji 903

Dodatek F Zasoby internetowe 907

Dodatek G Przewodnik po stylach PHP 909

Skorowidz 913

22

Analiza składni i łańcuchów

W tym rozdziale:

- Podział łańcuchów na elementy.
- Wyrażenia regularne.
- Definiowanie wyrażeń regularnych.
- Stosowanie wyrażeń regularnych w skryptach PHP.

Analiza składni polega na podziale całości na elementy składowe, zwykle dotyczy to podziału zdania na poszczególne wyrazy. PHP musi zanalizować napisany przez nas kod w pierwszym kroku procesu przekształcania go w dokument HTML. Czasami staniemy również przed problemem pobierania i weryfikacji danych zawartych w łańcuchach tekstowych. Może to być np. prosta lista oddzielona tabulatorami lub skomplikowany łańcuch, jakiego przeglądarka używa do swojej identyfikacji w obliczu serwera sieciowego. Można wówczas podzielić łańcuch na poszczególne elementy lub zastosować wyrażenie regularne. W rozdziale tym opisane zostały funkcje analizujące składnię oraz treść łańcuchów.

22.1. Podział łańcuchów na elementy

PHP udostępnia prosty model dzielenia łańcuchów. Wybrane przez nas znaki są uznawane za separatory, a fragmenty łańcucha, znajdujące się między separatorami, są uznawane za pojedyncze elementy. Z każdym pobranym elementem można zmieniać zestaw znaków separujących, co jest wygodne w przypadku nieregularnych łańcuchów — to znaczy tych, które nie są prostymi listami oddzielonymi przecinkami.

Listing 22.1 pobiera zdanie i dzieli je na poszczególne słowa za pomocą funkcji `strtok`, opisanej w rozdziale 12. W przypadku skryptu słowa są otoczone spacjami, znakami przestankowymi lub końca zdania. Cudzysłowy i apostrofy są uznawane za część wyrazu. Efekt działania przykładowego skryptu jest widoczny na rysunku 22.1.

Listing 22.1. *Podział łańcucha na elementy*

```
<?php
    /*
    ** Jeżeli przesłano zdanie, dokonuje jego analizy
    */
    if(isset($_REQUEST['sentence']))
    {
        $total=0;

        print("<b>Przesłany tekst:</b>");
        print("{$_REQUEST['sentence']}<br>\n<br>\n");

        //ustala znaki, które separują poszczególne elementy łańcucha
        $separators = " ,!.?";

        //pobiera kolejno wszystkie elementy
        for($token = strtok($_REQUEST['sentence'], $separators);
            $token !== FALSE;
            $token = strtok($separators))
        {
            //pomija puste elementy
            if($token != "")
            {
                // liczy wystąpienia każdego z wyrazów
                if(!isset($word_count[strtolower($token)]))
                {
                    $word_count[strtolower($token)]=1;
                }
                else
                {
                    $word_count[strtolower($token)]++;
                }
                $total++;
            }
        }

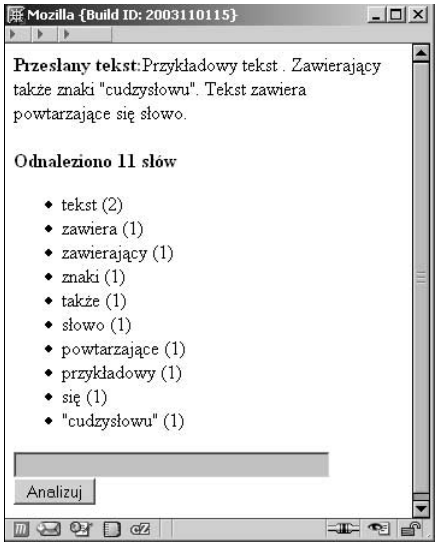
        //w pierwszej kolejności sortuje względem słów
        ksort($word_count);

        //następnie sortuje względem częstości występowania
        arsort($word_count);

        print("<b>Odnaleziono $total słów</b>\n");
        print("<ul>\n");
        foreach($word_count as $key=>$value)
        {
            print("<li>$key ($value)</li>\n");
        }
        print("</ul>\n");
    }

    print("<form action=\"{$_SERVER['PHP_SELF']}\" " .
        "method=\"post\">\n");
    print("<input name=\"sentence\" size=\"40\">\n");
    print("<input type=\"submit\" value=\"Analizuj\">\n");
    print("</form>\n");
?>
```

Rysunek 22.1.
Efekt działania skryptu z listingu 22.1



Warto zwrócić uwagę na rolę pętli `for` w powyższym przykładzie. Zamiast inkrementacji wartości całkowitej pobiera ona kolejne elementy łańcucha. Kiedy funkcja `strtok` trafi na koniec danych wejściowych, zwróci `FALSE`. Pierwszą myślą może być sprawdzanie w pętli wystąpienia wartości `FALSE` za pomocą operatora `!=`. Należy pamiętać, że pusty łańcuch ma wartość logiczną `FALSE`. Jeżeli dwa separatory następują po sobie, `strtok`, jak można się spodziewać, zwróci pusty łańcuch. Ponieważ nie chcemy, aby operacja dzielenia łańcucha została przerwana na pierwszym powtórzonym separatorze, konieczne jest sprawdzenie wystąpienia rzeczywistej wartości `FALSE` za pomocą operatora `!==`.

Funkcja `strtok` jest użyteczna tylko w najprostszych i najbardziej uporządkowanych przypadkach. Przykładem może być odczyt pliku tekstowego, separowanego tabulatorami. Algorytm polegać może wówczas na odczycie wiersza z pliku, podziale wiersza na elementy, stosując tabulator w roli separatora i przejściu do odczytu kolejnego wiersza z pliku.

22.2. Wyrażenia regularne

Na szczęście PHP udostępnia również o wiele doskonalsze narzędzie niż funkcja `strtok` — wyrażenia regularne. Używają one własnego języka do opisu wzorców, które są porównywane z łańcuchami. Kod źródłowy PHP zawiera implementację wyrażeń regularnych zgodnych z normą POSIX 1003.2. Norma ta umożliwia stosowanie wyrażeń starszego typu, ale sugeruje korzystanie z nowego typu, który zostanie tu opisany. Wszystkie funkcje związane z wyrażeniami regularnymi są opisane w rozdziale 12.

W 1999 r. Andrei Zmievski wzbogacił PHP o współpracę z wyrażeniami regularnymi, stosowanymi w języku Perl. Mają one dwie zalety w stosunku do wbudowanych wyrażeń regularnych PHP: ułatwiają kopiowanie wyrażeń ze skryptów Perl i wykonują się szybciej.

Dokładny opis wyrażeń regularnych wykracza poza ramy tego tekstu. Jest to zagadnienie godne następnej książki. Zostaną tu objaśnione podstawy oraz pokazane różne funkcje PHP, które używają wyrażeń regularnych. Świetnym źródłem informacji o wyrażeniach regu-

larnych jest rozdział 2. książki Ellie Quigley, *UNIX Shells by Example*. Wyrażenia regularne stosowane w Perlu są opisane w oficjalnej dokumentacji Perla pod adresem *http://www.perldoc.com/perl5.8.0/pod/perlre.html*. Następnie należy przeczytać w dokumentacji na stronie PHP o różnicach między implementacją tych wyrażeń w Perlu a w PHP: *http://www.php.net/manual/➥pcre.pattern.syntax.php*.

22.3. Definiowanie wyrażeń regularnych

Na najwyższym poziomie wyrażenia regularne składają się z jednej lub więcej gałęzi, oddzielonych znakiem pionowej kreski (|). Znak ten ma właściwości operatora logicznego LUB. Każda z gałęzi może odpowiadać testowanemu łańcuchowi. Kilka przykładów znajduje się w tabeli 22.1.

Tabela 22.1. *Warianty w wyrażeniach regularnych*

Próbka	Opis
`jabłko`	Odpowiada słowu „jabłko”.
`jabłko|piłka`	Odpowiada słowu „jabłko” lub „piłka”.
`początek|koniec|przerwa`	Odpowiada słowom „początek”, „koniec” lub „przerwa”.

Każdy z wariantów zawiera jeden lub więcej atomów. Po atomach mogą występować znaki modyfikujące możliwą liczbę kolejnych trafień dla danego atomu. Gwiazdka (*) oznacza, że atom może występować dowolną liczbę razy. Symbol dodawania (+) oznacza, że atom musi występować przynajmniej raz. Znak zapytania (?) oznacza, że atom może występować raz lub ani razu.

Alternatywnie atom może być związany, co oznacza, że następują po nim nawiasy klamrowe ({}), które zawierają liczby całkowite. Jeżeli nawiasy klamrowe zawierają pojedynczą liczbę, wówczas atom musi występować dokładnie tę liczbę razy. Jeżeli w nawiasach znajduje się pojedyncza liczba, po której następuje przecinek, atom musi występować tyle lub więcej razy. Jeżeli klamry zawierają dwie liczby oddzielone przecinkiem, atom musi występować liczbę razy zawartą w przedziale między tymi liczbami. W tabeli 22.2 przedstawione są przykłady repetycji.

Tabela 22.2. *Wzorce z repetycją w wyrażeniach regularnych*

Próbka	Opis
`a(b*)`	Odpowiada `a`, `ab`, `abb`, … — znak `a` plus dowolna liczba znaków `b`.
`a(b+)`	Odpowiada `ab`, `abb`, `abbb`, … — znak `a` plus jeden lub więcej znaków `b`.
`a(b?)`	Odpowiada `a` lub `ab` — znak `a`, po którym może wystąpić znak `b`.
`a(b{3})`	Odpowiada tylko `abbb`.
`a(b{2,})`	Odpowiada `abb`, `abbb`, `abbbb`, … — znak `a`, po którym następuje jeden lub więcej znaków `b`.
`a(b{2,4})`	Odpowiada `abb`, `abbb`, `abbbb` — znak `a`, po którym następują dwa, trzy lub cztery znaki `b`.

Atom jest szeregiem znaków, z których niektóre mają specjalne znaczenie, a inne oznaczają po prostu znak, który ma występować w łańcuchu. Kropka (`.`) zastępuje dowolny pojedynczy znak. Karetka (`^`) zastępuje początek łańcucha. Symbol dolara (`$`) zastępuje koniec łańcucha. Jeżeli w łańcuchu ma występować jeden ze znaków specjalnych (`^ > [ ] $ ( ) | * ? { } \`), należy je poprzedzić znakiem `\`. Tak naprawdę każdy znak poprzedzony znakiem `\` będzie traktowany dosłownie, nawet jeżeli nie ma on specjalnego znaczenia. Każdy znak nieposiadający specjalnego znaczenia zostanie uznany po prostu za znak, który ma występować w łańcuchu. Można również grupować atomy za pomocą nawiasów, aby były traktowane jako jeden atom.

Nawiasy kwadratowe (`[]`) służą do określania możliwego zakresu wartości. Może on mieć formę listy dozwolonych znaków lub zakresu określonego za pomocą myślnika (`-`). Jeżeli lista lub zakres są poprzedzone karetką (`^`), oznacza to wszystkie znaki spoza określonego zakresu. Należy zwrócić uwagę na to podwójne znaczenie znaku `^`.

Oprócz list i zakresów nawiasy kwadratowe mogą zawierać klasy znaków. Nazwy tych klas są dodatkowo otoczone dwukropkami, aby odpowiadać wszystkim znakom należącym do alfabetu `[:alpha:]`. Dostępne klasy to: `alnum`, `alpha`, `blank`, `cntrl`, `digit`, `graph`, `lower`, `print`, `punct`, `space`, `upper` i `xdigit`. Opisy tych klas znajdują się w dokumentacji `ctype`.

Oprócz tego, dwa dodatkowe kody w nawiasach kwadratowych określają początek lub koniec wyrazu. Są to odpowiednio `[:<:]` i `[:>:]`. Wyraz oznacza w tym przypadku każdą sekwencję znaków alfanumerycznych i znaków podkreślenia. Tabela 22.3 zawiera przykłady użycia nawiasów kwadratowych.

Tabela 22.3. *Nawiasy kwadratowe w wyrażeniach regularnych*

Próbka	Opis
`a.c`	Odpowiada `aac`, `abc`, `acc`, … — każdy trzyliterowy łańcuch zaczynający się na `a`, a kończący na `c`.
`^a.*`	Odpowiada każdemu łańcuchowi rozpoczynającemu się na literę `a`.
`[a-c]*x$`	Odpowiada `x`, `ax`, `bx`, `abax`, `abcx` — dowolny ciąg złożony z pierwszych trzech liter alfabetu, po których następuje `x`.
`b[yo]k`	Odpowiada wyrazom `byk` lub `bok`.
`[^Zz]{5}`	Odpowiada każdemu łańcuchowi o długości 5 znaków, który nie zawiera małego lub dużego `z`.
`[[:digit:]]`	Odpowiada każdej cyfrze, równoważnik wyrażenia `[0-9]`.
`[[:<:]]a.*`	Odpowiada każdemu wyrazowi, który zaczyna się na `a`.

22.4. Stosowanie wyrażeń regularnych w skryptach PHP

Podstawową funkcją do wykonywania wyrażeń regularnych jest `ereg`. Funkcja ta testuje łańcuch na danym wyrażeniu regularnym, zwracając `TRUE`, jeżeli wzorzec opisany w wyrażeniu regularnym występuje w testowanym łańcuchu. W ten sposób można sprawdzić, czy

łańcuch posiada określoną formę. Przykładem może być kontrola poprawności formatu kodu pocztowego, składającego się z dwóch cyfr, po których następują myślnik i trzy cyfry. Pokazuje to listing 22.2. Efekt dziania przykładowego skryptu widać na rysunku 22.2.

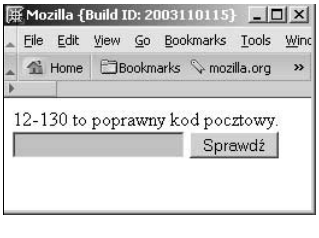
Listing 22.2. *Sprawdzanie kodu pocztowego*

```
<?php
    /*
    ** Sprawdza poprawność formatu kodu pocztowego
    ** skrypt ten sprawdza, czy dany ciąg jest kodem pocztowym
    ** który stanowią dwie cyfry, po których może pojawić się łącznik
    ** i kolejne trzy cyfry.
    */

    /*
    ** jeżeli przesłano kod, dokonuje jego analizy
    */
    if(isset($_REQUEST['kod']))
    {
        if(ereg("^([0-9]{2})(-[0-9]{3})?$", $_REQUEST['kod']))
        {
            print("{$_REQUEST['kod']} to poprawny kod pocztowy.<br>\n");
        }
        else
        {
            print("{$_REQUEST['kod']} <b>nie</b> " .
                "jest poprawnym kodem pocztowym.<br>\n");
        }
    }

    //tworzy formularz
    print("<form action=\"{$_SERVER['PHP_SELF']}\">\n");
    print("<input type=\"text\" name=\"kod\">\n");
    print("<input type=\"submit\" value=\"Sprawdź\">\n");
    print("</form>\n");
?>
```

Rysunek 22.2.
Efekt działania skryptu z listingu 22.2



Skrypt udostępnia formularz do wprowadzania kodu pocztowego. Musi on składać się z dwóch cyfr, po których następują myślnik i kolejne trzy cyfry. Działanie tego skryptu opiera się na następującym wyrażeniu regularnym:

```
^([0-9{2})(-[0-9]{3})?$
```

Do niego porównywane są informacje wprowadzone przez użytkownika. Pomocna będzie tu dokładna analiza wyrażenia.

Rozpoczyna się ono od karetki. Powoduje to szukanie trafień tylko od początku łańcucha. Jeżeli karetka zostałaby pominięta, kod pocztowy mógłby rozpoczynać się dowolną liczbą znaków, np. `abc12-456` i wciąż być zgodny z wyrażeniem regularnym. W podobny sposób znak dolara na końcu wyrażenia powoduje szukanie trafień od końca łańcucha. Zapobiega to trafieniom, takim jak `41-707abcd`. Kombinacja karetki i znaku dolara umożliwia wybieranie łańcuchów zawierających tylko wymagane znaki.

Pierwsze podwyrażenie to `([0-9]{2})`. Zakres w nawiasach kwadratowych pozwala na stosowanie tylko cyfr z zakresu od 0 do 9. Zawartość nawiasów klamrowych informuje, że cyfry muszą być dokładnie dwie.

Drugie podwyrażenie to `(-[0-9]{3})?`. W podobny sposób, jak w pierwszym wyrażeniu, określona została tu liczba cyfr na 3. Myślnik ma tu znaczenie dosłowne i musi poprzedzać 3 kolejne cyfry. Znak zapytania informuje, że całe podwyrażenie musi występować tylko raz lub w ogóle, co czyni podanie kolejnych trzech cyfr opcjonalnym.

W prosty sposób można przebudować takie wyrażenie, aby sprawdzało numery telefonów i daty. Wyrażenia regularne zapewniają świetny sposób kontroli wartości zmiennych zwracanych poprzez formularze. Zastępują one zagnieżdżone instrukcje `if` i przeszukiwanie łańcuchów funkcją `strpos`.

Można również sprawić, aby trafienia dotyczące poszczególnych podwyrażeń zostały zwrócone w tabeli. Jest to użyteczne w sytuacjach, kiedy trzeba podzielić łańcuch na części. Dobrym przykładem dla tej metody jest łańcuch, którym identyfikuje się przeglądarka. W łańcuchu tym zakodowane są: nazwa przeglądarki, wersja i typ komputera, na którym została uruchomiona. Pobranie tych informacji do oddzielnych zmiennych umożliwi dostosowanie strony do możliwości danej przeglądarki.

Listing 22.3 przedstawia skrypt, tworzący zestaw zmiennych, które pomagają ukryć stronę dla określonego typu przeglądarki. Dla przykładu uzależnimy zawartość hiperłącza od typu przeglądarki. Jeżeli użytkownik korzysta z Netscape Navigatora, łącze będzie wskazywało na stronę, z której można pobrać Microsoft Explorera. W innym wypadku łącze będzie wskazywało stronę, z której można pobrać Netscape Navigatora. Jest to przykład dostosowywania zawartości strony. Tą samą metodą można decydować również o użyciu zaawansowanych możliwości strony.

Listing 22.3. *Sprawdzanie http_user_agent*

```
<?php
    //sprawdza, z jakiej przeglądarki korzysta użytkownik
    //np. Mozilla/4.0 (compatible; MSIE 6.0; Windows NT 5.1; Q312461)
    ereg("^([[:alpha:]]+)/([[:digit:]\.]+)( .*)$",
        $_SERVER['HTTP_USER_AGENT'], $match);

    $browserName = $match[1];
    $browserVersion = $match[2];
    $browserDescription = $match[3];

    //szuka dowodów na to, że przeglądarka to MSIE
    if(eregi("msie", $browserDescription))
    {
        //szuka czegoś w stylu:
        //(compatible; MSIE 6.0; Windows NT 5.1; Q312461)
        eregi("MSIE ([[:digit:]\.]+);",
```

```
        $browserDescription, $match);

        $browserName = "MSIE";
        $browserVersion = $match[1];
    }

    print("Korzystasz z przeglądarki $browserName " .
        "wersja $browserVersion!<br>\n" .
        "Możesz spróbować również skorzystać z przeglądarki: ");

    if(eregi("mozilla", $browserName))
    {
        print("<a href=\"" .
            "http://www.microsoft.com/ie/download/default.asp\">");
        print("Internet Explorer");
        print("</a> ");
    }
    else
    {
        print("<a href=\"" .
            "http://www.netscape.com/computing/download/index.html" . "\">");
        print("Navigator");
        print("</a> ");
    }
    print("dla porównania.<br>\n");
?>
```

Główna funkcja `ereg` w skrypcie jest użyta bez warunku `if`. Zakłada ona, że przeglądarka dokona swojej identyfikacji co najmniej w postaci podania nazwy, ukośnika i numeru wersji. Tablicy `match` zostają przyporządkowane części rozpatrywanego łańcucha, które odpowiadają częściom wyrażenia regularnego. Wyrażenie to składa się z trzech podwyrażeń, odpowiadających nazwie, wersji i wszystkim pozostałym charakterystykom. Forma ta jest stosowana przez większość przeglądarek, w tym Navigatora i Internet Explorera. Jako że Internet Explorer zawsze identyfikuje się jako przeglądarka Mozilla (Netscape), należy wykonać dodatkowe kroki, aby określić, czy przeglądarka jest rzeczywiście produktem Netscape, czy go tylko udaje. Odpowiada za to wywołanie `eregi`.

Dlaczego element zerowy tablicy jest pomijany? Otóż przechowuje on podciąg, który został trafiony przez całe wyrażenie regularne. W omawianej sytuacji trafienia takie nie mają znaczenia. Zwykle element zerowy okazuje się przydatny, gdy poszukujemy określonego łańcucha, zawartego w ramach szerszego kontekstu. Przykładem może być przeszukiwanie treści strony w poszukiwaniu URL-u. Listing 22.4 pobiera stronę domową PHP i tworzy listę wszystkich łączy znajdujących się na tej stronie. Efekt działania jest widoczny na rysunku 22.3.

Listing 22.4. *Poszukiwanie adresów URL na stronie*

```
<?php
    //adres URL do pobrania linków
    $URL = "http://www.php.net/";

    //otwiera plik
    $page = fopen($URL, "r");

    print("Linki na stronie $URL<br>\n");
    print("<ul>\n");
```

Rysunek 22.3.
Efekt działania skryptu z listingu 22.4




```
    while(!feof($page))
    {
        //pobiera wiersz
        $line = fgets($page, 1024);

        //wykonuje pętle, o ile obecne są jeszcze jakieś linki
        while(eregi("href=\"[^\"]*\"", $line, $match))
        {
            //wyświetla adresy URL
            print("<li>{$match[0]}</li>\n");

            //usuwa adres URL z wiersza
            $replace = ereg_replace("\?", "\?", $match[0]);
            $line = ereg_replace($replace, "", $line);
        }
    }
    print("</ul>\n");

    fclose($page);
?>
```

Główna pętla skryptu pobiera wiersze tekstu ze strumienia pliku i szuka właściwości `href`. Jeżeli właściwość taka zostanie znaleziona w wierszu, jest on umieszczany w zerowym elemencie tablicy `match`. Następnie skrypt wyświetla zawartość tego elementu i usuwa go z wiersza za pomocą funkcji `ereg_replace`. Funkcja ta zamienia tekst pasujący do wyrażenia regularnego na dany łańcuch. W tym przypadku właściwość `HREF` zostaje zastąpiona łańcuchem pustym. Powodem usunięcia łącza po jego odnalezieniu jest możliwość znalezienia dwóch łączy w jednym wierszu kodu HTML. Funkcja `ereg` trafi wówczas tylko pierwszy podciąg. Rozwiązaniem jest odnalezienie i usunięcie wszystkich łączy.

Jak widać, w chwili usuwania łącza tworzona jest zmienna `replace`. Niektóre łącza mogą zawierać znak zapytania — dopuszczalny znak w URL-u, który oddziela nazwę pliku od zmiennych formularza. Jako że znak ten ma specjalne znaczenie w wyrażeniach regularnych, skrypt umieszcza przed nim znak `\`, aby umożliwić jego dosłowną interpretację.

Często używam funkcji `ereg_replace` do konwersji treści na potrzeby nowego kontekstu. Można na przykład zastosować `ereg_replace` do zastąpienia kilku spacji jedną. Pokazuje to listing 22.5. Efekt działania skryptu z listingu widać na rysunku 22.4.

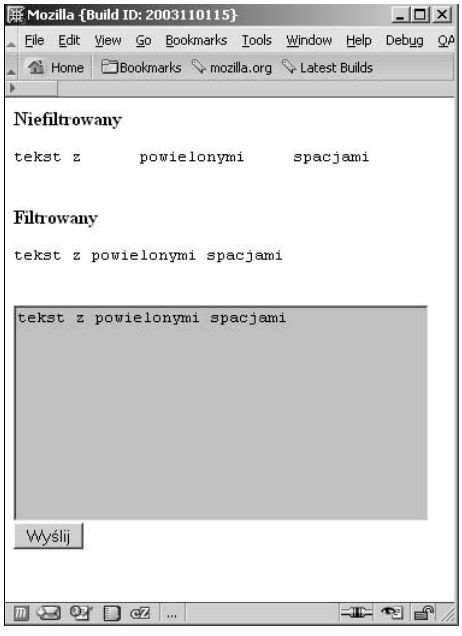
Listing 22.5. *Zamiana powielonych spacji*

```
<?php
    /*
    ** jeżeli przesłano tekst, ukazuje go
    */
    if(isset($_REQUEST['text']))
    {
        print("<b>Niefiltrowany</b><br>\n" .
            "<pre>{$_REQUEST['text']}</pre>" .
            "<br>\n");

            $_REQUEST['text'] = ereg_replace("[[:space:]]+",
            " ", $_REQUEST['text']);

        print("<b>Filtrowany</b><br>\n" .
            "<pre>{$_REQUEST['text']}</pre>" .
            "<br>\n");
    }
    else
    {
        $_REQUEST['text'] = "";
    }


    //tworzy formularz
    print("<form action=\"{$_SERVER['PHP_SELF']}\">\n" .
        "<textarea name=\"text\" cols=\"40\" rows=\"10\">" .
        "{$_REQUEST['text']}</textarea><br>\n" .
        "<input type=\"submit\" value=\"Wyślij\">\n" .
        "</form>\n");
?>
```



Rysunek 22.4.
Efekt działania skryptu z listingu 22.5