

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

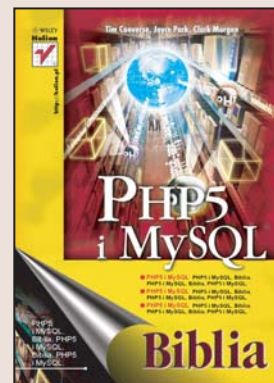
FRAGMENTY KSIĄŻEK ONLINE

PHP i MySQL. Biblia

Autorzy: Tim Converse, Joyce Park, Clark Morgan
Tłumaczenie: Robert Górczyński, Daniel Kaczmarek,
Maja Królikowska, Marek Pałczyński
ISBN: 83-7361-940-2

Tytuł oryginału: [PHP5 and MySQL Bible](#)

Format: B5, stron: 1080



Wyczerpujące omówienie najpopularniejszego środowiska do tworzenia dynamicznych witryn WWW

- Poznaj zasady programowania w PHP5 i w pełni wykorzystaj jego nowe możliwości
- Zastosuj bazę danych jako zaplecze dla stron WWW
- Zaprojektuj i stwórz dynamiczną witrynę WWW za pomocą PHP i MySQL

O PHP i MySQL słyszał już chyba każdy, kto zajmuje się projektowaniem witryn WWW. Skrypty napisane w języku PHP i korzystające z bazy danych MySQL „napędzają” tysiące dynamicznych stron WWW – sklepów, portali oraz aplikacji e-commerce. Duet PHP i MySQL to stabilne, wydajne i proste do opanowania środowisko o ogromnych możliwościach. Dostępne są nieodpłatnie, co jest jednym z głównych powodów ich popularności. Za ich pomocą można stworzyć zarówno proste wiadomości na stronie WWW, jak i rozbudowane systemy autoryzacji użytkowników, płatności elektronicznych, galerii i wiele innych aplikacji internetowych.

„PHP5 i MySQL. Biblia” to kompleksowe omówienie zagadnień związanych z tworzeniem witryn WWW z wykorzystaniem tych technologii. Przedstawia zasady programowania w języku PHP5 z uwzględnieniem reguł projektowania obiektowego. Opisuje bazę danych MySQL, język SQL oraz sposoby połączenia skryptów PHP z tabelami w bazie danych. Czytając ją, nauczysz się osadzać kod PHP w dokumentach HTML, korzystać z mechanizmów obsługi sesji oraz języka XML, korzystać z innych baz danych oraz zabezpieczać witryny WWW stworzone za pomocą PHP. Dowiesz się, jak zaimplementować obsługę wyjątków oraz jak usuwać błędy z kodu źródłowego.

- Typy danych, zmienne, polecenia i funkcje w PHP
- Operacje na tekstach, wartościach liczbowych i tablicach
- Administrowanie bazą MySQL
- Łączenie skryptów PHP z bazą danych
- Tworzenie elementów generowanych dynamicznie
- Programowanie obiektowe w PHP
- Mechanizmy obsługi sesji i plików cookie
- Korzystanie z biblioteki PEAR
- Obsługa wyjątków i usuwanie błędów z kodów źródłowych
- Korzystanie z baz danych PostgreSQL oraz Oracle
- Łączenie PHP z JavaScript i Javą
- Przetwarzanie plików XML i tworzenie usług sieciowych
- Generowanie grafiki za pomocą PHP
- Uwierzytelnianie użytkowników

**W tej książce znajdziesz wszystko, czego potrzebujesz
aby w pełni wykorzystać możliwości PHP i MySQL**

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



Spis treści

O autorach	27
Przedmowa	29
Część I Podstawy PHP	35
Rozdział 1. Dlaczego PHP i MySQL?	37
Co to jest PHP?	37
Co to jest MySQL?	38
Historia PHP	39
Historia MySQL	40
Dlaczego kochamy PHP?	40
Koszt	41
PHP jest łatwy	43
PHP można wbudować	44
PHP jest niezależny od platformy	46
PHP nie bazuje na znacznikach	46
PHP jest stabilny	46
PHP jest szybki	47
PHP jest otwarty	48
PHP dobrze współpracuje z innymi produktami	49
Szybki rozwój języka	50
Popularność PHP rośnie	50
PHP nie jest niczyją własnością	51
Społeczność PHP	52
Podsumowanie	53
Rozdział 2. Skrypty wykonywane na serwerze WWW	55
Statyczny HTML	55
Technologie wykonywane po stronie klienta	58
Skrypty wykonywane na serwerze	62
Do czego przydają się skrypty serwera	67
Podsumowanie	68
Rozdział 3. Rozpoczynamy pracę z PHP	69
Dzierżawa lub własny serwer	69
Wariant z dostawcą Internetu	69
Własny serwer — wady i zalety	73
Rozwiązania pośrednie	73

Instalowanie PHP	74
Zanim zaczniesz	74
Procedura instalacji	76
Narzędzia programistyczne	83
Podsumowanie	87
Rozdział 4. Dodajemy PHP do HTML	89
HTML jest gotowy na PHP	89
Przełączanie się z HTML do PHP	89
Kanoniczne znaczniki PHP	90
Krótkie znaczniki otwierające (w stylu SGML)	90
Witaj świecie	91
Wejście w tryb PHP i wyjście z niego	92
Dołączanie plików	93
Podsumowanie	95
Rozdział 5. Składnia i zmienne	97
PHP wiele wybacza	97
HTML to nie PHP	98
Składnia PHP bazuje na C	98
PHP nie przejmuje się odstępami	98
PHP jest czasami wrażliwy na wielkość liter	98
Instrukcje to wyrażenia zakończone średnikiem	99
Bloki	102
Komentarze	102
Komentarze wielowierszowe w stylu C	103
Komentarze jednowierszowe: # i //	103
Zmienne	103
PHP przyjął styl zmiennych Perl	104
Deklarowanie zmiennych	104
Przypisywanie zmiennym wartości	104
Zmiana wartości zmiennych	105
Nieprzypisane zmienne	105
Zasięg zmiennych	106
Możesz dowolnie zmieniać tryby pracy	107
Stałe	108
Typy w PHP nie są źródłem zmartwień	108
Brak deklaracji typów zmiennych	109
Automatyczna konwersja typów	109
Typy nadawane poprzez kontekst	109
Typy w PHP	109
Typy proste	110
Integer	110
Double	111
Boolean	113
NULL	114
String	115
Wyjście	119
Echo i print	119
Zmienne i ciągi	120
Podsumowanie	121

Rozdział 6. Sterowanie i funkcje	123
Wyrażenia logiczne	124
Stałe logiczne	124
Operatory logiczne	124
Operatory porównania	126
Operator trójskładnikowy	128
Instrukcje warunkowe	129
If-else	129
Switch	133
Pętle	134
Pętle ograniczone i nieograniczone	134
While	134
Do-while	135
For	136
Przykłady pętli	137
Break i continue	140
Pętle nieskończone	141
Składnia alternatywna	142
Przerywanie wykonania	142
Użycie funkcji	145
Zwracane wartości a efekty uboczne	145
Dokumentacja funkcji	146
Nagłówki w dokumentacji	147
Szukanie opisu funkcji	147
Definiowanie własnych funkcji	148
Czym jest funkcja?	148
Składnia definicji funkcji	148
Przykład definicji funkcji	149
Parametry formalne i parametry aktualne	150
Nieprawidłowa liczba argumentów	151
Funkcje a zasięg zmiennych	152
Zmienne globalne i lokalne	153
Zmienne statyczne	154
Wyjątki	155
Zasięg funkcji	156
Include oraz require	157
Rekurencja	158
Podsumowanie	160
Rozdział 7. Przekazywanie danych pomiędzy stronami	161
HTTP jest protokołem bezstanowym	161
Argumenty GET	162
Inne zastosowania adresów URL w stylu GET	164
Argumenty POST	166
Formatowanie zmiennych formularza	167
Konsolidacja formularzy i obsługujących je kodów	170
Używanie zmiennych tablicowych w połączeniu z formularzami	171
Tablice superglobalne PHP	174
Rozszerzony przykład: kalkulator do ćwiczeń	176
Podsumowanie	178

Rozdział 8. Ciągi	179
Ciągi w PHP	179
Dołączanie ciągów przy użyciu nawiasów klamrowych	180
Znaki i indeksy ciągu	181
Operatory dla ciągów	181
Złączenie i przypisanie	182
Składnia heredoc	182
Funkcje operujące na ciągach	183
Sprawdzanie ciągów	184
Szukanie znaków i podciągów	184
Porównywanie i przeszukiwanie	185
Przeszukiwanie	186
Wycinanie podciągu	187
Funkcje porządkujące	188
Zastępowanie ciągów	189
Funkcje zmiany wielkości liter	191
Funkcje znaków sterujących	192
Formatowanie danych	193
Rozszerzony przykład: kalkulator do ćwiczeń	195
Podsumowanie	198
Rozdział 9. Tablice i funkcje operujące na tablicach	199
Użycie tablic	199
Czym są tablice PHP?	200
Tworzenie tablic	202
Bezpośrednie przypisanie	202
Konstrukcja array()	202
Podawanie indeksów przy użyciu array()	203
Funkcje zwracające tablice	203
Odczytywanie wartości	204
Odczytywanie przy użyciu indeksu	204
Konstrukcja list()	204
Tablice wielowymiarowe	205
Informacje o tablicach	206
Usuwanie z tablicy	207
Iteracje	207
Obsługa iteracji	208
Użycie funkcji iteracyjnych	209
Nasza ulubiona metoda iteracji: foreach	209
Iteracje za pomocą current() i next()	211
Powtórne przeglądanie za pomocą reset()	212
Wypisywanie w odwrotnym porządku za pomocą end() i prev()	213
Pobieranie wartości kluczy za pomocą key()	213
Wartości puste i funkcja each()	214
Przeglądanie tablicy za pomocą array_walk()	215
Rozszerzony przykład: kalkulator do ćwiczeń	216
Podsumowanie	229

Rozdział 10. Liczby	231
Typy numeryczne	231
Operatory matematyczne	232
Operatory arytmetyczne	232
Operatory arytmetyczne i typy	232
Operator inkrementacji	233
Operator przypisania	234
Operatory porównania	234
Kolejność operacji i nawiasy	235
Proste funkcje matematyczne	236
Liczby losowe	236
Inicjowanie generatora	237
Przykład: losowy wybór	238
Rozszerzony przykład: kalkulator do ćwiczeń	240
Podsumowanie	245
Rozdział 11. Podstawowe pułapki PHP	247
Problemy związane z instalacją	247
Źródło pliku wyświetlane w przeglądarce	248
Blok PHP pokazuje się jako tekst lub przeglądarka chce zapisać plik	248
Nieodnaleziony serwer lub niemożliwe wyświetlenie strony	248
Problemy z wyświetlaniem	248
Całkowicie pusta strona	249
Dokument nie zawiera żadnych danych	249
Niekompletna lub nieprawidłowa strona	250
Kod PHP pokazuje się w przeglądarce	253
Niepowodzenie przy ładowaniu strony	254
Nieodnaleziona strona	254
Nieudane otwarcie pliku do włączenia	254
Błędy analizy składni	254
Komunikat błędu składni	254
Brakujący średnik	255
Brak znaku \$	255
Nieprawidłowa zmiana trybu	256
Nieoznaczone apostrofy	257
Niezakończone ciągi znaków	257
Inne błędy składni	258
Uprawnienia do plików	258
Błąd HTTP nr 403	258
Brak dołączanych plików	259
Ostrzeżenie przy włączaniu pliku	259
Nieprzypisane zmienne	260
Zmienna nie pokazuje się w wynikowym ciągu	260
Zmienna liczbowa nieoczekiwanie przyjmuje wartość zero	260
Jak zachowują się niezainicjowane zmienne	260
Nadpisane zmienne	262
Zmienna ma poprawną wartość, ale nie taką, jakiej oczekiwałeś	262
Problemy z funkcjami	263
Wywołanie niezdefiniowanej funkcji moja_funkcja()	263
Wywołanie niezdefiniowanej funkcji	264
Wywołanie niezdefiniowanej funkcji array()	264

Nie można ponownie zadeklarować funkcji	264
Nieprawidłowa liczba argumentów	265
Błędy matematyczne	265
Ostrzeżenie o dzieleniu przez zero	265
Niespodziewane wyniki działań	265
NaN (lub NAN)	265
Przekroczenie czasu oczekiwania	266
Podsumowanie	267

Część II PHP i bazy danych 271

Rozdział 12. Wybór bazy danych dla PHP 273

Czym jest baza danych?	273
Dlaczego używamy baz danych?	274
Utrzymanie i skalowalność	274
Przenośność	274
Unikanie nudnego programowania	274
Szukanie	275
Bezpieczeństwo	275
Architektura wielowarstwowa	275
Potencjalna wada: wydajność	276
Wybór bazy danych	276
Możesz nie mieć wyboru	276
Plikowe, relacyjne i obiektowo-relacyjne bazy danych	277
ODBC/JDBC kontra własne API	278
Zmiana bazy danych	278
Przegląd zaawansowanych funkcji	279
GUI	279
Podzapytania	279
Zapytanie SELECT INTO	279
Złożone złączenia	280
Wielowątkowość i blokowanie	280
Transakcje	280
Procedury i wyzwalacze	281
Indeksy	281
Klucze obce i więzy integralności	282
Replikacja bazy danych	282
Bazy danych obsługiwane przez PHP	282
Warstwa abstrakcji bazy danych (lub jej brak)	284
Wybieramy MySQL	285
Podsumowanie	286

Rozdział 13. Samouczek SQL 287

Relacyjne bazy danych i SQL	287
Standardy SQL	288
Podstawowe wyrażenia SQL	288
SELECT	289
INSERT	293
UPDATE	294
DELETE	294
Projekt bazy danych	294

Bezpieczeństwo i uprawnienia	297
Ustawianie uprawnień	297
Przechowywanie haseł w innym miejscu	298
Użycie dwóch warstw ochrony haseł	299
Tworzenie kopii bezpieczeństwa	300
Podsumowanie	300
Rozdział 14. Administracja bazą danych MySQL	303
Licencje MySQL	304
Instalowanie MySQL — przejście do wersji 4.	305
Czynności przed instalacją	305
Pobieranie MySQL	307
Instalowanie MySQL w Windows	307
Instalowanie MySQL w systemie Unix	308
Instalowanie MySQL w systemie Mac OS X	309
Czynności poinstalacyjne	309
Podstawowe komendy klienta MySQL	310
Administracja użytkownika MySQL	311
Programowanie lokalne	314
Samodzielna strona internetowa	314
Współdzielone utrzymywanie stron internetowych	315
PHPMyAdmin	315
Kopie zapasowe	319
Powielanie	320
Odzyskiwanie	323
mysamchk	324
mysqlcheck	325
Podsumowanie	325
Rozdział 15. Funkcje PHP i MySQL	327
Łączenie z MySQL	328
Tworzenie zapytań w MySQL	329
Pobieranie wyniku	330
Pobieranie opisu danych	333
Korzystanie z wielokrotnych połączeń	334
Kontrola błędów	335
Tworzenie baz danych MySQL za pomocą PHP	337
Typy danych MySQL	338
Funkcje MySQL	340
Podsumowanie	342
Rozdział 16. Wyświetlanie zapytań w tabelach	345
Tabele HTML i tabele bazy danych	346
Przekształcenie jeden w jeden	346
Przykład: wyświetlanie jednej tabeli	346
Przykładowe tabele	348
Ulepszanie wyświetlania	349
Złożone odwzorowania	352
Wiele zapytań albo skomplikowane wyświetlanie	352
Użycie kilku zapytań	353
Przykład skomplikowanego wyświetlania	355
Tworzenie przykładowych tabel	356
Podsumowanie	358

Rozdział 17. Tworzenie formularzy z zapytań	359
Formularze HTML	359
Podstawowy formularz zatwierdzania danych do bazy danych	360
Samoprzetwarzanie	362
Edytowanie danych w formularzu HTML	369
TEXT i TEXTAREA	369
CHECKBOX	371
RADIO	373
SELECT	377
Podsumowanie	379
Rozdział 18. Efektywność PHP i MySQL	381
Połączenia — ograniczanie i powtórne użycie	381
Przykład nieprawidłowego użycia: jedno połączenie na wyrażenie	382
Kilka wyników nie wymaga kilku połączeń	383
Trwałe połączenia	383
Indeksy i projekt tabel	384
Indeksowanie	384
Indeksowanie wszystkich pól	387
Inne rodzaje indeksów	388
Projekt tabeli	390
Przenoszenie pracy na serwer bazy danych	390
Baza jest szybsza od Ciebie	391
Przykład nieprawidłowego użycia: pętla zamiast warunku	391
Tworzenie pól daty i czasu	393
Szukanie ostatnio wstawionego wiersza	394
Podsumowanie	395
Rozdział 19. Pułapki tandemu PHP-MySQL	397
Brak połączenia	397
Problemy z uprawnieniami	399
Nieoznaczone apostrofy	401
Nieprawidłowe zapytania SQL	403
Pomyłki w nazwach	405
Pomyłki przy przecinkach	405
Ciągi nieotoczone apostrofami	405
Niezainicjowane zmienne	406
Zbyt mało danych, zbyt dużo danych	406
Funkcje języka SQL	407
mysql_affected_rows() kontra mysql_num_rows()	407
mysql_result()	408
OCIFetch()	408
Kontrola poprawności	408
Podsumowanie	409
Część III Zaawansowane funkcje i techniki	411
Rozdział 20. Programowanie zorientowane obiektowo w PHP	413
Co to jest programowanie zorientowane obiektowo?	414
Prosty pomysł	414
Opracowanie: obiekty jako typy danych	416
Opracowanie: dziedziczenie	416

Opracowanie: enkapsulacja	417
Opracowanie: konstruktory i destruktory	418
Terminologia	418
Podstawowe konstrukcje PHP dla OOP	420
Definiowanie klas	420
Dostęp do zmiennych składowych	421
Tworzenie egzemplarzy	421
Funkcje konstruktora	422
Dziedziczenie	423
Nadpisywanie funkcji	424
Połączone podklasy	424
Modyfikowanie i przypisywanie obiektów	426
Kwestie zasięgu	426
Zaawansowane funkcje OOP	427
Elementy publiczne, prywatne i chronione	427
Interfejsy	429
Stałe	429
Klasy abstrakcyjne	429
Symulowanie funkcji klas	430
Wywoływanie funkcji rodzica	431
Automatyczne wywołanie konstruktorów rodzica	432
Symulowanie metod przeciążenia	433
Serializacja	434
Funkcje introspekcji	436
Przegląd funkcji	437
Przykład: genealogia klas	439
Przykład: dopasowywanie zmiennych i kolumn bazy danych	442
Przykład: uogólniony test metod	444
Rozszerzony przykład: formularze HTML	447
Rozwiązywanie problemów	452
Symptom: zmienna składowa nie ma wartości w funkcji składowej	452
Symptom: błąd analizatora składni, oczekuję T_VARIABLE	452
Style OOP w PHP	453
Konwencje nazywania	453
Dodatki funkcji	453
Projektowanie dziedziczenia	454
Podsumowanie	455
Rozdział 21. Zaawansowane funkcje operujące na tablicach	457
Przekształcenia tablic	457
Pobieranie kluczy i wartości	458
Zamiana, odwracanie i mieszanie	459
Łączenie, dopełnianie, wycinanie i zastępowanie	460
Stosy i kolejki	461
Zamiana pomiędzy tablicą i zmiennymi	464
Sortowanie	465
Funkcje wyświetlające tablice	466
Podsumowanie	467

Rozdział 22. Łącuch i funkcje wyrażeń regularnych	469
Tokenizing i funkcje analizatora składni	470
Dlaczego wyrażenia regularne?	472
Wyrażenia regularne w PHP	473
Przykład wyrażeń stylu POSIX	473
Funkcje wyrażeń regularnych	475
Wyrażenia regularne zgodne z Perl	476
Przykład: prosty program do wyodrębniania hiperłączy	479
Wyrażenie regularne	479
Używanie wyrażenia w funkcji	481
Zaawansowane funkcje łańcuchów	483
Funkcje HTML	483
Mieszanie przy użyciu MD5	483
Łącuchy jako zbiory znaków	485
Funkcje podobne do łańcuchów	487
Podsumowanie	487
Rozdział 23. Funkcje systemu operacyjnego i dostępu do plików	489
Uprawnienia do plików PHP	489
Funkcje czytania i zapisywania plików	490
Otwarcie pliku	491
Czytanie pliku	494
Tworzenie mechanizmu pobierania plików przy użyciu fpasssthu()	496
Zapis do pliku	497
Zamknięcie pliku	498
Funkcje systemu plików i katalogów	498
feof	501
file_exists	501
filesize	501
Funkcje sieciowe	501
Funkcje logu systemowego	501
Funkcje DNS	502
Funkcje gniazd	502
Funkcje daty i czasu	502
Jeżeli nie znasz daty ani czasu	503
Jeżeli już odczytałeś datę i czas albo znacznik czasu	504
Funkcje konwersji kalendarza	505
Podsumowanie	506
Rozdział 24. Sesje, cookies i HTTP	507
Czym są sesje?	507
Co stanowi problem?	508
Dlaczego się tym zajmujemy?	508
Alternatywy sesji	508
Adres IP	509
Ukryte zmienne	509
Cookie	510
Jak działają sesje w PHP	510
Uaktywnianie sesji w PHP	511
Dystrybucja zmiennych sesji	511
Gdzie są przechowywane dane?	514

Przykładowy kod sesji	515
Funkcje obsługi sesji	518
Zagadnienia konfiguracji	520
Cookies	521
Funkcja setcookie()	522
Przykłady	523
Usuwanie cookie	524
Odczytywanie cookie	524
register_globals i nadpisywanie zmiennych	525
Pułapki cookie	526
Wysyłanie nagłówków HTTP	528
Przykład: przekierowanie	528
Przykład: uwierzytelnianie HTTP	529
Pułapki związane z nagłówkami	530
Pułapki i wykrywanie usterek	530
Podsumowanie	531
Rozdział 25. Typy i rodzaje konwersji	533
Typ zaokrąglający	533
Zasoby	534
Co to są zasoby?	534
Jak obsługiwać zasoby	535
Typ testujący	535
Przypisanie i koercja	535
Przekroczenie typu liczb całkowitych	540
Odnajdywanie największej liczby całkowitej	541
Podsumowanie	541
Rozdział 26. Zaawansowane użycie funkcji	543
Zmienna liczba argumentów	543
Argumenty domyślne	544
Tablice jako substytut wielu argumentów	544
Wiele argumentów w PHP4 i wersjach późniejszych	545
Wywołanie przez wartość	547
Wywołanie przez referencję	548
Zmienne jako nazwy funkcji	549
Bardziej skomplikowany przykład	549
Podsumowanie	553
Rozdział 27. Matematyka	555
Stałe matematyczne	555
Sprawdzanie liczb	556
Konwersja podstawy	557
Funkcje wykładnicze i logarytmy	560
Trygonometria	561
Arytmetyka o dowolnej dokładności (BC)	563
Przykład użycia funkcji o dowolnej dokładności	565
Konwersja obliczeń na dowolną dokładność	565
Podsumowanie	567

Rozdział 28. PEAR	569
Co to jest PEAR?	570
System pakietów PEAR	571
Próbka pakietów PEAR	571
Jak działa baza danych PEAR	572
Menedżer pakietów	572
Używanie menedżera	576
PHP Foundation Classes (PFC)	577
PHP Extension Code Library (PECL)	578
Styl kodowania PEAR	578
Wcięcia, odstępy i długość linii	578
Formatowanie struktur kontrolnych	579
Formatowanie funkcji i wywołań funkcji	580
Podsumowanie	581
Rozdział 29. Bezpieczeństwo	583
Możliwe ataki	584
Zniszczenie witryny	584
Dostęp do kodu źródłowego	585
Odczytywanie dowolnych plików	587
Uruchamianie dowolnych programów	590
Wirusy i inne e-stworzenia	591
Bezpieczeństwo poczty elektronicznej	592
Register-globals	593
Przekazywanie plików	595
Szyfrowanie	599
Szyfrowanie kluczem publicznym	600
Szyfrowanie pojedynczym kluczem	601
Szyfrowanie cookies	603
Mieszanie	604
Cyfrowe podpisywanie plików	605
Secure Sockets Layer	606
Dla Twojej informacji: bezpieczne witryny sieciowe	607
Podsumowanie	607
Rozdział 30. Konfiguracja	609
Podglądanie zmiennych środowiska	609
Poznajemy konfigurację PHP	609
Opcje kompilacji	610
Opcje kompilacji dla postaci CGI	614
Pliki konfiguracyjne Apache	616
Plik php.ini	618
Poprawianie wydajności PHP	622
Podsumowanie	624
Rozdział 31. Wyjątki i obsługa błędów	627
Obsługa błędów w PHP5	627
Błędy i wyjątki	628
Klasa wyjątku	629
Blok try/catch	630
Zgłaszanie wyjątku	630
Definiowanie własnych podklas Exception	631
Ograniczenia wyjątków w PHP	633

Inne metody obsługi błędów	633
Natywne błędy PHP	633
Definiowanie procedury obsługi błędów	635
Wyzwalanie błędów użytkownika	635
Odnutowywanie w dzienniku i debugowanie	636
Podsumowanie	637
Rozdział 32. Debugowanie	639
Ogólne strategie rozwiązywania problemów	640
Jednorazowo zmiana jednej rzeczy	640
Spróbuj odizolować problem	640
Upraszczaj, później rozbudowuj	640
Sprawdź oczywiste rzeczy	640
Dokumentuj swoje rozwiązania	641
Po naprawie przetestuj	641
Menażeria błędów	641
Błędy w trakcie kompilacji	641
Błędy w trakcie działania	641
Błędy logiczne	642
Używanie dzienników zdarzeń serwera sieciowego	642
Apache	642
IIS	644
Zgłaszanie błędów PHP i odnotowywanie w dzienniku	644
Zgłaszanie błędów	645
Zapisywanie błędów w dzienniku	645
Wybór błędów do zgłaszania lub odnotowania w dzienniku	646
Funkcje zgłaszania błędów	646
Wyświetlane komunikaty diagnostyczne	646
Używanie print_r()	647
Używanie syslog()	648
Odnutowywanie do dziennika we własnej lokalizacji	649
Używanie error_log()	650
Graficzne narzędzia do debugowania	651
Unikanie błędów na pierwszym miejscu	651
Odnajdywanie błędów, jeśli się pojawią	652
Podsumowanie	653
Rozdział 33. Styl	655
Zalety prawidłowego stylu	655
Czytelność	656
Komentarze	658
PHPDoc	659
Nazwy zmiennych i plików	659
Jednolitość stylu	661
Łatwość konserwacji	662
Unikaj magicznych liczb	662
Funkcje	662
Pliki dołączane	663
Interfejs obiektowy	663
Użycie programu kontroli wersji	664

Solidność	664
Niedostępność usługi	664
Niespodziewany typ zmiennej	665
Zwięzłość i wydajność	665
Używaj właściwych algorytmów	665
Poprawianie wydajności	666
Zwięzłość: zmniejszanie	667
Wskazówki na temat zwięzłości	668
Tryb HTML czy PHP?	670
Minimalny PHP	671
Maksymalny PHP	672
Średni PHP	672
Styl heredoc	673
Oddzielanie kodu od projektu	675
Funkcje	675
Arkusze stylów w PHP	675
Szablony i spójność stron	676
Podsumowanie	677
Część IV Połączenia	679
Rozdział 34. PostgreSQL	681
Dlaczego wybrać PostgreSQL?	681
Dlaczego tak czy owak obiektowo relacyjna?	683
Instalowanie PostgreSQL	683
Instalacja w systemie Linux	684
Ale czy jest to jeszcze baza danych?	685
Aż do prawdziwej pracy	687
PHP i PostgreSQL	688
Baza danych Cartoons	688
Podsumowanie	696
Rozdział 35. Oracle	697
Kiedy potrzebujesz Oracle?	697
Pieniądze	698
Inne konkurencyjne zasoby	698
Ogromne zestawy danych	699
Wiele dużych zapisów lub błędne przekształcanie danych	699
Wyzwalacze	699
Odpowiedzialność prawna	700
Dwuletnie perspektywy	700
Oracle i architektura sieciowa	700
Wyspecjalizowani członkowie zespołu	700
Wspólne konstruowanie baz danych	701
Ograniczone zmiany schematu	701
Narzędzia (lub ich brak)	701
Replikacja i poprawna praca mimo usterek	701
Buforowanie danych	702

Używanie funkcji OCI8	702
Sterowanie łańcuchami	703
Przetwarzanie i wykonywanie	703
Zgłaszanie błędów	704
Zarządzanie pamięcią	704
Prośzenie o wartości null	704
Pobieranie całych zestawów danych	704
Duże litery	704
Transakcyjność	705
Procedury składowane i kursory	706
Projekt: edytor	707
Projekt: wsadowy edytor produktów	715
Podsumowanie	723
Rozdział 36. Funkcje bazodanowe PEAR	725
Dyskusyjne zalety niezależności od baz danych	725
Rdzenne mechanizmy komunikacji z bazami danych	728
Abstrakcja bazodanowa	729
Zasady działania PEAR DB	730
Nazwy źródeł danych (DSN)	730
Połączenie	732
Wykonywanie zapytań	732
Odczytywanie wierszy	733
Rozłączanie	733
Kompletny przykład	733
Funkcje PEAR DB	735
Składowe klasy DB	735
Składowe klasy DB_Common	735
Składowe klasy DB_Result	736
Podsumowanie	736
Rozdział 37. E-mail	737
Informacje na temat architektury e-mail	737
Serwer TCP/IP	738
Mail Transfer Agent (MTA), czyli serwer SMTP	739
Kolejka poczty	740
Mail User Agent, czyli lokalny klient pocztowy	740
Program pobierający pocztę (serwer POP/IMAP)	740
Zarządca list wysyłkowych	742
Pobieranie poczty za pomocą PHP	742
Tworzenie od podstaw	743
Tworzenie na przykładzie	743
Tworzenie przez upiększanie	744
Wysyłanie poczty za pomocą PHP	744
Konfiguracja Windows	744
Konfiguracja Uniksa	745
Funkcja mail	745

Więcej na temat aplikacji pocztowych	747
Wysyłanie poczty z formularza	747
Wysyłanie poczty przy użyciu bazy danych	749
Wysyłanie załączników w poczcie MIME	750
Własna aplikacja pocztowa w PHP	752
Wysyłanie poczty z cronjob	754
Problemy w czasie korzystania z poczty	756
Podsumowanie	757
Rozdział 38. PHP i JavaScript	759
Tworzenie kodu JavaScript w PHP	759
Pojedynki obiektów	760
PHP nie analizuje wysyłanych danych	760
Kiedy używać JavaScript	761
PHP jako koło zapasowe do JavaScript	762
JavaScript statyczny kontra dynamiczny	764
Dynamiczna generacja formularzy	765
Przesyłanie danych z JavaScript do PHP	769
Podsumowanie	772
Rozdział 39. PHP i Java	775
PHP dla programistów Java	775
Podobieństwa	776
Różnice	777
Java Server Pages i PHP	778
Przewodnik po książce	779
Integrowanie języków PHP i Java	780
Java SAPI	780
Rozszerzenie Java	781
Obiekt Java	783
Błędy i wyjątki	784
Możliwe problemy	785
Bez żadnych ograniczeń	786
Podsumowanie	786
Rozdział 40. PHP i XML	789
Co to jest XML?	789
Praca z XML	792
Dokumenty i DTD	793
Struktura DTD	795
Analizatory kontrolujące i niekontrolujące poprawności	797
SAX kontra DOM	797
DOM	798
Stosowanie DOM XML	799
Funkcje DOM	799
SAX	801
Użycie SAX	802
Opcje SAX	803
Funkcje PHP dla SAX	804

API SimpleXML	805
Użycie SimpleXML	805
Funkcje SimpleXML	805
Przykładowa aplikacja SAX	806
Pułapki i wyszukiwanie błędów	812
Podsumowanie	812
Rozdział 41. Usługi sieciowe	815
Koniec programowania, jakie znamy	815
Smutna prawda o wymianie danych	816
Pełna prostota	816
REST, XML-RPC, SOAP, .NET	819
REST	820
XML-RPC	820
SOAP	821
Usługi .NET	823
Słabe strony dzisiejszych usług sieciowych	823
Duże i wolne	823
Prawdopodobne znaczne obciążenie	823
Standardy	824
Ukrywanie i wyszukiwanie	824
Kto za to płaci i w jaki sposób?	825
Projekt: klient REST	825
Projekt: serwer i klient SOAP	829
Podsumowanie	833
Rozdział 42. Grafika	835
Dostępne możliwości	835
Grafiki HTML	836
Tworzenie obrazków przy użyciu biblioteki gd	840
Czym jest biblioteka gd?	841
Formaty obrazków i przeglądarki	841
Wybór wersji	842
Instalacja	843
Zasady działania biblioteki gd	844
Funkcje	847
Obrazki i protokół HTTP	848
Przykład: fraktale	850
Rozwiązywanie problemów	858
Całkowicie pusta strona	858
Błąd „Headers already sent”	858
Zepsuty obrazek	859
Podsumowanie	860
Część V Studium przypadków	861
Rozdział 43. Dziennik sieciowy	863
Dlaczego dziennik?	863
Najprostszy dziennik	864
Dodanie HTML-owego narzędzia do edycji	869
Dołączenie bazy danych	871
Możliwe rozszerzenia	878
Podsumowanie	878

Rozdział 44. Uwierzytelnianie użytkowników	879
Projektowanie systemu uwierzytelniania użytkowników	879
Unikanie powszechnych problemów związanych z bezpieczeństwem	881
Wyłączenie register_globals	881
Kontrola długości napisów a bezpieczeństwo	882
Jednokierunkowe szyfrowanie haseł	883
Rejestracja	884
Zalogowanie i wylogowanie	890
Narzędzia użytkowników	894
Zapomniane hasła	895
Zmiana poufnych danych użytkowników	897
Edycja niechronionych danych użytkowników	902
Narzędzia administracyjne	905
Autoryzacja: basic auth, cookie, baza danych i IP	906
Logowanie jako użytkownik	907
Podsumowanie	909
Rozdział 45. System ocen użytkowników	911
Projekt początkowy	912
Dziedzina: witryna z cytatami	912
Możliwe oceny	912
Połączenie ocen z treścią	913
Zbieranie głosów	914
Agregowanie wyników	920
Rozszerzenia i możliwości	921
Podsumowanie	922
Rozdział 46. Quiz	923
Pojęcia wykorzystywane w tym rozdziale	923
Gra	924
Nasza wersja gry	924
Przykładowe ekrany	924
Zasady	926
Zagraj w to sam	927
Kod	927
Pliki źródłowe	927
Utworzenie bazy danych	952
Rozważania projektowe	956
Rozdzielenie kodu i wyświetlania	956
Trwałość danych	956
Obsługa wyjątków	957
Podsumowanie	957
Rozdział 47. Konwersja statycznych witryn HTML	959
Planowanie wielkiej modernizacji	959
Aby nie wyłączyć dziecka z kąpielą	960
Ocena techniczna	961
Przeprojektowanie interfejsu użytkownika	962
Planowanie nowego schematu bazy danych	965

Przenoszenie danych do bazy danych	969
Poprawianie danych	969
Wprowadzanie danych	970
Zbieranie danych	974
Szablony	977
Wydajność i buforowanie	984
Buforowanie	985
Podsumowanie	986
Rozdział 48. Wizualizacja danych przy użyciu diagramów Venna	987
Skalowane diagramy Venna	987
Zadanie	988
Struktura kodu	988
Wymagana trygonometria	989
Planowanie wyświetlania	992
Uproszczenie założeń	992
Określenie rozmiaru i skali	993
Wyświetlenie wyniku	998
Wizualizacja bazy danych	999
Testowanie	1003
Rozszerzenia	1004
Podsumowanie	1005
Dodatki	1007
Dodatek A PHP dla programistów C	1009
Dodatek B PHP dla hakerów Perl	1015
Dodatek C PHP dla programistów HTML	1021
Dodatek D Zasoby Sieci na temat PHP	1029
Skorowidz	1041

Rozdział 8.

Ciągi

W tym rozdziale:

- ♦ Ciągi w PHP
- ♦ Funkcje obsługi ciągów
- ♦ Rozszerzony przykład: kalkulator do ćwiczeń

Mimo że rysunki, dźwięki, animacje i aplety stają się coraz ważniejszą częścią sieci WWW, nadal bazuje ona na tekście. Podstawowym typem PHP przechowującym tekst jest typ `string`.

W niniejszym rozdziale opisane zostaną niemal wszystkie dostępne w PHP narzędzia do manipulowania ciągami (jedynie bardziej zaawansowane funkcje operujące na ciągach oraz dopasowywanie ciągów za pomocą wyrażeń regularnych zostanie opisane później, w rozdziale 22.). Na początku przedstawimy podstawowe informacje o ciągach, a następnie zaprezentujemy ich działanie w praktyce, kontynuując rozbudowę kalkulatora do ćwiczeń z rozdziału 7.

Ciągi w PHP

Ciągi to sekwencje znaków traktowane jako odrębna jednostka. Mogą być przypisywane do zmiennych, używane jako parametry funkcji, zwracane z funkcji lub wysyłane na wyjście i oglądane w przeglądarce klienta. Najprostszą metodą stworzenia ciągu w PHP jest otoczenie znaków cudzysłowami (`"`) bądź apostrofami (`'`):

```
$ciag = 'Ciąg';  
$inny_ciag = "Inny ciąg";
```

Różnica pomiędzy ciągami w cudzysłowach i apostrofach leży w interpretacji ich przez PHP. Jeżeli otoczysz ciąg apostrofami, prawie żadna interpretacja nie zostanie zastosowana, jeżeli zaś otoczysz cudzysłowami, PHP wklei wartości zmiennych w miejsce ich nazw i zamieni sekwencje sterujące na odpowiadające im znaki. Jeżeli np. wstawisz następujący kod do strony:

```
$wyrazenie = 'wszystko, co powiem';
$pytanie_1 =
    "Czy musisz brać $wyrazenie dosłownie?\n<BR>";
$pytanie_2 =
    'Czy musisz brać $wyrazenie dosłownie?\n<BR>';
echo $pytanie_1;
echo $pytanie_2;
```

powinieneś spodziewać się następującego wyniku:

```
Czy musisz brać wszystko, co powiem, dosłownie?
Czy musisz brać $wyrazenie dosłownie?\n
```



Szczegółowy opis traktowania przez PHP ciągów z apostrofami i cudzysłowami znajduje się w części „String” w rozdziale 5.

Dołączanie ciągów przy użyciu nawiasów klamrowych

W większości przypadków wystarczy umieścić zmienną w ciągu otoczonym cudzysłowami, aby wartość tej zmiennej została umieszczona jako część ciągu znaków w trakcie interpretowania kodu. Istnieją jednak dwie sytuacje, w których interpreter może wymagać bardziej szczegółowych wskazówek od programisty. Pierwsza z nich zachodzi wówczas, gdy zamierzenie programisty co do umiejscowienia zmiennej kłóci się z zasadami stosowanymi przez interpreter, druga natomiast ma miejsce wtedy, gdy dołączane wyrażenie nie jest zwykłą zmienną. W obydwu przypadkach wszelkie wątpliwości można rozwiązać umieszczając zmienną, która ma zostać dołączona, w nawiasach klamrowych {}.

PHP nie będzie na przykład miał trudności z interpretacją następującego kodu:

```
$sport = 'volleyball';
$plan = "W lecie będę grał w $sport";
```

W tej sytuacji analizator napotka symbol \$, po czym rozpocznie odczytywanie znaków składających się na nazwę zmiennej do momentu, gdy dotrze do znaku spacji występującego po słowie \$sport. Znaki spacji nie mogą znajdować się w nazwie zmiennej, zatem oczywiste jest, że zmienna nosi nazwę \$sport. PHP bez trudu ustali wartość tak zidentyfikowanej zmiennej ('volleyball') i umieści ją w ciągu znaków.

Czasami jednak umieszczanie znaku spacji zaraz za nazwą zmiennej nie leży w interesie programisty. Spójrzmy na poniższy przykład:

```
$sport1 = 'volley';
$sport2 = 'foot';
$sport3 = 'basket';
$plan1 = "W lecie będę grał w $sport1ball";      // Źłe
$plan2 = "Na jesieni będę grał w $sport2ball";   // Źłe
$plan3 = "W zimie będę grał w $sport3ball";     // Źłe
```

W tym przypadku programista nie osiągnie pożądanego efektu, ponieważ PHP zinterpretuje słowo \$sport1 jako część nazwy zmiennej \$sport1ball, która najprawdopodobniej będzie nieokreślona. Zamiast takiego zapisu należałoby zatem napisać:

```
$plan1 = "W lecie będę grał w {$sport1}ball";    // Dobrze
```

Dzięki takiemu zapisowi PHP najpierw obliczy wartość zmiennej znajdującej się w nawiasach klamrowych, a dopiero potem dołączy ją do ciągu.

Z analogicznych powodów PHP nie radzi sobie z dołączaniem wyrażeń zawierających zmienne złożone, takie jak tablice wielowymiarowe czy obiekty. Również wówczas konieczne jest użycie nawiasów klamrowych. Ogólna zasada stanowi, że jeżeli zaraz za znakiem { występuje znak \$, PHP najpierw obliczy wartość wyrażenia ciągnącego się aż do znaku }, po czym obliczony wynik dołączy do ciągu. (Jeżeli chcesz, by w ciągu pojawił się sam symbol {\$, musisz w tym celu obydwie znaki poprzedzić znakiem lewego ukośnika).



Inne sposoby radzenia sobie z takimi sytuacjami zostaną opisane w punkcie „Złączenie i przypisanie” w dalszej części niniejszego rozdziału.

Znaki i indeksy ciągu

W przeciwieństwie do większości języków programowania, PHP nie posiada osobnego typu znakowego. Zwykle funkcje, które w innych językach pobierają znak, w PHP spodziewają się ciągu o długości 1.

Można uzyskać znak z ciągu, traktując ciąg jak tablicę z pierwszym indeksem równym 0, wpisanym w nawiasach klamrowych zaraz po zmiennej ciągu. Uzyskany w ten sposób znak jest jednoliterowym ciągiem. Na przykład:

```
$ciag = "Podwojony";
for ($index = 0 ; $index < 9; $index++ ) {
    $ciag_do_wyswietlenia = $ciag{$index};
    print("$ciag_do_wyswietlenia$ciag_do_wyswietlenia");
}
```

daje w efekcie:

```
PPooddwwoojjoonnyy
```

Każda litera została wypisana dwa razy (liczba 9 wpisana jest w tym przykładzie dlatego, że nie wiemy jeszcze, jak określić długość ciągu — popatrz na opis funkcji `strlen()` w części „Sprawdzanie ciągów”).



We wcześniejszych wersjach PHP można było, zależnie od własnego uznania, umieszczać odpowiedni indeks w nawiasach kwadratowych i w taki sposób odczytywać pojedyncze znaki ciągu (np. `$ciag[3]` zamiast `$ciag{3}`). Wprawdzie cały czas można używać zapisu z nawiasami kwadratowymi (takiego samego jak w przypadku tablic), lecz mechanizm ten został już unieważniony i rekomendowane jest używanie do tego celu nawiasów klamrowych.

Operatory dla ciągów

PHP posiada tylko jeden prawdziwy operator dla ciągów: kropkę (.) — operator złączenia. Operator ten umieszczony pomiędzy ciągami tworzy nowy ciąg, będący ich sklejeniem. Na przykład:

```
$zdanie_1 = "Chciałbym podzielić się z wami ";  
$zdanie_2 = "moimi uwagami";  
print ($zdanie_1. $zdanie_2. "...");
```

da w efekcie:

Chciałbym podzielić się z wami moimi uwagami...

Zauważmy, że nie przekazujemy do instrukcji `print` wielu argumentów — przekazujemy jeden argument stworzony przez połączenie trzech ciągów. Pierwszy i drugi ciąg to zmienne, trzeci jest literałem.



Operator złączenia nie jest tym samym co operator `+` w języku Java, nie przeciąża innego operatora. Jeżeli pomylisz się i dodasz dwa ciągi za pomocą `+`, zostaną zinterpretowane jako liczby. Na przykład `jeden + dwa` będzie równe `0` (ponieważ nie udało się przeprowadzić prawidłowej konwersji ciągów).

Złączenie i przypisanie

Podobnie jak w przypadku operatorów arytmetycznych, PHP posiada operator skrócony (`.`), który jest złączeniem z przypisaniem. Wyrażenie:

```
$ciag .= $dodatek;
```

jest równoznaczne z:

```
$ciag = $ciag . $dodatek;
```

Zwróć uwagę, że w odróżnieniu od naprzemiennego w swej naturze dodawania i mnożenia, przy użyciu tego operatora nowy ciąg jest dodawany po prawej stronie starego. Jeżeli chcesz zmienić kolejność, musisz użyć dłuższej formy:

```
$ciag = $dodatek . $ciag;
```

Trzeba również pamiętać, że w trakcie złączania zmienne bez przypisanej wartości są traktowane jak ciągi puste. Zmienna `$ciag` pozostanie zatem niezmienniona, jeżeli zmiennej `$dodatek` nie zostanie przypisana żadna zmienna.

Składnia heredoc

Oprócz składni z apostrofami i cudzysłowami PHP udostępnia również dodatkową metodę definiowania wartości ciągów — jest to tak zwana *składnia heredoc*. Składnia ta okazuje się szczególnie przydatna w sytuacji, gdy trzeba zdefiniować obszerne fragmenty tekstu zawierające w sobie wartości zmiennych, ponieważ programista nie musi w jej przypadku poprzedzać apostrofów ani cudzysłowów znakiem ukośnika. Składnia ta przydaje się przede wszystkim do tworzenia stron zawierających formularze HTML.

W składni heredoc operatorem jest `<<<`. Zaraz po nim powinna nastąpić etykieta (bez cudzysłowów ani apostrofów) wskazująca rozpoczęcie ciągu złożonego z wielu wierszy. PHP będzie dołączać do ciągu kolejne wiersze aż do czasu, gdy w nowym wierszu natknie się na tę samą etykietę. Opcjonalnie etykieta zamykająca może posiadać na końcu znak średnika.

Spójrzmy na poniższy przykład:

```
$my_string_var = <<<EOT
Wszystko, co znajduje się w tym mało przydatnym
fragmencie tekstu, zostanie dołączone
do zmiennej mającej postać ciągu znaków,
tworzonego znak po znaku, wiersz po wierszu, aż
do osiągnięcia wiersza ostatniego, czyli właśnie tego.
EOT;
```

Zwróć uwagę, że zamykająca etykieta EOT nie może być w jakikolwiek sposób wcięta, ponieważ zostanie wówczas potraktowana jak jeszcze jeden fragment, który powinien zostać dołączony do ciągu. Etykieta nie musi mieć postaci EOT — może to być dowolna nazwa zgodna z regułami nazewnictwa zmiennych w PHP.

Dołączanie zmiennych realizowane jest w sposób identyczny, jak w ciągach otoczonych cudzysłowami. Zaletą składni heredoc jest natomiast fakt, że znaki apostrofów i cudzysłówów mogą być dołączane bez poprzedzania ich znakiem ukośnika i bez konieczności wcześniejszego zakańczania ciągu. Spójrzmy na kolejny przykład:

```
echo <<<ENDOFFORM
<FORM METHOD=POST ACTION="{$_ENV['PHP_SELF']}">
<INPUT TYPE=TEXT NAME=FIRSTNAME VALUE=$firstname>
<INPUT TYPE=SUBMIT NAME=SUBMIT VALUE=SUBMIT>
</FORM>
ENDOFFORM
```

Powyższy kod w bardzo prosty sposób zwróci do przeglądarki kod definiujący najprostsz formularz.

Funkcje operujące na ciągach

PHP zawiera wiele funkcji działających na ciągach. Jeżeli zamierzasz napisać własną funkcję, która analizuje ciąg znak po znaku, aby stworzyć nowy ciąg, pomyśl, które z tych zadań będą często wykonywane. Jeżeli takie wystąpią, prawdopodobnie istnieje wbudowana funkcja realizująca tę czynność.

W tej części przedstawimy podstawowe funkcje sprawdzające, porównujące, modyfikujące i drukujące ciągi. Jeżeli chcesz naprawdę dobrze poznać operacje na ciągach wykonywane w PHP, powinieneś przyswoić sobie co najmniej treści przedstawione w tym punkcie. Funkcje wyrażen regularnych oraz bardziej skomplikowane funkcje obsługi ciągów zostaną opisane w rozdziale 22.



Dla programistów C: powinniście znać wiele funkcji operujących na ciągach. Należy jedynie zapamiętać, że PHP zajmuje się przydziałem pamięci. Funkcje zwracające ciąg rezerwują pamięć na wynikowy ciąg i nie ma potrzeby wcześniej rezerwować pamięci.

sprawdzanie ciągów

Jakie pytania można zadać na temat ciągu? Na początek sprawdźmy przy użyciu funkcji `strlen()`, ile znaków zawiera.

```
$ciag = "Ten ciąg zawiera 26 znaków";
print("To ma". strlen($ciag)."znaków");
```

Uruchomienie tego kodu daje następujący wynik:

```
To ma 26 znaków
```

Odczytanie długości ciągu jest użyteczne w sytuacjach, gdy chcemy za pomocą pętli dostać się do wszystkich znaków ciągu. Bezużytecznym, ale pouczającym przykładem może być (używamy ciągu z poprzedniego przykładu):

```
for ($index=0; $index <strlen ($ciag); $index++)
    print("$ciag[$index]");
```

W przeglądarce pojawi się ciąg taki sam, jak na początku:

```
Ten ciąg zawiera 26 znaków
```

szukanie znaków i podciągów

Następną kategorią pytań na temat ciągów może być pytanie o ich zawartość. Funkcja `strpos()` szuka pozycji określonego znaku w ciągu, o ile w nim występuje.

```
$twister = "Król Karol kupił królowej Karolinie";
print("'k' występuje na pozycji ". strpos($twister, 'k'). "<BR>");
print("'q' występuje na pozycji". strpos($twister, 'q'). "<BR>");
```

Wykonanie tego przykładu daje w efekcie:

```
'k' występuje na pozycji 11
'q' występuje na pozycji
```

Pozycja litery 'q' pozostała niewypełniona, ponieważ funkcja `strpos()` zwraca `FALSE` w przypadku nieznaalezienia znaku, a `FALSE` zostało skonwertowane do pustego ciągu. Należy zwrócić uwagę, że funkcja `strpos()` odróżnia duże i małe litery.



Funkcja `strpos()` jest jednym z przykładów, kiedy brak ścisłego określenia typów może stanowić problem. Jeżeli nie zostanie odnaleziona wartość, funkcja zwraca wartość `FALSE`. Jeżeli odnaleziony zostanie pierwszy znak ciągu, funkcja zwróci zero (ponieważ indeksowanie rozpoczyna się od zera). Jeżeli użyjemy tych wartości w wyrażeniu logicznym, obie zostaną zinterpretowane jako `FALSE`. Jedynym sposobem rozróżnienia tych wartości jest użycie operatora sprawdzającego identyczność obiektów (`===`, wprowadzony w PHP4), który jest prawdziwy, jeżeli jego argumenty są identyczne i mają te same typy. Możesz go bezpiecznie użyć do porównania 0 z `FALSE`. Jeżeli używasz PHP3, musisz sprawdzić typ zwracanej wartości, używając funkcji np. `is_ integer()`.

Za pomocą funkcji `strpos()` można również szukać ciągów, nie tylko pojedynczych znaków. W tym celu wystarczy jedynie przekazać do niej ciąg wieloznakowy. Możesz także podać dodatkowy parametr numeryczny określający pozycję, od której funkcja ma rozpocząć szukanie.

Czy ciągi są niezienne?

W niektórych językach programowania (np. C) normalne jest manipulowanie ciągami przez ich bezpośrednią modyfikację, na przykład zapisywanie nowych znaków w środku ciągu. Inne języki (np. Java) trzymają programistów z daleka od takich operacji, tworząc klasę reprezentującą ciągi, które są *niezienne*. Można utworzyć nowy ciąg tylko poprzez stworzenie zmodyfikowanej kopii ciągu. Po jego utworzeniu nie można bezpośrednio zmieniać jego znaków.

W PHP ciągi mogą być zmieniane, ale w powszechnej praktyce jest traktowanie ciągów, jak gdyby były niezienne.

Można zmieniać ciąg, traktując go jako tablicę znaków i przypisywać bezpośrednio do tej tablicy:

```
$ciag = "abcdefg";
$ciag[5] = "X";
print ($ciag . "<BR>");
```

Daje to następujący wynik:

```
abcdeXg
```

Taka operacja jest jednak nieudokumentowana i nie występuje nigdzie w podręczniku, chociaż analogiczne odczytywanie znaków jest zamieszczone (i uaktualnione o mechanizm nawiasów klamrowych). Prawie wszystkie funkcje PHP operujące na ciągach zwracają zmienioną kopię ciągu, a nie przeprowadzają zmian na oryginale. Projektanci PHP preferują ten styl programowania. Radzimy nie modyfikować bezpośrednio ciągów, chyba że dzięki temu oszczędza się pamięć.

Możliwe jest też szukanie wstecz przy użyciu funkcji `strrpos()` (dodatkowe `r` pochodzi od słowa „reverse”, czyli w tył). Funkcja ta pobiera ciąg do szukania oraz pojedynczy znak, którego szukamy, i zwraca ostatnią pozycję, na której w pierwszym argumencie występuje argument drugi. W przeciwieństwie do `strpos()`, nie można podać podciagu do odszukania. Jeżeli użyjemy tej funkcji w naszym przykładowym zdaniu, odszukamy inne wystąpienie litery `k`:

```
$twister = "Król Karol kupił królowej Karolinie";
print("'k' występuje na pozycji ". strrpos($twister, 'k'). "<BR>");
```

W tym przypadku odnajdziemy pierwsze „k” w słowie „królowej”:

```
'k' występuje na pozycji 17
```

Porównywanie i przeszukiwanie

Często trzeba sprawdzać, czy dwa ciągi są identyczne. Szczególnie często w programach pracujących na danych wprowadzonych przez użytkownika.



Ciągi są takie same dla operatora `'=='`, jeżeli zawierają dokładnie taką samą sekwencję znaków. Operator ten nie wykonuje żadnych dokładniejszych porównań (np. sprawdzenia obszarów pamięci), zwraca natomiast baczność uwagę na wielkość liter.

Najprostszą metodą porównania ciągów jest zastosowanie operatora równości (`==`), który porównuje zarówno liczby, jak i ciągi.



Porównanie ciągów za pomocą operatora '==' (a także < i >) jest prawidłowe tylko w przypadku, gdy oba argumenty są takimi samymi ciągami i nie wykonano żadnej konwersji typów (więcej na ten temat w rozdziale 5.). Jedynie używając funkcji strcmp() (opisanej poniżej), otrzymujemy zawsze prawidłowe wyniki.

Podstawową funkcją do porównywania ciągów jest strcmp(). Posiada dwa ciągi jako argumenty, porównuje je znak po znaku aż do znalezienia różnicy. Funkcja zwraca wartość ujemną, jeżeli pierwszy ciąg jest „mniejszy” od drugiego, wartość dodatnią, gdy drugi ciąg jest mniejszy, oraz 0, jeżeli ciągi są identyczne.

Funkcja strcasecmp() działa w taki sam sposób, z tą różnicą, że nie bierze pod uwagę wielkości liter podczas porównywania ciągów. Wywołanie funkcji: strcmp("hej!", "HEJ!") powinno zwrócić 0.

Przeszukiwanie

Funkcje porównujące ciągi sprawdzają, czy ciągi są identyczne. Aby sprawdzić, czy jeden ciąg zawiera się w drugim, używamy funkcji strpos() (opiszemy ją później) lub strstr() (ewentualnie jednej z jej odmian).

Funkcja strstr() oczekuje w pierwszym parametrze ciągu do przeszukania, a w drugim ciągu do odszukania. Jeżeli operacja się powiedzie, funkcja zwróci fragment przeszukiwanego ciągu rozpoczynający się od pierwszego miejsca wystąpienia poszukiwanego fragmentu. Jeżeli nic nie zostanie odnalezione, zwracana jest wartość FALSE. W poniższym przykładzie zamieściliśmy jedno udane i jedno nieudane wywołanie funkcji.

```
$ciag_do_przeszukania = "pierwszaoladruqaola";
$ciag_do_odszukania = "ola";
print("Wynik szukania ciągu $ciag_do_odszukania: " .
      strstr($ciag_do_przeszukania, $ciag_do_odszukania) . "<br>");
$ciag_do_odszukania = "ula";
print("Wynik szukania ciągu $ciag_do_odszukania: " .
      strstr($ciag_do_przeszukania, $ciag_do_odszukania));
```

Wykonanie tego fragmentu da następujący wynik:

```
Wynik szukania ciągu ola: oladruqaola
Wynik szukania ciągu ula:
```

Puste miejsce za dwukropkiem w drugim wierszu wyniku jest wynikiem próby wydrukowania wartości FALSE, która została skonwertowana na pusty ciąg. Funkcja strstr() posiada alternatywną nazwę strchr(). Można używać dowolnej z nazw tej funkcji, wynik działania będzie identyczny. Podobnie jak strcmp(), strstr() posiada odmianę, która przy szukaniu ciągów identycznie traktuje małe i wielkie litery — stristr(). Funkcja ta działa identycznie jak strstr(), nie rozróżnia jednak wielkości liter. Dotychczas opisane funkcje operujące na ciągach zebraliśmy w tabeli 8.1.

Tabela 8.1. Funkcje porównujące, przeszukujące i badające ciągi

Funkcja	Opis
<code>strlen()</code>	Zwraca długość ciągu podanego jako argument wywołania i zwraca ją w postaci liczby całkowitej.
<code>strpos()</code>	Posiada dwa argumenty: ciąg do przeszukania i ciąg do znalezienia. Zwraca pozycję (licząc od zera) początku pierwszego miejsca wystąpienia ciągu lub wartość <code>FALSE</code> , jeżeli ciąg nie został odszukany. Można podać trzeci argument określający pozycję, od której ma rozpocząć szukanie.
<code>strrpos()</code>	Jak <code>strpos()</code> , ale przeszukuje ciąg od tyłu. Poszukiwany ciąg może składać się tylko z jednej litery. Nie ma opcjonalnych argumentów.
<code>strcmp()</code>	Spodziewa się dwóch ciągów jako argumentów. Zwraca 0, jeżeli ciągi te są identyczne. Jeżeli funkcja znajdzie różnicę, zwraca wartość ujemną, jeżeli kod ASCII pierwszego różniącego się znaku jest mniejszy w pierwszym ciągu, lub wartość dodatnią, jeżeli mniejszy kod ASCII znajduje się w drugim ciągu.
<code>strcasecmp()</code>	Działa tak jak <code>strcmp()</code> , traktując identycznie małe i wielkie litery.
<code>strstr()</code>	Przeszukuje pierwszy ciąg, szukając w nim drugiego. Zwraca fragment pierwszego ciągu rozpoczynający się od poszukiwanego fragmentu. Jeżeli nie ma drugiego ciągu w pierwszym, zwraca <code>FALSE</code> .
<code>strchr()</code>	Działa tak samo jak <code>strstr()</code> .
<code>stristr()</code>	Działa tak samo jak <code>strstr()</code> , nie rozróżniając wielkości liter.

Wycinanie podciągu

Wiele z funkcji operujących na ciągach ma na celu wybranie określonego podciągu lub modyfikację go w ciągu oryginalnym. Warto wiedzieć, że większość funkcji modyfikujących ciąg nie zmienia oryginalnego ciągu, ale zwraca zmienioną kopię, pozostawiając oryginał nietknięty.

Podstawową metodą wycięcia fragmentu ciągu jest użycie funkcji `substr()`, która zwraca ciąg będący określonym fragmentem ciągu przekazanego jako argument. Oprócz ciągu, na którym operuje, funkcja wymaga podania liczby określającej początek wycinanego ciągu. Trzeci argument jest opcjonalny i określa długość wynikowego podciągu. Jeżeli nie zostanie on podany, funkcja zwraca fragment od podanej pozycji (za pomocą drugiego argumentu) do końca ciągu. Należy pamiętać, że pierwszy znak ciągu znajduje się na pozycji 0.

Spójrzmy na następujący przykład:

```
echo (substr("Wycinanie ciągów w PHP jest proste", 23));
```

Zwróci on w wyniku „jest proste”; wyrażenie:

```
echo (substr("Wycinanie ciągów w PHP jest proste", 10, 6));
```

wypisze słowo „ciągów” — sześciznakowy ciąg rozpoczynający się od pozycji 10.

Oba argumenty numeryczne, pozycja początkowa i długość mogą być ujemne, lecz w obydwu przypadkach znaczenie ujemnego argumentu jest inne. Jeżeli pozycja początkowa jest ujemna, oznacza to, że początek podciągu jest określany względem końca ciągu. Pozycja -1 oznacza początek ciągu na ostatnim znaku, -2 na przedostatnim itd.

Można oczekiwać, że podanie ujemnej długości zinterpretowane będzie analogicznie do pozycji początkowej i podciąg będzie wyznaczany, odliczając wstecz od początkowego znaku ciągu. Nie jest to do końca prawdziwe. Znak określony przez indeks początkowy będzie pierwszym znakiem podciągu (a nie ostatnim), a jego długość określona będzie odliczeniem podanej liczby znaków od końca ciągu.

Spójrz na kilka przykładów, w których użyte zostały dodatnie i ujemne wartości:

```
$alfabet = "abcdefghijklmnop";
print ("3: ". substr( $alfabet, 3). "<BR>");
print ("-3: ". substr( $alfabet, -3). "<BR>");
print ("3,5: ". substr( $alfabet, 3, 5). "<BR>");
print ("3,-5: ". substr( $alfabet, 3, -5). "<BR>");
print ("-3,-5: ". substr( $alfabet, -3, -5). "<BR>");
print ("-3, 5: ". substr( $alfabet, -3, 5). "<BR>");
```

W wyniku otrzymujemy:

```
3: defghijklmnop
-3: nop
3,5: defgh
3,-5: defghijk
-3,-5:
-3, 5: nop
```



W przykładzie z pozycją początkową -3 i długością -5 pozycja końcowa znajduje się przed pozycją początkową, co jest sytuacją określaną jako „ciąg o ujemnej długości”. W podręczniku, dostępnym pod adresem <http://www.php.net/manual>, jest napisane, że w takiej sytuacji funkcja `substr()` zwróci ciąg zawierający jeden znak znajdujący się na pozycji początkowej. Zamiast tego widzimy, że PHP5 zwraca pusty ciąg. Należy szczególnie zwrócić uwagę na takie sytuacje.

Zauważmy, że pomiędzy funkcjami `substr()`, `strstr()` i `strpos()` występuje bliski związek. Funkcja `substr()` wycina podciąg, bazując na pozycji, `strstr()` wycina podciąg na podstawie zawartości, a `strpos()` znajduje położenie podciągu. Jeżeli jesteśmy pewni, że `$ciag` zawiera `$podciag`, wtedy wyrażenie:

```
strstr($ciag, $podciag)
```

jest równoważne wyrażeniu:

```
substr($ciag, strpos($ciag, $podciag))
```

Funkcje porządkujące

Mimo że funkcje `chop()`, `ltrim()` i `trim()` są zwykłymi funkcjami wycinającymi podciąg, używa się ich do porządkowania ciągów. Funkcje te wycinają znaki odstępów z początku, końca lub początku i końca ciągu. A oto przykład:

```

$oryginal = " To przechodzi ludzkie pojęcie ";
$f_chop = chop($oryginal);
$f_ltrim = ltrim($oryginal);
$f_trim = trim($oryginal);
print("Ciąg oryginalny: '$oryginal'<BR>");
print("Długość ciągu: ". strlen($oryginal). "<BR>");
print("Po funkcji chop: '$f_chop' <BR>");
print("Długość ciągu: ". strlen($f_chop). "<BR>");
print("Po funkcji ltrim: '$f_ltrim' <BR>");
print("Długość ciągu: ". strlen($f_ltrim). "<BR>");
print("Po funkcji trim: '$f_trim' <BR>");
print("Długość ciągu: ". strlen($f_trim). "<BR>");

```

W przeglądarce dostajemy:

```

Ciąg oryginalny: ' To przechodzi ludzkie pojęcie '
Długość ciągu: 33
Po funkcji chop: ' To przechodzi ludzkie pojęcie'
Długość ciągu: 30
Po funkcji ltrim: 'To przechodzi ludzkie pojęcie '
Długość ciągu: 32
Po funkcji trim: 'To przechodzi ludzkie pojęcie'
Długość ciągu: 29

```

Ciąg oryginalny ma trzy spacje na końcu (usuwane przez `chop()` i `trim()`) oraz jedną na początku (usuwana przez `ltrim()` i `trim()`). Dokładniej opiszemy zawartość okna przeglądarki po wykonaniu tego fragmentu. Powtarzające się odstępy zostały przez przeglądarkę połączone w jeden. Jeżeli jednak zajrzysz do źródła strony, to na końcu ciągu nadal znajdują się trzy odstępy.

Oprócz spacji funkcje te usuwają znaki zapisane jako sekwencje sterujące `\n`, `\r`, `\t` oraz `\0` (znaki końca wiersza, tabulatory oraz znak końca ciągu używany w programach napisanych w C).

Gdyby nazewnictwo funkcji było spójne, funkcja `chop()` powinna nazywać się `rtrim()`. Mimo że nazwa funkcji `chop()` (ang. *rąbać*) sugeruje destrukcyjne działanie, nie wpływa ona na ciąg przekazany jako argument. Po wykonaniu funkcji ciąg nie zmienia się.

Zastępowanie ciągów

Funkcje operujące na ciągach, które omówiliśmy do tej pory, zwracały fragment ciągu przekazanego jako argument, zamiast tworzyć całkowicie nowy ciąg. Teraz zajmiemy się funkcjami `str_replace()` i `substr_replace()`.

Funkcja `str_replace()` pozwala zamienić wszystkie miejsca wystąpienia podanego fragmentu ciągu na inny. Funkcja oczekuje trzech argumentów: ciągu do odszukania, ciągu, który zamieni szukany fragment, oraz ciągu źródłowego. Na przykład:

```

$pierwszy = "Birma jest nieco podobna do Rodezji.";
$drugi = str_replace("Rodezji", "Zimbabwe", $pierwszy);
$trzeci = str_replace("Birma", "Panama", $drugi);
print($trzeci);

```

Wynikiem tych operacji jest zdanie:

Panama jest nieco podobna do Zimbabwe.

Zastępowane są wszystkie miejsca wystąpienia podanego fragmentu. Jeżeli do ciągu PHP uda się wtłoczyć całą encyklopedię, można ją będzie zaktualizować jednym wywołaniem.

Musisz zwrócić uwagę na sytuację, gdy wystąpienia poszukiwanego ciągu zachodzą na siebie. Fragment programu:

```
$ciag = "ABA jest częścią ABABA";
$wynik = str_replace("ABA", "DEF", $ciag);
print("Wynik zamiany to '$wynik'<BR>");
```

wyświetli ciąg:

Wynik zamiany to 'DEF jest częścią DEFBA'

Jest to chyba najrozsądniejsze działanie.

Funkcja `str_replace()` odszukuje fragmenty ciągu do wymiany, porównując ciąg źródłowy z fragmentem podanym jako parametr; funkcja `substr_replace()` zamienia fragment ciągu znajdujący się na określonej pozycji. Można ją wywołać maksymalnie z czterema argumentami: ciągiem, na którym wykonuje się operacja, ciągiem, który jest wstawiany, początkową pozycją oraz parametrem opcjonalnym — długością fragmentu do wymiany. Na przykład:

```
print(substr_replace("ABCDEFGH", "-", 2, 3));
```

daje w wyniku:

AB-FG

Fragment CDE został zamieniony na pojedynczy znak minus. Możemy więc zastępować fragmenty ciągu ciągami o innej długości. Jeżeli nie zostanie podana długość fragmentu, wymieniony zostanie fragment ciągu od pozycji startowej do końca ciągu.

Funkcja `substr_replace()` pozwala również na używanie ujemnych wartości argumentów numerycznych. Są one traktowane identycznie jak ujemne wartości argumentów w funkcji `substr()`; opisaliśmy je w części „Wycinanie podciagu”. Należy pamiętać, że funkcje `str_replace()` i `substr_replace()` nie zmieniają w żaden sposób oryginalnych ciągów.

Mamy również kilka rzadziej używanych funkcji, które tworzą nowy ciąg na podstawie podanego jako parametr. Funkcja `strrev()` zwraca ciąg podany jako argument, ale pisany wspak. Funkcja `str_repeat()` tworzy ciąg zawierający ciąg źródłowy powtórzony daną liczbę razy. Przykładowo:

```
print(str_repeat("witaj ", 3));
```

daje w wyniku:

witaj witaj witaj

Funkcje przeszukujące i zamieniające ciąg zebrane zostały w tabeli 8.2.

Tabela 8.2. Funkcje przeszukujące i zamieniające ciąg

Funkcja	Działanie
<code>substr()</code>	Zwraca fragment ciągu opisany przez drugi argument określający pozycję początkową i opcjonalny trzeci argument określający długość. Fragment rozpoczyna się na pozycji startowej i ma długość podaną w trzecim argumentcie, a jeżeli nie został podany trzeci argument, obejmuje ciąg do końca. Ujemna wartość pozycji startowej oznacza, że jest określana od końca ciągu, ujemny parametr określający długość powoduje, że koniec podciągu jest wyznaczany przez podaną liczbę znaków od końca ciągu.
<code>chop()</code> lub <code>rtrim()</code>	Zwraca ciąg podany jako argument bez końcowych znaków odstępu. Znakami odstępu są znaki " ", <code>\n</code> , <code>\r</code> , <code>\t</code> i <code>\0</code> .
<code>ltrim()</code>	Zwraca ciąg podany jako argument bez początkowych znaków odstępu.
<code>trim()</code>	Zwraca ciąg podany jako argument bez początkowych i końcowych znaków odstępu.
<code>str_replace()</code>	Zamienia podany fragment ciągu na inny. Funkcja ma trzy argumenty: ciąg do odszukania, ciąg, na który jest on zamieniany, oraz ciąg bazowy. Zwraca kopię z zamienionymi na drugi argument wszystkimi miejscami wystąpienia pierwszego argumentu.
<code>substr_replace()</code>	Wstawia fragment podany jako argument na pozycję określoną przez parametry numeryczne. Funkcja posiada cztery argumenty: ciąg bazowy, fragment wstawiany, pozycję początkową i liczbę znaków do wymiany. Zwraca kopię ciągu podanego jako pierwszy argument, z ciągiem do zamiany wstawionym na określonej pozycji. Jeżeli opuszczony zostanie czwarty argument, koniec ciągu zostanie zamieniony na fragment przekazany w argumentcie. Ujemne wartości parametrów traktowane są identycznie jak w funkcji <code>substr()</code> .

Funkcje zmiany wielkości liter

Funkcje poniższe zmieniają wielkość liter z małych na duże, i odwrotnie. Pierwsze dwie operują na całym ciągu, kolejne — tylko na pierwszych literach.

strtolower()

Funkcja `strtolower()` zwraca ciąg ze wszystkimi małymi literami. Nie ma znaczenia, czy na początku cały ciąg był zapisany dużymi literami, czy duże i małe litery były wymieszane.

```
<?php
$oryginalny = "On NIE wie, że KRZYCZY";
$male = strtolower($oryginalny);
echo $male;
?>
```

Ten fragment programu zwróci ciąg: "on nie wie, że krzyczy".



Jeśli już wcześniej zmagaleś się z formularzem, który wymagał zaimplementowania rozbudowanych mechanizmów weryfikacji poprawności danych, być może zauważyłeś, że funkcja `strtolower()` okazuje się niezwykle użyteczna dla osób trwających w przekonaniu, iż ich adresy pocztowe zawierają wielkie litery. Równie użyteczne są funkcje opisywane dalej w tym punkcie.

strtoupper()

Funkcja `strtoupper()` zwraca ciąg z wszystkimi dużymi literami. Nie ma znaczenia, czy na początku cały ciąg był zapisany małymi literami, czy duże i małe litery były wymieszane.

```
<?php
$oryginalny = "napisz to wyraźnie";
echo("<B>".strtoupper($oryginalny)."</B>");
?>
```

ucfirst()

Funkcja `ucfirst()` zmienia pierwszą literę ciągu na dużą.

```
<?php
$oryginalny = "przykładowe zdanie.";
echo(ucfirst($oryginalny));
?>
```

ucwords()

Funkcja `ucwords()` zamienia pierwsze litery wyrazów w ciągu na duże.

```
<?php
$oryginalny = "miasto nowy jork";
echo(ucwords($oryginalny));
?>
```



Zarówno `ucwords()`, jak i `ucfirst()` nie konwertują liter na małe. Zmieniają tylko właściwe początkowe litery na duże. Jeżeli w środku ciągu wystąpią duże litery, nie zostaną zamienione.

Funkcje znaków sterujących

Jedną z charakterystycznych cech PHP jest to, że potrafi się on komunikować niemal ze wszystkimi programami. PHP działa jako język łączący strony WWW z bazami danych, serwerami LDAP, połączeniami poprzez gniazda sieciowe czy protokół HTTP. Zwykle dialog taki jest realizowany dwuetapowo. Najpierw tworzony jest ciąg komunikatu (na przykład zapytanie do bazy danych), następnie jest wysyłany do programu, z którym realizowana jest komunikacja. Często program ten w specjalny sposób interpretuje niektóre znaki. Aby były one potraktowane jako część literału, a nie kod sterujący, należy je specjalnie oznaczyć.

Wielu użytkowników uaktywnia opcję *magic-quotes*, dzięki czemu znaki apostrofów i cudzysłowów automatycznie poprzedzane są znakiem ukośnika przed wstawieniem ciągów do bazy danych. Jeśli rozwiązanie takie okaże się jednak niewygodne lub niepożądane, istnieją też funkcje usuwające i dodające znaki `\`. Funkcja `addslashes()` oznacza apostrofy, cudzysłowy, lewe ukośniki oraz znaki NULL za pomocą lewego ukośnika. Zwykle jest to konieczne przy wysyłaniu znaków do bazy danych.

```
<?php
$escapedstring = addslashes("Po angielsku zdanie - 'I'm a dog.' znaczy 'Jestem psem.'");
$query = "INSERT INTO test (quote) values ('$escapedstring')";
$result = mysql_query($query) or die(mysql_error());
?>
```

Dzięki temu serwer SQL nie potraktuje apostrofu przed "I" jak zakończenia wyrażenia. Jeżeli odczytujesz dane z bazy, musisz użyć funkcji `stripslashes()`, aby usunąć znaki sterujące.

```
<?php
$query = "SELECT quote FROM test WHERE ID = 1";
$result = mysql_query($query) or die (mysql_error());
$new_row = mysql_fetch_array($result);
$quote = stripslashes($new_row[0]);

echo $quote;
?>
```

Funkcja `quotemeta()` przetwarza wszystkie znaki specjalnego znaczenia w poleceniach powłoki Unix: '.', '\', '+', '*', '?', '[', '^', ']', '(', '\$', ')', ' '.

Na przykład:

```
$literal= 'Znaki ($, *) mają dla mnie szczególne znaczenie \n<BR>';
$qm = quotemeta($literal);
echo $qm;
```

Wykonanie tego fragmentu da w wyniku:

```
Znaki \(\$, \*) mają dla mnie szczególne znaczenie \n
```



Funkcje sterujące specyficzne dla HTML opisano w punkcie „Zaawansowane funkcje obsługi ciągów” w rozdziale 22.

Formatowanie danych

Podstawowymi konstrukcjami używanymi do drukowania danych są `print` i `echo`, które zostały opisane w rozdziale 5. Zwykle wartości zmiennych są wypisywane poprzez wbudowywanie zmiennych w ciągi otoczone cudzysłowami (i są zamieniane na wartości) i przekazywanie takiego ciągu do instrukcji `echo` lub `print`.

Jeżeli potrzebujesz dokładniej sformatowanego tekstu, PHP udostępnia funkcje `printf()` i `sprintf()`, które działają analogicznie do funkcji o tych samych nazwach w C. Obie funkcje mają takie same argumenty: specjalny ciąg formatujący (opisany poniżej), a następnie dowolną liczbę argumentów, które zostaną wklejone we właściwe miejsca ciągu formatującego.

Jedyną różnicą pomiędzy `printf()` i `sprintf()` jest to, że `printf()` wysyła wynik bezpośrednio na wyjście, a `sprintf()` zwraca wynik jako ciąg.



Dla programistów C: funkcja `sprintf()` różni się od jej wersji z C jeszcze jednym szczegółem. W C należy utworzyć ciąg wynikowy, PHP sam utworzy ten ciąg.

Najbardziej skomplikowaną częścią tych funkcji jest ciąg formatujący. Każdy znak, jaki umieścisz w tym ciągu, pojawi się w ciągu wynikowym, oprócz sekwencji znaków rozpoczynających się od %. Znak % oznacza początek „specyfikacji konwersji”, która wskazuje sposób wydruku argumentu odpowiadającego tej specyfikacji.

Po znaku % można umieścić pięć elementów określających sposób konwersji (niektóre opcjonalnie): wypełnienie, wyrównanie, minimalna szerokość, dokładność oraz typ.

- ♦ Pojedynczy (opcjonalny) znak wypełnienia może być zerem lub spacją. Znak ten jest używany do wypełniania nieużywanego miejsca, które zarezerwowano, określając minimalną szerokość. Jeżeli znak ten nie zostanie podany, wolne miejsce zostanie wypełnione spacjami.
- ♦ Opcjonalny znak *wyrównania* (-) wskazuje, czy wyświetlana wartość powinna być wyrównywana do strony prawej, czy do lewej. Jeżeli został umieszczony, wartość jest wyrównana do lewej strony; jeżeli go nie ma, wartość jest wyrównana do prawej strony.
- ♦ Opcjonalna liczba określająca *minimalną szerokość*; wartość zajmuje co najmniej tyle miejsca. Jeżeli będzie potrzebne więcej miejsca do wydrukowania wartości, przekroczona zostanie szerokość pola.
- ♦ Opcjonalne określenie szerokości części ułamkowej. Składa się z kropki i liczby określającej dokładność drukowania liczb rzeczywistych (dla innych wartości nie ma znaczenia).
- ♦ Pojedynczy znak określający sposób, w jaki ma być interpretowany typ zmiennej. Znak *f* oznacza drukowanie liczby rzeczywistej, *s* oznacza drukowanie ciągu, reszta znaków (*b*, *c*, *d*, *o*, *x*, *X*) oznacza, że wartość jest traktowana jak liczba całkowita i drukowana w różnych formatach. Formaty te to: format binarny dla litery *b*, *c* oznacza drukowanie znaku o odpowiednim kodzie ASCII, *o* oznacza liczbę oktalną, *x* to liczba heksagonalna (pisana małymi literami), a *X* to liczba heksagonalna pisana dużymi literami.

Przykładem może być drukowanie tej samej liczby rzeczywistej na różne sposoby:

```
<pre>
<?php
$wartosc = 3.14159;
printf("%f, %10f, %-010f, %2.2f\n", $wartosc, $wartosc, $wartosc, $wartosc);
?>
</pre>
```

Wynik działania tego fragmentu:

```
3.141590,          3.141590, 3.1415900000000000,  3.14
```

Konstrukcja `<pre></pre>` w HTML wskazuje przeglądarce, aby nie formatowała tak oznaczonego bloku (nie usuwała wielokrotnych odstępów itp.).

Rozszerzony przykład: kalkulator do ćwiczeń

W niniejszym punkcie rozszerzymy przykładowy kalkulator do ćwiczeń utworzony w rozdziale 5., wykorzystując szereg funkcji operujących na ciągach do przetwarzania ciągów wysyłanych z formularza użytkownika. W poprzednim przykładzie udało nam się jedynie przekazać zmienną będącą ciągiem znaków z formularza HTML do strony PHP mającej za zadanie odebrać i obsłużyć ten ciąg. W tym punkcie zaimplementujemy mechanizmy analizy otrzymanego ciągu. (W końcowej części niniejszego punktu dowiesz się, dlaczego nasz przykład wymaga dalszego rozszerzania w następnych rozdziałach).

Listing 8.1 prezentuje kod formularza, w którym użytkownik ma wpisać wykonywane ćwiczenie. Jest on bardzo podobny do formularza zaprezentowanego w rozdziale 7. — zmianie uległa jedynie docelowa strona z kodem obsługującym formularz.

Listing 8.1. Formularz początkowy

```
<HTML>
<HEAD>
<STYLE TYPE="text/css">
<!--
BODY, P {color: black; font-family: verdana; font-size: 10 pt}
H1      {color: black; font-family: arial; font-size: 12 pt}
-->
</STYLE>
</HEAD>
<BODY>
<TABLE BORDER=0 CELLPADDING=10 WIDTH=100%>
<TR>
<TD BGCOLOR="#F0F8FF" ALIGN=Center VALIGN=Top WIDTH=150>
</TD>
<TD BGCOLOR="#FFFFFF" ALIGN=Left VALIGN=Top WIDTH=83%>
<H1>Kalkulator do ćwiczeń (przekazywanie ciągu znaków)</H1>
<P>Wpisz rodzaj ćwiczenia, a kalkulator powie ci, jak długo powinienes je wykonywać,
<BR>by spalić kilogram tłuszczu.</P>
<FORM METHOD="post" ACTION="wc_handler_str.php">
<INPUT TYPE="text" SIZE=50 NAME="exercise">
<BR><BR>
<INPUT TYPE="submit" NAME="submit" VALUE="Spalaj się!">
</FORM>
</TD>
</TR>
</TABLE>
</BODY>
</HTML>
```

Na listingu 8.2 przedstawiono zmienioną stronę obsługującą formularz, która wyświetla odpowiednie statystyki w zależności od ćwiczenia wpisanego przez użytkownika.



W poniższym kodzie celowo użyliśmy superglobalnej tablicy `$_POST` do odczytania wartości ciągu znaków przekazanego z formularza. Oznacza to jednak, że kod z listingu 8.2 nie będzie działać w PHP w wersji wcześniejszej niż 4.1. Aby przystosować skrypt do wcześniejszej wersji języka, należy zamiast `$_POST` użyć `$HTTP_POST_VARS`, pamiętając jednak, że obecnie długie nazwy zmiennych są oficjalnie nieważne i w którymś momencie przestaną być w ogóle obsługiwane.

Listing 8.2. Skrypt obsługi formularza przy użyciu funkcji ciągów

```

<?php

$exercise_str = $_POST['exercise'];    // Uwaga! Działa tylko w PHP 4.1+
// Upewniamy się, że użytkownik przestrzega reguł
if (strlen($exercise_str) > 50) {
    echo "Nie stosujesz się do reguł. Wstydź się!";
    exit;
}
// Próba przetworzenia ciągu wejściowego
// -----
// Usunięcie ewentualnych spacji na początku i na końcu ciągu
$exercise_str = trim($exercise_str);
// Przekształcenie wszystkich liter w małe, by ułatwić porównywanie ciągów
$exercise_str = strtolower($exercise_str);

// Przypisanie liczby godzin, przez jaką trzeba wykonywać dane ćwiczenie,
// by spalić kilogram tłuszczu
if ($exercise_str == 'jazda na rowerze' || $exercise_str == 'jazda na rowerze górskim') {
    $hours = '5 godzin i 40 minut';
} elseif ($exercise_str == 'bieg' ||
    $exercise_str == 'jogging') {
    $hours = '4 godziny i 30 minut';
} elseif ($exercise_str == 'piłka nożna' ||
    $exercise_str == 'futbol') {
    $hours = '4 godziny i 30 minut';
} elseif ($exercise_str == 'aerobik') {
    $hours = '7 godzin i 30 minut';
} else {
    // Wyzerowanie wszystkich innych ćwiczeń
    $exercise_str = '';
    $hours = '';
}

// Utworzenie zdania
// -----

if ($exercise_str != "" && $hours != "") {
    $message = 'Jeden kilogram tłuszczu można spalić, gdy ' . $exercise_str . ' jest
    uprawiany(a) ' .
        'przez ' . $hours . '.';
} elseif ($exercise_str == "" && $hours == "") {
    // Jeżeli ćwiczenia nie ma na naszej liście, wyświetlany jest
    // domyślny komunikat.
    $message = 'Brak danych dla podanego ćwiczenia.';
} else {
    // Istnieją jeszcze dwie możliwości
    // 1. Rozpoznamy ćwiczenie, lecz nie mamy dla niego statystyki
    // 2. Nie rozpoznamy ćwiczenia, ale jakimś cudem mamy dla niego statystyki
    // Żadna z nich nie powinna się zdarzyć, ale na wszelki wypadek...
    $message = 'Coś poszło bardzo źle!';
}

// Utworzenie strony
// -----
$page_str = <<< EOPAGE
<HTML>
<HEAD>
<STYLE TYPE="text/css">

```

```

<!--
BODY, P {color: black; font-family: verdana; font-size: 10 pt}
H1      {color: black; font-family: arial; font-size: 12 pt}
-->
</STYLE>
</HEAD>

<BODY>
<TABLE BORDER=0 CELLPADDING=10 WIDTH=100%>
<TR>
<TD BGCOLOR="#F0F8FF" ALIGN=CENTER VALIGN=TOP WIDTH=150>
</TD>
<TD BGCOLOR="#FFFFFF" ALIGN=LEFT VALIGN=TOP WIDTH=83%>
<H1>Obsługa kalkulatora do ćwiczeń - część 2</H1>
<P>Kalkulator informuje: "$message"</P>
</TD>
</TR>
</TABLE>
</BODY>
</HTML>
EOPAGE;

echo $page_str;

?>

```

Kod z listingu 8.2 wykonuje następujące czynności:

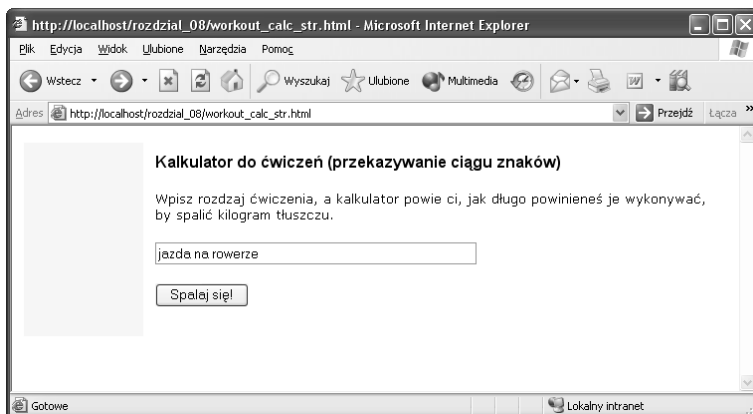
1. Odczytuje ciąg znaków przekazany z formularza HTML.
2. Próbuje doprowadzić otrzymany ciąg znaków do standardowej postaci, usuwając znaki spacji z początku i końca, przekształcając wszystkie litery na małe i rozpoznając niektóre synonimy.
3. Na podstawie oczyszczonego i odpowiednio przekształconego ciągu znaków wyszukuje statystyki dla podanego ćwiczenia.
4. Tworzy stronę z odpowiedzią przy użyciu składni heredoc.

Rysunki 8.1 i 8.2 przedstawiają efekt działania kalkulatora, gdy użytkownik wpisze jazdę na rowerze.

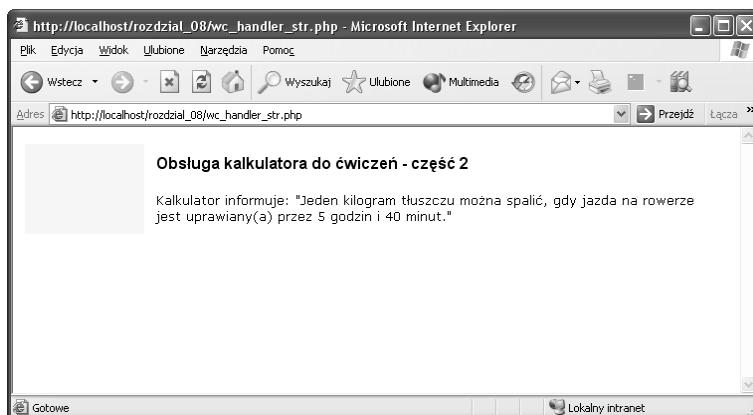
Nie znudził Ci się jeszcze kalkulator do ćwiczeń? Główny problem polega na tym, że nasz skrypt najprawdopodobniej nie będzie wiedział, jak obsłużyć różne dyscypliny sportu wpisywane przez użytkowników w sposób inny, niż wyświetlanie komunikatu domyślnego. Ponadto jeśli podniesiesz poziom raportowania błędów do E_ALL, wyświetlane będą ostrzeżenia informujące o braku inicjacji zmiennych. Użytkownik nie otrzymuje również żadnych wskazówek odnośnie dyscyplin, jakie rozpoznaje kalkulator.

Podsumowując, dopóki nie utworzymy systemu (takiego jak wyszukiwarka), który będzie umiał sobie radzić z dowolnymi tekstami, generowanie przez kod wyników w zależności od wartości wpisanych przez użytkowników mija się w praktyce z celem. Jeśli użytkownik ma dokonywać wyborów, należy zakres dostępnych wartości odpowiednio ograniczyć. Tekstu wpisywanego przez użytkownika można używać wówczas, gdy ma on być przeglądany przez innego użytkownika (np. zapisywany w tym celu do bazy danych lub przesyłany pocztą elektroniczną).

Rysunek 8.1.
Formularz początkowy



Rysunek 8.2.
Odpowiedź



Aby elegancko obsługiwać wartości, które może wybierać użytkownik, powinno się używać innych elementów interfejsu HTML niż pola tekstowe — np. pól wyboru, pól opcji czy list rozwijanych. Z kolei aby móc korzystać z tych elementów interfejsu po stronie klienta, w większym stopniu trzeba zacząć używać tablic. W tym właśnie kierunku pójdziemy w rozdziale 9.

Podsumowanie

Ciągi są sekwencjami znaków. Jest to jeden z sześciu podstawowych typów danych w PHP. W przeciwieństwie do niektórych innych języków, nie ma tu osobnego typu znakowego, pojedynczy znak zachowuje się jak ciąg o długości 1. Ciągi w kodzie są otaczane apostrofami (') lub cudzysłowami ("). Ciągi otoczone apostrofami są interpretowane prawie dosłownie, a ciągi otoczone cudzysłowami interpretują kilka sekwencji sterujących i automatycznie wstawiają wartości zmiennych.

Podstawowym operatorem ciągów jest '.', który łączy dwa ciągi. Istnieje dodatkowo spora gama funkcji, które umożliwiają sprawdzanie, porównywanie, szukanie, wycinanie, zamianę zawartości ciągu. PHP udostępnia także wyrażenia regularne, zgodne ze standardem POSIX i Perl (i opisane w rozdziale 22.), do zaawansowanych zastosowań.