

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, MySQL i Apache dla każdego. Wydanie II

Autor: Julie C. Meloni

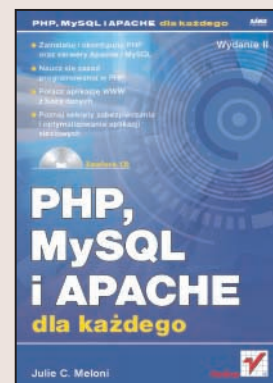
Tłumaczenie: Adam Byrtek (wprowadzenie, rozdz. 1 – 14),

Jarosław Dobrzański (rozdz. 15 – 32)

ISBN: 83-7361-877-5

Tytuł oryginału: [Sams Teach Yourself PHP, MySQL and Apache All in One, 2nd Edition](#)

Format: B5, stron: 588



Stwórz własną dynamiczną witrynę WWW

- Zainstaluj i skonfiguruj PHP oraz serwery Apache i MySQL
- Naucz się zasad programowania w PHP
- Połącz aplikację WWW z bazą danych
- Poznaj sekrety zabezpieczania i optymalizowania aplikacji sieciowych

Dynamiczne witryny WWW spotykamy w sieci coraz częściej. Po mechanizmy bazodanowe i technologie skryptowe działające po stronie serwera sięgają już nie tylko twórcy portali i sklepów internetowych, ale także ci, którym jeszcze do niedawna wystarczył zwykły, statyczny HTML. Wśród technologii wykorzystywanych do tworzenia dynamicznych stron WWW od dawna prym wiedzie duet PHP i MySQL, instalowany na serwerach WWW pracujących pod kontrolą Apache'a. Te właśnie narzędzia są najczęściej wykorzystywane do tworzenia galerii, forów dyskusyjnych, giełd ogłoszeniowych i wielu innych aplikacji WWW.

„PHP, MySQL i Apache dla każdego. Wydanie II” to podręcznik dla wszystkich, którzy chcą poznać zasady tworzenia dynamicznych witryn WWW z wykorzystaniem najpopularniejszych obecnie technologii. Książka opisuje proces instalacji i konfiguracji Apache'a, MySQL-a i PHP na serwerze oraz na stacji roboczej, elementy języka PHP oraz zasady stosowania języka SQL. Na praktycznych przykładach przedstawia możliwości wykorzystywania PHP i bazy danych do tworzenia elementów dynamicznych stron WWW oraz całych projektów. Czytając ją, nauczysz się zabezpieczać aplikacje WWW oraz poprawiać wydajność ich działania.

- Instalowanie i konfigurowanie narzędzi
- Programowanie w języku PHP
- Funkcje, tablice i obiekty
- Przetwarzanie danych z formularzy
- Obsługa sesji oraz systemu plików
- Dynamiczne generowanie grafiki
- Integracja PHP z MySQL
- Operacje na danych w tabelach i język SQL
- Tworzenie prostych projektów aplikacji WWW
- Monitorowanie pracy aplikacji
- Mechanizmy bezpieczeństwa i uwierzytelniania użytkowników
- Poprawa wydajności aplikacji

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



Spis treści

O Autorach	15
Podziękowania	17
Wprowadzenie	19
Część I Podstawy	23
Rozdział 1. Szybka instalacja	25
Instalacja w systemie Linux/Unix	25
Instalacja MySQL	26
Instalacja Apache	27
Instalacja PHP	27
Instalacja w systemie Windows	29
Instalacja MySQL	29
Instalacja Apache	30
Instalacja PHP	32
Rozwiązywanie problemów	34
Rozdział 2. Instalacja i konfiguracja MySQL	37
Wersja aktualna i przyszłe wersje MySQL	37
Jak zdobyć MySQL	38
Instalacja MySQL w systemie Linux/Unix	38
Instalacja MySQL w systemie Windows	39
Rozwiązywanie problemów instalacji	43
Podstawy bezpieczeństwa	44
Uruchamianie MySQL	45
Zabezpieczanie połączenia MySQL	45
Wprowadzenie do systemu uprawnień MySQL	46
Dwustopniowa autoryzacja	47
Korzystanie z systemu uprawnień	48
Dodawanie użytkowników	48
Usuwanie uprawnień	50
Podsumowanie	51
Pytania i odpowiedzi	51
Warsztaty	52
Test	52
Odpowiedzi	52
Ćwiczenia	52

Rozdział 3. Instalacja i konfiguracja Apache	53
Wersja aktualna i przyszłe wersje Apache	53
Wybór sposobu instalacji	54
Kompilacja kodu źródłowego	54
Instalacja dystrybucji binarnej	54
Instalacja Apache w systemie Linux/Unix	55
Pobieranie kodu źródłowego	55
Rozpakowanie kodu źródłowego	55
Przygotowania do kompilacji Apache	56
Budowanie i instalacja Apache	57
Instalacja Apache w systemie Windows	57
Format pliku konfiguracyjnego Apache	60
Dyrektywy	60
Pojemniki	62
Instrukcje warunkowe	63
ServerRoot	64
Pliki konfiguracyjne dla katalogów	64
Pliki dziennika Apache	65
access_log	65
error_log	66
Pozostałe pliki	66
Polecenia związane z Apache	66
Serwer Apache	66
Skrypt kontrolny Apache	68
Pierwsze uruchomienie Apache	68
Sprawdzanie pliku konfiguracyjnego	68
Uruchamianie Apache	69
Rozwiązywanie problemów	69
Inny serwer WWW	69
Brak uprawnień do portu	70
Dostęp zabroniony	70
Złe parametry grupy	71
Podsumowanie	71
Pytania i odpowiedzi	71
Warsztaty	72
Test	72
Odpowiedzi	72
Ćwiczenia	72
Rozdział 4. Instalacja i konfiguracja PHP	73
Wersja aktualna i przyszłe wersje PHP	73
Kompilacja PHP w systemie Linux/Unix	74
Dodatkowe opcje konfiguracyjne	75
Integracja PHP z Apache w systemie Linux/Unix	75
Instalacja PHP w systemie Windows	76
Integracja PHP z Apache w systemie Windows	76
Plik php.ini	77
Testowanie	78
Gdzie znaleźć pomoc	78
Podstawy skryptów PHP	80
Początek i koniec bloku instrukcji PHP	81
Instrukcja echo i funkcja print()	82
Łączenie HTML i PHP	83
Komentarze w kodzie PHP	83
Podsumowanie	85
Pytania i odpowiedzi	85

Warsztaty	86
Test	86
Odpowiedzi	86
Ćwiczenia	86
Część II Struktura języka PHP	87
Rozdział 5. Podstawowe elementy języka PHP	89
Zmienne	89
Zmienne globalne i superglobalne	91
Typy danych	92
Zmiana typu za pomocą settype()	94
Zmiana typu poprzez rzutowanie	95
Po co sprawdzać typ?	97
Operatory i wyrażenia	97
Operator przypisania	98
Operatory arytmetyczne	98
Operator konkatencji	98
Złożony operator przypisania	99
Inkrementacja i dekrementacja wartości zmiennej całkowitej	100
Operatory porównania	101
Tworzenie złożonych wyrażen za pomocą operatorów logicznych	102
Kolejność operatorów	103
Stałe	104
Stałe predefiniowane	105
Podsumowanie	105
Pytania i odpowiedzi	105
Warsztaty	106
Test	106
Odpowiedzi	107
Ćwiczenia	107
Rozdział 6. Sterowanie przepływem w PHP	109
Zmiana przepływu	110
Instrukcja if	110
Użycie klauzuli else w instrukcji if	111
Użycie klauzuli else if w instrukcji if	111
Instrukcja switch	113
Operator ?	114
Pętle	115
Instrukcja while	115
Instrukcja do...while	116
Instrukcja for	117
Przerywanie pętli za pomocą instrukcji break	118
Pomijanie iteracji za pomocą instrukcji continue	120
Zagnieżdżone pętle	121
Bloki kodu PHP	122
Podsumowanie	124
Pytania i odpowiedzi	124
Warsztaty	124
Test	125
Odpowiedzi	125
Ćwiczenie	126

Rozdział 7. Funkcje	127
Czym jest funkcja?	127
Wywoływanie funkcji	128
Definiowanie funkcji	129
Zwracanie wartości przez funkcje użytkownika	131
Zasięg zmiennych	132
Wywoływanie zmiennych za pomocą instrukcji global	133
Przechowywanie wartości pomiędzy wywołaniami funkcji za pomocą instrukcji static	135
Więcej o argumentach	137
Przypisywanie zmiennym wartości domyślnych	137
Przekazywanie zmiennych przez referencję	139
Sprawdzanie istnienia funkcji	140
Podsumowanie	142
Pytania i odpowiedzi	142
Warsztaty	142
Test	142
Odpowiedzi	143
Ćwiczenie	143
Rozdział 8. Tablice i obiekty	145
Czym jest tablica?	145
Tworzenie tablic	146
Tworzenie tablic asocjacyjnych	147
Tworzenie tablic wielowymiarowych	147
Niekóre funkcje operujące na tablicach	149
Tworzenie obiektu	151
Własności obiektów	152
Metody obiektów	153
Konstruktor	155
Dziedziczenie	155
Podsumowanie	157
Pytania i odpowiedzi	157
Warsztaty	157
Test	157
Odpowiedzi	158
Ćwiczenia	158
Część III Pierwsze kroki z kodem	159
Rozdział 9. Ciągi znaków, data i czas	161
Formatowanie ciągów znaków	162
Funkcja printf()	162
Zamiana argumentów	168
Przechowywanie sformatowanych ciągów znaków	169
Analizowanie ciągów znaków	170
Uwaga na temat indeksowania ciągów znaków	170
Sprawdzanie długości ciągu za pomocą funkcji strlen()	170
Znajdowanie podciągu za pomocą funkcji strpos()	171
Określanie pozycji podciągu za pomocą funkcji strpos()	171
Pobieranie części ciągu za pomocą funkcji substr()	172
Podział ciągu na słowa za pomocą funkcji strtok()	172
Operacje na ciągach znaków	174
Oczyszczanie ciągu za pomocą funkcji trim(), ltrim(), rtrim() i strip_tags()	174
Zmiana fragmentu ciągu za pomocą funkcji substr_replace()	175
Zamiana podciągów za pomocą funkcji str_replace()	176

Zmiana wielkości liter	176
Zawijanie tekstu za pomocą funkcji wordwrap() i nl2br()	177
Dzielenie ciągów za pomocą funkcji explode()	179
Funkcje operujące na dacie i czasie	179
Pobieranie bieżącej daty za pomocą funkcji time()	179
Konwersja znacznika czasu za pomocą funkcji getdate()	180
Formatowanie znacznika czasu za pomocą funkcji date()	181
Tworzenie znacznika czasu za pomocą funkcji mktime()	183
Weryfikacja daty za pomocą funkcji checkdate()	184
Podsumowanie	185
Pytania i odpowiedzi	185
Warsztaty	185
Test	185
Odpowiedzi	186
Ćwiczenia	186
Rozdział 10. Formularze	189
Tworzenie prostego formularza	189
Przekazywanie informacji w tablicach	191
Łączenie kodu HTML i PHP w jednym skrypcie	194
Zapisywanie informacji o stanie w ukrytym polu	196
Przekierowania	197
Wysyłanie poczty elektronicznej	199
Konfiguracja systemu	199
Tworzenie formularza	200
Skrypt wysyłający wiadomość	201
Formatowanie wiadomości za pomocą HTML	202
Przesyłanie plików	204
Tworzenie formularza wysyłającego plik	204
Skrypt obsługujący przesłany plik	206
Podsumowanie	207
Warsztaty	208
Test	208
Odpowiedzi	208
Ćwiczenia	208
Rozdział 11. Cookies i sesje	209
Wprowadzenie do cookies	209
Anatomia cookie	210
Tworzenie cookie	211
Usuwanie cookie	212
Wprowadzenie do sesji	212
Otwieranie sesji	213
Zmienne sesyjne	214
Przekazywanie identyfikatora sesji w adresie	218
Niszczanie sesji i usuwanie zmiennych	218
Zastosowania sesji	219
Obsługa zarejestrowanych użytkowników	219
Obsługa ustawień użytkownika	219
Podsumowanie	220
Pytania i odpowiedzi	220
Warsztaty	220
Test	220
Odpowiedzi	221
Ćwiczenie	221

Rozdział 12. Pliki i katalogi	223
Dołączanie plików za pomocą funkcji include()	223
Zwracanie wartości z dołączonego dokumentu	224
Instrukcja include() wewnątrz struktur sterujących	225
Użycie include_once()	226
Dyrektywa include_path	226
Weryfikacja plików	227
Sprawdzanie, czy dany plik istnieje, za pomocą funkcji file_exists()	227
Plik czy katalog?	228
Sprawdzanie uprawnień pliku	228
Sprawdzanie rozmiaru pliku za pomocą funkcji filesize()	229
Pobieranie informacji o danych związanych z plikiem	229
Funkcja wyświetlająca informacje o pliku	230
Tworzenie i usuwanie plików	231
Otwieranie plików do zapisu, odczytu i dopisywania	232
Odczytywanie danych z pliku	233
Odczytywanie wierszy za pomocą funkcji fgets() i feof()	233
Odczytywanie określonej ilości danych za pomocą funkcji fread()	234
Odczytywanie kolejnych znaków za pomocą funkcji fgetc()	236
Zapisywanie i dopisywanie danych do pliku	237
Zapisywanie danych do pliku za pomocą funkcji fwrite() i fputs()	237
Blokowanie plików za pomocą funkcji flock()	238
Operacje na katalogach	239
Tworzenie katalogów za pomocą funkcji mkdir()	239
Usuwanie katalogu za pomocą rmdir()	240
Otwieranie katalogu za pomocą funkcji opendir()	240
Odczytywanie zawartości katalogu za pomocą funkcji readdir()	240
Podsumowanie	242
Pytania i odpowiedzi	242
Warsztaty	242
Test	242
Odpowiedzi	243
Ćwiczenia	243
Rozdział 13. Środowisko serwera	245
Otwieranie potoku za pomocą funkcji popen()	245
Przekazywanie poleceń do powłoki za pomocą funkcji exec()	247
Wykonywanie poleceń za pomocą funkcji system() lub passthru()	249
Podsumowanie	250
Pytania i odpowiedzi	250
Warsztaty	251
Test	251
Odpowiedzi	251
Rozdział 14. Obrazki	253
Proces powstawania obrazka	253
Kilka słów o kolorze	253
Konieczne zmiany w PHP	254
Pobieranie dodatkowych bibliotek	254
Rysowanie nowego obrazka	255
Rysowanie kształtów i linii	255
Wypełnianie kształtów kolorem	257
Rysowanie wykresów	258
Modyfikacja istniejących obrazków	262
Tworzenie obrazków na podstawie danych przesłanych przez użytkownika	264
Podsumowanie	268

Pytania i odpowiedzi	268
Warsztaty	268
Test	269
Odpowiedzi	269
Ćwiczenie	269

Część IV Integracja PHP i MySQL271

Rozdział 15. Tajniki procesu projektowania bazy danych 273

Rola dobrego projektu bazy danych	273
Typy relacji między tabelami	274
Relacje jeden do jednego	275
Relacje jeden do wielu	275
Relacje wiele do wielu	276
Normalizacja	277
Problemy z tabelą prostą	278
Pierwsza postać normalna	278
Druga postać normalna	279
Trzecia postać normalna	280
Postępowanie zgodnie z procesem projektowania	280
Podsumowanie	281
Pytania i odpowiedzi	282
Warsztaty	282
Test	282
Odpowiedzi	282
Ćwiczenie	282

Rozdział 16. Podstawowe polecenia SQL 283

Typy danych w MySQL	284
Liczbowe typy danych	284
Typy czasu i daty	285
Typy łańcuchowe	286
Składnia tworzenia tabel	287
Używanie polecenia INSERT	288
Bliższe spojrzenie na INSERT	288
Stosowanie polecenia SELECT	290
Porządkowanie wyników zwracanych przez SELECT	291
Ograniczanie wyników	291
Używanie WHERE w zapytaniach	292
Stosowanie operatorów w klauzuli WHERE	293
Porównywanie łańcuchów za pomocą LIKE	294
Selekcja z kilku tabel	294
Używanie JOIN	296
Modyfikowanie rekordów za pomocą polecenia UPDATE	298
Warunkowe instrukcje UPDATE	300
Stosowanie bieżących wartości kolumn z UPDATE	300
Używanie polecenia REPLACE	301
Stosowanie polecenia DELETE	302
Warunkowa instrukcja DELETE	303
Często stosowane funkcje MySQL operujące na ciągach tekstowych	304
Funkcje długości i konkatencji	304
Funkcje przycinające i dopełniające	306
Funkcje lokalizacji i pozycji	308
Funkcje operujące na podciągach	308
Funkcje modyfikujące ciągi	309

Korzystanie z funkcji daty i czasu w MySQL	310
Operowanie na dniach	311
Operowanie na miesiącach i latach	313
Operowanie na tygodniach	314
Operowanie na godzinach, minutach i sekundach	315
Formatowanie daty i czasu w MySQL	316
Działania arytmetyczne na datach w MySQL	318
Funkcje specjalne i możliwości w zakresie konwersji	320
Podsumowanie	322
Pytania i odpowiedzi	323
Warsztaty	324
Test	324
Odpowiedzi	325
Ćwiczenie	325
Rozdział 17. Interakcja z MySQL z poziomu PHP	327
Łączenie się z MySQL poprzez PHP	327
Używanie <code>mysql_connect()</code>	328
Wykonywanie zapytań	329
Odbieranie komunikatów o błędach	330
Operowanie na danych z bazy MySQL	331
Wstawianie danych z poziomu PHP	331
Pobieranie danych z bazy w PHP	335
Pozostałe funkcje MySQL w PHP	336
Podsumowanie	337
Warsztaty	337
Test	337
Odpowiedzi	337
Część V Proste projekty	339
Rozdział 18. Zarządzanie prostą listą mailingową	341
Opracowywanie mechanizmu subskrypcji	341
Tworzenie tabeli subskrybentów	342
Tworzenie formularza subskrypcji	342
Budowa mechanizmu mailingu	348
Podsumowanie	351
Pytania i odpowiedzi	352
Warsztaty	352
Test	352
Odpowiedzi	352
Rozdział 19. Tworzenie internetowej książki adresowej	353
Planowanie i tworzenie tabel w bazie danych	353
Tworzenie menu	356
Tworzenie mechanizmu dodawania rekordów	356
Przeglądanie rekordów	361
Tworzenie mechanizmu usuwania rekordów	367
Uzupełnianie istniejących rekordów	369
Podsumowanie	374
Warsztaty	375
Test	375
Odpowiedzi	375
Ćwiczenia	375

Rozdział 20. Tworzenie prostego forum dyskusyjnego	377
Projektowanie tabel w bazie danych	377
Tworzenie formularzy wprowadzania danych i skryptów	378
Wyświetlanie listy tematów	381
Wyświetlanie postów w temacie	385
Dodawanie postu w wybranym temacie	388
Podsumowanie	391
Pytania i odpowiedzi	392
Warsztaty	392
Test	392
Odpowiedzi	392
Ćwiczenie	393
Rozdział 21. Tworzenie witryny sklepu internetowego	395
Planowanie i tworzenie tabel w bazie danych	395
Wstawianie rekordów do tabeli sklep_kategorie	397
Wstawianie rekordów do tabeli sklep_artykuly	397
Wstawianie rekordów do tabeli sklep_art_rozmiar	398
Wstawianie rekordów do tabeli sklep_art_kolor	399
Wyświetlanie kategorii artykułów	399
Wyświetlanie artykułów	402
Podsumowanie	405
Warsztaty	405
Test	405
Odpowiedzi	406
Rozdział 22. Tworzenie mechanizmu koszyka z zakupami	407
Planowanie i tworzenie tabel	407
Integracja koszyka z witryną sklepową	409
Dodawanie artykułów do koszyka	412
Przeglądanie zawartości koszyka	413
Usuwanie artykułów z koszyka	416
Sposoby dokonywania płatności i sekwencja kasowa	416
Tworzenie formularza kasowego	417
Realizowanie czynności kasowych	417
Podsumowanie	418
Warsztaty	418
Test	419
Odpowiedzi	419
Rozdział 23. Tworzenie prostego kalendarza	421
Tworzenie prostego kalendarza wyświetlanego na ekranie	421
Sprawdzenie danych przesłanych przez użytkownika	422
Tworzenie formularza HTML	423
Tworzenie tabeli kalendarza	425
Dodawanie terminów do kalendarza	428
Tworzenie biblioteki kalendarza	435
Podsumowanie	440
Pytania i odpowiedzi	441
Warsztaty	441
Test	441
Odpowiedzi	441
Ćwiczenie	441

Rozdział 24. Ograniczanie dostępu do aplikacji	443
Istota uwierzytelniania	443
Uwierzytelnianie klienta	444
Możliwości funkcjonalne modułu uwierzytelniającego serwera Apache	445
Uwierzytelnianie bazujące na plikach	446
Kontrola dostępu bazująca na pliku bazy danych	448
Apache jako narzędzie kontroli dostępu	449
Wprowadzanie reguł dostępu	449
Interpretacja reguł dostępu	451
Wiązane zastosowanie metod kontroli dostępu	452
Ograniczenie dostępu na podstawie metod HTTP	453
Ograniczenie dostępu na podstawie wartości cookies	454
Tworzenie tabeli uprawnionych użytkowników	454
Tworzenie formularza logowania i skryptu	455
Sprawdzanie cookie uwierzytelniającego	458
Podsumowanie	459
Pytania i odpowiedzi	459
Warsztaty	460
Test	460
Odpowiedzi	460
Ćwiczenie	460
Rozdział 25. Monitorowanie i prowadzenie dzienników aktywności serwera	461
Standardowe odnotowywanie dostępu do serwera	461
Ustalanie treści dzienników	462
Odnnotowywanie dostępu w plikach	465
Odnnotowywanie dostępu w programie	466
Standardowy tryb odnotowywania błędów serwera Apache	467
Odnnotowywanie błędów w pliku	467
Odnnotowywanie błędów w programie	467
Demon syslog jako argument	467
Dyrektywa LogLevel	468
Zarządzanie dziennikami serwera Apache	468
Ustalanie nazw hostów	469
Rotacja dzienników	469
Łączenie i podział dzienników	470
Analiza dzienników	470
Monitorowanie dzienników błędów	470
Odnnotowywanie informacji w bazie danych	471
Tworzenie tabeli w bazie danych	471
Tworzenie skryptu PHP odnotowującego dane	471
Tworzenie przykładowych raportów	472
Podsumowanie	475
Pytania i odpowiedzi	475
Warsztaty	475
Test	475
Odpowiedzi	475
Rozdział 26. Lokalizacja aplikacji	477
Internacjonalizacja i lokalizacja	477
Zestawy znaków	478
Modyfikacje środowiska	479
Zmiany w konfiguracji serwera Apache	479
Zmiany w konfiguracji PHP	480
Zmiany w konfiguracji MySQL	480
Tworzenie zlokalizowanej struktury strony	481
Podsumowanie	486

Pytania i odpowiedzi	487
Warsztaty	487
Test	487
Odpowiedzi	487

Część VI Administrowanie i dostrajanie489

Rozdział 27. Poprawianie wydajności i wirtualny hosting na serwerze Apache 491

Kwestie skalowalności	492
Ograniczenia systemu operacyjnego	492
Ustawienia serwera Apache związane z wydajnością	494
Testowanie serwera pod obciążeniem przy użyciu ApacheBench	495
Aktywne dostrajanie wydajności	497
Odwzorowywanie plików w pamięci	497
Rozkład obciążenia	498
Buforowanie	498
Redukcja ilości transmitowanych danych	498
Ustawienia sieci	499
Zapobieganie nadużyciom	499
Roboty	499
Implementacja wirtualnego hostingu	500
Wirtualny hosting bazujący na adresach IP	501
Wirtualny hosting bazujący na nazwach	501
Masowy hosting wirtualny	503
Podsumowanie	504
Pytania i odpowiedzi	505
Warsztaty	506
Test	506
Odpowiedzi	506

Rozdział 28. Bezpieczny serwer WWW 507

Potrzeba bezpieczeństwa	507
Protokół SSL	508
Rozwiązanie kwestii poufności	508
Rozwiązanie kwestii nienaruszalności	510
Rozwiązanie kwestii uwierzytelniania	510
Uzyskiwanie i instalacja narzędzi SSL	513
OpenSSL	513
Moduł mod_ssl serwera Apache	514
Zarządzanie certyfikatami	515
Tworzenie pary kluczy	515
Tworzenie prośby o podpisanie certyfikatu	517
Tworzenie certyfikatu podpisanego przez nas samych	518
Konfiguracja SSL	518
Uruchamianie serwera	519
Podsumowanie	519
Pytania i odpowiedzi	519
Warsztaty	520
Test	520
Odpowiedzi	520

Rozdział 29. Optymalizacja i dostrajanie MySQL 521

Tworzenie zoptymalizowanej platformy	521
Stosowanie funkcji benchmark()	522
Opcje inicjalizacyjne MySQL	523
Kluczowe parametry startowe	524

Optymalizacja struktury tabel	525
Optymalizacja zapytań	525
Korzystanie z polecenia FLUSH	527
Korzystanie z polecenia SHOW	528
Pobieranie informacji o bazach danych i tabelach	529
Pobieranie informacji o strukturze tabel	530
Pobieranie statusu systemu	531
Podsumowanie	533
Pytania i odpowiedzi	534
Warsztaty	534
Test	534
Odpowiedzi	535
Ćwiczenia	535
Rozdział 30. Aktualizacja oprogramowania	537
Trzymanie ręki na pulsie	537
Kiedy aktualizować?	538
Aktualizacja MySQL	539
Aktualizacja Apache	539
Modyfikowanie Apache bez dokonywania aktualizacji	540
Aktualizacja PHP	541
Podsumowanie	541
Warsztaty	541
Test	542
Odpowiedzi	542
Część VII Spojrzenie w przyszłość	543
Rozdział 31. Możliwości funkcjonalne i wsteczna kompatybilność PHP 5.0	545
Co było złe w PHP 4?	545
Nowy model obiektowy	546
Pozostałe nowe możliwości	548
SQLite	548
Obsługa XML-a	549
A więc kiedy instalować PHP 5?	549
Kompatybilność wsteczna	549
Podsumowanie	550
Pytania i odpowiedzi	550
Rozdział 32. Możliwości funkcjonalne i wsteczna kompatybilność MySQL 4.1	551
Korzystanie z podzapytań	551
Przykład systemu śledzącego czas pracy z zastosowaniem podzapytań	552
Ulepszenia dotyczące internacjonalizacji	555
Inne nowe możliwości funkcjonalne	555
Spojrzenie w przyszłość — MySQL 5.0	556
Podsumowanie	556
Pytania i odpowiedzi	557
Warsztaty	557
Test	557
Odpowiedzi	557
Dodatki	559
Skorowidz	561

Rozdział 5.

Podstawowe elementy języka PHP

W tym rozdziale zaczniemy poznawać język skryptowy PHP. Czytelnicy niebędący programistami z początku mogą poczuć się nieco przytłoczeni liczbą informacji, nie ma jednak powodów do obaw, zawsze można wrócić do tego rozdziału później. Lepiej skupić się na zrozumieniu głównych pojęć, niż starać się dokładnie zapamiętać omówione konstrukcje. Na pewno pojawią się one w tej książce jeszcze nie raz, więc będzie jeszcze szansa ich przyswojenia.

Doświadczeni programiści powinni przynajmniej przejrzeć ten rozdział, gdyż omawiamy w nim kilka cech szczególnych dla PHP, dotyczących zmiennych globalnych, typów danych i ich konwersji.

W tym rozdziale omówimy:

- ◆ Zmienne — czym są, dlaczego są potrzebne i jak z nich korzystać
- ◆ Definiowanie zmiennych i odczytywanie ich wartości
- ◆ Typy danych
- ◆ Niektóre często używane operatory
- ◆ Tworzenie wyrażeń za pomocą operatorów
- ◆ Definiowanie stałych i korzystanie z nich

Zmienne

Zmienne to zdefiniowane przez programistę „pojemniki”, które mogą przechowywać jakąś wartość: liczbę, ciąg znaków, obiekt, tablicę czy wartość logiczną. Zmienna to podstawowe pojęcie w programowaniu. Gdyby nie istniały zmienne, wszystkie wartości

należałoby zapisać bezpośrednio w skrypcie. Dodanie do siebie dwóch liczb i wyświetlenie rezultatu to już skrypt, rozwiązujący jakiś problem:

```
echo (2 + 4);
```

Jednak ten fragment kodu jest przydatny tylko dla tych, którzy chcą znać sumę liczb 2 i 4. Można oczywiście napisać kolejny skrypt, wyświetlający sumę liczb, powiedzmy, 3 i 5. Oczywiście takie podejście do programowania staje się absurdalne, wtedy właśnie przychodzą z pomocą zmienne.

Dzięki zmiennym możemy tworzyć szablony operacji (takich jak dodawanie dwóch liczb) bez zwracania uwagi na to, na jakich dokładnie wartościach operujemy. Wartość zostanie przypisana zmiennej w trakcie działania skryptu, być może na podstawie danych dostarczonych przez użytkownika, pobranych z bazy danych, czy w zależności od wcześniejszego przebiegu skryptu. Innymi słowy, zmiennych należy używać wszędzie tam, gdzie dane mogą ulec zmianie.

Zmienna składa się z wybranej przez programistę nazwy, poprzedzonej znakiem dolara (\$). Nazwy zmiennych mogą składać się z liter, cyfr i znaku podkreślenia (_), nie mogą zawierać spacji. Nazwa musi zaczynać się literą lub znakiem podkreślenia. Oto kilka prawidłowo nazwanych zmiennych:

```
$a;
$długa_nazwa_zmiennej;
$a2453;
$spiacyzzzz;
```



Nazwy zmiennych powinny coś mówić o ich zawartości, dobrze też, gdy ich nazewnictwo jest spójne. Na przykład, jeśli skrypt przechowuje informacje o nazwie użytkownika i hasła, nie należy umieszczać nazwy w zmiennej \$n, a hasła w zmiennej \$h, te nazwy zmiennych niewiele mówią o ich zawartości. Modyfikując skrypt kilka tygodni później, możesz pomyśleć, że \$n oznacza „numer”, nie „nazwę”, a \$h oznacza „historię”, nie „hasło”. A jak współpracownik ma się domyślić, co oznaczają nazwy \$n i \$h? Zmienne można nazywać zgodnie z dowolną konwencją, jeśli tylko nazwy będą opisowe i zgodne z jakąś regułą, którą mogą stosować inni.

Średnik (;), nazywany także *terminatorem instrukcji*, służy do zakończenia instrukcji PHP. Średniki w poprzednim fragmencie kodu nie były częścią nazwy zmiennej, stanowiły koniec opcjonalnej instrukcji, informującej PHP o tym, że zmienna jest „gotowa do działania”. Aby zadeklarować zmienną, wystarczy w skrypcie umieścić odwołanie do niej. Na ogół w momencie deklaracji zmiennej przypisywana jest jakaś wartość początkowa:

```
$num1 = 8;
$num2 = 23;
```

Powyżej zadeklarowaliśmy dwie zmienne i przypisaliśmy im wartość za pomocą operatora przypisania (=). Przypisywanie wartości zmiennym omawiamy bardziej szczegółowo w podrozdziale „Operatory i wyrażenia”, w dalszej części tego rozdziału. Po przypisaniu, zmienne można traktować w ten sam sposób, jak wartość, którą reprezentują. Innymi słowy, instrukcja

```
echo $num1;
```

jest równoznaczna z instrukcją

```
echo 8;
```

tak długo, jak długo `$num1` ma wartość 8.

Zmienne globalne i superglobalne

Oprócz zasad nazywania zmiennych istnieją także zasady określające dostępność zmiennych. Przede wszystkim wartość każdej zmiennej jest widoczna tylko wewnątrz funkcji lub skryptu, w którym się znajduje. Na przykład, jeśli w pliku *skryptA.php* zmiennej `$imie` została przypisana wartość `jan`, a w pliku *skryptB.php* także korzystamy ze zmiennej `$imie`, możemy bez obaw przypisać jej wartość `anna`, nie będzie to miało wpływu na *skryptA.php*. Zmienna `$imie` jest *lokalna* w obrębie każdego skryptu, a jej wartości w różnych skryptach są od siebie niezależne.

Zmienną `$imie` można też zdefiniować jako *globalną*. Jeśli zmienna `$imie` zostanie zdefiniowana jako globalna zarówno w pliku *skryptA.php*, jak i *skryptB.php*, a te dwa skrypty będą w jakiś sposób powiązane (na przykład jeden z nich wywołuje lub dołącza drugi), zmienna będzie miała jedną wspólną wartość. Zmienne globalne omówimy bardziej szczegółowo w rozdziale 6., „Sterowanie przepływem w PHP”.

Oprócz zdefiniowanych przez programistę zmiennych globalnych, PHP zawiera kilka predefiniowanych zmiennych, nazywanych *superglobalnymi*. Są one dostępne we wszystkich skryptach. Każda ze zmiennych superglobalnych to tak naprawdę tablica:

- ♦ `$_GET` zawiera zmienne przesłane do skryptu za pomocą metody GET.
- ♦ `$_POST` zawiera zmienne przesłane do skryptu za pomocą metody POST.
- ♦ `$_COOKIE` zawiera zmienne przesłane do skryptu za pomocą *cookies*.
- ♦ `$_FILES` zawiera zmienne przesłane do skryptu poprzez wysłanie pliku.
- ♦ `$_SERVER` zawiera informacje takie jak wysłane nagłówki, ścieżkę do pliku czy adres skryptu.
- ♦ `$_ENV` zawiera wartości zmiennych środowiskowych serwera.
- ♦ `$_REQUEST` zawiera zmienne przesłane do skryptu przez użytkownika.
- ♦ `$_SESSION` zawiera zmienne zarejestrowane w aktualnej sesji.

W treści książki będziemy korzystać ze zmiennych superglobalnych wszędzie tam, gdzie to tylko możliwe. Korzystanie ze zmiennych superglobalnych poprawia bezpieczeństwo aplikacji, zmniejszając prawdopodobieństwo wstawienia obcych danych do skryptu. Pisząc skrypty tak, aby akceptowały tylko dane, których się spodziewamy, przekazane w odpowiedni sposób (na przykład za pomocą formularza wysyłanego metodą POST, czy sesji), możemy uniknąć wielu problemów.

Typy danych

Różne typy danych zajmują różną ilość pamięci, mogą też być traktowane w różny sposób w trakcie wykonywania skryptu. Dlatego właśnie niektóre języki zobowiązują programistę do wcześniejszego określenia typu danych, które zmienna może przechowywać. Jednak PHP jest językiem z luźnym typowaniem, a więc typ zmiennej jest określany w momencie przypisywania jej wartości.

Takie automatyczne typowanie ma swoje wady i zalety. Z jednej strony daje dużą elastyczność w użyciu zmiennych — zmienna przechowująca ciąg znaków w dalszej części skryptu może zawierać liczbę lub dowolny inny typ danych. Z drugiej strony, może to prowadzić do problemów w przypadku większych skryptów, gdy spodziewamy się konkretnego typu zmiennej, a zmienna zawiera dane innego typu. Na przykład założmy, że napisaliśmy kod operujący na tablicy. Jeśli odpowiednia zmienna zamiast tablicy zawiera wartość liczbową, każda operacja tablicowa na tej zmiennej spowoduje błąd.

W tabeli 5.1 opisujemy podstawowe typy danych, dostępne w PHP.

Tabela 5.1. Podstawowe typy danych

Typ	Przykład	Opis
Boolean	true	Wartość logiczna, określona jedną ze specjalnych stałych: true (prawda) albo false (fałsz).
Integer	5	Liczba całkowita.
Float lub Double	3.234	Liczba zmiennopozycyjna.
String	"witaj"	Ciąg znaków.
Object		Obiekt, egzemplarz klasy.
Array		Tablica, uporządkowany zbiór kluczy i wartości.
Resource		Odnośnik do zewnętrznego zasobu (na przykład bazy danych).
NULL		Niezdefiniowana zmienna.

Zmienne typu resource są często zwracane przez funkcje odpowiadające za współpracę z zewnętrznymi aplikacjami lub plikami. Na przykład w rozdziale 17., „Interakcja z MySQL z poziomu PHP” wspominamy o „identyfikatorze zasobu MySQL”. Typ NULL jest zarezerwowany dla zmiennych, które zostały zadeklarowane, ale nie przypisano im żadnej wartości.

Aby sprawdzić typ zmiennej, można użyć wbudowanej funkcji `gettype()`. Zwróci ona ciąg znaków reprezentujący typ zmiennej, umieszczonej w nawiasie funkcji. Na listingu 5.1 przypisujemy zmiennej wartości różnych typów, a następnie wyświetlamy rezultat funkcji `gettype()`. Komentarze mówią, jakiego typu jest zmienna w danym momencie.



Wywoływanie funkcji omawiamy bardziej szczegółowo w rozdziale 7., „Funkcje”.

Listing 5.1. Sprawdzanie typu zmiennej

```
1: <?php
2: $test; // deklaracja bez przypisania wartości
3: echo gettype($test); // null
4: echo "<br>";
5: $test = 5;
6: echo gettype($test); // integer
7: echo "<br>";
8: $test = "pięć";
9: echo gettype($test); // string
10: echo "<br>";
11: $test = 5.024;
12: echo gettype($test); // double
13: echo "<br>";
14: $test = true;
15: echo gettype($test); // boolean
16: echo "<br>";
17: $test = array("jabłko", "pomarańcza", "gruszka");
18: echo gettype($test); // array
19: echo "<br>";
20: ?>
```

Powyższy kod należy umieścić w pliku *gettype.php*, w katalogu nadrzędnym serwera WWW. Po otwarciu skryptu w przeglądarce internetowej, powinien pojawić się następujący rezultat:

```
NULL
integer
string
double
boolean
array
```

Deklarując zmienną `$test` w linii 2., nie przypisaliśmy jej żadnej wartości, a więc pierwsze wywołanie funkcji `gettype()` w linii 3. zwraca wartość `NULL`. Następnie, za pomocą operatora przypisania (`=`), nadajemy zmiennej kolejne wartości, wyświetlając rezultat funkcji `gettype()`. W linii 5. zmiennej została przypisana wartość całkowita, mówiąc krótko, liczba bez wartości dziesiętnej¹. W linii 8. zmiennej przypisujemy ciąg znaków, który zawsze powinien być otoczony pojedynczymi lub podwójnymi znakami cudzysłowu (`'` lub `"`). Wartość `double`, przypisana zmiennej `$test` w linii 11., to liczba zmiennopozycyjna (zawierająca kropkę dziesiętną). Wartość logiczna, przypisana zmiennej `$test` w linii 14., może przyjmować jedną z dwóch możliwych wartości: `true` lub `false`. W linii 17. do utworzenia tablicy wykorzystano funkcję `array()`, którą omówimy dokładniej w rozdziale 8., „Tablice i obiekty”. Przykładowa tablica zawiera trzy elementy, a funkcja `gettype()` zgodnie z prawdą zwraca wartość „array”.

¹ W PHP, podobnie jak w języku angielskim, część całkowita oddzielana jest od części dziesiętnej kropką, nie przecinkiem — *przyp. red.*

Zmiana typu za pomocą `settype()`

Język PHP posiada funkcję `settype()`, umożliwiającą zmianę typu zmiennej. Zmienną i nazwę nowego typu należy przekazać jako argumenty funkcji `settype()` (w nawiasie, oddzielone przecinkiem), jak w poniższym przykładzie:

```
settype($zmienna, 'nowy typ');
```

Kod na listingu 5.2 konwertuje wartość 3.14 (typu `float`) do każdego z czterech omówionych wcześniej typów danych.

Listing 5.2. *Zmiana typu zmiennej za pomocą `settype()`*

```
1: <?php
2: $nieokreslona = 3.14;
3: echo gettype($nieokreslona); // double
4: echo " ma wartość $nieokreslona<br>"; // 3.14
5: settype($nieokreslona, 'string');
6: echo gettype($nieokreslona); // string
7: echo " ma wartość $nieokreslona<br>"; // 3.14
8: settype($nieokreslona, 'integer');
9: echo gettype($nieokreslona); // integer
10: echo " ma wartość $nieokreslona<br>"; // 3
11: settype($nieokreslona, 'double');
12: echo gettype($nieokreslona); // double
13: echo " ma wartość $nieokreslona<br>"; // 3
14: settype($nieokreslona, 'bool');
15: echo gettype($nieokreslona); // boolean
16: echo " ma wartość $nieokreslona<br>"; // 1
17: ?>
```



Zgodnie z podręcznikiem PHP, wartość „double” (a nie „float”) zostanie zwrócona nawet wtedy, gdy zmienna jest typu `float`.

Za każdym razem za pomocą funkcji `gettype()` sprawdzamy nowy typ zmiennej, a następnie wyświetlamy jej wartość. Gdy ciąg „3.14” zostanie zamieniony w linii 8. na liczbę całkowitą, części dziesiętnej nie można już odzyskać. To właśnie dlatego zmienna `$nieokreslona`, gdy zamienimy ją z powrotem na `double` w linii 11., ma wartość 3. Wreszcie w linii 14. konwertujemy zmienną na wartość logiczną. Wartość `true` jest reprezentowana przez 1, a wartość `false` — przez pusty ciąg znaków. A więc w linii 16. zostanie wyświetlona wartość 1.

Powyższy kod należy umieścić w pliku `settype.php`, w katalogu nadrzędnym serwera WWW. Po otwarciu skryptu w przeglądarce internetowej, powinien pojawić się następujący rezultat:

```
double ma wartość 3.14
string ma wartość 3.14
integer ma wartość 3
double ma wartość 3
boolean ma wartość 1
```

Zmiana typu poprzez rzutowanie

Zmiana typu zmiennej za pomocą funkcji `gettype()` różni się od zmiany typu poprzez rzutowanie przede wszystkim tym, że w tym drugim przypadku tworzona jest kopia zmiennej, a oryginał pozostaje bez zmian. Aby zmienić typ za pomocą rzutowania, należy podać jego nazwę, zawartą w nawiasie, przed zmienną, której dotyczy rzutowanie. Na przykład poniższa instrukcja tworzy kopię zmiennej `$oryginalna` odpowiedniego typu (`integer`) i przypisuje jej wartość do zmiennej `$nowa`. Zmienna `$oryginalna` będzie wciąż dostępna, a jej typ nie ulegnie zmianie:

```
$nowa = (integer) $oryginalna;
```

Kod na listingu 5.3 zmienia typ za pomocą rzutowania.

Listing 5.3. *Rzutowanie zmiennej*

```
1: <?php
2: $nieokreslona = 3.14;
3: $chwila = (double) $nieokreslona;
4: echo gettype($chwila); // double
5: echo " ma wartość $chwila<br>"; // 3.14
6: $chwila = (string) $nieokreslona;
7: echo gettype($chwila); // string
8: echo " ma wartość $chwila<br>"; // 3.14
9: $chwila = (integer) $nieokreslona;
10: echo gettype($chwila); // integer
11: echo " ma wartość $chwila<br>"; // 3
12: $chwila = (double) $nieokreslona;
13: echo gettype($chwila); // double
14: echo " ma wartość $chwila<br>"; // 3.14
15: $chwila = (boolean) $nieokreslona;
16: echo gettype($chwila); // boolean
17: echo " ma wartość $chwila<br>"; // 1
18: echo "<hr>";
19: echo "typ zmiennej początkowej: ";
20: echo gettype($nieokreslona); // double
21: ?>
```

Typ zmiennej `$nieokreslona` pozostaje bez zmian (`double`) w trakcie działania skryptu, czego dowodzi funkcja `gettype()` w linii 20., wyświetlająca typ zmiennej.

W praktyce w trakcie rzutowania zmiennej `$nieokreslona` tworzona jest jej kopia, skonwertowana na odpowiedni typ, która następnie zostaje przypisana zmiennej `$chwila`. Rzutowania występują w liniach 3., 6., 9., 12. i 15. Operujemy jedynie na kopii zmiennej `$nieokreslona`, a więc początkowa wartość zmiennej pozostaje bez zmian (w przeciwieństwie do kodu na listingu 5.2, gdzie w linii 8. zmienna została skonwertowana na typ `integer`).

Powyższy kod należy umieścić w pliku `testtype.php`, w katalogu nadrzędnym serwera WWW. Po otwarciu skryptu w przeglądarce internetowej, powinien pojawić się następujący rezultat:

```
double ma wartość 3.14
string ma wartość 3.14
integer ma wartość 3
double ma wartość 3.14
boolean ma wartość 1
typ zmiennej początkowej: double
```

Teraz, gdy już zaprezentowaliśmy dwa sposoby zmiany typu zmiennej, zastanówmy się, w jakich sytuacjach może to być użyteczne. Na pewno nie będziemy korzystać z tej możliwości zbyt często, PHP automatycznie rzutuje zmienne w zależności od kontekstu, w jakim zostaną użyte. Jednak takie automatyczne rzutowanie jest tymczasowe, a czasem zachodzi potrzeba permanentnej zmiany typu zmiennej.

Na przykład liczby, wpisane przez użytkownika w formularzu HTML, zostaną przekazane do skryptu jako ciągi znaków. Przy próbie dodania do siebie dwóch takich ciągów, PHP w celu przeprowadzenia dodawania skonwertuje je na liczby. A więc

```
"30cm" + "40cm"
```

w rezultacie da 70.



Ogólnym terminem „liczba” określamy zarówno liczby całkowite, jak i zmiennopozycyjne. A więc jeśli użytkownik w formularzu poda wartości zmiennopozycyjne, reprezentowane przez ciągi „3.14cm” i „4.12cm”, rezultatem dodawania będzie 7.26.

Podczas rzutowania ciągu znaków na liczbę całkowitą lub zmiennopozycyjną, PHP zignoruje wszystkie znaki nienumeryczne. Fragment od pierwszego znaku nienumerycznego do końca ciągu zostanie obcięty. A więc ciąg „30cm” zostanie zamieniony na „30”, jednak ciąg „6m2cm” będzie miał wartość liczbową „6”, cała reszta zostanie obcięta.

Czasem warto samodzielnie „oczyścić” dane przesłane przez użytkownika przed ich użyciem w skrypcie. Załóżmy, że użytkownik miał przesłać liczbę. Możemy zasymulować taką sytuację, nadając wartość odpowiedniej zmiennej:

```
$test = "30cm";
```

Jak widać, użytkownik dodał do liczby jednostkę — zamiast wpisać „30”, podał wartość „30cm”. Aby upewnić się, że wartość zmiennej będzie całkowita, można dokonać rzutowania:

```
$nowytest = (integer) $test;
echo "Pudełko ma szerokość $nowytest centymetrów.";
```

Oto rezultat działania powyższego kodu:

```
Pudełko ma szerokość 30 centymetrów.
```

Gdyby nie zastosowano rzutowania i wyświetlono początkową wartość zmiennej \$test, rezultat byłby inny:

```
Pudełko ma szerokość 30cm centymetrów.
```

Trzeba przyznać, że to zdanie wygląda trochę dziwnie.

Po co sprawdzać typ?

Kiedy znajomość typu zmiennej może okazać się przydatna? Zdarzają się sytuacje, gdy operujemy na danych pobranych z jakiegoś obcego źródła. W rozdziale 7. omówimy, jak korzystać z funkcji. Dane są często przekazywane pomiędzy funkcjami, które mogą odbierać informacje za pomocą argumentów. Aby zapewnić poprawne działanie funkcji, dobrze wcześniej sprawdzić, czy dane do niej przekazane są odpowiedniego typu. Na przykład funkcja spodziewająca się argumentu typu „resource” nie będzie dobrze działała, gdy otrzyma zamiast tego ciąg znaków.

Operatory i wyrażenia

Pokazaliśmy, jak przypisywać zmiennym wartości, a także jak sprawdzać i zmieniać typ zmiennych. Jednak bez możliwości przeprowadzania operacji na danych, żaden język programowania nie byłby zbyt użyteczny. Operatory to symbole, za pomocą których można manipulować danymi przechowywanymi w zmiennych. Dzięki temu na podstawie istniejących danych można otrzymać nową wartość, sprawdzić poprawność danej i tak dalej. Wartość, na której działa operator nazywamy operandem.



Operator to symbol (lub kilka symboli), który użyty wraz z pewnymi wartościami tworzy nową wartość.

Operand to wartość wykorzystywana w operatorze. Operator posiada na ogół co najmniej dwa operandy.

W tym prostym przykładzie operator, wraz z dwoma operandami, tworzy nową wartość:

(4 + 5)

Liczby całkowite 4 i 5 to operandy. Zastosowany na nich operator dodawania (+) zwraca wartość całkowitą 9. Operator na ogół znajduje się pomiędzy dwoma operandami, choć od tej zasady jest kilka wyjątków.

Operator wraz z operandami tworzy *wyrażenie*, choć wyrażenie nie musi zawierać operatora. Tak naprawdę w PHP wyrażenie jest zdefiniowane jako wszystko to, co może zostać użyte jako wartość. A więc wyrażeniami są stałe całkowite — np. 654, zmienne — np. \$user, i wywołania funkcji — np. gettype(). Na przykład wyrażenie (4 + 5) w rzeczywistości składa się z dwóch wyrażeń składowych (4 i 5) i operatora (+). Gdy wyrażenie tworzy nową wartość, mówimy, że *zwraca* tę wartość. A więc wyrażenie można także traktować jako wartość. W naszym przykładzie wyrażenie (4 + 5) zwraca 9.



Wyrażenie to dowolna kombinacja wywołań funkcji, wartości i operatorów, zwracająca wartość. Łatwo zapamiętać, że jeśli pewnego fragmentu można użyć jako wartości, to jest to wyrażenie.

Po wstępie teoretycznym omówimy najpopularniejsze operatory występujące w języku PHP.

Operator przypisania

Operator przypisania, reprezentowany przez symbol =, był już przez nas wykorzystywany podczas deklaracji zmiennych. Odczytuje on wartość operandu po prawej stronie operatora i przypisuje do zmiennej po lewej stronie:

```
$imie = "adam";
```

Po wykonaniu tego fragmentu zmienna `$imie` będzie zawierała wartość "adam". Taka konstrukcja to także wyrażenie, choć z pozoru może się wydawać, że operator po prostu modyfikuje zmienną, nic nie zwracając. W rzeczywistości takie wyrażenie zawsze zwraca kopię prawego operandu. A więc

```
echo $imie = "adam";
```

przypisze zmiennej `$imie` wartość "adam", ale także wyświetli ciąg znaków "adam".

Operatory arytmetyczne

Operatory arytmetyczne robią dokładnie to, czego można się spodziewać po ich nazwie — przeprowadzają działania arytmetyczne. W tabeli 5.2 podajemy ich listę, wraz z przykładami użycia.

Tabela 5.2. Operatory arytmetyczne

Operator	Nazwa	Przykład	Rezultat
+	dodawanie	10+3	13
-	odejmowanie	10-3	7
/	dzielenie	10/3	3.3333333333333
*	mnożenie	10*3	30

Operator konkatencji

Operator konkatencji oznaczamy pojedynczą kropką (.). Traktując oba operandy jako ciągi znaków, operator skleja lewy operand z prawym. A więc wyrażenie

```
"witaj"." świecie"
```

zwróci wartość

```
"witaj świecie"
```

Warto zauważyć, że oba słowa są oddzielone spacją tylko dlatego, że występuje ona w drugim operandzie („ świecie” zamiast „świecie”). Operator konkatencji po prostu dołącza jeden ciąg do drugiego, nie rozdzielając ich w żaden sposób. A więc próba połączenia dwóch ciągów niepoprzedzonych lub zakończonych spacją, np.

```
"witaj"."świecie"
```

da następujący rezultat:

```
"witajświecie"
```

Niezależnie od typów zawartych w operandach, oba są traktowane jako ciągi znaków, a więc rezultat zawsze będzie ciągiem znaków. Operatorem konkatencji w treści książki bardzo często używamy do sklejenia ciągu znaków z wartością jakiegoś wyrażenia:

```
$cm = 212;
echo "szerokość: " . ($cm/100) . " metrów";
```

Złożony operator przypisania

Istnieje tylko jeden operator przypisania, jednak PHP posiada kilka operatorów złożonych, modyfikujących wartość zmiennej po lewej stronie operatora i zwracających jej nową wartość. Regułą jest, że operatory nie zmieniają wartości operandów — ta reguła nie dotyczy złożonych operatorów przypisania (i oczywiście standardowego operatora przypisania). Operatory złożone składają się z odpowiedniego symbolu, po którym następuje znak równości. Są znacznym ułatwieniem w sytuacjach, w których konieczne byłoby użycie dwóch osobnych operatorów. Na przykład chcąc zwiększyć wartość zmiennej, której początkowa wartość to 4, o kolejne 4, możemy napisać:

```
$x = 4;
$x = $x + 4; // $x ma teraz wartość 8
```

Można też użyć złożonego operatora przypisania (+=), aby zwiększyć wartość zmiennej:

```
$x = 4;
$x += 4; // $x ma teraz wartość 8
```

Każdy operator arytmetyczny, a także operator konkatencji, posiada odpowiadający mu operator złożony. W tabeli 5.3 podajemy ich listę wraz z przykładami użycia.

Tabela 5.3. Niektóre złożone operatory przypisania

Operator	Przykład	Zapis równoważny
+=	<code>\$x += 5</code>	<code>\$x = \$x + 5</code>
-=	<code>\$x -= 5</code>	<code>\$x = \$x - 5</code>
/=	<code>\$x /= 5</code>	<code>\$x = \$x / 5</code>
*=	<code>\$x *= 5</code>	<code>\$x = \$x * 5</code>
%=	<code>\$x %= 5</code>	<code>\$x = \$x % 5</code>
.=	<code>\$x .= " test"</code>	<code>\$x = \$x . " test"</code>

W każdym z przykładów w tabeli 5.3 wartość zmiennej `$x` zostaje zmodyfikowana na podstawie wartości prawego operandu. Od tej pory zmienna `$x` będzie zawierała nową wartość. Na przykład:

```
$x = 4;
$x += 4; // $x ma teraz wartość 8
$x += 4; // $x ma teraz wartość 12
$x -= 3; // $x ma teraz wartość 9
```

W dalszej części książki będziemy często stosowali operatory złożone. Złożony operator konkatencji często występuje we fragmentach, w których tekst konstruowany jest dynamicznie. Dobry przykład to zastosowanie pętli do dynamicznego tworzenia kodu HTML reprezentującego tabelę.

Inkrementacja i dekrementacja wartości zmiennej całkowitej

Podczas programowania w PHP często zachodzi potrzeba zwiększenia lub zmniejszenia o jeden wartości zmiennej całkowitej. Dobrym przykładem jest zliczanie iteracji pętli. Znamy już dwa sposoby osiągnięcia tego — użycie operatora dodawania

```
$x = $x + 1; // wartość $x zostaje zwiększona o 1
```

lub złożonego operatora przypisania

```
$x += 1; // wartość $x zostaje zwiększona o 1
```

W każdym z tych przypadków nowa wartość zostaje przypisana do zmiennej `$x`. Wyrażenia tego typu są bardzo często spotykane, dlatego PHP zawiera specjalne operatory, służące do zwiększania i zmniejszania wartości zmiennej o 1. To operatory *postinkrementacji* oraz *postdekrementacji*, pierwszy z nich składa się z dwóch symboli plus, umieszczonych za nazwą zmiennej:

```
$x++; // wartość $x zostaje zwiększona o 1
```

Powyższe wyrażenie zwiększa wartość zmiennej `$x` o jeden. Użycie dwóch symboli minus spowoduje zmniejszenie wartości zmiennej:

```
$x--; // wartość $x zostaje zmniejszona o 1
```

Warto pamiętać, że operatory postinkrementacji i postdekrementacji najpierw zwracają wartość operandu, a dopiero potem ją modyfikują (stąd przedrostek „post”):

```
$x = 3;  
$y = $x++ + 3;
```

W takiej sytuacji `$y` przyjmuje wartość 6 ($3 + 3$), a dopiero wtedy zmienna `$x` zostaje zwiększona.

W niektórych wyrażeniach warunkowych zachodzi potrzeba zmniejszenia (lub zwiększenia) wartości zmiennej jeszcze przed przeprowadzeniem testu. W tym celu PHP zapewnia operatory preinkrementacji i predekrementacji. Działają w ten sam sposób, jak wcześniej omówione operatory, jednak symbole je reprezentujące muszą poprzedzać zmienną:

```
++$x; // wartość $x zostaje zwiększona o 1  
--$x; // wartość $x zostaje zmniejszona o 1
```

Gdyby użyć tych operatorów w wyrażeniu warunkowym, wartość zmiennej zostanie zmniejszona jeszcze przed przeprowadzeniem testu. Na poniższym przykładzie zmienna `$x` zostaje zwiększona przed sprawdzeniem, czy jest mniejsza od 4:

```
$x = 3;  
++$x < 4; // zwraca wartość logiczną false
```

Liczba 4 nie jest mniejsza od 4, a więc wyrażenie warunkowe zwraca `false`.

Operatory porównania

Operatory porównania sprawdzają wartość operandów, zwracając wartość logiczną `true`, jeśli test się powiódł, a `false` w przeciwnym razie. Takie wyrażenia są bardzo przydatne w strukturach sterujących, takich jak instrukcje `if` czy `while`. Omówimy je w rozdziale 6.

Na przykład, aby sprawdzić, czy wartość zmiennej `$x` jest mniejsza niż 5, można użyć operatora „mniejsze niż”:

```
$x < 5
```

Jeśli `$x` ma wartość 3, wyrażenie zwróci wartość `true`. Jeśli `$x` zawiera wartość 7, wyrażenie zwróci `false`.

W tabeli 5.4 prezentujemy dostępne operatory porównania.

Tabela 5.4. *Operatory porównania*

Operator	Nazwa	Zwraca prawdę, jeśli...	Przykład (<code>\$x = 4</code>)	Rezultat
<code>==</code>	równe	lewa strona jest równa prawej	<code>\$x == 5</code>	<code>false</code>
<code>!=</code>	różne	lewa strona nie jest równa prawej	<code>\$x != 5</code>	<code>true</code>
<code>===</code>	identyczne	lewa strona jest równa prawej i są tego samego typu	<code>\$x === 4</code>	<code>true</code>
<code>!==</code>	nie identyczne	lewa strona jest różna prawej lub są różnego typu	<code>\$x !== "4"</code>	<code>false</code>
<code>></code>	większe	lewa strona jest większa niż prawa	<code>\$x > 4</code>	<code>false</code>
<code>>=</code>	większe lub równe	lewa strona jest większa lub równa prawej	<code>\$x >= 4</code>	<code>true</code>
<code><</code>	mniejsze	lewa strona jest mniejsza niż prawa	<code>\$x < 4</code>	<code>false</code>
<code><=</code>	mniejsze lub równe	lewa strona jest mniejsza lub równa prawej	<code>\$x <= 4</code>	<code>true</code>

Powyższe operatory najczęściej stosuje się wraz z wartościami całkowitymi lub zmiennopozycyjnymi, jednak operatora równości można użyć także do porównania dwóch ciągów znaków. Warto poświęcić chwilę, aby dobrze zrozumieć różnicę między operatorami `==` i `=`. Operator `==` sprawdza równość, podczas gdy `=` przypisuje zmiennej wartość. Dobrze też zapamiętać, że operator `===` sprawdza równość zarówno wartości, jak i typów.

Tworzenie złożonych wyrażeń za pomocą operatorów logicznych

Operatory logiczne przeprowadzają działania logiczne na operandach. Na przykład operator alternatywy, oznaczany dwiema kreskami (||) lub po prostu słowem `or` (lub), zwraca wartość logiczną „prawda”, jeśli którykolwiek z operandów jest prawdziwy:

```
true || false
```

Powyższe wyrażenie zwróci wartość `true`.

Operator koniunkcji, oznaczany dwoma znakami ampersand (&&) lub po prostu słowem `and` (i), zwraca wartość logiczną „prawda” tylko wówczas, gdy oba operandy są prawdziwe:

```
true && false
```

Powyższe wyrażenie zwróci wartość `false`. W rzeczywistości operatorów logicznych nie używa się na stałych, ale na dwóch lub więcej wyrażeniach, mających wartość logiczną. Na przykład:

```
($x > 2) && ($x < 15)
```

Powyższe wyrażenie zwróci wartość `true` tylko wtedy, gdy wartość `$x` jest większa od 2 i mniejsza od 15. Używamy nawiasów, aby zwiększyć czytelność kodu i określić kolejność obliczania wartości wyrażeń. W tabeli 5.5 prezentujemy dostępne operatory logiczne.

Tabela 5.5. Operatory logiczne

Operator	Nazwa	Zwraca prawdę, jeśli...	Przykład	Rezultat
	alternatywa (lub)	lewa lub prawa strona jest prawdziwa	<code>true false</code>	<code>true</code>
<code>or</code>	alternatywa (lub)	lewa lub prawa strona jest prawdziwa	<code>true or false</code>	<code>true</code>
<code>xor</code>	różnica symetryczna (<code>xor</code>)	lewa lub prawa strona jest prawdziwa, ale nie obie	<code>true xor true</code>	<code>false</code>
&&	koniunkcja (i)	lewa i prawa strona jest prawdziwa	<code>true && false</code>	<code>false</code>
<code>and</code>	koniunkcja (i)	lewa i prawa strona jest prawdziwa	<code>true and false</code>	<code>false</code>
!	negacja (nie)	pojedynczy operand nie jest prawdziwy	<code>! true</code>	<code>false</code>

Dociekliwi na pewno zastanawiają się, czemu istnieją dwie wersje operatora koniunkcji i alternatywy. To dobre pytanie, odpowiemy na nie w następnym podrozdziale.

Kolejność operatorów

PHP zazwyczaj odczytuje wyrażenia od lewej do prawej. Jednak w przypadku wyrażeń składających się z więcej niż jednego operatora, stosowanie tej zasady może prowadzić do niejasności. Rozważmy prosty przykład:

```
4 + 5
```

Nie ma tu miejsca na żadne niejasności, PHP po prostu doda 4 do 5. Co jednak w przypadku takiego wyrażenia, złożonego z dwóch operatorów:

```
4 + 5 * 2
```

Tutaj pojawia się problem. Czy silnik PHP powinien dodać 4 do 5, a następnie pomnożyć całość przez 2, co da rezultat 18? A może oznacza to, że do 4 należy dodać wynik mnożenia 5 razy 2, co da rezultat 14? Gdyby czytać wyrażenie od lewej do prawej, prawdziwe byłoby pierwsze stwierdzenie. Jednak PHP przypisuje poszczególnym operatorom różne priorytety, a ponieważ operator mnożenia ma większy priorytet niż dodawania, to właśnie drugie stwierdzenie jest prawdziwe. A więc wynikiem wyrażenia jest 4 plus rezultat mnożenia 5 razy 2.

Możemy jednak zmienić kolejność operatorów, zawierając wyrażenia w nawiasach. W poniższym przykładzie najpierw przeprowadzone zostanie dodawanie, a dopiero potem mnożenie:

```
(4 + 5) * 2
```

W przypadku wyrażeń złożonych, niezależnie od priorytetów operatorów, warto stosować nawiasy, aby uczynić kod bardziej czytelnym i uchronić się od poważnych błędów, takich jak naliczenie w sklepie internetowym podatku od niewłaściwej kwoty. Oto lista dotychczas omówionych operatorów, w kolejności priorytetów (od największego do najmniejszego):

```
++, --, (typ)  
/, *, %  
+, -  
<, <=, ==, >  
==, ===, !=  
&&  
||  
=, +=, -=, /=, *=, %=, .=  
and  
xor  
or
```

Jak widać, operatory `or` i `and` mają mniejszy priorytet niż odpowiadające im operatory `||` i `&&`. A więc można użyć odpowiedniej wersji, aby określić kolejność obliczania wyrażenia złożonego. Poniższe wyrażenia są równoważne, ale drugie z nich jest o wiele bardziej czytelne:

```
$x and $y || $z  
$x && ($y || $z)
```

Co więcej, poniższe wyrażenie jest jeszcze bardziej czytelne:

```
$x && ($y or $z)
```

Podkreślmy jeszcze raz, wszystkie trzy wersje są równoważne.

Priorytety to jedyne, co odróżnia operatory && i and (a także || i or), nie zawsze jednak warto polegać na priorytetach. W większości sytuacji użycie nawiasów sprawia, że kod staje się bardziej czytelny i mniej podatny na błędy. A więc w dalszej części książki będziemy stosowali operatory || i &&, kolejność wyrażen oznaczając za pomocą nawiasów.

Stałe

Użycie zmiennych to bardzo elastyczny sposób przechowywania danych, ich wartość można zmieniać w każdym miejscu skryptu. Czasem jednak zachodzi potrzeba stosowania wartości niezmiennych w trakcie działania skryptu — wtedy przydatne stają się *stałe*. Aby utworzyć stałą, należy użyć wbudowanej funkcji PHP `define()`. Wartość stałej pozostaje niezmienną, aż do następnego użycia `define()`. Aby zdefiniować stałą, w nawiasie w wywołaniu funkcji należy umieścić nazwę stałej i jej wartość, oddzielone przecinkiem:

```
define("NAZWA_STALEJ", 42);
```

Wartością stałej może być liczba, ciąg znaków czy wartość logiczna. Warto postępować zgodnie z konwencją i nazywać stałe wielkimi literami. Aby poznać wartość stałej, wystarczy użyć jej nazwy (nie należy poprzedzać jej symbolem dolara). Na listingu 5.4 pokazujemy, jak zdefiniować i sprawdzić wartość stałej:

Listing 5.4. Definiowanie stałej

```
1: <?php
2: define("BIEZACY_ROK", "2005");
3: echo "Mamy teraz rok ".BIEZACY_ROK;
4: ?>
```



Korzystając ze stałych, powinniśmy pamiętać, że można użyć ich w dowolnym miejscu skryptu, a także w dołączanych skryptach zewnętrznych.

Warto zauważyć, że w linii 3. użyliśmy operatora konkatencji, aby skleić wartość stałej z ciągiem "Mamy teraz rok ". Jest to możliwe dzięki temu, że PHP traktuje stałą w taki sam sposób, jak ciąg znaków, który reprezentuje.

Powyższy kod należy umieścić w pliku *stala.php*, w katalogu nadrzędnym serwera WWW. Po otwarciu skryptu w przeglądarce internetowej, powinien pojawić się następujący rezultat:

```
Mamy teraz rok 2005
```

Funkcja `define()` może przyjmować jeszcze trzeci argument, będący wartością logiczną. Określa on, czy w nazwie stałej powinna być rozróżniana wielkość liter. Domyślnie wielkość liter ma znaczenie, jednak przekazując `true` jako trzeci argument, możesz to zmienić. Wtedy wartość stałej, zdefiniowanej w następujący sposób:

```
define("BIEZACY_ROK", "2005", true);
```

możemy otrzymać, nie zwracając uwagi na wielkość liter:

```
echo biezacy_rok;  
echo BiEZaCY_Rok;  
echo BIEZACY_ROK;
```

Powyższe wyrażenia są równoważne, każde z nich wypisze wartość 2005. Dzięki temu możemy uczynić kod nieco bardziej przyjaznym dla innych programistów, korzystających z naszych stałych. Jednak z drugiej strony, biorąc pod uwagę to, że w nazwach stałych domyślnie *jest* rozróżniana wielkość liter, może to tylko przysporzyć kłopotów innym programistom. A więc jeśli nie ma ważnego powodu, aby postępować inaczej, lepiej w nazwach stałych brać pod uwagę wielkość liter, a także nazywać stałe wielkimi literami. To prosta (i ogólnie przyjęta) konwencja.

Stałe predefiniowane

PHP daje dostęp do pewnych wbudowanych stałych. Na przykład `__FILE__` zwraca nazwę pliku, zawierającego obecnie wykonywany skrypt. Stała `__LINE__` zwraca bieżącą linię w pliku. Te stałe są szczególnie użyteczne do wyświetlenia komunikatu o błędzie. Aby poznać wersję PHP działającą na serwerze, można użyć stałej `PHP_VERSION`.

Podsumowanie

W tym rozdziale omówiliśmy niektóre podstawowe cechy języka PHP. Opisaliśmy zmienne, zastosowanie operatora przypisania do zmiany ich wartości, wspomnieliśmy o zasięgu zmiennych oraz wbudowanych zmiennych superglobalnych. Omówiliśmy najpopularniejsze operatory i pokazaliśmy, jak za ich pomocą tworzyć złożone wyrażenia. Wreszcie pokazaliśmy, jak definiować i pobierać wartość stałych.

Teraz, gdy już opanowaliśmy podstawy PHP, w następnym rozdziale możemy ruszyć do przodu. Pokażemy, jak pisać skrypty podejmujące decyzje i powtarzające pewne zadania. Będziemy opierać się na wiedzy zdobytej w tym rozdziale.

Pytania i odpowiedzi

- P:** *Kiedy przydaje się znajomość typu zmiennej?*
- O:** Często typ zmiennej ma wpływ na to, jakie operacje można na niej przeprowadzać. Na przykład nie można stosować operacji tablicowych na ciągu znaków. Warto

też przed wykonaniem operacji arytmetycznych sprawdzić, czy odpowiednie zmienne są typu całkowitego czy zmiennopozycyjnego (nawet jeśli silnik PHP sam przeprowadzi odpowiednią konwersję).

P: *Czy nazywając zmienne, powinienem stosować się do jakichś konwencji?*

O: Podstawowym celem powinno zawsze być zwiększenie czytelności kodu. Nazwa zmiennej `$ab123245` nic nie mówi o jej znaczeniu w skrypcie, łatwo też o pomyłkę w zapisie takiej nazwy. Nazwy zmiennych powinny być krótkie i opisowe.

Nazwa zmiennej `$n` najprawdopodobniej nie będzie nic znaczyć, gdy wrócimy do kodu po miesiącu. O wiele lepszą nazwą dla zmiennej jest `$nazwapliku`.

P: *Czy priorytetów operatorów należy uczyć się na pamięć?*

O: Czemu nie, chociaż najprawdopodobniej szkoda na to czasu. Korzystając z nawiasów, możemy zarówno uczynić kod bardziej czytelnym, jak i wybrać swoją własną kolejność obliczania wyrażeń.

Warsztaty

Warsztaty mają na celu utrwalenie i sprawdzenie zdobytej wiedzy, powinny też pokazać, jak zastosować ją w praktyce.

Test

1. Która z poniższych nazw zmiennych nie jest poprawna?

```
$wartosc_od_uzytownika
$666666xyz
$xyz666666
$_licznik_
$cos_waznego
$nazwa-pliku
```

2. Jaki będzie rezultat działania poniższego kodu?

```
$num = 33;
(boolean) $num;
echo $num;
```

3. Jaki będzie rezultat działania poniższej instrukcji?

```
echo gettype("4");
```

4. Jaki będzie rezultat działania poniższego fragmentu?

```
$test = 5.5466;
settype($test, "integer");
echo $test;
```

5. Które z poniższych instrukcji nie są wyrażeniami?

```
4;  
gettype(44);  
5/12;
```

6. Które z instrukcji w pytaniu 5. zawierają operator?

7. Co zwróci poniższe wyrażenie?

```
5 < 2
```

Jaki będzie typ zwróconej wartości?

Odpowiedzi

1. Nazwa zmiennej `$66666xyz` nie jest poprawna, gdyż nie zaczyna się literą lub znakiem podkreślenia. Nazwa zmiennej `$cos_waznego` nie jest poprawna, gdyż zawiera spację. Zmienna `$nazwa-pliku` jest także niepoprawna, gdyż zawiera niedozwolony znak (-).
2. Kod wyświetli liczbę 33. W czasie rzutowania na wartość logiczną tworzona jest kopia wartości zmiennej `$num`, jednak sama zmienna nie jest modyfikowana.
3. Instrukcja wyświetli ciąg znaków "string".
4. Kod wyświetli wartość 5. Gdy liczba zmiennopozycyjna zostaje zamieniona na całkowitą, obcięta zostaje cała część dziesiętna.
5. Każda z tych instrukcji to wyrażenie, każda zwraca jakąś wartość.
6. Instrukcja `5/12` zawiera operator dzielenia.
7. Wyrażenie zwróci wartość logiczną `false`.

Ćwiczenia

1. Utwórz skrypt zawierający co najmniej pięć różnych zmiennych. Wypełnij je wartościami różnych typów, a następnie zastosuj funkcję `gettype()`, aby poznać ich typ.
2. Przypisz wartości dwóm zmiennym. Użyj operatorów porównania, aby sprawdzić, czy pierwsza wartość jest
 - ♦ taka sama jak druga,
 - ♦ mniejsza niż druga,
 - ♦ większa niż druga,
 - ♦ mniejsza lub równa drugiej.

Rezultat każdego z testów wyświetl w przeglądarce. Zmień wartości zmiennych i uruchom skrypt ponownie.