

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. Receptury

Autorzy: David Sklar, Adam Trachtenberg

Tłumaczenie: Tomasz Jarzębowski

ISBN: 83-7361-117-7

Tytuł oryginału: [PHP Cookbook](#)

Format: B5, stron: 648



PHP to prosty ale jednocześnie bardzo użyteczny język skryptowy dostępny na zasadach open source. W ostatnich latach stał się jednym z ważniejszych narzędzi programistycznych w Internecie. Ponad milion witryn, od stron wielkich korporacji po strony prywatne, wykorzystuje PHP do dynamicznego generowania zawartości stron WWW. PHP zawiera zbiór bardzo przydatnych funkcji, ma prostą składnię, obsługuje wielu różnych systemów operacyjnych oraz usług sieciowych, co czyni go idealnym narzędziem do szybkiego tworzenia dynamicznych stron WWW.

Książka zawiera unikalny zbiór przykładów i rozwiązań problemów spotykanych na co dzień w trakcie programowania w języku PHP. Dla każdego problemu zostało przedstawione wyczerpujące rozwiązanie lub „receptura” – krótki fragment kodu, który od zaraz można zastosować w aplikacjach. Jednak ta książka to nie tylko gotowy do wykorzystania kod, znajdziesz tu także wyjaśnienie, jak i dlaczego przedstawiony kod działa, a to ułatwi dostosowanie go do rozwiązania podobnych, a nie tylko identycznych problemów.

Receptury z niniejszej książki rozwiązują nie tylko proste problemy, jak wysyłanie zapytania do bazy danych lub przetwarzanie adresów URL, ale także całe programy wykonujące złożone zadania, na przykład wyświetlanie tabel HTML lub generowanie wykresów słupkowych.

Książka zawiera ponad 250 receptur obejmujących między innymi:

- pracę z prostymi typami danych: tekstami, liczbami, datami, czasem i tablicami,
- bloki tworzenia programów w PHP: zmienne, funkcje, klasy i obiekty,
- programowanie sieciowe, czyli tworzenie formularzy, dostęp do baz danych i XML,
- użyteczne funkcje jak: wyrażenia regularne, szyfrowanie i bezpieczeństwo, grafika, internacjonalizacja i lokalizacja a także usługi sieciowe,
- pracę z plikami i katalogami,
- wiersz poleceń PHP i PHP-GTK,
- PEAR (PHP Extension and Application Repository)

Książka zawiera wiele kodu użytecznego dla wszystkich programistów PHP, od nowicjuszy po weteranów. Zamiast śledzić listy dyskusyjne, dokumentację dostępną w Internecie oraz inne źródła, wystarczy zajrzeć do „PHP. Receptury”, by znaleźć rozwiązania typowych problemów.



Spis treści

Wstęp	13
Rozdział 1. Łańcuchy znaków	21
1.0. Wprowadzenie	21
1.1. Uzyskiwanie dostępu do podłańcuchów znaków	24
1.2. Zastępowanie podłańcuchów znaków	25
1.3. Przetwarzanie łańcucha znaków znak po znaku	26
1.4. Odwracanie kolejności słów lub znaków w łańcuchu znaków	28
1.5. Poszerzanie i zwężanie tabulatorów	29
1.6. Kontrolowanie wielkości liter	31
1.7. Umieszczanie funkcji i wyrażeń wewnątrz łańcuchów znaków	32
1.8. Odcinanie od łańcuchów znaków znaków niewidocznych	33
1.9. Parsowanie danych oddzielanych przecinkami	35
1.10. Parsowanie danych o stałej szerokości	36
1.11. Dzielenie łańcuchów znaków	38
1.12. Łamanie tekstu do określonej długości linii.....	41
1.13. Przechowywanie danych binarnych w łańcuchach znaków	42
Rozdział 2. Liczby	45
2.0. Wprowadzenie	45
2.1. Sprawdzanie, czy łańcuch znaków zawiera poprawną liczbę	46
2.2. Porównywanie liczb zmiennopozycyjnych	47
2.3. Zaokrąglanie liczb zmiennopozycyjnych	48

2.4. Wykonywanie operacji na seriach liczb całkowitych	49
2.5. Generowanie liczb losowych z danego przedziału.....	50
2.6. Generowanie ważonych liczb losowych.....	52
2.7. Obliczanie logarytmów.....	53
2.8. Obliczanie potęg	54
2.9. Formatowanie liczb	55
2.10. Wyświetlanie słów w liczbie mnogiej	56
2.11. Obliczanie wartości funkcji trygonometrycznych.....	57
2.12. Obliczanie funkcji trygonometrycznych w stopniach, a nie w radianach	58
2.13. Obsługa bardzo dużych lub bardzo małych liczb.....	59
2.14. Przekształcanie liczb z jednego systemu liczbowego na inny	61
2.15. Wykonywanie obliczeń na liczbach systemów innych niż dziesiętny	62
Rozdział 3. Daty i czas	65
3.0. Wprowadzenie	65
3.1. Sprawdzanie aktualnej daty i czasu	67
3.2. Przekształcanie elementów daty i czasu w znaczniki czasu epoki	69
3.3. Przekształcanie znacznika czasu epoki w elementy czasu i daty	71
3.4. Wyświetlanie daty lub czasu w określonym formacie.....	71
3.5. Obliczanie różnicy między dwiema datami.....	77
3.6. Obliczanie różnicy między dwiema datami.....	79
3.7. Znajdowanie dnia tygodnia, miesiąca lub roku oraz numeru tygodnia w roku	80
3.8. Weryfikacja poprawności daty	82
3.9. Parsowanie dat i czasu z łańcuchów znaków	84
3.10. Dodawanie lub odejmowanie czasu od daty	86
3.11. Wyznaczanie czasu w strefach czasowych.....	88
3.12. Uwzględnianie czasu zimowego.....	92
3.13. Generowanie czasu o wysokiej precyzji	94
3.14. Generowanie przedziałów czasowych.....	95
3.15. Stosowanie kalendarzy innych niż gregoriański.....	96
3.16. Program Calendar	98
Rozdział 4. Tablice	101
4.0. Wprowadzenie	101
4.1. Tworzenie tablicy zaczynającej się od indeksu różnego od 0	104
4.2. Przechowywanie w tablicy wielu elementów pod jednym kluczem.....	105
4.3. Inicjowanie tablicy liczbami całkowitymi z określonego przedziału.....	106
4.4. Iterowanie przez kolejne elementy tablicy	107

4.5. Usuwanie elementów z tablicy.....	110
4.6. Zmianianie rozmiaru tablicy.....	112
4.7. Łączenie tablic.....	114
4.8. Przekształcanie tablicy w łańcuch znaków.....	116
4.9. Wyświetlanie zawartości tablicy z przecinkami.....	117
4.10. Sprawdzanie, czy klucz jest w tablicy.....	119
4.11. Sprawdzanie, czy element jest w tablicy.....	119
4.12. Znajdowanie pozycji elementu w tablicy.....	121
4.13. Znajdowanie elementów, które spełniają odpowiednie warunki.....	122
4.14. Znajdowanie elementu tablicy o największej lub najmniejszej wartości.....	123
4.15. Odwracanie tablicy.....	124
4.16. Sortowanie tablicy.....	125
4.17. Sortowanie tablicy na podstawie porównywalnych pól.....	127
4.18. Sortowanie wielu tablic.....	129
4.19. Sortowanie tablicy przy użyciu metody, a nie funkcji.....	131
4.20. Ustawianie elementów tablicy w kolejności losowej.....	132
4.21. Tasowanie talii kart.....	133
4.22. Usuwanie z tablicy powtarzających się elementów.....	134
4.23. Wyznaczanie sumy, przecięcia lub różnicy między dwiema tablicami.....	135
4.24. Wyznaczanie wszystkich kombinacji elementów tablicy.....	137
4.25. Znajdowanie wszystkich permutacji tablicy.....	139
4.26. Program — wyświetlanie tablicy w tabeli HTML z kolumnami ułożonymi w poziomie....	142
Rozdział 5. Zmienne.....	145
5.0. Wprowadzenie.....	145
5.1. Unikanie pomyłek między operatorami == i =.....	146
5.2. Ustalanie wartości domyślnej.....	147
5.3. Wymiana wartości bez używania zmiennych tymczasowych.....	148
5.4. Tworzenie dynamicznej nazwy zmiennej.....	149
5.5. Stosowanie zmiennych statycznych.....	150
5.6. Współdzielenie zmiennych pomiędzy procesami.....	152
5.7. Enkapsulacja złożonych typów danych do postaci łańcucha znaków.....	154
5.8. Wyświetlanie zawartości zmiennej w postaci łańcuchów znaków.....	155
Rozdział 6. Funkcje.....	159
6.0. Wprowadzenie.....	159
6.1. Uzyskiwanie dostępu do parametrów funkcji.....	160
6.2. Ustawianie domyślnych wartości parametrów funkcji.....	161

6.3. Przekazywanie wartości przez odwołanie	163
6.4. Stosowanie parametrów nazwanych	164
6.5. Tworzenie funkcji pobierających zmienną liczbę argumentów	166
6.6. Zwracanie wartości przez referencję	168
6.7. Zwracanie więcej niż jednej wartości	169
6.8. Pomijanie wybranych wartości zwracanych	170
6.9. Zwracanie błędu	172
6.10. Wywoływanie funkcji zależnie od wartości zmiennych	173
6.11. Dostęp do zmiennej globalnej wewnątrz funkcji	174
6.12. Tworzenie funkcji dynamicznych	176
Rozdział 7. Klasy i obiekty	177
7.0. Wprowadzenie	177
7.1. Tworzenie egzemplarzy obiektów	181
7.2. Definiowanie konstruktorów obiektów	182
7.3. Usuwanie obiektu	183
7.4. Klonowanie obiektów	184
7.5. Przypisywanie odwołań do obiektów	185
7.6. Wywoływanie metod obiektu zwracanego przez inną metodę	186
7.7. Dostęp do metod przesłoniętych	186
7.8. Przeciążanie właściwości	188
7.9. Wykorzystanie wielopostaciowości metod	190
7.10. Sprawdzanie metod i właściwości obiektów	192
7.11. Dodawanie właściwości do obiektu podstawowego	194
7.12. Dynamiczne tworzenie klas	195
7.13. Dynamiczne tworzenie egzemplarzy obiektów	196
Rozdział 8. Podstawy programowania na potrzeby WWW	199
8.0. Wprowadzenie	199
8.1. Zapisywanie danych cookie	201
8.2. Odczytywanie danych cookie	202
8.3. Usuwanie danych cookie	203
8.4. Odsyłanie do innej strony	204
8.5. Śledzenie przebiegu sesji	205
8.6. Zapis sesji w bazie danych	206
8.7. Pozyskiwanie informacji o przeglądarkach	210
8.8. Konstruowanie zapytania metody GET	211
8.9. Proste uwierzytelnianie HTTP	214

8.10. Uwierzytelnianie z wykorzystaniem danych cookie	216
8.11. Wymuszenie przesłania danych do przeglądarki	219
8.12. Buforowanie danych wyjściowych	220
8.13. Przesyłanie danych z użyciem kompresji gzip	221
8.14. Ukrywanie komunikatów o błędach	222
8.15. Obsługa błędów	224
8.16. Tworzenie własnych procedur obsługi błędów	225
8.17. Zapisywanie błędów w dzienniku	227
8.18. Unikanie błędów powtórnego przesłania nagłówka	229
8.19. Rejestrowanie wartości zmiennych	230
8.20. Odczyt zmiennych środowiskowych	232
8.21. Ustawianie wartości zmiennych środowiskowych	233
8.22. Odczyt zmiennych konfiguracyjnych.....	234
8.23. Ustawianie wartości zmiennych konfiguracyjnych	236
8.24. Komunikacja w ramach serwera Apache	237
8.25. Określanie wydajności kodu.....	238
8.26. Program — aktywowanie i dezaktywowanie stron internetowych użytkowników	242
8.27. Program — monitorowanie aktywności użytkowników	244
Rozdział 9. Formularze	251
9.0. Wprowadzenie	251
9.1. Przetwarzanie danych pochodzących z formularza	253
9.2. Weryfikacja danych formularza	255
9.3. Formularze wielostronicowe	257
9.4. Powtórne wyświetlanie formularzy	260
9.5. Zabezpieczenie przed wielokrotnym przesyłaniem tego samego formularza	263
9.6. Obsługa przesyłanych plików	265
9.7. Zabezpieczanie kodu analizującego dane formularza.....	267
9.8. Obsługa znaków specjalnych w danych użytkownika	269
9.9. Obsługa zmiennych zawierających w nazwie znak kropki.....	271
9.10. Elementy formularza o większej liczbie opcji.....	272
9.11. Listy rozwijane zawierające daty	273
Rozdział 10. Dostęp do baz danych	275
10.0. Wprowadzenie	275
10.1. Bazy danych w plikach tekstowych	280
10.2. Bazy danych DBM	282
10.3. Zestawianie połączeń z bazami danych SQL.....	286

10.4. Przesyłanie zapytań do baz danych SQL	288
10.5. Odczyt wierszy bez użycia pętli	290
10.6. Wprowadzanie zmian w bazach danych SQL	292
10.7. Efektywne zwielokrotnianie zapytań	294
10.8. Określanie liczby udostępnionych wierszy	296
10.9. Obsługa znaków specjalnych	297
10.10. Zapisywanie informacji o przebiegu programu oraz komunikatów o błędach	299
10.11. Automatyczne dobieranie wartości identyfikatorów	302
10.12. Programowe konstruowanie zapytań	303
10.13. Tworzenie odsyłaczy do wielostronicowych wyników zapytania	307
10.14. Buforowanie zapytań i ich wyników	311
10.15. Program — wielowątkowa lista dyskusyjna	313
Rozdział 11. Automatyzacja pracy w sieci	321
11.0. Wprowadzenie	321
11.1. Pobieranie stron metodą GET	323
11.2. Pobieranie stron metodą POST	326
11.3. Pobieranie stron wymagających danych cookie	328
11.4. Pobieranie stron wymagających przesłania odpowiednich nagłówek	330
11.5. Pobieranie stron w protokole HTTPS	331
11.6. Analizowanie danych HTTP	331
11.7. Oznaczanie elementów strony WWW	334
11.8. Wyodrębnianie odsyłaczy z plików HTML	336
11.9. Przekształcanie tekstu ASCII w kod HTML	337
11.10. Przekształcanie kodu HTML do postaci tekstu ASCII	338
11.11. Usuwanie znaczników HTML i PHP	339
11.12. Wykorzystanie szablonów Smarty	340
11.13. Analiza dziennika pracy serwera WWW	342
11.14. Program — wyszukiwanie błędnych odsyłaczy	345
11.15. Program — wyszukiwanie nowych odsyłaczy	347
Rozdział 12. XML	351
12.0. Wprowadzenie	351
12.1. Ręczne generowanie kodu XML	354
12.2. Generowanie kodu XML z użyciem rozszerzenia DOM	355
12.3. Analiza dokumentu XML z zastosowaniem rozszerzenia DOM	359
12.4. Analiza kodu XML za pomocą interfejsu SAX	362
12.5. Przekształcanie dokumentu XML za pomocą arkusza XSLT	367

12.6. Przesyłanie żądań XML-RPC	370
12.7. Odbieranie żądań XML-RPC	373
12.8. Przesyłanie żądań SOAP	377
12.9. Odbieranie żądań SOAP	380
12.10. Wymiana danych w formacie WDDX	382
12.11. Odczyt arkuszy RSS	384
Rozdział 13. Wyrażenia regularne	387
13.0. Wprowadzenie	387
13.1. Różnice pomiędzy funkcjami <code>ereg</code> i <code>preg</code>	391
13.2. Dopasowywanie wyrazów	392
13.3. Wyszukiwanie n-tego wystąpienia danej wartości	393
13.4. Obszerne i ograniczone dopasowania	395
13.5. Sprawdzanie adresów poczty elektronicznej	397
13.6. Wyszukiwanie linii pliku spełniających określone kryteria	400
13.7. Wyszukiwanie tekstu wewnątrz znaczników HTML	401
13.8. Obsługa znaków specjalnych w wyrażeniach regularnych	403
13.9. Odczytywanie rekordów rozdzielanych określonymi symbolami	404
Rozdział 14. Szyfrowanie i bezpieczeństwo połączeń	407
14.0. Wprowadzenie	407
14.1. Przechowywanie haseł w innym miejscu niż plików witryny	409
14.2. Ukrywanie danych za pomocą kodowania	410
14.3. Weryfikacja danych za pomocą skrótu	411
14.4. Przechowywanie haseł	412
14.5. Sprawdzanie złożoności hasła	413
14.6. Sposoby postępowania w przypadku utraty haseł	415
14.7. Szyfrowanie i deszyfrowanie danych	417
14.8. Zapamiętywanie zaszyfrowanych danych w pliku lub bazie danych	422
14.9. Współużytkowanie zaszyfrowanych danych z inną witryną	425
14.10. Wykrywanie połączenia SSL	427
14.11. Szyfrowanie poczty za pomocą GPG	428
Rozdział 15. Grafika	431
15.0. Wprowadzenie	431
15.1. Rysowanie linii, prostokątów i wielokątów	434
15.2. Rysowanie łuków, elips i okręgów	436
15.3. Rysowanie linii ze wzorem	438

15.4. Rysowanie tekstu.....	439
15.5. Rysowanie wyśrodkowanego tekstu.....	442
15.6. Dynamicznie generowanie obrazów	447
15.7. Pobieranie i ustawianie koloru przezroczystości	449
15.8. Bezpieczne udostępnianie obrazów	450
15.9. Program — generowanie wykresów słupkowych z wyników głosowania.....	452
Rozdział 16. Internacjonalizacja i lokalizacja tworzonych aplikacji	457
16.0. Wprowadzenie	457
16.1. Wyświetlanie nazw dostępnych stref językowych	459
16.2. Korzystanie z konkretnej strefy językowej.....	459
16.3. Ustawianie domyślnej strefy.....	460
16.4. Dostosowanie tekstów komunikatów	461
16.5. Formatowanie dat i czasu	464
16.6. Wyświetlanie walut.....	466
16.7. Dostosowywanie obrazów do potrzeb mieszkańców określonej strefy językowej	468
16.8. Lokalizacja dołączanych plików	470
16.9. Zarządzanie zasobami przeznaczonymi dla różnych stref językowych	470
16.10. Wykorzystanie rozszerzenia gettext.....	472
16.11. Odczyt i zapis znaków Unicode.....	473
Rozdział 17. Usługi internetowe	475
17.0. Wprowadzenie	475
17.1. Wysyłanie poczty elektronicznej.....	476
17.2. Wysyłanie poczty MIME	479
17.3. Odczytywanie poczty za pomocą protokołów IMAP lub POP3	481
17.4. Wysyłanie wiadomości do grup dyskusyjnych.....	484
17.5. Odczytywanie wiadomości z grup dyskusyjnych	486
17.6. Pobieranie i wysyłanie plików za pomocą protokołu FTP	491
17.7. Wyszukiwanie adresów przy użyciu serwerów LDAP	493
17.8. Wykorzystanie serwera LDAP do autoryzacji użytkowników.....	495
17.9. Przeprowadzanie sprawdzania DNS	498
17.10. Sprawdzanie, czy serwer działa	499
17.11. Pobieranie informacji o nazwie domeny	501
Rozdział 18. Pliki	503
18.0. Wprowadzenie	503
18.1. Tworzenie lub otwieranie lokalnego pliku	507

18.2. Tworzenie tymczasowego pliku	508
18.3. Zdalne otwieranie pliku	509
18.4. Odczyt ze standardowego wejścia.....	510
18.5. Odczyt plików do łańcucha znaków	510
18.6. Zliczanie wierszy, akapitów i rekordów w pliku.....	512
18.7. Przetwarzanie każdego wyrazu z pliku	514
18.8. Odczyt wybranego wiersza z pliku	516
18.9. Przetwarzanie pliku wstecz liniami lub akapitami.....	517
18.10. Pobieranie z pliku losowego wiersza	517
18.11. Przemieszczanie wszystkich wierszy w pliku	518
18.12. Przetwarzanie pól tekstowych o zmiennej długości.....	519
18.13. Odczytywanie plików konfiguracyjnych.....	520
18.14. Odczyt lub zapis z konkretnego położenia w pliku	522
18.15. Usuwanie ostatniego wiersza z pliku	523
18.16. Modyfikacja pliku bez użycia pliku tymczasowego.....	525
18.17. Opróżnianie bufora	527
18.18. Zapis na standardowe wyjście	528
18.19. Jednoczesny zapis do wielu uchwytów plików	529
18.20. Znaki specjalne powłoki.....	530
18.21. Przekazywanie wejścia do programu	531
18.22. Odczyt standardowego wyjścia z programów	532
18.23. Odczyt standardowego wyjścia błędów z programu	534
18.24. Blokowanie pliku.....	535
18.25. Odczyt i zapis skompresowanych plików.....	537
18.26. Program unzip	539
Rozdział 19. Katalogi	543
19.0. Wprowadzenie	543
19.1. Pobieranie i ustawianie czasu plików	546
19.2. Pobieranie informacji o pliku.....	547
19.3. Zmiana praw lub właściciela pliku.....	549
19.4. Podział nazwy pliku na części składowe	550
19.5. Usuwanie pliku.....	551
19.6. Kopiowanie lub przemieszczanie pliku.....	551
19.7. Przetwarzanie wszystkich plików w katalogu	552
19.8. Pobranie listy plików spełniających pewien wzorzec	554
19.9. Przetwarzanie wszystkich plików w katalogu	555

19.10. Tworzenie nowych katalogów	557
19.11. Usuwanie katalogu i jego zawartości	558
19.12. Program — wyświetlanie listy plików w katalogu jako strony WWW	559
19.13. Program — wyszukiwanie tekstu w witrynie	563
Rozdział 20. PHP po stronie klienta	567
20.0. Wprowadzenie	567
20.1. Przetwarzanie argumentów programu	571
20.2. Przetwarzanie argumentów za pomocą klasy getopt	572
20.3. Odczyt z klawiatury	576
20.4. Odczyt haseł	577
20.5. Wyświetlanie widgetu w oknie	579
20.6. Wyświetlanie kilku widgetów w jednym oknie	581
20.7. Odpowiedź na działania użytkownika	583
20.8. Wyświetlanie menu	585
20.9. Program — powłoka z wierszem poleceń	588
20.10. Program — wyświetlanie warunków pogodowych	591
Rozdział 21. Rozszerzenie PEAR	599
21.0. Wprowadzenie	599
21.1. Korzystanie z zarządcy pakietów PEAR	601
21.2. Znajdowanie pakietów PEAR	604
21.3. Znajdowanie informacji o pakiecie	605
21.4. Instalacja pakietów PEAR	606
21.5. Instalacja pakietów PECL	608
21.6. Aktualizacja pakietów PEAR	609
21.7. Usuwanie zainstalowanych pakietów PEAR	611
21.8. Dokumentowanie klas za pomocą PHPDoc	611
Skorowidz	615

5

Zmienne

5.0. Wprowadzenie

Zgodnie z logiką warunkową zmienne są tym, co czyni programy komputerowe silnymi i elastycznymi. Jeżeli potraktujemy zmienną jako opatrzone nazwą pudełko, które przechowuje jakąś wartość, możemy powiedzieć, że PHP pozwala na tworzenie dziurawych, pustych pudełek, pudełek, które zawierają nazwy innych pudełek, pudełek wypełnionych liczbami lub łańcuchami znaków, pudełek przechowujących tablice innych pudełek, pudełek wypełnionych obiektami oraz innych odmian pudełek, jakie tylko można sobie wyobrazić.

Zmienna może być zarówno ustawiona, jak i nieustawiona. Zmienna z przypisaną do niej jakąkolwiek wartością: `true` lub `false`, pustą lub niepustą, jest ustawiona. Funkcja `isset()` zwraca wartość `true`, jeżeli przekazano do niej zmienną, która jest ustawiona. Jedynym sposobem, aby zmienną ustawioną przekształcić w zmienną nieustawioną, jest wywołanie wobec niej funkcji `unset()`. Do `unset()` można przekazywać skalary, tablice i obiekty. Do funkcji tej można również przekazać jednocześnie wiele zmiennych, aby wszystkie uczynić nieustawionymi:

```
unset($vegetables);
unset($vegetables[12]);
unset($earth, $moon, $stars);
```

Jeśli zmienna jest obecna w łańcuchu zapytania URL, to będzie ona ustawiona nawet wówczas, jeśli nie przypisano do niej żadnej wartości. A zatem:

```
http://www.przyklad.com/set.php?szympansy=&malpy=12
```

spowoduje ustawienie `$_GET['malpy']` na 12, a `$_GET['szympansy']` będzie pustym łańcuchem znaków.

Wszystkie nieustawione zmienne są również puste. Zmienne ustawione mogą być puste lub niepuste. Puste zmienne mają wartość logiczną równą `false`, są to: liczba całkowita `0`, zmienna typu `double` `0.0`, pusty łańcuch znaków, łańcuch znaków postaci „0”, wartość logiczna `false`, tablica nie zawierająca żadnych elementów, obiekt bez zmiennych i metod oraz wartość `NULL`. Wszystkie inne zmienne są niepuste. Dotyczy to również łańcucha znaków „00” oraz łańcucha „ ”, który zawiera tylko znak spacji.

Zmienne mogą być odczytywane jako mające wartość albo `true`, albo `false`. Powyżej wymieniono wszystkie wartości, które przez PHP traktowane są jako równoważne wartości logicznej `false`. Każda inna wartość jest wartością `true`. Różnica pomiędzy tym, co puste, a tym, co fałszywe polega na tym, że tylko zmienne mogą być puste. Stałe oraz wartości zwracane przez funkcję mogą mieć wartość `false`, ale nie mogą być puste. Na przykład poniższa instrukcja jest prawidłowa, ponieważ `$first_name` jest zmienną:

```
if (empty($first_name)) { .. }
```

Natomiast poniższe dwa przykłady spowodują wygenerowanie błędu parsera, ponieważ `0` (stała) oraz wartość zwracana przez funkcję `get_first_name()` nie mogą być puste:

```
if (empty(0)) { .. }  
if (empty(get_first_name())) { .. }
```

5.1. Unikanie pomyłek między operatorami `==` i `=`

Problem

Podczas porównywania zmiennej i stałej chcemy uniknąć przypadkowego przypisania wartości.

Rozwiązanie

Napiszemy:

```
if (12 == $dwarves) { ... }
```

zamiast

```
if ($dwarves == 12) { ... }
```

Umieszczenie stałej po lewej stronie wywoła błąd parsera w razie zastosowania operatora przypisania. Innymi słowy, PHP wyświetli błąd, jeżeli napiszemy:

```
if (12 = $dwarves) { ... }
```

natomiast:

```
if ($dwarves = 12) { ... }
```

zostanie wykonane bez ostrzeżeń, czyli `12` zostanie przypisane do zmiennej `$dwarves`, a następnie uruchomiony zostanie kod wewnątrz bloku (ponieważ wynikiem operacji `$dwarves = 12` jest wartość `12`, traktowana jako `true`).

Analiza

Umieszczenie stałej po lewej stronie porównania wymusza przyrównanie do typu stałej. Powoduje to problemy w sytuacji, gdy liczba całkowita porównywana jest ze zmienną mogącą być liczbą całkowitą lub łańcuchem znaków. Jeśli `$dwarves` ma wartość `0`, wyrażenie `0 == $dwarves` ma wartość `true`. Jednak taką samą wartość będzie miało to wyrażenie w sytuacji, gdy `$dwarves` będzie mieć wartość `śpioch`. Jeżeli liczba całkowita (`0`) znajdzie się po lewej stronie operatora porównania, PHP przekształci przed wykonaniem operacji porównania to, co jest po prawej stronie (łańcuch znaków `śpioch`) na liczbę całkowitą (`0`). Aby tego uniknąć, należy zastosować operator tożsamości `0 === $dwarves`.

Zobacz również

Dokumentacja operatora `=` jest dostępna pod adresem <http://www.php.net/language.operators.assignment>, a dokumentacja operatorów `==` i `===` — na stronie <http://www.php.net/manual/language.operators.comparison.php>.

5.2. Ustalanie wartości domyślnej

Problem

Chcemy przypisać wartość domyślną do zmiennej, która jeszcze nie posiada żadnej wartości. Często zdarza się, że chcemy w sposób bezpośredni przypisać wartość domyślną zmiennej, która może zostać nadpisana danymi z formy wejściowej lub przez zmienną środowiskową.

Rozwiązanie

Należy zastosować funkcję `isset()` w celu przypisania wartości domyślnej zmiennej, która może już posiadać wartość:

```
if (!isset($cars)) { $cars = $default_cars; }
```

Aby nadać nowej zmiennej wartość (na przykład wartość domyślną), należy zastosować operator potrójny (`a ? b : c`):

```
$cars = isset($_REQUEST['cars']) ? $_REQUEST['cars'] : $default_cars;
```

Analiza

Stosowanie funkcji `isset()` ma zasadnicze znaczenie podczas przypisywania wartości domyślnych. Bez niej wartość nie mająca charakteru domyślnego nie może być zerem lub jakąkolwiek inną wartością, obliczaną jako `false`. Rozważmy następujące przypisanie:

```
$cars = $_REQUEST['cars'] ? $_REQUEST['cars'] : $default_cars;
```

Jeżeli `$_REQUEST['cars']` ma wartość 0, `$cars` jest ustawiona na `$default_cars` i to pomimo tego, że 0 może być prawidłową wartością zmiennej `$cars`.

Aby w łatwy sposób ustawić jednocześnie wiele wartości domyślnych, można zastosować tablicę z wartościami domyślnymi. Kluczami w tablicy wartości domyślnych są nazwy zmiennych, a wartości znajdujące się w tablicy są wartościami domyślnymi dla każdej zmiennej:

```
$defaults = array('imperatorzy' => array('Rudolf II', 'Kaligula'),
                 'warzywo'      => 'seler',
                 'hektary'      => 15);

foreach ($defaults as $k => $v) {
    if (! isset($GLOBALS[$k])) { $GLOBALS[$k] = $v; }
}
```

Ponieważ zmienne są ustawione w globalnej przestrzeni nazw, poprzedni kod nie zadziała, jeśli będziemy chcieli ustawić wartości domyślne obowiązujące jedynie wewnątrz funkcji. Aby wykonać tę operację, należy zastosować zmienne zmiennych:

```
foreach ($defaults as $k => $v) {
    if (! isset($$k)) { $$k = $v; }
}
```

Zobacz również

Dokumentacja dotycząca funkcji `isset()` jest dostępna pod adresem <http://www.php.net/isset>; zmienne zmiennych są omówione w recepturze 5.4 oraz pod adresem <http://www.php.net/language.variables.variable>.

5.3. Wymiana wartości bez używania zmiennych tymczasowych

Problem

Chcemy wymienić wartości w dwóch zmiennych bez użycia dodatkowych zmiennych do przechowywania tych wartości.

Rozwiązanie

Aby zamienić zmienne `$a` i `$b`, napiszemy:

```
list($a, $b) = array($b, $a);
```

Analiza

Konstrukcja językowa PHP o nazwie `list()` umożliwia przypisywanie wartości z tablicy do pojedynczych zmiennych. Odpowiednik `list()` po prawej stronie wyrażenia, czyli

`array()`, umożliwia tworzenie tablic z pojedynczych wartości. Przypisanie tablicy zwróconej przez funkcję `array()` do zmiennych w `list()` umożliwia zmianę kolejności tych wartości. Działa to również w przypadku więcej niż dwóch wartości:

```
list($yesterday,$today,$tomorrow) = array($today,$tomorrow,$yesterday);
```

Metoda ta nie jest szybsza niż zastosowanie zmiennych tymczasowych, dlatego należy stosować ją dla zachowania czytelności kodu, a nie zwiększenia jego szybkości.

Zobacz również

Dokumentacja funkcji `list()` jest dostępna pod adresem <http://www.php.net/list>, a funkcji `array()` — pod adresem <http://www.php.net/array>.

5.4. Tworzenie dynamicznej nazwy zmiennej

Problem

Chcemy dynamicznie utworzyć nazwę zmiennej. Na przykład chcemy zastosować nazwy zmiennych, które odpowiadają nazwom pól pochodzących z zapytania do bazy danych.

Rozwiązanie

Zastosujemy składnię zmiennej zmiennych PHP: wstawimy znak `$` przed zmienną, której wartość wyznacza nazwę zmiennej, którą chcemy utworzyć:

```
$animal = 'tygrysy';
$tygrysy = 103;
print $$animal;
103
```

Analiza

Wykonanie powyższego przykładu spowoduje wyświetlenie wartości 103. Ponieważ `$animal = 'tygrysy'`, `$$animal` ma wartość `$tygrysy`, a jej wartością jest z kolei 103.

Stosując nawiasy klamrowe, możemy utworzyć bardziej skomplikowane wyrażenia, które będą wskazywać nazwy zmiennych:

```
$stooges = array('Moe','Larry','Curly');
$stooge_moe = 'Moses Horwitz';
$stooge_larry = 'Louis Feinberg';
$stooge_curly = 'Jerome Horwitz';

foreach ($stooges as $s) {
    print "$s naprawdę nazywa się ${'stooge_'.strtolower($s)}.\n";
}
Moe naprawdę nazywa się Moses Horwitz.
Larry naprawdę nazywa się Louis Feinberg.
Curly naprawdę nazywa się Jerome Horwitz.
```

PHP oblicza wyrażenie pomiędzy nawiasami klamrowymi i stosuje je jako nazwę zmiennej. Wyrażenie może nawet zawierać wywołania funkcji, na przykład `strtolower()`.

Zmienne zmiennych są także użyteczne przy przechodzeniu kolejno przez zmienne o podobnych nazwach. Załóżmy, że wysyłamy zapytanie do tabeli bazy danych, w której znajdują się pola `title_1`, `title_2` itd. Jeżeli chcemy sprawdzić, czy tytuł odpowiada którejkolwiek z tych wartości, najprostszym rozwiązaniem jest przejście przez nie kolejno w następujący sposób:

```
for ($i = 1; $i <= $n; $i++) {  
    $t = "title_$i";  
    if ($title == $$t) { /* jeśli się równa */ }  
}
```

Oczywiście, pierwszym rozwiązaniem przychodzącym na myśl byłoby przechowywanie tych wartości w tablicy, ale jeżeli utrzymujemy stary kod, który wykorzystuje ten właśnie sposób (i nie możemy go zmienić), zmienne zmiennych okazują się wówczas pomocne.

Zastosowanie nawiasów klamrowych jest również niezbędne podczas wyjaśniania niejasności dotyczących elementów tablicy. Zmienna zmiennych `$$donkeys[12]` może mieć dwa znaczenia. Pierwsze to „odczytaj to, co znajduje się w dwunastym elemencie tablicy `$donkeys` i użyj tego jako nazwy zmiennej”. Napiszemy wówczas `$$donkeys[12]`. Drugie znaczenie to „zastosuj jako nazwę tablicy to, co znajduje się w skalarze `$donkeys` i poszukaj dwunastego elementu tej tablicy”. Napiszemy wówczas `$$donkeys}[12]`.

Zobacz również

Dokumentacja zmiennych zmiennych jest dostępna pod adresem <http://www.php.net/lan-guage.variables.variable>.

5.5. Stosowanie zmiennych statycznych

Problem

Chcemy zachować wartość zmiennej lokalnej pomiędzy kolejnymi wywołaniami funkcji.

Rozwiązanie

Zadeklarujemy zmienną jako statyczną przy użyciu słowa kluczowego `static`:

```
function track_times_called( ) {  
    static $i = 0;  
    $i++;  
    return $i;  
}
```

Analiza

Zadeklarowanie zmiennej jako `static` spowoduje, że jej wartość zostanie zapamiętana przez funkcję. Jeżeli więc funkcja zostanie wywołana w późniejszym czasie, nadal będziemy mogli uzyskać dostęp do wartości tej zapisanej zmiennej. Funkcja `pc_check_the_count()` z przykładu 5.1 wykorzystuje zmienne statyczne do zliczania liczby piłek odbitych i przepuszczonych przez baseballistę grającego na pozycji pałkarza.

Przykład 5.1. Funkcja `pc_check_the_count()`

```
function pc_check_the_count($pitch) {
    static $strikes = 0;
    static $balls  = 0;

    switch ($pitch) {
        case 'foul':
            if (2 == $strikes) break; // nic się nie dzieje, jeśli pitcher
                                     ↙nie uderzył 2 razy
            // w przeciwnym wypadku przejdź do przypadku 'strike'
        case 'strike':
            $strikes++;
            break;
        case 'ball':
            $balls++;
            break;
    }

    if (3 == $strikes) {
        $strikes = $balls = 0;
        return 'wyliminowany';
    }
    if (4 == $balls) {
        $strikes = $balls = 0;
        return 'biegnie';
    }
    return 'uderza';
}

$what_happened = check_the_count($pitch);
```

W funkcji `pc_check_the_count()` to, co zdarzy się z pałkarzem, zależy od wartości licznika uderzeń `$pitch` znajdującego się wewnątrz funkcji w instrukcji `switch`. Moglibyśmy zamiast tego zwracać liczby piłek odbitych i przepuszczonych, lecz wymagałoby to także sprawdzania w różnych miejscach kodu, czy zawodnik powinien zostać wyliminowany, czy ma biec do bazy, czy też pozostać na miejscu.

Zmienne statyczne zachowują swoje wartości pomiędzy kolejnymi wywołaniami funkcji, jednak wartość ta jest zapamiętywana tylko na czas pojedynczego wywołania skryptu. Zmienna zadeklarowana jako `static` dostępna w czasie jednego wywołania nie utrzyma swojej wartości do chwili kolejnego wywołania tej samej strony.

Zobacz również

Dokumentacja zmiennych statycznych jest dostępna pod adresem <http://www.php.net/lan-guage.variables.scope>.

5.6. Współdzielenie zmiennych pomiędzy procesami

Problem

Chcemy poznać sposób współdzielenia informacji pomiędzy procesami, aby zapewnić szybki dostęp do współdzielenia danych.

Rozwiązanie

Zapiszemy dane w segmencie pamięci wspólnej i zagwarantujemy wyłączny dostęp do niego za pomocą semafora:

```
$semaphore_id = 100;
$segment_id   = 200;
// pobiera uchwyt do semafora związanego z pożądanym
// segmentem pamięci wspólnej
$sem = sem_get($semaphore_id,1,0600);
// zapewnia sobie wyłączny dostęp do semafora
sem_acquire($sem) or die("Nie mogę przejąć semafora ");
// pobiera uchwyt do pożądanego segmentu pamięci wspólnej
$shm = shm_attach($segment_id,16384,0600);
// pozyskuje wartość z segmentu pamięci wspólnej
$population = shm_get_var($shm,'population');
// operuje na wartości
$population += ($births + $immigrants - $deaths - $emigrants);
// z powrotem zapisuje wartość w segmencie pamięci wspólnej
shm_put_var($shm,'population',$population);
// zwalnia uchwyt do segmentu pamięci wspólnej
shm_detach($shm);
// zwalnia semafor, by inne procesy mogły go pozyskiwać
sem_release($sem);
```

Analiza

Segment pamięci wspólnej jest częścią pamięci RAM w komputerze, do której dostęp mogą mieć różne procesy (na przykład większa liczba procesów serwera WWW, który obsługuje zapytania). Semafor sam zapewnia, że inne procesy nie zachodzą na siebie w momencie, gdy jeden z nich ma dostęp do segmentu pamięci wspólnej. Przed wykorzystaniem przez proces segmentu pamięci musi on przejąć sterowanie semaforem. Jeśli proces został już wykonany, semafor jest zwalniany po to, by umożliwić wykorzystanie go przez inny proces.

Aby przejąć sterowanie semaforem, należy zastosować funkcję `sem_get()`, która znajduje jego identyfikator. Pierwszym argumentem tej funkcji jest liczba całkowita będąca kluczem semafora. Kluczem może być jakakolwiek liczba całkowita pod warunkiem, że wszystkie programy, które będą miały dostęp do tego konkretnego semafora, stosują ten sam klucz. Jeżeli semafor z wyszczególnionym kluczem jeszcze nie istnieje, zostaje on utworzony, ustawiana jest maksymalna liczba procesów, które mogą mieć dostęp do semafora, wskazywana przez drugi argument funkcji `sem_get()` (w tym przypadku jest to 1),

a następnie ustawiane są uprawnienia semaforów wskazywane przez trzeci argument funkcji (0600). Uprawnienia te działają tak jak uprawnienia plików, tak więc 0600 oznacza, że użytkownik, który utworzył semafor, może go odczytywać oraz zapisywać do niego. Mówiąc o „użytkowniku”, mamy na myśli nie tylko proces, który utworzył semafor, ale również każdy inny proces z tym samym identyfikatorem użytkownika. Uprawnienia o wartości 0600 powinny być wystarczające dla większości zastosowań, w których procesy serwera WWW działają jako procesy tego samego użytkownika.

Funkcja `sem_get()` zwraca identyfikator, który wskazuje na odpowiedni semafor systemu. Identyfikator ten należy zastosować do uzyskania kontroli nad semaforem za pomocą funkcji `sem_acquire()`. Funkcja ta czeka aż do momentu, w którym semafor będzie mógł zostać przejęty (na przykład wtedy, gdy inne procesy zwolnią semafor), po czym zwróci wartość `true`. W przypadku błędu zwrócona zostanie wartość `false`. Błędy mogą wynikać z nieprawidłowego określenia uprawnień, lub też mogą powstać, gdy ilość pamięci potrzebna do utworzenia semafora jest zbyt mała. Gdy tylko semafor zostanie przejęty, można odczytać zawartość segmentu pamięci wspólnej.

Najpierw za pomocą funkcji `shm_attach()` należy ustawić łącze do konkretnego segmentu pamięci wspólnej. Tak jak w przypadku funkcji `sem_get()`, pierwszym argumentem `shm_attach()` jest również klucz w postaci liczby całkowitej. Tym razem jednak identyfikuje on pożądaną segment, a nie semafor. Jeżeli segment o wskazanym kluczu nie istnieje, zostanie on utworzony przy zastosowaniu pozostałych argumentów. Drugim argumentem (16384) jest rozmiar segmentu w bajtach, natomiast ostatni argument (0600) określa uprawnienia dostępu do segmentu. Wywołanie `shm_attach(200,16384,0600)` spowoduje utworzenie segmentu pamięci wspólnej o pojemności 16 kilobajtów, z którego odczyt i do którego zapis będzie mógł wykonywać tylko ten użytkownik, który go utworzył. Funkcja zwraca identyfikator potrzebny do odczytu i zapisu do segmentu pamięci wspólnej.

Po połączeniu z segmentem można z niego pobierać zmienne za pomocą funkcji `shm_get_var($shm, 'population')`. Funkcja ta przeszukuje segment pamięci wspólnej identyfikowany przez `$shm` i pobiera wartość zmiennej noszącej nazwę `population`. W pamięci wspólnej można przechowywać zmienne dowolnego typu. Gdy tylko zmienna zostanie pobrana, można na niej wykonywać operacje tak jak na innych zmiennych. Wywołanie funkcji `shm_put_var($shm, 'population', $population)` spowoduje umieszczenie wartości zmiennej `$population` z powrotem w segmencie pamięci wspólnej jako zmiennej o nazwie `population`.

Zakończyliśmy już pracę z pamięcią wspólną. Należy się teraz z nią rozłączyć za pomocą funkcji `shm_detach()` i zwolnić semafor za pomocą funkcji `sem_release()`, aby mogły z niego korzystać inne procesy.

Główną zaletą pamięci wspólnej jest to, że działa ona bardzo szybko. Ponieważ jednak znajduje się ona w pamięci RAM, nie może pomieścić zbyt dużo danych, a poza tym dane te zostaną one utracone, jeżeli komputer zostanie zrestartowany (chyba że podjęte zostaną specjalne czynności zmierzające do zapisania informacji z pamięci wspólnej na dysk przed wyłączeniem komputera oraz ich ponownego załadowania do pamięci podczas uruchamiania). Poza tym pamięć wspólna nie jest dostępna w systemie Windows.

Zobacz również

Receptura 8.27 zawiera program, który korzysta z pamięci wspólnej; dokumentacja pamięci wspólnej oraz funkcji semafora jest dostępna pod adresem <http://www.php.net/sem>.

5.7. Enkapsulacja złożonych typów danych do postaci łańcucha znaków

Problem

Chcemy utworzyć łańcuch znaków reprezentujący tablicę lub obiekt w celu przechowywania go w pliku lub bazie danych. Łańcuch ten powinien umożliwiać łatwe przywrócenie tablicy lub obiektu w pierwotnej postaci.

Rozwiązanie

Zastosujemy funkcję `serialize()`, aby zakodować zmienne i ich wartości do postaci tekstowej:

```
$pantry = array('cukier' => '2 kg.', 'masło' => '3 kostki');
$fp = fopen('/tmp/spizarnia', 'w') or die ("Nie można otworzyć spizarni");
fputs($fp, serialize($pantry));
fclose($fp);
```

Aby odtworzyć zmienne, zastosujemy funkcję `unserialize()`:

```
$new_pantry = unserialize(join(' ', file('/tmp/spizarnia')));
```

Analiza

Powstały po serializacji łańcuch znaków, który zostanie odtworzony jako zmienna `$pantry`, ma następującą postać:

```
a:2:{s:6:"cukier";s:5:"2 kg.";s:5:"masło";s:8:"3 kostki";}
```

Łańcuch ten przechowuje wystarczającą ilość informacji, aby przywrócić wszystkie wartości znajdujące się w tablicy, jednak sama nazwa zmiennej nie jest przechowywana w reprezentacji powstałej w wyniku serializacji.

Podczas przesyłania serializowanych danych w ramach adresu URL z jednej strony WWW na inną należy wywołać względem tych danych funkcję `urlencode()`, aby upewnić się, że znajdujące się w nich metaznaki URL zostaną właściwie obsłużone:

```
$shopping_cart = array('Bagietka z makiem' => 2,
                      'Zwykła bagietka' => 1,
                      'Lody' => 4);
print '<a href="next.php?cart="'.urlencode(serialize($shopping_cart)).
">Dalej</a>';
```

Ustawienia konfiguracyjne `magic_quotes_gpc` oraz `magic_quotes_runtime` mają wpływ na dane przekazywane do funkcji `unserialize()`. Jeżeli `magic_quotes_gpc` ma wartość `on`, dane przekazywane w adresach URL, zmienne `POST` lub `cookies` muszą przed deserializacją zostać poddane działaniu funkcji `stripslashes()`:

```
$new_cart = unserialize(stripslashes($cart)); // jeśli magic_quotes_gpc
                                                ↳ ma wartość on
$new_cart = unserialize($cart);             // jeśli magic_quotes_gpc
                                                ↳ ma wartość off
```

Jeżeli ustawienie `magic_quotes_runtime` ma wartość `on`, serializowane dane przechowywane w pliku muszą być podczas zapisu poddane działaniu funkcji `addslashes()`, a podczas odczytu — funkcji `stripslashes()`:

```
$fp = fopen('/tmp/cart','w');
fputs($fp,addslashes(serialize($a)));
fclose($fp);

// jeśli magic_quotes_runtime ma wartość on
$new_cart = unserialize(stripslashes(join(',',file('/tmp/cart'))));
// jeśli magic_quotes_runtime ma wartość off
$new_cart = unserialize(join(',',file('/tmp/cart')));
```

Jeżeli `magic_quotes_runtime()` ma wartość `on`, wówczas również serializowane dane odczytywane z bazy danych muszą zostać poddane działaniu funkcji `stripslashes()`:

```
mysql_query(
    "INSERT INTO cart (id,data) VALUES (1,'".addslashes(serialize($cart))."')");

$r = mysql_query('SELECT data FROM cart WHERE id = 1');
$ob = mysql_fetch_object($r);
// jeśli magic_quotes_runtime ma wartość on
$new_cart = unserialize(stripslashes($ob->data));
// jeśli magic_quotes_runtime ma wartość off
$new_cart = unserialize($ob->data);
```

Serializowane dane przesyłane do bazy danych zawsze muszą zostać poddane działaniu funkcji `addslashes()` (lub innej odpowiadającej jej funkcji właściwej dla bazy danych), aby zapewnić, że zostaną one prawidłowo zapisane.

Zobacz również

Receptura 10.7 zawiera informacje dotyczące obsługi danych przesyłanych do bazy danych.

5.8. Wyświetlanie zawartości zmiennej w postaci łańcuchów znaków

Problem

Chcemy kontrolować wartości przechowywane w zmiennej. Może ona być skomplikowaną tablicą zagnieżdżoną lub obiektem, więc nie można tak po prostu jej wyświetlić ani przechodzić kolejno przez jej elementy.

Rozwiązanie

Zastosujemy funkcję `print_r()` lub `var_dump()`:

```
$array = array("name" => "Franek", 12, array(3, 4));

print_r($array);
Array
(
    [name] => Franek
    [0] => 12
    [1] => Array
        (
            [0] => 3
            [1] => 4
        )
)

var_dump($array);
array(3) {
    ["name"]=>
    string(6) "Franek"
    [0]=>
    int(12)
    [1]=>
    array(2) {
        [0]=>
        int(3)
        [1]=>
        int(4)
    }
}
```

Analiza

Dane wyświetlane przez funkcję `print_r()` są bardziej zwarte i łatwiejsze do odczytania. Natomiast dane wyświetlane przez funkcję `var_dump()` zawierają typ danych oraz długość każdej zmiennej.

Funkcje te przechodzą przez kolejne zmienne w sposób rekurencyjny, dlatego jeżeli w zmiennej znajduje się odwołanie wskazujące na nią samą, może to doprowadzić do powstania pętli nieskończonej. Na szczęście obie funkcje zawsze wstrzymują się przed wyświetlaniem bez końca informacji o zmiennych. Jeżeli funkcja `print_r()` napotka zmienną odczytaną już wcześniej, wyświetli łańcuch `*RECURSION*`, zamiast ponownie wyświetlać informacje o zmiennej i kontynuować przechodzenie kolejno przez resztę informacji, które zostały jeszcze do wyświetlenia. Jeżeli funkcja `var_dump()` napotka tę samą zmienną więcej niż trzy razy, wygeneruje błąd krytyczny i zakończy działanie skryptu. Rozważmy tablice `$user_1` i `$user_2`, które odwołują się nawzajem do siebie przez elementy `friend`:

```
$user_1 = array('name' => 'Maks Białystok',
               'username' => 'maks');

$user_2 = array('name' => 'Leon Blum',
               'username' => 'leon');
```

```
// Maks i Leon są przyjaciółmi
$user_2['friend'] = &$user_1;
$user_1['friend'] = &$user_2;

// Maks i Leon mają pracę
$user_1['job'] = 'Kanciarz';
$user_2['job'] = 'Księgowy';
```

Wywołanie funkcji `print_r($user_2)` spowoduje wyświetlenie następujących danych:

```
Array
(
    [name] => Leon Blum
    [username] => leon
    [friend] => Array
        (
            [name] => Maks Białystok
            [username] => maks
            [friend] => Array
                (
                    [name] => Leon Blum
                    [username] => leon
                    [friend] => Array
                        (
                            *RECURSION*
                        )
                    [job] => Księgowy
                )
            [job] => Kanciarz
        )
    [job] => Księgowy
)
```

Jeżeli funkcja `print_r()` po raz drugi napotka odwołanie do `$user_1`, wówczas wyświetli łańcuch znaków `*RECURSION*` i nie zejdzie już dalej do tablicy. Następnie funkcja kontuuje swoje działanie, wyświetlając pozostałe elementy tablic `$user_1` i `$user_2`.

W razie napotkania rekurencji funkcja `var_dump()` zachowuje się w inny sposób:

```
array(4) {
  ["name"]=>
  string(9) "Leon Blum"
  ["username"]=>
  string(4) "leon"
  ["friend"]=>
  &array(4) {
    ["name"]=>
    string(14) "Maks Białystok"
    ["username"]=>
    string(4) "maks"
    ["friend"]=>
    &array(4) {
      ["name"]=>
      string(9) "Leon Blum"
      ["username"]=>
      string(4) "leon"
      ["friend"]=>
      &array(4) {
        ["name"]=>
        string(14) "Maks Białystok"
        ["username"]=>
```

```

string(4) "maks"
["friend"]=>
&array(4) {
    ["name"]=>
    string(9) "Leon Blum"
    ["username"]=>
    string(4) "leon"
    ["friend"]=>
    &array(4) {
        ["name"]=>
        string(14) "Maks Białystok"
        ["username"]=>
        string(4) "maks"
        ["friend"]=>
        &array(4) {
            ["name"]=>
            string(9) "Leon Blum"
            ["username"]=>
            string(4) "leon"
            ["friend"]=>
            &array(4) {

```


Fatal error: Nesting level too deep - recursive dependency?
in <code>var-dump.php</code> on line <code>15</code>

Zatrzymanie rekurencji nastąpi dopiero wówczas, gdy funkcja `var_dump()` napotka odwołanie do `$user_1` po raz czwarty. Gdy to nastąpi, funkcja wygeneruje błąd krytyczny i żadne inne wartości zmiennych nie zostaną wyświetlone ani nie zostanie wykonana żadna inna część skryptu.

Pomimo tego, że funkcje `print_r()` i `var_dump()` wyświetlają wyniki swego działania, a nie zwracają ich jako wartości zwracanej, dane te i tak można przechwycić, unikając ich drukowania dzięki zastosowaniu buforowania danych wyjściowych:

```

ob_start();
var_dump($user);
$dump = ob_get_contents();
ob_end_clean();

```

Wynik działania funkcji `var_dump($user)` zostanie umieszczony w zmiennej `$dump`.

Zobacz również

Receptura 8.12 opisuje buforowanie danych wyjściowych; receptura 10.8 przedstawia sposób obsługi błędów w module *PEAR DB* polegający na zastosowaniu buforowania danych wyjściowych funkcji `print_r()` w celu zachowania komunikatów błędów; dokumentacja funkcji `print_r()` jest dostępna pod adresem <http://www.php.net/print-r>, a dokumentacja funkcji `var_dump()` — pod adresem <http://www.php.net/var-dump>.