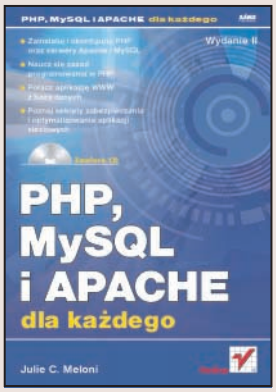
PHP, MySQL i Apache dla każdego. Wydanie II



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 wystarczał 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

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;
$dluga_nazwa_zmiennej;
$a2453;
$spiacyZZZZ;
```

Na marginesie

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 haśle, 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 załóżmy, 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.

Na marginesie | 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 `$test` w linii 2., nie przypisaliśmy jej żadnej wartości, a więc pierwsze wywołanie funkcji `gettype()` w linii 3. zwraca wartość `NULL`. Następnie, za pomocą operatora przypisania (=), nadajemy zmiennej kolejne wartości, wyświetlając rezultat funkcji `gettype()`. W linii 5. zmiennej została przypisana wartość całkowita, mówiąc krótko, liczba bez wartości dziesiętnej[1]. W linii 8. zmiennej przypisujemy ciąg znaków, który zawsze powinien być otoczony pojedynczymi lub podwójnymi znakami cudzysłowu (' lub "). Wartość `double`, przypisana zmiennej `$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.

Na marginesie

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`.

Na marginesie

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 konkatenacji

Operator konkatenacji 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 konkatenacji 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. Operatora konkatenacji 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 konkatenacji, 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
+=	$x += 5	$x = $x + 5
-=	$x -= 5	$x = $x - 5
/=	$x /= 5	$x = $x / 5
*=	$x *= 5	$x = $x * 5
%=	$x %= 5	$x = $x % 5
.=	$x .= " test"	$x = $x." test"

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 konkatenacji 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 ($x = 4)	Rezultat
==	równe	lewa strona jest równa prawej	$x == 5	false
!=	różne	lewa strona nie jest równa prawej	$x != 5	true
===	identyczne	lewa strona jest równa prawej i są tego samego typu	$x === 4	true
!==	nie identyczne	lewa strona jest różna prawej lub są różnego typu	$x !== "4"	false
>	większe	lewa strona jest większa niż prawa	$x > 4	false
>=	większe lub równe	lewa strona jest większa lub równa prawej	$x >= 4	true
<	mniejsze	lewa strona jest mniejsza niż prawa	$x < 4	false
<=	mniejsze lub równe	lewa strona jest mniejsza lub równa prawej	$x <= 4	true

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	true || false	true
or	alternatywa (lub)	lewa lub prawa strona jest prawdziwa	true or false	true
xor	różnica symetryczna (xor)	lewa lub prawa strona jest prawdziwa, ale nie obie	true xor true	false
&&	koniunkcja (i)	lewa i prawa strona jest prawdziwa	true && false	false
and	koniunkcja (i)	lewa i prawa strona jest prawdziwa	true and false	false
!	negacja (nie)	pojedynczy operand nie jest prawdziwy	! true	false

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 wyrażeń 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 niezmieniona, 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 konkatenacji, 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_uzytkownika
$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 `$666666xyz` nie jest poprawna, gdyż nie zaczyna się literą lub znakiem podkreślenia. Nazwa zmiennej `$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.