PHP to dobry, łatwy i wygodny w zastosowaniu język programowania. W wersji 5. wyposażony został w obiektowość z prawdziwego zdarzenia, stąd możliwe jest pisanie w pełni obiektowych aplikacji WWW. W celu zachowania wstecznej kompatybilności lub też umożliwienia programistom wyboru stylu programowania nadal możliwe jest tworzenie aplikacji WWW w sposób tradycyjny (liniowy z zastosowaniem funkcji).

Rozdział ten wprowadzi Cię w arkana sztuki programowania w PHP. Dowiesz się, jaka jest struktura skryptu PHP (czyli strony zawierającej kod PHP), w jaki sposób można wyświetlać dane w oknie przeglądarki oraz jak stosować zmienne, stałe, instrukcje sterujące wykonaniem programu, pętle oraz inne elementy języka PHP. Rozdział ten nie ma na celu zanudzenia czytelnika mnóstwem teorii — głównym celem jest szybkie i bezbolesne zapoznanie go z językiem programowania PHP.

Struktura skryptu

Skrypt PHP to nic innego, jak tylko normalna strona WWW, z tą jednak różnicą, że zawierająca kod PHP. Kod ten umieszczony jest w specjalnych sekcjach ograniczonych znacznikami <?php i ?> — pierwszy z nich otwiera sekcję kodu PHP, drugi zamyka sekcję. Istnieje też możliwość użycia znaczników krótkich — wtedy znacznik otwierający sekcję kodu nie zawiera ciągu php. To, czy można używać znaczników krótkich, czy też nawet znaczników rodem z ASP lub JSP, zależy od konfiguracji PHP. Zaleca się jednak używanie znaczników normalnych. Listing 4.1 przedstawia przykładowy skrypt PHP.

Listing 4.1. Przykładowy skrypt PHP

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN"

  "http://www.w3.org/TR/html4/strict.dtd">

<html lang="pl">

<head>

  <meta http-equiv="Content-Type" content="text/html; charset=utf-8">

  <title>Powitanie!!!</title>

</head>

<body>

  <h1>Powitanie</h1>

  <p>

  <?php

    echo "Witaj w świecie PHP!";

  ?>

  </p>
```

```
</body>

</html>
```

Listing 4.1 przedstawia kod strony WWW zawierający tylko jedną sekcję PHP. Większa część kodu to jednak standardowe znaczniki HTML tworzące strukturę najprostszej (poprawnej) strony zgodnej ze standardem HTML 4. Jak już zostało to wspomniane wcześniej, nie będziemy omawiać typowych konstrukcji HTML-a. Przypomnijmy tylko, że składa się on z deklaracji typu dokumentu (<!DOCTYPE>), nagłówka (<head>) oraz treści (<body>). W sekcji <head> określany jest sposób kodowania znaków (w naszym przypadku jest to standard UTF-8) oraz tytuł strony (znacznik <title>).

Ilość sekcji PHP nie jest ograniczona, mogą one być umieszczane w dowolnym miejscu strony. Sekcja kodu PHP z listingu 4.1 składa się tylko z jednej instrukcji echo. Instrukcja ta ma za zadanie wyświetlenie ciągu znaków umieszczonego w cudzysłowie (lub w apostrofach), a następującego po tej instrukcji. W języku PHP każdą instrukcję kończymy średnikiem. Zamiast instrukcji echo możemy użyć też print — to kwestia przyzwyczajenia i nawyków programisty, ponieważ w podstawowym zastosowaniu instrukcje te niczym się od siebie nie różnią. Zauważmy, że napis umieszczony po instrukcji echo wyświetlony zostanie dokładnie w tym miejscu strony, w którym umieszczony został w kodzie — pokazuje to rysunek 4.1.

Rysunek 4.1. Strona WWW wygenerowana na podstawie kodu z listingu 4.1

W jaki sposób uzyskać efekt jak na rysunku 4.1? Otóż kod z listingu 4.1 należy zapisać w pliku z rozszerzeniem php (pliki z tym rozszerzeniem serwer WWW przekazywał będzie interpreterowi PHP — tak jak to skonfigurowaliśmy podczas instalacji PHP) i umieścić w folderze, którego zawartość udostępniona jest przez serwer WWW.

Obejrzyjmy dodatkowo, jak wygląda kod źródłowy tak wygenerowanej strony. W większości przeglądarek należy w tym celu wybrać z menu Widok (ang. View) pozycję Źródło (ang. Source) lub Źródło strony (ang. Page source). Sposób wyświetlania kodu źródłowego strony zależy od rodzaju przeglądarki, jednak treść strony będzie taka jak przedstawiono to na listingu 4.2.

Listing 4.2. Kod strony po przetworzeniu przez PHP

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN"

  "http://www.w3.org/TR/html4/strict.dtd">

<html lang="pl">

<head>

  <meta http-equiv="Content-Type" content="text/html; charset=utf-8">

  <title>Powitanie!!!</title>

</head>

<body>

  <h1>Powitanie</h1>

  <p>

  Witaj w świecie PHP!  </p>



</body>

</html>
```

Zniknęły wszelkie znaczniki PHP i pozostał jedynie czysty kod HTML. Taka jest bowiem istota działania języków skryptowych pracujących po stronie serwera. Zadaniem skryptu PHP jest wygenerowanie takiego kodu, który będzie mógł być zrozumiały dla przeglądarki. W naszym przypadku w miejsce skryptu zawartego pomiędzy znacznikami <?php i ?> została wstawiona wytworzona przez niego treść, czyli napis Witaj w świecie PHP!.

Kod z listingu 4.1 wyświetla zwykły, niesformatowany tekst. W PHP istnieje możliwość wyświetlania również tekstu sformatowanego, czyli zawierającego znaczniki HTML. Przykład z listingu 4.3 jest modyfikacją przykładu z listingu 4.1 polegającą na wprowadzeniu do wyświetlanego tekstu definicji koloru.

Listing 4.3. Zastosowanie formatowania tekstu w wyświetlanym napisie

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN"

  "http://www.w3.org/TR/html4/strict.dtd">
```

```
<html lang="pl">

<head>

  <meta http-equiv="Content-Type" content="text/html; charset=utf-8">

  <title>Powitanie!!!</title>

</head>

<body>

  <h1>Powitanie</h1>

  <p>

  <?php

   // Teraz formatowanie

   echo '<span style="color:red">Witaj w świecie PHP!</span>';

  ?>

  </p>

</body>

</html>
```

Jako że definicja koloru w atrybucie style znacznika span musi być ujęta w cudzysłów, tekst do wyświetlenia przez instrukcję echo umieszczony został w apostrofach — efekt końcowy jest taki sam, jak przy zastosowaniu cudzysłowu.

Stronę WWW wygenerowaną w oparciu o kod z listingu 4.3 przedstawia rysunek 4.2.

Rysunek 4.2. Wyświetlanie sformatowanego tekstu

W kodzie z listingu 4.3 powyżej instrukcji echo umieszczony został wiersz rozpoczynający się podwójnym znakiem ukośnika (ang. slash) — jest to komentarz. Komentarze stanowią nieodłączną część każdego kodu źródłowego, zawierają ważne informacje, przydatne np. podczas poprawiania i rozwoju oprogramowania, i mają na celu uratowanie twórcy programu przed zagubieniem w stworzonym jakiś czas temu i prawie już zapomnianym kodzie. W języku PHP istnieje kilka możliwości umieszczania komentarzy. Ten pokazany w przykładzie pochodzi z języka C++ i rozpoczyna się podwójnym znakiem ukośnika — wszystkie elementy po tych znakach są ignorowane przez interpreter — a jego zasięg ograniczony jest do końca wiersza.

Oprócz tego można również stosować komentarze blokowe oraz jednowierszowe uniksowe. Komentarz blokowy zaczyna się od sekwencji znaków /*, a kończy sekwencją */. Wszystko to, co znajduje się pomiędzy tymi znakami, zostanie zignorowane przez analizator składniowy PHP. Przykład poprawnego zastosowania wyglądać może następująco:

```
<?php

/*W tym miejscu wyświetlamy napis powitalny*/

echo ("<h2> Witaj w świecie PHP!</h2>");

/*Koniec kodu PHP*/

?>
```

Po przetworzeniu przez PHP takiego kodu do przeglądarki zostanie wysłana jedynie linia zawierająca ciąg znaków:

```
<h2> Witaj w świecie PHP!</h2>
```

Komentarzy tego typu nie wolno zagnieżdżać, czyli we wnętrzu jednego komentarza blokowego nie wolno umieszczać kolejnego.

Komentarz jednowierszowy typu uniksowego ma takie samo działanie, jak komentarz jednowierszowy omówiony wyżej, jedynie jego wygląd jest inny. Zaczyna się on od znaku # i obowiązuje do końca danej linii skryptu. Przykład wykorzystania tego typu komentarza wygląda następująco:

```
<?php

# W tym miejscu wyświetlamy napis powitalny

echo ("<h2> Witaj w świecie PHP!</h2>");

# Koniec kodu PHP

?>
```

Komentarz jednowierszowy uniksowy może być również zagnieżdżony wewnątrz blokowego.

Każdy plik z rozszerzeniem php w momencie odwołania się do niego przez przeglądarkę WWW przekazywany jest przez serwer WWW interpreterowi PHP, dlatego też nie jest wskazane, aby rozszerzenie php dodawać do

plików nie zawierających kodu PHP, ponieważ może to spowodować spowolnienie w serwowaniu strony — interpreter będzie niepotrzebnie angażowany.

Zwykle sekcje kodu PHP są bardziej rozbudowane — składają się z kilku, kilkudziesięciu lub nawet kilkuset linii kodu. Jeżeli chcemy użyć krótkiej sekcji (a dokładnie sekcji jednowierszowej — jak w opisywanych przykładach), możemy posłużyć się tzw. sekcją wyrażenia. Sekcja wyrażenia, znana np. z JSP, służy do wstawiania w kod (X)HTML ciągów znaków wygenerowanych za pomocą np. funkcji lub wyrażenia. Listing 4.4 zawiera przykład zastosowania sekcji wyrażenia.

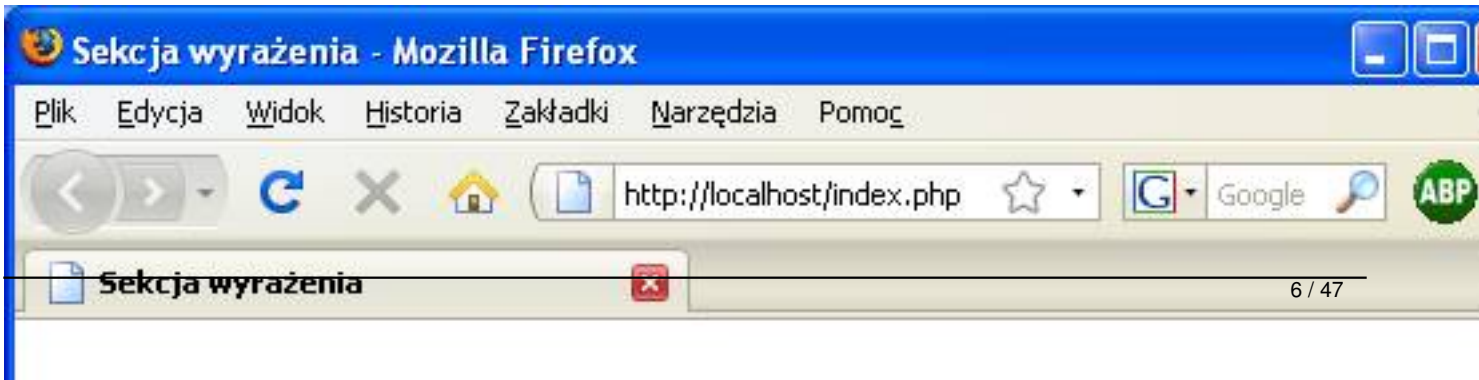
Listing 4.4. Przykład zastosowania sekcji wyrażenia

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN"

  "http://www.w3.org/TR/html4/strict.dtd">

<html lang="pl">

<head>

  <meta http-equiv="Content-Type" content="text/html; charset=utf-8">

  <title>Powitanie!!!</title>

</head>

<body>

  <h1>Powitanie</h1>

  <p>

    W basenie jest <?= (125 * 10) - 3; ?> litrów wody...

  </p>

</body>

</html>
```

Aby wyświetlić wynik wyrażenia, nie trzeba używać instrukcji echo ani print, należy jednak pamiętać o składni sekcji wyrażenia — dokładnie tak, jak na listingu 4.4.

Efektem, który uzyskamy, będzie wstawienie wartości obliczonej na podstawie wyrażenia arytmetycznego (125 * 10) – 3, czyli 1247, w miejsce określone w kodzie HTML. Zastosowanie sekcji wyrażeń może w niektórych przypadkach znacznie uprościć kod strony. Pamiętać należy, że sekcje wyrażeń można stosować, gdy konfiguracja PHP dopuszcza użycie krótkich znaczników ograniczających kod PHP (opcja ta jest domyślnie wyłączona, sposób jej włączenia zostanie opisany w kolejnym podpunkcie).

Rysunek 4.3 przedstawia efekt zastosowania sekcji wyrażenia.



Rysunek 4.3. Wynik zastosowania sekcji wyrażenia

Kod generujący pojedynczą stronę WWW nie musi być w całości umieszczany w jednym pliku (skrypcie) — czasem wręcz wskazane jest, aby zastosować rozbicie skryptu na kilka plików, tym bardziej że często różne strony wchodzące w skład jednej aplikacji wykonują te same operacje. Gdyby nie stosować w takim przypadku modularyzacji kodu aplikacji (podziału na moduły — tutaj pliki z tzw. wspólnym kodem), trzeba by było w każdym skrypcie generującym pojedynczą stronę przepisać ten sam kod. Pomijając fakt, że pliki niepotrzebnie zwiększyłyby swoją objętość, nie byłoby dobrze, gdyby się nagle okazało, że kod, który tak misternie i z takim mozołem wstawiany był do 25 skryptów, zawiera błąd… W przypadku zastosowania modularyzacji kod ten byłby umieszczony w jednym pliku, który byłby włączany do pozostałych 25 plików i poprawienie ewentualnego błędu w tym skrypcie spowodowałoby poprawne działanie owych 25 przykładowych skryptów.

Istnieje kilka sposobów dołączania plików z kodem PHP — opisano je w dalszej części kursu.

Język PHP jest bardzo elastyczny, również pod względem składni — dopuszcza on tzw. składnię alternatywną w przypadku niektórych instrukcji. W tym kursie postawiono na stosowanie składni najbardziej popularnej, zbliżonej do języków takich jak C, C++ czy Java. Więcej informacji na temat składni alternatywnej czytelnik znajdzie w dokumentacji PHP na stronie http://www.php.net.

Znaczniki PHP

Wiemy już, że treść skryptu PHP musi być oddzielona od kodu HTML specjalnymi znacznikami. Dokładniej rzecz ujmując, każdy fragment kodu PHP (nawet jeśli nie ma wokół niego kodu HTML) musi być ujęty w znaczniki, tak aby aparat wykonawczy PHP wiedział, że dana sekcja to instrukcje PHP, a nie inne dane. Podstawowe znaczniki już poznaliśmy, tak naprawdę jest ich jednak nieco więcej. W sumie możemy wyróżnić cztery różne typy:

- znaczniki kanoniczne,
- znaczniki typu SGML,
- znaczniki typu ASP,
- znaczniki skryptów HTML.

Znaczniki kanoniczne

Znaczniki kanoniczne to standardowe znaczniki PHP, które poznaliśmy już wcześniej. Znacznik otwierający to <?php, natomiast zamykający to ?>. Schematyczna konstrukcja wykorzystująca ten typ ma zatem następującą postać:

```
<?php

  //tutaj kod skryptu

?>
```

Znaczniki kanoniczne są rozpoznawane zawsze, niezależnie od tego, jakie opcje są włączone w pliku konfiguracyjnym php.ini. Jest to również zalecany sposób umieszczania skryptów w kodzie HTML i właśnie ten

sposób będzie wykorzystywany w przykładach w dalszej części kursu. **Znaczniki typu SGML**

Znaczniki typu SGML to znaczniki w postaci skróconej. Znacznik otwierający to <?, zamykający — ?>. Schematyczna konstrukcja wykorzystująca ten typ będzie miała postać:

```
<?

  //tutaj kod skryptu

?>
```

Jest to najkrótsza forma znaczników bloku PHP, jakie można zastosować. Aby jednak móc korzystać z tego sposobu, należy włączyć rozpoznawanie tych znaczników, co można zrobić, wykorzystując jeden z przedstawionych niżej sposobów:

1.

W przypadku PHP w wersji 3 można wywołać funkcję short_tags().

2.

Można włączyć opcję enable-short-tags przed kompilacją pakietu (./configure --enable-short-tags).

3.

W pliku konfiguracyjnym php.ini można umieścić linię short_open_tag = On.

Wykorzystywanie tego typu znaczników, choć możliwe, nie jest jednak zalecane. **Znaczniki typu ASP**

Znaczniki typu ASP są znane użytkownikom technologii ASP. Znacznik otwierający to <%, zamykający — %>. Schematyczna konstrukcja wykorzystująca ten typ będzie więc miała postać:

```
<%

  //tutaj kod skryptu

%>
```

Aby móc korzystać z tego typu wyróżnienia bloków PHP, należy w pliku konfiguracyjnym włączyć opcję asp_tags = On. **Znaczniki skryptów HTML**

Ta postać jest dobrze znana użytkownikom (X)HTML. Jest to typowy znacznik <script> z parametrem language ustawionym na wartość php oraz (o ile kod ma być zgodny ze standardem HTML 4) parametrem type ustawionym na text/php. Znacznik otwierający będzie miał zatem postać <script language="php" type="text/php">, natomiast zamykający — </script>. W związku z tym schematyczna konstrukcja wykorzystująca ten typ znaczników to:

```
<script language="php" type="text/php">

  //tutaj kod skryptu

</script>
```

Postać ta, podobnie jak kanoniczna, jest rozpoznawana standardowo i nie wymaga włączania żadnych dodatkowych opcji konfiguracyjnych. Należy pamiętać, że atrybut language jest niepoprawny dla HTML w wersji

4.1 oraz XHTML.

Typy danych i zmienne

Zmienne są podstawowymi elementami każdego ze współczesnych języków programowania. Pozwalają na definiowanie struktur danych — od tych najprostszych, przechowujących wartość liczbową, do tych bardziej złożonych (tablic, obiektów itp.). W języku PHP pod pojęciem zmiennej rozumiemy symbol, któremu możemy przypisać pewną wartość.

PHP nie jest językiem, w którym zmienne muszą być deklarowane przed pierwszym użyciem. Zmienna zostaje utworzona i zdefiniowana w momencie jej pierwszego użycia; może to mieć miejsce w dowolnym miejscu w kodzie skryptu. PHP nie jest również językiem ścisłej kontroli typów rozumianych jako zbiory wartości, które mogą być przyjmowane przez zmienne. Typ zmiennej określany jest przez PHP w zależności od kontekstu, w jakim zmienna została użyta. Tego typu podejście może być przyczyną zamieszania w kodzie, dlatego też zwraca się uwagę — szczególnie początkującym programistom — na przemyślane nazywanie zmiennych oraz komentowanie kodu.

Nazwa (identyfikator) zmiennej w PHP zaczyna się od znaku $, po którym może wystąpić mała litera, duża litera lub znak podkreślenia, a dalej dowolna liczba dużych i małych liter, cyfr i znaków podkreślenia. Ważne jest, żeby nazwy zmiennych były używane konsekwentnie — o ile PHP w przypadku słów kluczowych i innych predefiniowanych elementów dopuszcza zmianę wielkości liter, o tyle w przypadku zmiennych nie jest to wskazane.

Oto kilka przykładów zmiennych:

```
$a = 0;

$zmiennaNapisowa = "Nieokreślony ciąg znaków";

$b = $a + 1;

$innyNapis = "Również nieokreślony ciąg znaków";

$tmp = $zmiennaNapisowa;

$wynik = true;
```

Jak widać, zmienne użyte w przykładzie przyjmują różne typy wartości: liczby całkowite jako stałe literalne (0), liczby całkowite jako wynik działania (w trzecim wierszu zmiennej $b przypisana zostanie wartość zmiennej $a powiększona o 1), napisy (ciągi znaków ujęte w cudzysłów lub apostrofy), wartości innych zmiennych oraz wartość boolowską true. Występujące w PHP typy danych możemy podzielić następująco:

- typy skalarne,
- typy złożone,
- typy specjalne.

Typy skalarne

Typy skalarne to inaczej typy proste. Dzielimy je na następujące rodzaje:

-

typ boolean,
-

typ integer,
-

typ float,
-

typ string.

Typ boolean

Jest to typ logiczny, który może przyjmować tylko dwie wartości — true (prawda) oraz false (fałsz). Ten typ wykorzystywany jest przy konstruowaniu wyrażeń logicznych oraz sprawdzaniu warunków. **Typ integer**

Jest to typ całkowitoliczbowy, dzięki któremu można reprezentować zarówno dodatnie, jak i ujemne liczby całkowite. Liczby te mogą być zapisane w trzech różnych formatach: dziesiętnym, ósemkowym (oktalnym) lub szesnastkowym (heksadecymalnym). Domyślnie stosowany jest format dziesiętny. Jeżeli chcemy uzyskać liczbę ósemkową, poprzedzamy ją znakiem 0 (zero). Liczbę szesnastkową należy poprzedzić znakami 0x. Przy zapisie wartości szesnastkowych można wykorzystywać zarówno duże, jak i małe litery od a do f. Przykładowo, wszystkie zaprezentowane poniżej wartości są typu integer:
-

123 dodatnia całkowita wartość dziesiętna 123,
-

-123 ujemna całkowita wartość dziesiętna –123,
-

012 dodatnia całkowita wartość ósemkowa równa 10 dziesiętnie,
-

-024 ujemna całkowita wartość ósemkowa równa 20 dziesiętnie,
-

0xFF dodatnia całkowita wartość szesnastkowa równa 255 dziesiętnie,
-

-0x0f ujemna całkowita wartość szesnastkowa równa –15 dziesiętnie.

Maksymalny zakres typu całkowitego (czyli zakres wartości, jaki można za jego pomocą przedstawić), podobnie jak w języku C, zależy od platformy sprzętowo-systemowej, na jakiej uruchamiane jest PHP. Typowo jest to 32-bitowa liczba ze znakiem, czyli zakres wartości od -2^{31} do $2^{31} - 1$. W przypadku przekroczenia zakresu wartość jest konwertowana do typu float. **Typ float**

Typ float reprezentuje zarówno dodatnie, jak i ujemne liczby rzeczywiste (zmiennopozycyjne, zmiennoprzecinkowe). Ich zakres, podobnie jak dla typu integer, jest zależny od platformy sprzętowo-systemowej, z reguły są to wartości od $-1{,}8 * 10^{308}$ do $1{,}8 * 10^{308}$. Typową reprezentacją jest zapis z kropką dziesiętną, czyli np. 1.5. Można również używać notacji wykładniczej w postaci X.YeZ, gdzie X to część całkowita, Y — dziesiętna, natomiast Z to wykładnik potęgi liczby 10. Zapis taki oznacza to samo, co $X.Y * 10^Z$. Przykładowo wszystkie zaprezentowane poniżej wartości są typu float:

-

dodatnia wartość rzeczywista 1,1,
-

-1.1 ujemna wartość rzeczywista 1,1,
-

0.1E2 dodatnia wartość rzeczywista 10,
-

-1.0E-2 dodatnia wartość rzeczywista –0.01.

Typ string

Typ string to typ łańcuchowy, który służy do zapamiętywania sekwencji znaków. Nie ma ograniczenia długości tego ciągu. Do zapamiętywania pojedynczego znaku jest wykorzystywany jeden bajt. Wynika z tego, że PHP (do wersji 5. włącznie) bezpośrednio nie obsługuje standardu Unicode (zmieni się to w wersji 6.). Łańcuch znaków można utworzyć na trzy sposoby:
-

z użyciem znaków apostrofu,
-

z użyciem znaków cudzysłowu,
-

z użyciem składni heredoc.

Wykorzystanie znaków apostrofu

Pierwszym sposobem deklaracji łańcucha znakowego jest ujęcie go w znaki apostrofu. PHP praktycznie nie dokonuje interpretacji takiego ciągu znaków, jeśli zatem, na przykład, wyświetlamy go na ekranie, pojawi się w większości przypadków w niezmienionej postaci. Przykładem takiego ciągu jest 'abc'.

Wyjątkiem od tej zasady jest specjalne traktowanie samego znaku apostrofu. Jeśli chcemy uzyskać go w napisie, koniecznie musimy poprzedzić go znakiem (lewy ukośnik). Podobnie jeżeli chcemy uzyskać sekwencję ', należy napisać \'. Ta zasada dotyczy także samego znaku ukośnika. **Wykorzystanie znaków cudzysłowu**

Drugim sposobem deklaracji łańcucha znakowego jest ujęcie go w znaki cudzysłowu. Przykładem takiego ciągu jest "abc". Takie ciągi są przetwarzane przez PHP. Najważniejsze jest to, że jeśli w tego typu sekwencji znajdzie się określenie zmiennej, zostanie ono zmienione na wartość przez nią reprezentowaną. **Wykorzystanie składni heredoc**

W przypadku składni heredoc łańcuch znakowy należy rozpocząć od sekwencji <<<, po której musi nastąpić identyfikator. Tego identyfikatora należy następnie użyć w celu zasygnalizowania końca łańcucha znakowego. Dla nazwy identyfikatora obowiązują takie same zasady jak przy nazewnictwie zmiennych. Może się ona zaczynać wyłącznie od znaku podkreślenia lub litery i może zawierać dowolną kombinację liter, cyfr i znaków podkreślenia. Linia kończąca nie może natomiast zawierać żadnych innych znaków oprócz identyfikatora i średnika. Uwaga ta dotyczy wszystkich znaków, również spacji, tabulatorów itp.! Przykład definicji łańcucha znakowego korzystającej ze składni heredoc jest następujący:

$napis = <<<ID1

Tutaj rozpoczyna się napis

ID1;

Zmienne zawarte w ciągu znaków korzystającym z opisywanej składni, podobnie jak w przypadku składni z cudzysłowami, zostaną zamienione na odpowiadające im wartości. Jak łatwo się domyślić, ten sposób tworzenia napisów jest najczęściej wykorzystywany, kiedy mają one mieć znaczną długość i składają się z wielu linii. **Typy złożone**

Typy złożone dzielą się na dwa rodzaje. Są to:

-

typ array (tablicowy),

-

typ object (obiektowy).

Tablice zostaną omówione w jednym z kolejnych podpunktów, natomiast obiektom i programowaniu obiektowemu poświęcony jest cały rozdział 8. **Typy specjalne**

Wyróżniamy dwa rodzaje typów specjalnych. Są to:

-

typ resource,

-

typ null.

Typ resource

Typ resource jest typem specjalnym wskazującym, że zmienna przechowuje odwołanie do zasobu zewnętrznego utworzonego za pomocą specjalnych funkcji. W tym kursie nie będziemy zajmować się tego rodzaju danymi. **Typ null**

Typ null jest typem specjalnym informującym o tym, że dana zmienna nie przechowuje żadnej wartości (jest pusta). Został on wprowadzony w PHP w wersji 4. Jego użycie jest praktycznie takie samo, jak w innych językach programowania. Jeżeli chcemy ustawić zmienną na null, piszemy:

$zmienna = null;

Wielkość liter nie ma przy tym znaczenia, prawidłowe są zatem zapisy: null, NULL, Null czy nawet całkowicie niepraktyczny nUlL. **Zmiana typu zmiennej**

Ponieważ typ zmiennej ustalany jest przez PHP na bieżąco, można użyć tej samej zmiennej do przechowywania wartości różnych typów — np. po raz pierwszy zmienna może być użyta do przechowywania liczby, a później można jej przypisać ciąg znaków. **Tablice**

O ile tworzenie i używanie zmiennych typów podstawowych nie jest trudne (pokazuje to wcześniejszy przykład), o tyle w przypadku tablic potrzebna będzie dodatkowa porcja informacji.

Tablice to struktury danych przechowujące zbiór danych, zwykle jednego typu, dostępnych przez indeks. Tablice mogą mieć postać wektora (czyli będą to tablice jednowymiarowe), ale dopuszcza się też tworzenie tablic wielowymiarowych (zazwyczaj nie wykracza się poza tablice dwuwymiarowe).

Struktury tablicowe w PHP można tworzyć albo na bieżąco przypisując wartości dla poszczególnych indeksów, albo za pomocą konstrukcji językowej array(). Ponadto w PHP można tworzyć tablice zwykłe (indeksowane kolejnymi nieujemnymi liczbami całkowitymi) oraz asocjacyjne — indeksowane kluczem, którym może być również ciąg znaków.

W przypadku tablic zwykłych, jeśli do utworzenia ma być użyte słowo kluczowe array, konstrukcja przyjmie schematyczną postać:

```
$tablica = array(wartość1, wartość2,..., wartośćN);
```

$tablica to zmienna typu tablicowego, dzięki której będziemy mogli się do tej struktury odwoływać, natomiast wartość1, wartość2 itd. to wartości znajdujące się w kolejnych komórkach. W przypadku dużej liczby wartości w celu zwiększania czytelności stosuje się również zapis w postaci:

```
$tablica = array(

  wartość1,

  wartość2,

  ...,

  wartośćN);
```

Jeśli natomiast ma zostać zastosowany zapis uproszczony, stosowana jest konstrukcja:

```
$tablica[0] = wartość1;

$tablica[1] = wartość1;

…

$tablica[N-1] = wartośćN;
```

Tablice asocjacyjne tworzy się, podobnie jak w przypadku tablic klasycznych indeksowanych numerycznie, za pomocą słowa kluczowego array, jednak konstrukcja ma nieco inną postać. Schematycznie wygląda to następująco:

```
$tablica = array(

  klucz1 => wartość1,

  klucz2 => wartość2,

  ...

  kluczN => wartośćN

);
```

Prosty przykład tworzenia i użycia tablic przedstawiony został na listingu 4.5, natomiast w dalszej części kursu

pojawią się przykłady bardziej zaawansowane.

Listing 4.5. Tworzenie i użycie tablic

```
<?php

 // Tablica klasyczna tworzona wprost

 $tab1[0] = "jeden";

 $tab1[1] = "dwa";

 $tab1[2] = "trzy";


 // Tablica klasyczna tworzona za pomocą konstrukcji array()

 $tab2 = array("cztery", "pięć", "sześć");


 // Tablica asocjacyjna tworzona za pomocą konstrukcji array()

 $tab3 = array('pierwszy' => 1,

              'drugi'    => 2,

              'trzeci'   => 3);


 echo "Zawartość tablicy tab1:n";

 echo $tab1[0], " ", $tab1[1], " ", $tab1[2], "n";


 echo "Zawartość tablicy tab2:n";

 echo $tab2[0], " ", $tab2[1], " ", $tab2[2], "n";


 echo "Zawartość tablicy tab3:n";

 echo $tab3['pierwszy'], " ", $tab3['drugi'],

     " ", $tab3['trzeci'], "n";


 //Zmiana zawartości komórki o indeksie 1 w tablicy tab1

 $tab1[1] = 10;
```

```
  echo "Zawartość tablicy tab1 po zmnianie:n";

  echo $tab1[0], " ", $tab1[1], " ", $tab1[2], "n";

?>
```

PHP udostępnia szereg funkcji (pojęcie funkcji wyjaśnione zostało w dalszej części rozdziału) pozwalających na sprawdzanie zawartości i właściwości zmiennych, zarówno typów prostych, jak i złożonych — funkcje te są sukcesywnie wprowadzane w miarę potrzeb w kolejnych rozdziałach. **Zmienne zmienne**

Tytuł punktu jest dziwny, ale w bardzo trafny sposób oddaje sens pewnej interesującej konstrukcji językowej dopuszczalnej w PHP. Wykorzystując właściwość PHP polegającą na tworzeniu zmiennych w momencie ich pierwszego użycia, programista ma możliwość tworzenia zmiennych, których identyfikatorem mogą być wartości innych zmiennych. Oto przykład:

```
<?php

  $a = "początek";

  $$a = " zdania";

  echo $a.$$a; // "początek zdania"

?>
```

W pierwszym wierszu tworzona jest zmienna napisowa $a i przypisuje się jej napis „początek”. W drugim wierszu tworzona jest zmienna, której identyfikator jest po prostu wartością zmiennej $a, i zmiennej tej przypisywany jest ciąg znaków „zdania”. Wyświetlając zawartość obu zmiennych (użyto tutaj opisanego dalej operatora konkatenacji napisów) w sposób pokazany w wierszu trzecim, uzyska się napis umieszczony w komentarzu do tego wiersza.

Zmienne tworzone w ten sposób wprowadzają element programowania dynamicznego do PHP i wbrew temu, co by się mogło wydawać, często znajdują zastosowanie podczas generowania stron WWW o strukturze silnie uzależnionej od danych. **Zmienne predefiniowane**

PHP udostępnia szereg zmiennych predefiniowanych. Zmienne te są osiągalne z każdego skryptu i najczęściej są uzależnione od środowiska pracy (systemu, serwera WWW itd.). Najważniejsze zmienne predefiniowane, stanowiące źródło wielu cennych informacji, a w wielu przypadkach wręcz niezbędnych do poprawnego tworzenia aplikacji WWW, zgromadzone zostały w tzw. tablicach superglobalnych. Są to tablice asocjacyjne, indeksowane identyfikatorem zmiennej predefiniowanej. Sposób wykorzystania poszczególnych tablic superglobalnych zostanie wprowadzony i wyjaśniony w miarę potrzeby w następnych rozdziałach. Tablice superglobalne to (pełne zestawienie tych tablic można znaleźć w dokumentacji PHP): **$GLOBALS**

Jest to tablica zawierająca odniesienie do każdej zmiennej zdefiniowanej użytkownika, która ma zasięg globalny dla danego skryptu (temat zasięgu i widoczności zmiennych zostanie przedstawiony bliżej podczas omawiania funkcji w lekcji 6.). Kluczami tej tablicy są nazwy, a wartościami kluczy wartości zmiennych. Tablica $GLOBALS została wprowadzona w PHP 3. **$_SERVER**

Jest to tablica zawierająca informacje ustawiane przez serwer WWW. Można z niej odczytać m.in. dane dotyczące połączenia, które wywołało dany skrypt, np. adres IP, port, wartości nagłówków HTTP itp. Zmienna ta jest dostępna

od PHP 4.1.0. W wersjach wcześniejszych wykorzystywana była do tego celu zmienna $HTTP_SERVER_VARS.
$_GET

Tablica zawierająca dane przekazane do serwera WWW za pomocą metody GET. Zmienna $_GET jest dostępna od PHP 4.1.0. W wersjach wcześniejszych wykorzystywana była do tego celu zmienna $HTTP_GET_VARS.
$_POST

Tablica zawierająca dane przekazane do serwera WWW za pomocą metody POST. Zmienna $_POST jest dostępna od PHP 4.1.0. W wersjach wcześniejszych wykorzystywana była do tego celu zmienna $HTTP_POST_VARS. **$_COOKIE**

Tablica zawiera cookies przekazane z serwera WWW. Zmienna $_COOKIE jest dostępna od PHP 4.1.0. W wersjach wcześniejszych wykorzystywana była do tego celu zmienna $HTTP_COOKIE_VARS. **$_FILES**

Tablica zawierająca elementy przekazane do skryptu za pomocą metody POST podczas przesyłania plików do serwera. Zmienna $_FILES jest dostępna od PHP 4.1.0. W wersjach wcześniejszych wykorzystywana była do tego celu zmienna $HTTP_POST_FILES. **$_ENV**

Tablica zawierająca wartości zmiennych środowiskowych przekazanych z systemu, na którym działa PHP. Zmienna $_ENV jest dostępna od PHP 4.1.0. W wersjach wcześniejszych wykorzystywana była do tego celu zmienna $HTTP_ENV_VARS. **$_REQUEST**

Tablica asocjacyjna zawierająca dane z $_GET, $_POST i $_COOKIE. Zmienna $_REQUEST jest dostępna od PHP 4.1.0, nie ma odpowiednika we wcześniejszych wersjach. Do wersji PHP 4.3.0 $_REQUEST zawierała również dane z tablicy $_FILES. **$_SESSION**

Tablica asocjacyjna zawierająca dane związane z bieżącą sesją (sesje zostały opisane w rozdziale 8.). Zmienna $_SESSION jest dostępna od PHP 4.1.0. W wersjach wcześniejszych wykorzystywana była do tego celu zmienna $HTTP_SESSION_VARS.

Na listingu 4.6 przedstawiono sposób wyświetlenia zawartości jednej z tablic superglobalnych — tutaj jest to tablica $_SERVER.

Listing 4.6. Wyświetlenie zawartości tablicy superglobalnej $_SERVER (kod oraz rezultat)

```
<?php


  print_r($_SERVER);


?>
```

```
Array

(
```

[HTTP_ACCEPT] => */*

[HTTP_ACCEPT_LANGUAGE] => pl

[HTTP_ACCEPT_ENCODING] => gzip, deflate

[HTTP_USER_AGENT] => Mozilla/4.0 (compatible; MSIE 6.0; Windows NT 5.1; SV1; .NET CLR 1.1.4322; .NET CLR 2.0.50727)

[HTTP_HOST] => 192.168.20.135

[HTTP_CONNECTION] => Keep-Alive

[HTTP_CACHE_CONTROL] => no-cache

[PATH] => /sbin:/usr/sbin:/bin:/usr/bin

[SERVER_SIGNATURE] =>

[SERVER_SOFTWARE] => Apache/2.2.0 (Fedora)

[SERVER_NAME] => 192.168.20.135

[SERVER_ADDR] => 192.168.20.135

[SERVER_PORT] => 80

[REMOTE_ADDR] => 192.168.20.1

[DOCUMENT_ROOT] => /var/www/html

[SERVER_ADMIN] => root@localhost

[SCRIPT_FILENAME] => /home/rafal/public_html/index.php

[REMOTE_PORT] => 1887

[GATEWAY_INTERFACE] => CGI/1.1

[SERVER_PROTOCOL] => HTTP/1.1

[REQUEST_METHOD] => GET

[QUERY_STRING] =>

[REQUEST_URI] => /~rafal/

[SCRIPT_NAME] => /~rafal/index.php

[PHP_SELF] => /~rafal/index.php

[REQUEST_TIME] => 1150548791

) Wyrażenia

W zasadzie wszystko w języku PHP to wyrażenie: przypisanie wartości do zmiennej, dodanie dwóch zmiennych,

ale również sama zmienna lub stała. Cechą charakterystyczną wyrażeń jest wartość — każdy element języka, który ma wartość, może być określany jako wyrażenie. W kursie tym — zgodnie z obietnicą dotyczącą rezygnacji z nadmiaru teorii — nie będzie długiego wykładu na temat wyrażeń, skupimy się głównie na ich zastosowaniu, stąd tylko te parę zdań wprowadzenia.

Najpopularniejszą formą wyrażenia jest porównanie (operatory porównania opisane są w następnym punkcie), wykorzystywane najczęściej w instrukcjach sterujących przebiegiem programu, podczas podejmowania decyzji. Porównania mają wartości typu boolowskiego, natomiast inne wyrażenia mogą mieć różne wartości.

Tak jak w przypadku zmiennych predefiniowanych różne rodzaje wyrażeń wprowadzane będą w miarę opisywania kolejnych zagadnień związanych z tworzeniem dynamicznych stron WWW.

Operatory

Język PHP jest językiem programowania bazującym na wyrażeniach, a w przypadku większości wyrażeń najistotniejszym elementem jest operator. Operatory w PHP można podzielić na jednoargumentowe oraz dwuargumentowe. Ze względu na zastosowanie operatory w PHP dzieli się na arytmetyczne, przypisania, porównania, inkrementacji, dekrementacji, bitowe, logiczne, kontroli błędów, wykonania poleceń systemowych, napisowe oraz tablicowe.

W punkcie tym przedstawione zostaną tylko najbardziej popularne operatory: arytmetyczne, przypisania, porównania, inkrementacji, dekrementacji, logiczne i napisowe. Pozostałe grupy operatorów są opisane szczegółowo w dokumentacji PHP.

Operatory arytmetyczne

Operatory arytmetyczne są w przypadku PHP podobne jak w matematyce, jedynie mnożenie i dzielenie oznaczane jest inaczej:

-

operator dodawania — „+”

-

operator odejmowania — „-”

-

operator mnożenia — „*”

-

operator dzielenia — „/”

-

operator modulo (reszta z dzielenia) — „%”

Działanie tych operatorów sprowadza się do wykonywania znanych każdemu typowych działań arytmetycznych. Jeśli zatem chcemy dodać do siebie dwie zmienne lub liczbę do zmiennej, wykorzystujemy operator +, do mnożenia operator * itd. Listing 4.7 przedstawia zastosowanie operatorów arytmetycznych.

Listing 4.7. Przykład zastosowania operatorów arytmetycznych

```
<?php

  $a = 5;

  $b = $a + 3; // $b zawiera 8
```

```
$b = $b / 2; // $b zawiera 4

$a = $a * $b; // $a zawiera 20

?>
```

Na początku zmiennej $a przypisano wartość 5. Następnie zmiennej $b przypisano wynik operacji arytmetycznej $a + 3, czyli 8. W linii trzeciej zmienna $b zmieniała wartość na wynik wyrażenia $b / 2, czyli 4. W ostatnim wierszu zmienna $a otrzymała wartość wynikającą z działania $a * $b, czyli 20.

Do operatorów arytmetycznych należy również %, przy czym, jak zostało to zaznaczone wyżej, nie oznacza on obliczania procentów, ale dzielenie modulo, czyli resztę z dzielenia. Przykładowo, działanie 10 % 3 da w wyniku 1. Trójka zmieści się bowiem w dziesięciu 3 razy, pozostawiając resztę 1 (3 * 3 = 9, 9 + 1 = 10). Podobnie 21 % 8 = 5, bowiem 2 * 8 = 16, 16 + 5 = 21. **Operatory inkrementacji i dekrementacji**

Operatory inkrementacji i dekrementacji są operatorami jednoargumentowymi i mają za zadanie zwiększenie lub zmniejszenie wartości zmiennej o jeden. Występują w dwóch postaciach: przedrostkowej i przyrostkowej (mówimy też o preinkrementacji i postinkrementacji oraz predekrementacji i postdekrementacji). Jeżeli zastosowane są przy samodzielnej zmiennej, ich działanie jest identyczne, natomiast zastosowane przy zmiennej wchodzącej w skład wyrażenia dają zgoła inny efekt. W przypadku preinkrementacji najpierw wartość zmiennej jest zwiększana o jeden, a potem używana w wyrażeniu, natomiast w przypadku postinkrementacji najpierw wartość zmiennej pobierana jest do wartościowania wyrażenia, a później zwiększana o jeden.

Przykład preinkrementacji:

```
++$a;
```

Przykład postinkrementacji:

```
$a++;
```

Identycznie działa operator dekrementacji, z tą różnicą, że oczywiście zmniejsza wartość zmiennej o jeden.

Przykład predekrementacji:

```
--$a;
```

Przykład postdekrementacji:

```
$a--;
```

Operatory bitowe

Operatory bitowe służą, jak sama nazwa wskazuje, do wykonywania operacji na bitach. Są to:

- iloczyn bitowy (inaczej koniunkcja bitowa, operacja AND) — &;
- suma bitowa (inaczej alternatywa bitowa, operacja OR) — |;
- negacja bitowa (inaczej uzupełnienie do jedynki, operacja NOT) — ~;

suma bitowa modulo 2 (inaczej alternatywa bitowa wykluczająca, różnica symetryczna, operacja XOR) — ^;
-

przesunięcie bitowe w prawo — >>;
-

przesunięcie bitowe w lewo — <.

Wszystkie operacje z wyjątkiem jednoargumentowej negacji bitowej wymagają dwóch argumentów. **Iloczyn bitowy**

Iloczyn bitowy to operacja powodująca, że włączone pozostają tylko te bity, które były włączone w obu argumentach. Wynik operacji AND na pojedynczych bitach obrazuje tabela 4.1. Jeśli zatem wykonamy operację:

$liczba = 179 & 38;

to zmiennej $liczba zostanie przypisana wartość 34.

Tabela 4.1. Wyniki operacji AND dla pojedynczych bitów

Argument 1	Argument 2	Wynik
1	1	1
1	0	0
0	1	0
0	0	0

Dlaczego 34? Najłatwiej pokazać to, jeśli obie wartości (czyli 179 i 38) przedstawi się w postaci dwójkowej. 179 w postaci dwójkowej to 10110011 natomiast 38 to 00100110. Operacja AND będzie zatem miała postać:

10110011 (179)

00100110 (38)

00100010 (34)

Wynikiem jest więc 34. **Suma bitowa**

Suma bitowa to operacja powodująca, że pozostają włączone te bity, które były włączone przynajmniej w jednym z argumentów. Wynik operacji OR na pojedynczych bitach obrazuje tabela 4.2. Jeśli zatem wykonamy operację:

$liczba = 34 | 65;

to zmiennej $liczba zostanie przypisana wartość 99.

Tabela 4.2. Wyniki operacji OR dla pojedynczych bitów

Argument 1	Argument 2	Wynik
1	1	1
1	0	1
0	1	1
0	0	0

0

0

Jeśli obie liczby rozpiszemy w postaci dwójkowej, otrzymamy 00100010 (34) i 01000001 (65). Zatem całe działanie będzie miało postać:

00100010 (34)

01000001 (65)

01100011 (99) **Negacja bitowa**

Negacja bitowa powoduje zmianę stanu bitów. Czyli tam, gdzie dany bit miał wartość 0, będzie miał 1, natomiast tam, gdzie miał wartość 1, będzie miał 0. Działanie operacji NOT na pojedynczych bitach obrazuje tabela 4.3.

Tabela 4.3. Wyniki operacji NOT dla pojedynczych bitów

Argument

Wynik

1

0

0

1

Bitowa różnica symetryczna

Bitowa różnica symetryczna, czyli operacja XOR, powoduje, że włączone zostają te bity, które miały różne stany w obu argumentach, a pozostałe zostają wyłączone. Wynik operacji XOR na pojedynczych bitach obrazuje tabela 4.4.

Tabela 4.4. Wyniki operacji XOR dla pojedynczych bitów

Argument 1

Argument 2

Wynik

1

1

0

1

0

1

0

1

1

0

0

0

Wykonanie przykładowej operacji:

$liczba = 34 ^ 118;

spowoduje przypisanie zmiennej $liczba wartości 84. Jeśli bowiem zapiszemy obie wartości w postaci dwójkowej, to 34 przyjmie postać 00100010, natomiast 118 — 01110110. Operacja XOR będzie zatem wyglądała następująco:

00100010 (34)

01110110 (118)

01010100 (84) **Przesunięcie bitowe w lewo**

Przesunięcie bitowe w lewo to operacja polegająca na przesunięciu wszystkich bitów argumentu znajdującego się z lewej strony operatora w lewo o liczbę miejsc wskazaną przez argument znajdujący się z jego prawej strony. Tym samym wykonanie przykładowej operacji:

$liczba = 84 << 1;

spowoduje przypisanie zmiennej $liczba wartości 168. Działanie 84 << 1 oznacza bowiem: przesuń wszystkie bity wartości 84 o jedno miejsce w prawo. Skoro 84 w postaci dwójkowej ma postać 01010100, to po przesunięciu powstanie 10101000, czyli 168.

Warto zauważyć, że przesunięcie bitowe w lewo odpowiada mnożeniu wartości przez wielokrotność liczby 2. Czyli przesunięcie w lewo o jedno miejsce to pomnożenie przez 2, o dwa miejsca — przez 4, o trzy miejsca — przez 8 itd. **Przesunięcie bitowe w prawo**

Przesunięcie bitowe w prawo polega na przesunięciu wszystkich bitów argumentu znajdującego się z lewej strony operatora w prawo o liczbę miejsc wskazaną przez argument, który znajduje się z prawej strony operatora. A zatem wykonanie operacji:

$liczba = 84 >> 1;

spowoduje przypisanie zmiennej $liczba wartości 42. Oznacza to bowiem przesunięcie wszystkich bitów wartości 01010100 o jedno miejsce w prawo, czyli powstanie wartości 00101010 dwójkowo (42 dziesiętnie).

Tu również należy zwrócić uwagę, że przesunięcie bitowe w prawo odpowiada podzieleniu wartości przez wielokrotność liczby 2. Czyli przesunięcie w prawo o jedno miejsce to podzielenie przez 2, o dwa miejsca — przez 4, o trzy miejsca — przez 8 itd., przy czym część ułamkowa jest tracona. **Operatory logiczne**

Operatory logiczne wykorzystywane są do tworzenia wyrażeń logicznych z wyrażeń mających wartości typu boolowskiego (np. wyrażeń porównujących). Operatory logiczne mają dokładnie takie samo znaczenie jak podstawowe funkcje logiczne i znajdują zastosowanie podczas tworzenia złożonych wyrażeń warunkowych wykorzystywanych w instrukcjach sterujących przebiegiem programu (opisanych w dalszej części rozdziału).

Operatory logiczne:

- logiczne I (iloczyn) — „and” lub „&&”;
- logiczne LUB (suma)— „or” lub „||”;
- logiczne NIE (negacja) — „!” (operator jednoargumentowy, umieszczany przed argumentem);
- logiczne ALBO (WYŁĄCZNIE-LUB, alternatywa wykluczająca) — „xor”.

Przykłady zastosowania tych operatorów pojawią się w dalszych rozdziałach, gdzie będą wprowadzane w miarę konieczności wraz z objaśnieniem. Warto jednak przypomnieć sobie podstawy logiki. **Iloczyn logiczny**

Wynikiem operacji AND (iloczyn logiczny) jest wartość true wtedy i tylko wtedy, kiedy oba argumenty mają wartość true. W każdym innym przypadku wynikiem jest false. Przedstawia to tabela 4.5.

Tabela 4.5. Logiczny iloczyn

Argument 1	Argument 2	Wynik
true	true	true

true
false
false

false
true
false

false
false
false

Suma logiczna

Wynikiem operacji OR (suma logiczna, alternatywa logiczna) jest wartość true wtedy i tylko wtedy, kiedy przynajmniej jeden z argumentów ma wartość true. W przypadku kiedy oba argumenty mają wartość false, jest ona również wynikiem całej operacji. Przedstawia to tabela 4.6.

Tabela 4.6. Logiczna suma

Argument 1	Argument 2	Wynik
true	true	true
true	false	true
false	true	true
false	false	false

true
false
false
false

Logiczna alternatywa wykluczająca

Wynikiem operacji XOR (logiczna alternatywa wykluczająca, różnica symetryczna) jest wartość true wtedy i tylko wtedy, kiedy oba argumenty mają różną wartość. W przypadku kiedy argumenty mają równą wartość, wynikiem jest false. Przedstawia to tabela 4.7.

Tabela 4.7. Logiczna suma

Argument 1	Argument 2	Wynik
true	true	false
true	false	true
false	true	true
false	false	false

Negacja logiczna

Operacja NOT (logiczna negacja) zamienia wartość argumentu na przeciwną. Jeśli więc argument miał wartość true, będzie miał false i odwrotnie — jeśli miał wartość false, będzie miał true. Obrazuje to tabela 4.8.

Tabela 4.8. Logiczna negacja

Argument	Wynik
true	
true	
false	
True	

Operatory przypisania

Operator przypisania jest w zasadzie jeden i jest oznaczany za pomocą znaku „=”. Operator przypisania był już bez objaśnień używany w przykładach do tego rozdziału — łatwo się domyślić, że za jego pomocą możemy nadać zmiennej wartość. Jako operatory przypisania można również przedstawić operatory złożone, wykonujące dwie operacje naraz: +=, -=, *= itd. Złożone operatory przypisania są po prostu skróconą formą operacji zwiększania lub zmniejszania wartości danej zmiennej o wartość wyrażenia (zmiennej, stałej itp.).

Przykładowy zapis:

$a += 5

tłumaczymy jako:

$a = $a + 5

Oznacza on: przypisz zmiennej $a wartość wynikającą z dodawania $a + 5.

W PHP występuje cała grupa tego typu operatorów (są one zebrane w tabeli 4.9). Schematycznie możemy przedstawić ich znaczenie następująco:

arg1 op= arg2

oznacza działanie

arg1 = arg1 op arg2

czyli $a += $b oznacza $a = $a + $b, $a *= $b oznacza $a = $a * $b, $a %= $b oznacza $a = $a % $b itd.

Tabela 4.9. Operatory przypisania i ich znaczenie w PHP

Argument 1	Operator	Argument 2	Znaczenie

x

=

y

x = y

x

+=

y

x = x + y

x

-=

y

x = x – y

x

*=

y

x = x * y

x

/=

y

x = x / y

x

%=

y

x = x % y

x

```
.=

y

x = x . y


x

<<=

y

x = x << y


x

>>=

y

x = x >> y


x

&=

y

x = x & y


x

|=

y

x = x | y


x

^=

y

x = x ^ y
```

Widoczny na listingu 4.8 przykład wyjaśnia, w czym rzecz.

Listing 4.8. Złożona operacja przypisania

```
<?php

 $a = 2;

 $a = $a + 5; // zwiększenie $a o 5

 echo 'Wartość zmiennej $a po operacji $a = $a + 5; ';

 echo $a;

 $a += 5; // to jest również zwiększenie $a o 5, ale krócej zapisane

 echo 'Wartość zmiennej $a po operacji $a += 5; ';

 echo $a;
```

?> **Operatory porównywania**

Operatory porównania (relacyjne) są najczęściej wykorzystywane do konstruowania wyrażeń porównujących, znajdujących zastosowanie w instrukcjach sterujących wykonaniem programu. Wyrażenia porównujące mają wartości typu boolowskiego — tworząc takie wyrażenie, zadajemy sobie pytanie, czy np. prawdą jest, że A jest większe od B.

Oto podstawowe operatory porównania:

- równość „==” — porównanie dwóch wyrażeń z wykorzystaniem tego operatora daje true, jeżeli wyrażenia te są równe co do wartości;
- identyczność „===” — true, gdy porównywane wyrażenia są równe co do wartości i typu;
- różność „<>” lub „!=” — true, gdy porównywane wyrażenia są różne co do wartości;
- różność „!==” — true, gdy wyrażenia są różne co do wartości i typu;
- mniejszość „<” — true, gdy pierwsze wyrażenie ma mniejszą wartość niż drugie;
- większość „>” — true, gdy pierwsze wyrażenie jest większe od drugiego;
- mniejszość lub równość „<=” — true, gdy pierwsze wyrażenie ma wartość mniejszą lub równą wartości wyrażenia drugiego;
- większość lub równość „>=” — true, gdy pierwsze wyrażenie ma wartość większą lub równą wartości wyrażenia drugiego.

Przykłady zastosowania tych operatorów będą się pojawiać w kolejnych skryptach, głównie podczas używania

instrukcji warunkowych i pętli. **Operator łączenia napisów**

Na koniec operator łączenia napisów (inaczej operator konkatenacji). Jest to operator dwuargumentowy, oznaczany jako „.”. Poniższy przykład (listing 4.9) pokazuje zastosowanie tego operatora.

Listing 4.9. Zastosowanie operatora łączenia napisów

```
<?php

  $napis = " punktu";

  echo "Koniec".$napis."<br />";
```

?> **Priorytety operatorów**

Oprócz znajomości samych operatorów niezbędna jest jeszcze wiedza o tym, jakie mają one priorytety, czyli jaka jest kolejność wykonywania reprezentowanych przez nie operacji. Wiemy np. z matematyki, że mnożenie jest silniejsze od dodawania, zatem najpierw mnożymy, potem dodajemy. W PHP jest podobnie, siła każdego operatora jest ściśle określona. Zostało to przedstawione w tabeli 4.10 (uwzględnione zostały również dotychczas nieopisywane operatory). Im wyższa pozycja w tabeli, tym wyższy priorytet operatora. Operatory znajdujące się na jednym poziomie (w jednym wierszu) mają ten sam priorytet.

Tabela 4.10. Priorytety operatorów

Lp.	Rodzaje operatorów	Symbole
1	Tworzenie obiektów	new
2	Indeks tablicy	[]
3	Inkrementacja, dekrementacja	++, --
4		

Negacje, konwersje typów, kontrola błędów
!, ~, ++, --, (int), (float), (string), (array), (object), @

5
Mnożenie, dzielenie, reszta z dzielenia
*, /, %

6
Dodawanie, odejmowanie, łączenie łańcuchów znakowych
+, -, .

7
Przesunięcia bitowe
<<, >>

8
Relacje (mniejsze, większe, mniejsze lub równe, większe lub równe)
<, >, <=, >=

9
Relacje (równe, identyczne, różne, nieidentyczne)
==, ===, !=, !==

10
Iloczyn bitowy
&

11
Bitowa różnica symetryczna
^

12

Suma bitowa
|

13
Iloczyn logiczny
&&

14
Suma logiczna
||

15
Warunkowy
? :

16
Operatory przypisania
=, +=, -=, *=, /=, .=, %=, &=, ^=, |=, <<=, >>=

17
Iloczyn logiczny
and

18
Logiczna różnica symetryczna
xor

19
Suma logiczna
or

20

Rozdzielanie wyrażeń

,

Sterowanie wykonaniem programu

Gdy wie się już, czym są zmienne, stałe i jakiego typu wartości mogą przyjmować zmienne, następnym krokiem w poznawaniu każdego nowego języka programowania jest poznanie struktur sterujących wykonaniem programu.

Świat byłby o wiele wspanialszy, niż jest, gdyby wszystkie programy wykonywały się instrukcja po instrukcji, od początku do końca jedną drogą… Niestety, jest tak, że czasem trzeba podjąć decyzję, co robić dalej w zaistniałej sytuacji. Na przykład program poprosi użytkownika o podanie liczby dodatniej — a co się stanie, jeżeli użytkownik poda liczbę ujemną lub zero? Albo prowadząc obliczenia matematyczne, program dochodzi do momentu dzielenia jednej wartości przez drugą — może się okazać, że dzielnik jest równy zero. Co trzeba zrobić, aby w takiej sytuacji program (skrypt) nie „wyłożył się”? Można mnożyć podobne przykłady, oczywiście niekoniecznie dotyczące sytuacji tak drastycznych — można np. podejmować decyzję, jaki napis wyświetlić, jeśli użytkownik jest zalogowany, a jaki, gdy użytkownik jeszcze się nie zalogował itp. Instrukcja warunkowa if oraz instrukcja wyboru switch, o których to instrukcjach traktuje ten punkt, są zaraz po instrukcji przypisania najczęściej używanymi instrukcjami każdego algorytmicznego języka programowania.

Na początek przykład z dzieleniem przez zero. Program wykonuje obliczenia arytmetyczne w oparciu o dane przekazane przez użytkownika (np. za pośrednictwem formularza umieszczonego na stronie WWW). Może się zdarzyć, że dane będą tak dobrane, że dzielnik (zakładając, że gdzieś po drodze wykonywane będzie dzielenie) będzie równy 0. W takiej sytuacji próba wykonania dzielenia zakończy się niepowodzeniem, które — o ile w domyślnej konfiguracji PHP nie spowoduje drastycznego zakończenia skryptu — będzie skutkowało błędnym działaniem skryptu (użytkownik nie otrzyma wyniku lub wynik nie będzie prawdziwy).

Rozwiązaniem problemu będzie postępowanie według następującego algorytmu: przed każdą operacją dzielenia należy sprawdzić, czy dzielnik jest równy 0 — jeżeli tak, to użytkownik zostaje o tym fakcie poinformowany i program kończy obliczenia; jeżeli nie, to obliczenia są kontynuowane. **Instrukcja if**

W języku PHP zostanie to zapisane w następujący sposób (przy założeniu, że zmienne $a i $b zawierają wyniki wcześniejszych obliczeń oraz że $a jest dzielną, a $b dzielnikiem):

```
if ($b == 0) {

  die("Próba dzielenia przez zero");

}

$c = $a / $b; // Kontynuacja obliczeń
```

Instrukcja (w zasadzie funkcja — o funkcjach można przeczytać w dalszej części kursu) die spowoduje wypisanie komunikatu „Próba dzielenia przez zero” i natychmiastowe zakończenie skryptu w przypadku, gdy $b będzie równa 0.

Instrukcja warunkowa, która pojawiła się w przykładzie, ma ogólną postać:

```
if (warunek)

  instrukcja;
```

Instrukcja warunkowa, zwykle nazywana po prostu instrukcją if, w takiej postaci działa w następujący sposób: jeśli

warunek jest spełniony (w wyniku wartościowania, czyli obliczenia wyrażenia stanowiącego warunek, otrzymamy wartość true), wykonywana jest instrukcja i program kontynuuje wykonanie; jeśli zaś warunek nie jest spełniony (wartość false), program kontynuuje wykonanie z pominięciem instrukcji. Instrukcja wykonywana, gdy warunek jest spełniony, może być instrukcją prostą lub złożoną (blokiem instrukcji). Instrukcja złożona składa się z jednej lub więcej linii kodu (instrukcji prostych i złożonych) zamkniętych w nawiasach { i }. Instrukcja prosta to pojedyncza instrukcja, np. przypisania itp. To oznacza, że jeżeli w bloku if (gdy warunek jest prawdziwy) ma wystąpić wiele instrukcji, należy użyć konstrukcji w postaci:

```
if (warunek){

  instrukcja1;

  instrukcja2;

  instrukcjaN;
```

} **Instrukcja if…else**

Wracając do przykładu dzielenia przez zero, warunek stanowi wyrażenie porównujące wartość zmiennej $b z zerem. Wyrażenie to wygeneruje wartość true, jeśli $b będzie zawierała 0. Jeśli zmienna $b będzie zawierała wartość różną od zera, wyrażenie zwróci wartość false. Po analizie tego warunku wiadomo już, że skrypt zakończy obliczenia, gdy $b będzie zawierała 0, natomiast obliczenia będą kontynuowane, gdy $b będzie zawierała wartość różną od zera. Zauważ, że zamiast instrukcji die, która powoduje zakończenie wykonywania skryptu i prawdę mówiąc jej użycie nie jest najbardziej eleganckim rozwiązaniem, można było zastosować print lub echo. Jednak w takiej sytuacji skrypt nadal nie działałby tak, jak chciałby jego twórca — w zasadzie działałby tak, jakby w ogóle nie było sprawdzenia, czy $b zawiera 0, ponieważ po wyświetleniu informacji program kontynuowałby wykonanie. Aby takiej sytuacji (i podobnym przypadkom) zapobiec, stosuje się drugi wariant instrukcji warunkowej. Ma ona postać jak niżej:

```
if (warunek)

  instrukcja1;

else

  instrukcja2;
```

instrukcja1 i instrukcja2 mogą być zarówno instrukcjami prostymi, jak i złożonymi. Gdy w bloku if lub else występuje wiele instrukcji, należy je ująć w nawias klamrowy — zgodnie z podanym wyżej schematem.

W tym wariancie, jeżeli spełniony jest warunek, wykonywana jest instrukcja1, w przeciwnym wypadku instrukcja2. Po wykonaniu jednej z tych instrukcji program kontynuuje działanie. Przykład z dzieleniem przez zero można zatem zapisać w następujący sposób:

```
if ($b == 0) {

  print("Dzielenie przez zero - obliczenia zostały przerwane");

} else {

  $c = $a / $b;

  ...
```

```
}
```

Kod dokonujący obliczeń wykonany zostanie tylko wtedy, gdy $b będzie miała wartość różną od zera.

Instrukcja switch

Instrukcja if umożliwia wykonywanie zadań w zależności od dwóch przypadków: gdy warunek jest spełniony lub gdy warunek nie jest spełniony — są więc tylko dwie drogi do wyboru. A co w przypadku, gdy będzie trzeba wykonać różne operacje w zależności od tego, czy pewna zmienna przyjmuje wartości np. 0, 1, 2, 3 itd.? Można oczywiście zastosować tzw. kaskadę instrukcji if — będzie to jak najbardziej poprawne formalnie rozwiązanie, ale spowoduje znaczne zaciemnienie kodu. Owe kaskady instrukcji if stosuje się zwykle wtedy, kiedy nie można zastosować instrukcji switch, która została zaprezentowana w tej części punktu. Instrukcja switch, nazywana instrukcją wyboru, umożliwia wykonanie określonej czynności (lub zbioru czynności — czyli zwykle instrukcji prostej lub złożonej) w zależności od wartości, jaką wygeneruje badane wyrażenie. Instrukcja switch ma postać:

```
switch (wyrażenie) {

  case wartość1:

    instrukcja1;

    break;

  case wartość2:

    instrukcja2;

    break;



  ...



  default:

    instrukcjaN;

}
```

Każda z wartości występujących po słowie kluczowym case porównywana jest kolejno z wartością obliczoną na podstawie wyrażenia — jeżeli określone wartości będą równe, wykonana będzie odpowiednia instrukcja, a po niej instrukcja break, która spowoduje opuszczenie bloku instrukcji switch. Jeśli żadna z wartości po słowach case nie będzie pasowała do wartości obliczonej w wyrażeniu, wykonana zostanie instrukcja umieszczona po słowie kluczowym default. Klauzula ze słowem kluczowym default nie jest obowiązkowa, ale warto ją stosować — jest to w pewnym sensie odpowiednik sekcji else w instrukcji warunkowej. Z oczywistych względów zwykle umieszcza się sekcję default na końcu bloku instrukcji switch.

To, co odróżnia instrukcję switch od innych instrukcji (choćby od poznanej instrukcji warunkowej), to brak konieczności stosowania nawiasów { i } (zwanych również nawiasami składniowymi), jeżeli po słowie kluczowym case i przed słowem kluczowym break umieszczamy większą ilość instrukcji.

Listing 4.10 przedstawia przykładowe zastosowanie instrukcji switch.

Listing 4.10. Zastosowanie instrukcji switch

```
<?php

 // ...

 $b = abs($b); // zmienna $b zawiera wartość uzyskaną np. od użytkownika
 switch ($a) {
  case 0:
   echo "Zero";
   break;
  case 1:
   echo "Jeden";
   break;
  case 2:
   echo "Dwa";
   break;
  default:
   echo "Podałeś liczbę spoza przedziału <0, 2>";
 }

 // ...

?>
```

Umieszczanie instrukcji break po każdej instrukcji (lub grupie instrukcji) wchodzących w skład klauzuli case jest koniecznością — po zakończeniu obsługi jednego z możliwych przypadków sterowanie jest przekazywane do pierwszej instrukcji po bloku instrukcji switch (instrukcja break powoduje opuszczenie instrukcji switch). Gdyby nie umieszczać słowa break po grupie instrukcji, wykonywany byłby kod obsługi kolejnego przypadku, co niekoniecznie byłoby zgodne z założeniami twórcy programu. Czasem jednak nie wprowadza się słowa break właśnie w celu umożliwienia wykonania jednej instrukcji (grupy instrukcji) dla kilku przypadków. Listing 4.11 zawiera przykładowy kod demonstrujący przedstawioną sytuację.

Listing 4.11. Celowe pominięcie słowa break w instrukcji switch

```
<?php

switch ($a) {

  case 0:

  case 1:

  case 2:

    // W przypadku, gdy zmienna $a zawierać będzie wartość 0, 1 lub 2 wykonany

    // zostanie ten sam fragment kodu.

    echo "Zero, jeden lub dwa";

    break;

  default:

    echo " Podałeś liczbę spoza przedziału <0, 2>";

}

?>
```

Instrukcję switch można zastąpić złożoną instrukcją warunkową if…else if, w której po if występują dodatkowe bloki else if (w dowolnej liczbie). Schematyczna postać takiej konstrukcji to:

```
if (warunek1){

  instrukcje1;

}

else if (warunek2){

  instrukcje2;

}

else if (warunek3){

..instrukcje3;

}

...

else if (warunekN){

..instrukcjeN;

}
```

```
else{

..instrukcjeM;

}
```

Oznacza to: jeżeli warunek1 jest prawdziwy, to zostaną wykonane instrukcje1. W przeciwnym wypadku, jeżeli jest prawdziwy warunek2, to zostaną wykonane instrukcje2. Jeśli natomiast jest prawdziwy warunek3, to zostaną wykonane instrukcje3 itd. Jeżeli żaden z warunków nie będzie prawdziwy, to zostaną wykonane instrukcjeM. Ostatni blok else jest jednak opcjonalny i nie musi być stosowany.

Operator warunkowy

Operator warunkowy pozwala na ustalenie wartości wyrażenia w zależności od prawdziwości danego warunku. Ma on postać:

```
warunek ? wartość1 : wartość2
```

Oznacza to: jeśli warunek jest prawdziwy, podstaw za wartość całego wyrażenia wartość1, w przeciwnym wypadku za wartość wyrażenia podstaw wartość2. Spójrzmy na kod z listingu 4.12 i zastanówmy się, jaka wartość zostanie wyświetlona w instrukcji echo. Innymi słowy, jaka będzie wartość zmiennej $wynik.

Listing 4.12. Użycie operatora warunkowego

```
<?php

$liczba = 100;

$wynik = ($liczba < 0) ? -1 : 1;

echo($wynik);

?>
```

Najważniejsza jest tu oczywiście linia $wynik = ($liczba < 0) ? –1 : 1;. Po lewej stronie operatora przypisania = znajduje się zmienna ($wynik), natomiast po stronie prawej wyrażenie warunkowe, czyli linia ta oznacza: przypisz zmiennej $wynik wartość wyrażenia warunkowego. Jak jest ta wartość? Trzeba przeanalizować samo wyrażenie: ($liczba < 0) ? –1 : 1. Oznacza ono, zgodnie z tym, co zostało napisane w poprzednim akapicie: jeżeli wartość zmiennej $liczba jest mniejsza od 0, przypisz wyrażeniu wartość –1, w przeciwnym przypadku (zmienna $liczba większa lub równa 0) przypisz wyrażeniu wartość 1. Ponieważ zmiennej $liczba przypisaliśmy wcześniej wartość 100, wartością całego wyrażenia będzie 1 i ta właśnie wartość zostanie przypisana zmiennej $wynik.

Pętle

Pisząc różnego rodzaju programy, programiści często spotykają się z sytuacją, w której program musi wykonać określoną z góry (lub nie) ilość razy ten sam kod — często ma to miejsce podczas operacji na tablicach. Oczywiście można by przepisać potrzebne linie kodu tyle razy, ile jest to wymagane, ale nie zawsze (prawie nigdy…) znamy ilość powtórzeń danego fragmentu kodu lub ilość ta jest bardzo duża.

Do rozwiązywania problemów związanych z koniecznością wielokrotnego, cyklicznego wykonywania tego samego fragmentu kodu służą pętle, czyli instrukcje iteracyjne. Pętla jest specjalną konstrukcją językową, dzięki której można uzyskać właściwą liczbę powtórzeń wykonania danego fragmentu kodu. Język PHP dostarcza cztery różne pętle. Każdy z czterech rodzajów pętli znajduje zastosowanie w różnych przypadkach, jednak przy pewnych modyfikacjach można używać różnych pętli wymiennie. Sztuka polega na tym, aby dobrać w trakcie tworzenia kodu taką pętlę, której zastosowanie ułatwi (a nie utrudni — na upartego można zawsze stosować jeden rodzaj pętli) rozwiązanie danego problemu.

Pętla for

Zacznijmy od pętli for, której pierwotnym zastosowaniem były operacje powtarzane z góry znaną ilość razy. W języku PHP — tak jak i w języku C, C++ czy Java — pętla for może mieć bardziej skomplikowane (nawet nietypowe) zastosowanie, na razie jednak skupimy się na jej podstawowym zastosowaniu.

Pętla for ma postać:

for (wyrażenie_inicjujące; wyrażenie_warunkowe; wyrażenie iteracyjne)

instrukcja;

Pojęcia użyte w powyższej definicji oznaczają:

- wyrażenie_inicjujące — wyrażenie, którego zadaniem jest zwykle ustalenie warunku początkowego (warunków początkowych) pętli;
- wyrażenie_warunkowe — od jego wartości (true lub false) zależy, czy pętla będzie wykonywana dalej, czy też zostanie zakończona;
- wyrażenie_iteracyjne — zwykle jest to instrukcja modyfikująca licznik pętli, czyli wpływająca na kontynuację lub zakończenie pętli (licznik pętli jest często sprawdzany w wyrażeniu_warunkowym);
- instrukcja — instrukcja (prosta lub złożona) stanowiąca tzw. ciało pętli — jest to kod, który będzie powtarzany w pętli.

Pętla for działa według następującego algorytmu: przed rozpoczęciem wykonywania kodu pętli wartościowane jest wyrażenie_inicjujące. Dalej sprawdzana jest wartość wyrażenia_warunkowego — jeżeli jest to true, pętla może być kontynuowana, w przeciwnym razie wykonanie pętli zostaje zakończone i sterowanie przekazywane jest do następnej instrukcji po pętli. Po sprawdzeniu wyrażenia_warunkowego i uzyskaniu wartości true wykonywana jest instrukcja stanowiąca ciało pętli. Po wykonaniu tej części kodu wartościowane jest wyrażenie_iteracyjne (najczęściej jest to modyfikacja licznika pętli) i znowu obliczane jest wyrażenie_warunkowe, którego rolę już znamy — jeżeli wyrażenie to zwróci wartość false, pętla zostanie zakończona, jeśli true — wykonane zostanie ciało pętli itd.

Należy zauważyć, że wyrażenie_inicjujące wartościowane jest tylko raz, przed wykonaniem pętli.

Na listingu 4.13 przedstawiono przykład użycia pętli for do wypisania zawartości jednowymiarowej tablicy liczb całkowitych, zapełnionej wcześniej wartościami od 1 do 4.

Listing 4.13. Przykład użycia pętli for

```
<?php

 // Tworzymy tablicę poprzez przypisanie jej kolejnych elementów

 $tab[0] = 1;

 $tab[1] = 2;
```

```
  $tab[2] = 3;

  $tab[3] = 4;



  // Wypisujemy zawartość tablicy

  for ($i = 0; $i < 4; $i++) {

    print($tab[$i].'<br />');

  }

?>
```

Dołączenie znacznika
 do elementu tablicy wypisywanego przez instrukcję print ma na celu eleganckie wyświetlenie wyniku — jest to swego rodzaju kosmetyka.

Pętla while

Dużo prostszą składniowo od pętli for jest pętla while. Jest ona przeznaczona do wykonywania operacji, w przypadku których nie wiemy, nie jesteśmy pewni lub nie obchodzi nas, ile razy kod stanowiący ciało pętli ma zostać wykonany.

Pętla while ma postać:

```
while (warunek)

  instrukcja;
```

warunek i instrukcja mają w tym przypadku podobne znaczenie, jak w instrukcji warunkowej if. Pętla while działa więc według następującego algorytmu: najpierw sprawdzana jest wartość wyrażenia stanowiącego warunek. Jeżeli wyrażenie to generuje wartość true, wykonywane jest ciało pętli (instrukcja). Po każdorazowym wykonaniu ciała pętli zawsze sprawdzany jest warunek — jeżeli wyrażenie warunkowe zwróci wartość false, pętla zostaje zakończona (ciało pętli nie będzie wykonywane) i sterowanie zostaje przekazane do następnej instrukcji po pętli. Krótko (choć może za dużo żargonu): pętla while wykonuje się tak długo, jak długo warunek jest spełniony.

Ważne jest umieszczenie w ciele pętli while kodu, który będzie bezpośrednio wpływał na zmianę wyniku wyrażenia warunkowego — chodzi o to, że jeżeli warunek nie będzie modyfikowany i cały czas będzie generował wartość true, wpadniemy w pułapkę pętli nieskończonej. W przypadku dynamicznej strony WWW efekt ten będzie się przejawiał komunikatem o błędzie serwera związanym z upływem czasu przeznaczonego na odpowiedź (ang. timeout), natomiast w przypadku aplikacji samodzielnej (mała dygresja, choć — biorąc pod uwagę PHP-GTK — być może całkiem na miejscu) użytkownik będzie miał wrażenie, że program się „zawiesił”.

Na listingu 4.14 przedstawiono modyfikację skryptu z listingu 4.13 — zamiast pętli for użyta została pętla while.

Listing 4.14. Przykład zastosowania pętli while

```
<?php

  // Tworzymy tablicę poprzez przypisanie jej kolejnych elementów

  $tab[0] = 1;

  $tab[1] = 2;
```

```
  $tab[2] = 3;

  $tab[3] = 4;



  // Wypisujemy zawartość tablicy

  $i = 0;

  while ($i < 4) {

    print($tab[$i++].'<br />');

  }



?>
```

Zmienna $i, której przypisywana jest wartość 0 przed rozpoczęciem pętli, jest licznikiem pętli — przy każdej iteracji (jednokrotnym wykonaniu ciała pętli) jest ona inkrementowana (jej wartość zwiększana jest o 1). Wartość tej zmiennej sprawdzana jest w wyrażeniu warunkowym — jeśli jest ona mniejsza od 4, ciało pętli zostaje wykonane. Gdyby wartość tej zmiennej nie była modyfikowana, mielibyśmy przykład pętli nieskończonej.

Warto zwrócić uwagę na instrukcję print, a zwłaszcza na odwołanie do elementu tablicy: $tab[$i++]. Odwołanie to składa się z dwóch faz: najpierw pobierany jest z tablicy element o indeksie $i, a następnie zmienna $i jest zwiększana o jeden — jest to typowy przykład zastosowania postinkrementacji. Gdyby zastosować zapis postaci $tab[++$i] (preinkrementacja), to najpierw wartość zmiennej $i byłaby zwiększana o jeden, a potem dopiero zostałby odczytany element tablicy o indeksie $i.

Pętla do…while

Pętla do…while jest modyfikacją pętli while — zmiana polega na tym, że warunek zakończenia pętli sprawdzany jest po każdorazowym wykonaniu ciała pętli. Innymi słowy, jeżeli porównamy pętlę while i pętlę do…while, obie mające identyczne ciało i ten sam warunek zakończenia, to w przypadku gdy warunek nie będzie spełniony, pętla while nie wykona się ani razu, natomiast pętla do… while wykona kod stanowiący ciało pętli dokładnie jeden raz.

Pętla do…while ma postać:

```
do {

  instrukcja;

} while (warunek);
```

Tak jak w przypadku poprzednich pętli przykład z listingu 4.15 będzie modyfikacją kodu z listingu 4.13.

Listing 4.15. Przykład użycia pętli do… while

```
<?php

  // Tworzymy tablicę poprzez przypisanie jej kolejnych elementów

  $tab[0] = 1;
```

```
$tab[1] = 2;

$tab[2] = 3;

$tab[3] = 4;



// Wypisujemy zawartość tablicy

$i = 0;

do {

  print($tab[$i++].'<br />');

} while ($i < 4);



?>
```

Pętla do… while nie cieszy się zbytnio popularnością wśród programistów — bywa używana głównie w przypadkach, gdy implementowany algorytm wykorzystuje główną właściwość tej pętli (wykonanie ciała przed wartościowaniem i sprawdzeniem wyrażenia warunkowego).

Pętla foreach

Jak wiadomo, w języku PHP występują zarówno tablice „klasyczne” (indeksowane kolejnymi liczbami całkowitymi nieujemnymi), jak i tablice asocjacyjne, indeksowane kluczem, który może być ciągiem znaków. O ile z przeglądaniem elementów tablicy zwykłej i operowaniem nimi nie ma najmniejszych problemów — wiemy, jak działają opisane wcześniej pętle i umiemy je wykorzystać — o tyle w przypadku tablic asocjacyjnych próba wykonania kodu jak dla tablicy zwykłej zakończy się niepowodzeniem.

Specjalnie dla tablic w PHP 4 wprowadzona została pętla foreach. W poprzednich wersjach PHP (co zresztą pozostawiono do dziś w celu zachowania wstecznej kompatybilności) do przeglądania tablicy trzeba było stosować np. pętlę while z odpowiednio przygotowanym wyrażeniem warunkowym i pamiętać o ustawianiu kursora tablicy na jej początku przed rozpoczęciem pętli.

Pętla foreach może występować w jednej z dwóch postaci:

```
foreach (wyrażenie_tablicowe as wartość)

  instrukcja;
```

oraz:

```
foreach (wyrażenie_tablicowe as klucz => wartość)

  instrukcja;
```

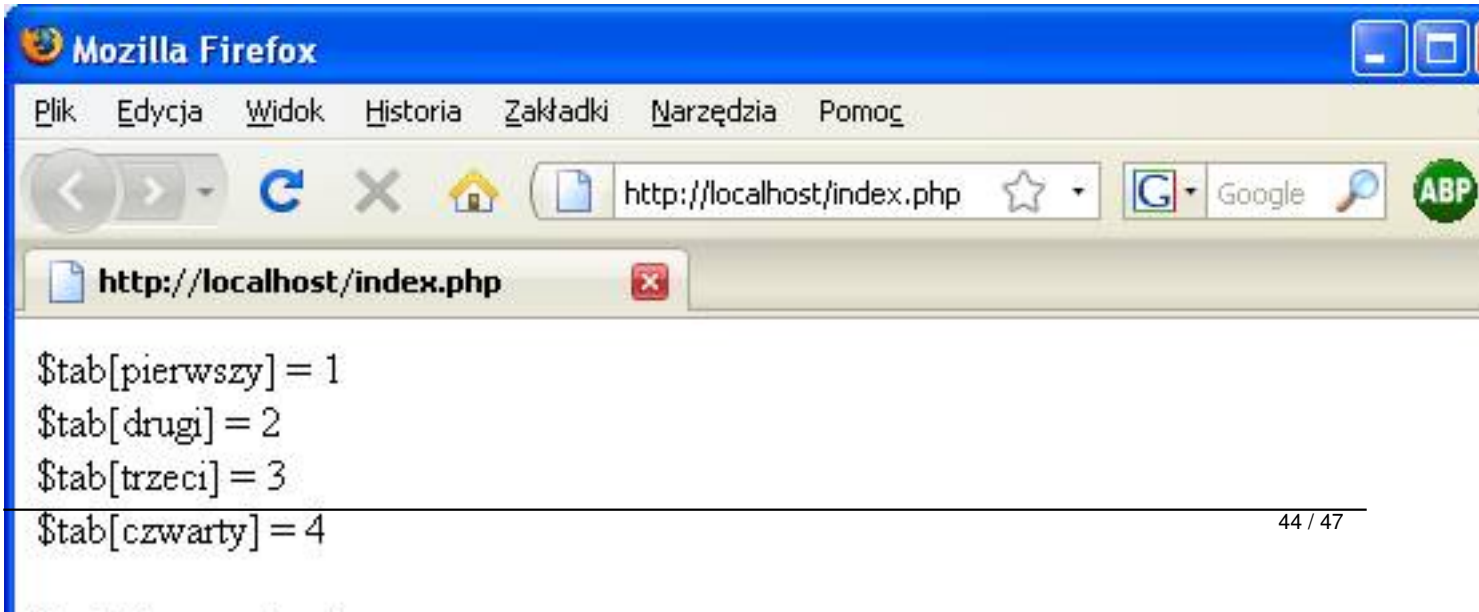
wyrażenie_tablicowe jest najczęściej po prostu zmienną tablicową (podaje się tutaj identyfikator tablicy), klucz to zmienna, w której umieszczony zostanie klucz aktualnie przeglądanego elementu tablicy, natomiast wartość to zmienna, w której umieszczona zostanie wartość aktualnie przeglądanego elementu. Jak wynika z definicji pętli foreach, pobieranie klucza nie jest konieczne. Zaletą zastosowania tego typu pętli jest brak potrzeby obliczania ilości wykonań — pętla ta wykona się tyle razy, ile będzie elementów w tablicy.

Listing 4.16 zawiera przykład użycia pętli foreach dla tablicy asocjacyjnej. Dodatkowo, dla porównania, kod z listingu 4.16 zawiera pętlę while, realizującą dokładnie to samo zadanie, co w tym przypadku pętla foreach.

Listing 4.16. Przykład użycia pętli foreach

```
<?php

   $tab = array('pierwszy' => 1,

              'drugi'    => 2,

              'trzeci'   => 3,

              'czwarty'  => 4);


 // Wypisujemy zawartość tablicy metodą tradycyjną

 reset($tab); // ustawienie kursora tablicy na jej początku

 while (list($klucz, $wartosc) = each($tab))

   echo '$tab['.$klucz.'] = '.$wartosc.'<br />';


 echo '<br />';


 // Wypisujemy zawartość tablicy za pomocą pętli foreach

 foreach($tab as $klucz => $wartosc)

   echo '$tab['.$klucz.'] = '.$wartosc.'<br />';


?>
```

Efekt działania skryptu z listingu 4.16 został przedstawiony na rysunku 4.4.



Rysunek 4.4. Efekt wykonania skryptu z listingu 4.16

Funkcje

Funkcje są bardzo ważnym elementem języka PHP — na tyle ważnym, że już we wcześniejszych przykładach i opisach z trudem unikano tego określenia. W funkcjach umieszcza się fragment kodu, który może być używany w różnych momentach i miejscach programu, czasem nawet w różnych kontekstach. Funkcja stanowi swego rodzaju czarną skrzynkę — nie interesuje nas najczęściej, co jest w środku, ważne jest, jakie informacje funkcji przekażemy i jaką wartość funkcja zwróci. Funkcje w PHP trochę przypominają ich odpowiedniki w matematyce — funkcja przekształca zbiór argumentów w zbiór wartości.

Funkcję definiuje się raz, najczęściej na początku skryptu, a później wywołuje się ją dowolną ilość razy, w zależności od potrzeb i konieczności.

Ogólna definicja funkcji ma postać:

```
function nazwa (argument1, argument2, /* ... */, argumentN) {

  instrukcja;

  return wartość_zwracana;

}
```

Nazwa funkcji może się składać z małych lub dużych liter, cyfr i znaków podkreślenia, natomiast musi się zaczynać od litery lub znaku podkreślenia. W przeciwieństwie do zmiennych nazwa funkcji nie musi zaczynać się od znaku $.

Argumenty funkcji są opcjonalne — można definiować funkcje bezargumentowe. Opcjonalne jest również użycie instrukcji return. Instrukcja ta ma za zadanie zakończyć wykonywanie funkcji i zwrócić wartość obliczoną przez funkcję (w instrukcji, która może być pojedynczą instrukcją lub zbiorem instrukcji), natomiast można tworzyć funkcje, które nie będą zwracać żadnych wartości (jeśli użytkownik programował kiedyś w języku Pascal, choć język ten już dawno stracił znaczenie w zastosowaniach produkcyjnych, to tego typu funkcje kojarzą mu się zapewne z procedurami). Na listingu 4.17 przedstawiono przykładową funkcję obliczającą wartość bezwzględną liczby przekazanej jako argument i jej wywołanie.

Listing 4.17. Przykładowa funkcja i jej wywołanie

```
<?php

  // Funkcja obliczająca wartość bezwzględną argumentu

  function rwAbs($liczba) {

    if ($liczba < 0)

      $liczba = -$liczba;

    return $liczba;

  }
```

```
  // Wywołanie funkcji - zmienna $wynik zawiera wartość zwróconą przez funkcję rwAbs

  $wynik = rwAbs(-15);



  echo $wynik;

?>
```

W ciele przykładowej funkcji posługujemy się argumentem $liczba tak, jakby to była zwykła zmienna — możemy tak zrobić, ponieważ w momencie wywołania funkcji wartość -15 zostaje skopiowana do nowo utworzonej wewnątrz funkcji zmiennej $liczba. W tym przypadku mamy zatem do czynienia z tzw. przekazywaniem argumentów (parametrów) przez wartość (kopiowanie wartości).

Istnieje jednak drugi rodzaj przekazywania argumentów funkcji, określany jako przekazywanie przez referencję. Moglibyśmy go użyć, gdy jako argument przekazywana była zmienna (przypomina to przekazywanie argumentów przez adres w C) — wtedy musielibyśmy zmienić ciało funkcji, ponieważ manipulacja zmienną $liczba, a w szczególności zmiana jej wartości, spowodowałaby zmianę wartości zmiennej zewnętrznej, przekazanej jako argument (parametr) funkcji. Przykład przekazywania parametru przez referencję (modyfikacja poprzedniego przykładu) został pokazany na listingu 4.18.

Listing 4.18. Przykład funkcji (przekazywanie parametrów przez referencję)

```
<?php

  // Funkcja obliczająca wartość bezwzględną argumentu

  function rwAbs(&$liczba_arg) {

    $liczba = $liczba_arg;

    if ($liczba < 0)

      $liczba = -$liczba;

    return $liczba;

  }



  // Wywołanie funkcji - zmienna $wynik zawiera wartość zwróconą przez funkcję rwAbs

  $a = -15;

  $wynik = rwAbs($a);



  echo $wynik."<br />";

?>
```

Argumenty funkcji mogą mieć zdefiniowane przez użytkownika wartości domyślne argumentów. Mechanizm ten umożliwia wywołanie funkcji tak, jakby nie przyjmowała żadnych argumentów, oczywiście w przypadku gdy nie ma potrzeby zmiany wartości argumentu. Przykład funkcji z argumentem z wartością domyślną został pokazany na listingu 4.19.

Listing 4.19. Przykład funkcji z parametrem z wartością domyślną

```
<?php

  function pokaz($napis = "To jest napis") {

    echo $napis;

  }



  pokaz();

?>
```

Tematyka funkcji nie została w tym punkcie wyczerpana — warto przejrzeć dokumentację PHP.

Warto stosować funkcje we własnych skryptach — pozwalają one uporządkować kod i są pierwszym krokiem do pełnej modularyzacji programu. Poza tym w razie jakiegoś błędu w ciele funkcji poprawki wprowadzane są tylko raz i w jednym miejscu. **Funkcje predefiniowane**

PHP dostarcza dużą liczbę predefiniowanych funkcji, które są bardzo przydatne, wręcz niezbędne do tworzenia dobrych, działających aplikacji WWW. Funkcje te zostały podzielone na grupy tematyczne — ułatwia to znacznie odnalezienie funkcji wykonującej żądane zadanie.

Pełną listę funkcji dostępnych w PHP (i ich szczegółowy opis) zawiera oczywiście dokumentacja PHP (http://www.php.net).