Zmienne i tablice - Podstawy języka PHP

W tym rozdziale dowiesz się, jak PHP przechowuje dane oraz pozwala na zarządzanie nimi.

Dane

PHP, jak każdy inny język programowania, operuje na danych. Niektóre z nich są zapisane na sztywno w skrypcie. Niemniej każda rzecz, która reprezentuje jakąkolwiek informację, zwana jest wyrażeniem. Oto prosty przykład:

```
8
```

To jest wyrażenie liczbowe reprezentujące liczbę całkowitą.

```
6.454
```

To jest wyrażenie liczbowe reprezentujące ułamek, czyli liczbę zmiennoprzecinkową.

```
0x6F44
```

To jest wyrażenie reprezentujące liczbę zapisane w systemie szesnastkowym.

```
07647
```

To jest wyrażenie reprezentujące liczbę zapisane w systemie ósemkowym.

'To jest tekst bez znaków specjalnych'
"To też jest tekst, ale ze znakami specjalnymi"

Powyżej mamy dwa wyrażenia reprezentujące tekst. Pomiędzy nimi istnieje istotna różnica. Apostrofy korzystają z jednego tylko znaku specjalnego: \' - pozwala on oznaczyć wewnątrz tekstu znak apostrofu. Gdybyśmy zapomnieli o poprzedzającym backslashu, PHP uznałby, że w tym momencie kończy się wyrażenie tekstowe i dalej jest już normalny skrypt.

Cudzysłów posiada więcej takich kodów formatujących:

```
 "\n - tak robimy zejście do nowej linijki w systemach Linux"
"\r - tak w systemach Mac"
"\r\n - a tak w systemach operacyjnych Windows"
"Wyświetlimy cudzysłów: \" ..."
"Wstawimy tabulator: \t"
```

W obu przypadkach, aby wyświetlić backslash, należy napisać go podwójnie:

```
 'Oto backslash: \\ '
```

Otrzymamy dzięki temu tekst:

```
 Oto backslash: \
```

Oto bardziej skomplikowane wyrażenie:

```
 5 + 7
```

Reprezentuje ono sumę dwóch mniejszych wyrażeń.

```
'Tekst A '.'Tekst B'
```

Oto wyrażenie będące połączeniem dwóch mniejszych wyrażeń tekstowych. Znaki kropka oraz plus to tzw. **operatory**. Wykonują one pewne operacje na dwóch innych wyrażeniach i zwracają jakąś wartość, zatem same także stają się wyrażeniem. Podobnie, jak w matematyce, obowiązuje pewna kolejność ich wykonywania, a zmieniać ją możemy za pomocą nawiasów:

```
5 * (6 + 8)
```

Wyrażeniem nazywamy dowolny element języka PHP reprezentujący jakąś informację, którą można przetwarzać

Pamiętaj, że PHP potrafi automatycznie rozpoznawać typ wyrażenia i w razie potrzeby odpowiednio go konwertować. Dla przykładu instrukcja *echo* wymaga danych tekstowych. W skrypcie:

```
<?php

echo 7;

?>
```

Interpreter dokona konwersji liczby na tekst, a dopiero potem spowoduje jego wyświetlenie.

Funkcje

Informatyka wiele zawdzięcza matematyce. W programowaniu występuje wiele pojęć zaczerpniętych bezpośrednio od królowej nauk. Jednym z nich jest funkcja, do której możemy wprowadzać parametry, a w zamian otrzymujemy jakiś wynik. Poniższy skrypt będzie pierwszym "naprawdę" dynamicznym, jaki stworzymy. Skorzystamy w nim z dwóch funkcji, aby wyświetlić aktualny czas:

```
<?php

echo 'Dzisiaj mamy: '.date('d.m.Y').'<br/>';
echo 'Od 1.1.1970 minęło: '.time().' sekund';

?>
```

Jak widać, składnia funkcji jest następująca: *nazwaFunkcji(parametry)*. Jeśli funkcja nie posiada parametrów, nawiasy są puste. Jeżeli jest ich więcej, niż jeden, oddzielamy je od siebie przecinkiem. Zaś sam parametr jest niczym innym, jak... wyrażeniem. Jest nim także funkcja, dlatego możemy ją wpleść w nasz tekst za pomocą operatora kropki.

Uwaga!
echo oraz jego synonim **print** nie są funkcjami ani wyrażeniami! Są to instrukcje języka PHP i to tłumaczy, dlaczego nie musimy stosować przy nich nawiasów.

Zmienne

Innym pojęciem matematycznym jest zmienna reprezentująca informację nieznaną w trakcie pisania skryptu. Jest to taki pojemnik, do którego możemy w trakcie wykonywania skryptu wstawiać wszelkie informacje, zapamiętując je w ten sposób. Bez zmiennych nie można mówić o jakimkolwiek przetwarzaniu danych, ponieważ komputer musi mieć jakąś możliwość umieszczania gdzieś wyników obliczeń oraz przechowywania niezbędnych programom danych.

Każda zmienna posiada własną, unikalną nazwę, która jednoznacznie ją identyfikuje. Język PHP wymaga, aby zaczynała się ona od znaku dolara, a następnie od litery lub podkreślenia. Dalsze znaki mogą być także cyframi. Można stosować duże litery, ale dla interpretera ma to istotne znaczenie. *$zmienna* i *$Zmienna* to dwie różne zmienne. Przykłady prawidłowych nazw zmiennych:

```
$a, $b, $foo, $_50, $_foo, $moja_zmienna, $mojaZmienna3
```

Przykłady nieprawidłowych nazw:

$5, $

PHP zezwala na używanie w nazwach zmiennych także znaków o kodach od 128 do 255, wśród których znajdują się m.in. polskie litery.

Aby przypisać wartość do zmiennej, należy ułożyć wyrażenie z operatorem **=**. Po lewej stronie umieszczamy naszą zmienną, a po prawej inne wyrażenie określające wstawianą wartość. Oto, jak wygląda to w praktyce:

```
<?php

// inicjujemy zmienna $czas aktualnym czasem w sekundach od 1.1.1970
  $czas = time();

echo 'Od 1.1.1970 minęło '.$czas.' sekund<br/>';
echo 'Od 1.1.1970 minęło '.($czas / 60).' minut<br/>';
echo 'Od 1.1.1970 minęło '.($czas / 3600).' godzin';

?>
```

W powyższym przykładzie stworzyliśmy zmienną *$czas*, wprowadzając do niej aktualny czas. Następnie wykorzystaliśmy ją w obliczeniach. Poznaliśmy w ten sposób jedno z zastosowań zmiennych. Zachowaliśmy w nich wynik działania jednej sekcji programu, aby potem używać go wielokrotnie gdzie indziej bez konieczności każdorazowego odwoływania się do funkcji i zbędnego liczenia po sto razy tego samego. Wprawdzie pobieranie aktualnego czasu nie jest czasochłonną operacją, ale na razie jesteśmy jeszcze na początkowym etapie nauki i musimy dbać o prostotę przykładów.

PHP, w przeciwieństwie do innych języków programowania, ma bardzo liberalne reguły stosowania zmiennych. W ogóle nie trzeba ich nigdzie deklarować, a interpreter sam na bieżąco dopasuje typ informacji do naszych potrzeb. Dana zmienna jest tworzona podczas pierwszego jej wykorzystania w skrypcie. Jest sporo sytuacji, w których zachowanie to jest pożądane, lecz normalnie może utrudnić ono pracę. Wyobraź sobie taką sytuację: programista potrafi najczęściej bardzo szybko pisać i od czasu do czasu zdarza mu się wcisnąć dwa klawisze w złej kolejności. Powstaje literówka. Jeżeli zostanie ona popełniona podczas wpisywania nazwy, PHP w teorii nie zgłosi tego jako błędu i programista będzie musiał spędzić dużo czasu na jej odnalezienie. Słowo "teoria" nie znalazło się tu przypadkowo. Podczas instalowania PHP wspominaliśmy o poziomach raportowania błędów. Im wyższy poziom, tym większej ilości rzeczy czepia się PHP. Na poziomie *E_ALL* zdefiniowanym w rekomendowanym pliku *php.ini* takie beztroskie podejście do zmiennych nie jest tolerowane. Tutaj PHP wymaga już, aby podczas pierwszego użycia zmiennej została jej przypisana jakaś wartość, ponieważ inaczej otrzymamy powiadomienie (ang. *notice*) o próbie odwołania się do nieistniejącej zmiennej. Popatrzmy sobie na ten przykład:

```
<?php

echo $a * 16 + 5;

?>
```

Zmienna *$a* nie została tu wcześniej zadeklarowana. Otwórz swój plik *php.ini*, odnajdź dyrektywę *error_reporting* i zmień jej wartość na *E_ALL | ~E_NOTICE*. Wyłączysz w ten sposób wyświetlanie powiadomień. Zrestartuj serwer i uruchom powyższy skrypt. Wynikiem powinno być "5". PHP bez pytania podstawił do *$a* wartość neutralną 0. Przywróć teraz poprzedni poziom (*E_ALL | E_STRICT*) i ponownie uruchom ten skrypt. Oprócz wyniku, ujrzysz też komunikat:

```
Notice: Undefined variable: a in D:\Serwer\www\katalog\twojskrypt.php on line 3
```

Sytuację można rozwiązać, ręcznie inicjując zmienną *$a* wartością 0:

```
<?php
```

```
$a = ;

echo $a * 16 + 5;

?>
```

Zalecane jest, aby wszystkie skrypty pracowały bezproblemowo przy włączonych powiadomieniach. Jeżeli zajdzie sytuacja, że odwołanie się do zmiennej, która może jeszcze nie istnieć, jest potrzebne, istnieje kilka sposobów "oszukania" PHP, lecz poznamy je w dalszej części podręcznika.

Początkujący programiści mają tendencję do tworzenia dużej liczby tzw. *zmiennych tymczasowych*, które nie wnoszą absolutnie niczego do programu poza wydłużeniem kodu i zmniejszeniem wydajności. Po każdym etapie przetworzenia jakiejś informacji, umieszczana jest ona w nowej zmiennej. Takie podejście jest nieprawidłowe. Oto przykłady "złych" skryptów:

```
<?php

$tekst = 'To jest jakiś tekst';
$tekstMaly = strtolower($tekst);
$tekstBezpieczny = addslashes($tekstMaly);
echo $tekstBezpieczny;

?>
```

```
<?php

$format = 'd.m.Y';
echo date($format);

?>
```

W pierwszym skrypcie niepotrzebnie po każdym etapie przetwarzania tekstu tworzymy dla niego nową zmienną. Możemy to poprawić na dwa sposoby. Pierwszy:

```
<?php

$tekst = 'To jest jakiś tekst';
$tekst = strtolower($tekst);
$tekst = addslashes($tekst);
echo $tekst;

?>
```

Drugi, w którym w ogóle opuszczamy zmienne:

```
<?php

echo addslashes(strtolower('To jest jakiś tekst'));

?>
```

W drugim "złym" skrypcie w ogóle niepotrzebnie tworzymy zmienną; przecież format daty możemy wpisać bezpośrednio do funkcji.

```
<?php
echo date('d.m.Y');
?>
```

W tym jednak przypadku może być to uznane za kwestię dyskusyjną. Jeżeli nasz skrypt bardzo często będzie

formatować różne daty, a my będziemy chcieli mieć możliwość jej prostego konfigurowania, warto pokusić się o zapisanie formatu w jakiejś zmiennej. W ten sposób poprzez zmianę jej wartości w jednym miejscu zostanie ona uwzględniona w całym skrypcie.

Nauczenie się, kiedy warto użyć zmiennych, a kiedy nie, to kwestia praktyki. W niniejszym podręczniku będziemy zwracali na tę kwestię baczną uwagę. Jeśli zajdzie potrzeba użycia zmiennych tymczasowych - wyjaśnimy, dlaczego, bowiem całkowite rezygnowanie z ich użycia także może rodzić wiele problemów.

Typy zmiennych

W sekcji poświęconej rodzajom danych w PHP dowiedziałeś się, że istnieje pewne rozróżnienie na tekst i liczby. Skoncentrujemy się teraz na poznaniu większej ilości typów oraz pokazaniu, jak PHP dokonuje konwersji między nimi.

Istnieją trzy kategorie typów: *wielkości skalarne*, *typy złożone* oraz *typy specjalne*. Dokumentacja wymienia jeszcze jedną, lecz stworzoną na jej własne potrzeby do zaznaczania niektórych rzeczy. Powiemy o niej później.

Wielkości skalarne

Pierwszym typem skalarnym jest *liczba całkowita*. Jej angielskim określeniem jest *integer* albo *int*. Może być ona zapisana w trzech systemach liczbowych: dziesiętnym, szesnastkowym albo ósemkowym:

```
<?php
$a = 1234; // liczba całkowita
  $a = -123; // liczba całkowita ujemna
  $a = 0123; // zapis ósemkowy (odpowiednik dziesiętnego 83)
  $a = 0x1A; // zapis szesnastkowy (odpowiednik dziesiętnego 26)
 ?>
```

Możemy także korzystać z wartości ułamkowych zwanych *liczbami zmiennoprzecinkowymi* (ang. *floating point numbers* albo skrótowo *float*). Przy ich zapisywaniu obowiązują reguły języka angielskiego, więc części całkowite od ułamkowych oddzielamy za pomocą kropki. Także i tu mamy do wyboru kilka sposobów zapisu:

```
<?php
$a = 1.234;
$a = 1.2e3;
$a = 7E-10;
?>
```

Specjalnie wyróżniony został tzw. typ logiczny (*boolean*) przyjmujący jedynie wartości **FALSE** i **TRUE**. Jest on bardzo ważny, ponieważ wiele funkcji generuje właśnie w nim rezultat, czy operacja się powiodła. Wyrażenia porównawcze także generują wartości logiczne.

Ostatnim z typów skalarnych jest ciąg tekstowy (ang. *string*). Zdążyliśmy już powiedzieć nieco o nim, np. że istnieją dwie składnie zapisywania ciągów. Ta oparta na apostrofach posiada minimalny zestaw kodów formatujących pozwalających na wstawienie do tekstu innych apostrofów oraz ukośników wstecznych:

```
<?php

echo 'To jest tekst zapisany w apostrofach. Kody formatujące pozwalają
umieścić w tekście wyłącznie inne apostrofy: \' albo backslashe: \\. Wszystko inne,
np. \n zostanie wyświetlone jako zwyczajny tekst';

?>
```

Uruchom powyższy skrypt i porównaj sobie wynik z przeglądarki z treścią wpisaną w apostrofy. Spróbuj usunąć backslash sprzed apostrofu i zobacz, co się stanie. Skrypt się nie uruchomi, ponieważ wystąpił błąd składni. PHP myśli, że tekst kończy się już tutaj i dalej jest inna część algorytmu, co nie jest oczywiście prawdą. Więcej możliwości formatowania posiada tekst ograniczony cudzysłowami:

```
<?php

echo "To jest tekst zapisany w apostrofach. Za pomocą kodów formatujących możemy
umieszczać wiele rzeczy: \" \\ \n \r \$";

?>
```

Cudzysłowy zezwalają na "proste" umieszczanie w nich wartości zmiennych (zm. tymczasowa stworzona dla zilustrowania zagadnienia):

```
<?php

$czas = time();

echo "Aktualny czas w sekundach: $czas sek.";

?>
```

Jednak nie nadużywaj tego. Wstawianie zmiennych w ten sposób jest kilka razy wolniejsze, niż łączenie ich z ciągiem operatorem kropki.

```
<?php

// mozna takze użyć apostrof
  echo "Aktualny czas w sekundach: ".time()." sek.";

?>
```

Niektórzy początkujący programiści niezbyt rozumieją ideę tej możliwości i próbują wykorzystywać ciągi do wprowadzania wartości zmiennych jako parametrów do funkcji (zm. tymczasowa użyta dla zilustrowania zagadnienia):

```
<?php

$formatDaty = 'd.m.Y';
echo date("$formatDaty");

?>
```

Unikaj takiej konstrukcji, jak ognia. Choć PHP ją akceptuje, nie jest to prawidłowe użycie tej struktury języka. Więcej... przy złożonych typach powoduje zniekształcenie danych. Jeżeli spotkasz kogoś piszącego w ten sposób, poinformuj go o tym.

Inne typy

Typami złożonymi są w PHP tablice oraz obiekty. Tablice poznamy jeszcze w tym rozdziale, natomiast obiektami oraz samym programowaniem obiektowym w dalszej części podręcznika.

Istnieją jeszcze dwa typy specjalne: *resource* oraz *NULL*. Pierwszy z nich reprezentuje wszelkiego rodzaju połączenia z bazami danych, otwarte przez PHP pliki itd. Drugi to wartość pusta. Za jego pomocą możemy "za symulować", że zmienna nie istnieje lub nie zawiera wartości. Pojawia się w trzech sytuacjach:

- Do zmiennej przypisana została stała **NULL**.
- Do zmiennej nie została przypisana jeszcze żadna wartość (zgłaszane jest wtedy powiadomienie)
- Zmienna została zniszczona poleceniem **unset()**.

Przyjrzyjmy się wspomnianemu w trzecim punkcie poleceniu. Czasem jakąś zmienną trzeba zniszczyć. Najlepiej nadaje się do tego polecenie **unset()**:

```
<?php

$zmienna = 4856;

unset($zmienna);

echo $zmienna;

?>
```

Zmiennej już nie ma, dlatego polecenie **echo** nie pokaże nic.

Konwersja typów

PHP potrafi sam rozpoznać typ informacji przypisanej do zmiennej oraz automatycznie konwertować go w zależności od potrzeb. Przykładowo liczby ułamkowe użyte tam, gdzie potrzeba całkowitych, są zaokrąglane w górę do zera. Wartości logiczne mogą być reprezentowane cyframi 0 (**FALSE**) oraz 1 (**TRUE**). Ciągi tekstowe mogą być konwertowane do liczb, jeżeli pierwszy z ich znaków jest cyfrą. W przeciwnym przypadku PHP dobiera wartość 0.

Możemy sami wymusić konwersję typów:

```
<?php

// wyświetl liczbę całkowitą jako ułamek
  echo (float) 10;

?>
```

W nawiasie przed wartością piszemy angielską nazwę typu: *integer*, *int*, *string*, *boolean*, *float*, *double* (liczba zmiennoprzecinkowa mogąca przybierać znacznie większe wartości). Unikaj jakiejkolwiek konwersji typów złożonych: tablic, obiektów, zasobów, gdyż w każdym z tych przypadków informacje zostają całkowicie utracone; zamiast nich zwracana jest po prostu nazwa typu złożonego - to dlatego ostrzegaliśmy przed wprowadzaniem wartości zmiennych do funkcji przez cudzysłowy.

PHP posiada funkcję **gettype()** zwracającą nazwę aktualnego typu danych zawartych w zmiennej:

```
<?php

$a = 547;

echo 'Typ zmiennej $a to: '.gettype($a);

?>
```

Więcej o operatorach

Dotychczas poznałeś już kilka operatorów, np. **+** czy **=**. Jak zdążyłeś już zauważyć, po obu stronach takiego operatora stoją wyrażenia, na których wykonuje on operacje i zwraca wynik. Sam jest zatem wyrażeniem. Operatory mają określoną kolejność wykonywania wzorowaną na matematyce. Możemy ją naginać do własnych potrzeb, stosując nawiasy.

Oto wykaz najciekawszych operatorów na początek.

Operator	Nazwa	Składnia	Opis
/	Dzielenie	wyrażenie / wyrażenie	Reprezentuje wynik dzielenia. Drugie wyrażenie nie może być zerem.
%	Dzielenie modulo	wyrażenie % wyrażenie	Reprezentuje resztę z dzielenia. Drugie wyrażenie nie może być zerem.
*	Mnożenie	wyrażenie * wyrażenie	Reprezentuje iloczyn dwóch wyrażeń.
+	Dodawanie	wyrażenie + wyrażenie	Reprezentuje sumę dwóch wyrażeń.

-	Odejmowanie	wyrażenie - wyrażenie	Reprezentuje różnicę dwóch wyrażeń.
.	Łączenie (Konkatenacja)	wyrażenie . wyrażenie	Reprezentuje połączenie dwóch wyrażeń w ciąg tekstowy.
++	Inkrementacja (zwiększenie)	**$zmienna++**	Reprezentuje wartość zmiennej, a następnie zwiększa ją o 1.
++	Inkrementacja (zwiększenie)	**++$zmienna**	Reprezentuje wartość zmiennej powiększoną o 1.
--	Dekrementacja (zmniejszenie)	**$zmienna--**	Reprezentuje wartość zmiennej, a następnie zmniejsza ją o 1.
--	Dekrementacja (zmniejszenie)	**--$zmienna**	Reprezentuje wartość zmiennej pomniejszoną o 1.

Zwróć uwagę na cztery ostatnie pozycje. Wyszczególnione zostały tam tzw. operatory jednoargumentowe, czyli takie, które operują wyłącznie na jednym wyrażeniu. W dodatku koniecznie musi być ono zmienną, ponieważ modyfikują one jej wartość, powiększając lub zmniejszając o jeden. Każdy z tych operatorów został podany podwójnie, ponieważ w zależności od tego, czy postawimy go przed, czy po zmiennej, otrzymamy nieco inne rezultaty. Porównaj:

```
<?php
// najpierw składnia $zmienna++

$zmienna = 5;

echo 'Stan 1: '.($zmienna++).'<br/>';
echo 'Stan 2: '.$zmienna.'<br/><br/>';

// teraz składnia ++$zmienna
   echo 'Restart zmiennej...<br/>';
$zmienna = 5;

echo 'Stan 1: '.(++$zmienna).'<br/>';
echo 'Stan 2: '.$zmienna.'<br/>';

?>
```

Skrypt ten wygeneruje nam kilka linijek:

```
Stan 1: 5
Stan 2: 6

Restart zmiennej...
Stan 1: 6
Stan 2: 6
```

Przeanalizuj wyniki działania. Okazuje się, że w składni *$zmienna++* najpierw dostajemy wartość zmiennej, a dopiero potem zwiększamy ją o jeden (dlatego zmiana widoczna jest dopiero w drugim stanie). *++$zmienna* najpierw powiększa, potem zwraca, w efekcie czego otrzymujemy w obu stanach liczbę "6". Identyczna zasada obowiązuje operator --.

Zajmijmy się teraz przypisywaniem danych do zmiennej. Wiemy już, że operator przypisania po lewej stronie wymaga zmiennej, po prawej wyrażenia, którego wartość trzeba w niej umieścić. Skoro operator, jest to też wyrażenie, lecz jaką wartość ono reprezentuje. Okazuje się, że tę, która jest przypisywana. Możemy wobec tego zastosować sprytną sztuczkę, o której wbrew pozorom wie niezbyt wielu programistów. Zainicjujmy pięć zmiennych naraz tą samą wartością:

```
<?php

$a = $b = $c = $d = $e = 5;

?>
```

PHP najpierw przypisze "5" do zmiennej $e, zwracając jednocześnie "5" tak, by mogło być ono przypisane do $d, potem do $c, $b i na końcu $a. W ten sposób jednym wielkim wyrażeniem zainicjowaliśmy pięć zmiennych naraz.

Poznany już operator przypisania nie jest jedynym, jaki istnieje w PHP. Aby ułatwić modyfikację wartości zmiennych o liczby inne niż jeden, stworzono całą gamę operatorów łączących w sobie przypisywanie oraz jakąś operację matematyczną. Oto i one.

Operator	Składnia	Równoważna postać	Opis
/=	$zmienna /= wyrazenie	$zmienna = $zmienna / wyrażenie	Dzieli zmienną przez wyrażenie i umieszcza w niej wynik. Wyrażenie nie może być zerem.
%=	$zmienna %= wyrazenie	$zmienna = $zmienna / wyrażenie	Umieszcza w zmiennej resztę z dzielenia tej zmiennej przez wyrażenie, które oczywiście nie może być zerem.
*=	$zmienna *= wyrazenie	$zmienna = $zmienna * wyrażenie	Mnoży zmienną przez wyrażenie i zapisuje w niej wynik.
+=	$zmienna += wyrazenie	$zmienna = $zmienna + wyrażenie	Dodaje do zmiennej wyrażenie i zapisuje w niej wynik.
-=	$zmienna -= wyrazenie	$zmienna = $zmienna - wyrażenie	Odejmuje od zmiennej wyrażenie i zapisuje w niej wynik.
.=	$zmienna .= wyrazenie	$zmienna = $zmienna * wyrażenie	Dołącza do zmiennej tekstowej nowy fragment.

Te operatory po prostu skracają zapis i czynią go czytelniejszym. Warto o nich pamiętać szczególnie przy operacjach na ciągach tekstu, kiedy nasz algorytm składa jakąś treść, doczepiając kolejne jej partie do wybranej zmiennej:

```
<?php

$tekst = 'Litwo, ojczyzno moja! Ty jesteś jak zdrowie<br/>';
$tekst .= 'Ile Cię trzeba cenić, ten tylko się dowie<br/>';
$tekst .= 'Kto Cię stracił, dziś piękność twą w całej ozdobie<br/>';
$tekst .= 'Widzę i opisuję, bo tęsknię po tobie.<br/>';

echo $tekst;

?>
```

Rezultat:

Litwo, ojczyzno moja! Ty jesteś jak zdrowie
Ile Cię trzeba cenić, ten tylko się dowie
Kto Cię stracił, dziś piękność twą w całej ozdobie
Widzę i opisuję, bo tęsknię po tobie.

Kolejne partie tekstu były doklejane do właściwej zmiennej tak, że na końcu otrzymaliśmy spójny i kompletny tekst. Identycznie jest z pozostałymi operatorami. += dodaje wartość do zmiennej, -= odejmuje itd.

Na tym zakończymy na razie temat operatorów, lecz to jeszcze nie wszystko. Już niedługo zapoznamy się z operatorami służącymi do porównywania danych oraz podstawami operatorów logicznych. Będą nam one potrzebne przy omawianiu instrukcji warunkowych i pętli.

Tablice

Zaznajomimy się teraz z tablicami, pierwszym złożonym typem danych. Cofnijmy się do czasów szkoły podstawowej/gimnazjum na lekcję matematyki i przypomnijmy nasz pierwszy kontakt z pojęciem funkcji. Była tam mowa, że funkcję można przedstawiać w postaci tabelki:

x	0	1	2	3	4	5	6	7	8
y	5	3	8	7	9	24152		19	

Spróbujmy przenieść taką tabelkę w świat programowania. Widzimy, że wszystkie "igreki" są ze sobą powiązane, ponieważ odgrywają podobne role bycia wynikami funkcji. Gdyby to zapisać jako zupełnie osobne zmienne, mielibyśmy problem z późniejszym dostaniem się do nich. Popatrz na to w ten sposób: użytkownik wprowadza argument, a my mamy dla niego odnaleźć wartość spośród tych podanych. Skąd PHP może wiedzieć, że akurat ta grupa zmiennych jest ze sobą w jakiś sposób powiązania, a tym bardziej znać regułę wybierania "tej właściwej"? Na razie nie jest to w ogóle możliwe. Co nam pozostaje? Zapisać wszystkie wartości w strukturze zwanej **tablicą**.

Tablica to zmienna zbiorcza, grupująca pojedyncze elementy mające właściwości zmiennych, do których odwoływać możemy się za pomocą indeksów.

Wiemy już, że tablica jest zmienną, która grupuje sobie mniejsze zmienne opisywane przez ich indeksy (wartości *x* w naszej tabelce funkcji). Najważniejsze jest jednak to, że przy odwoływaniu się do nich, potrzebny nam indeks możemy określić zwyczajnym wyrażeniem! To oznacza, że zadanie wymienione akapit wyżej staje się realne. Zanim je zaprogramujemy, nieco informacji o składni tablic. Na początek utwórzmy pustą tablicę:

```
<?php

$tablica = array();

?>
```

Spróbujmy wprowadzić do niej wartości naszej funkcji:

```
<?php

$tablica = array( => 5, 3, 8, 7, 9, 24, 15, 2, 19);

?>
```

Początkowe "0" określa, od jakiej liczby zacząć numerowanie kolejnych elementów. Po strzałce wymieniamy ich wartości oddzielone przecinkiem. Teraz spróbujmy dostać się do jakiejś z nich.

```
<?php

$tablica = array( => 5, 3, 8, 7, 9, 24, 15, 2, 19);

echo 'Pod numerem 5 kryje się wartość '.$tablica[5];

?>
```

Pomiędzy nawiasami kwadratowymi wprowadzić musimy wyrażenie określające indeks tablicy, który chcielibyśmy odczytać. Możemy teraz pokusić się o napisanie skryptu losującego elementy:

```
<?php

$tablica = array( => 5, 3, 8, 7, 9, 24, 15, 2, 19);

echo 'Pod numerem 5 kryje się wartość '.$tablica[rand(, 8)];

?>
```

W tym skrypcie za wybranie indeksu tablicy odpowiada funkcja *rand()* zwracająca losową[1] wartość z tablicy.

Indeksy tablicy wcale nie muszą być numeryczne. PHP dopuszcza także tekstowe wersje. Mamy wtedy do czynienia z tablicami asocjacyjnymi.

```
<?php

$artykul = array(
```

```
'tytul' => 'Tytuł artykułu',
'data' => date('d.m.Y'),
'tresc' => 'To jest treść artykułu'
);

echo '<h1>'.$artykul['tytul'].'</h1>';
echo '<p>Napisany dnia '.$artykul['data'].'</p>';
echo '<p>'.$artykul['tresc'].'</p>';

?>
```

Mogą istnieć także tablice mieszane, w których występują zarówno indeksy tekstowe, jak i numeryczne. Pamiętaj, że każdy element tablicy zachowuje się, jak zwykła zmienna, dlatego także możesz przypisywać do niego dowolne wartości już po utworzeniu tablicy.

```
<?php

$tablica = array( => 5, 3, 8, 7, 9, 24, 15, 2, 19);

// modyfikuj losowy element tablicy
   $tablica[rand(, 8)] = 6;

echo '<pre>';
var_dump($tablica);
echo '</pre>';

?>
```

Najpierw przypisaliśmy do losowego elementu tablicy nową wartość: 6. Całość wyświetlamy funkcją *var_dump()*. Przydaje się ona przy poszukiwaniu błędów w skrypcie. Potrafi zaprezentować w czytelnej formie każdy typ danych, więc możemy za jej pomocą kontrolować, czy na danym etapie wykonywania rezultaty są takie, jakie być powinny. Podobne działanie ma funkcja *print_r()*.

Zobacz teraz, jak zachowuje się *$tablica*, kiedy próbujemy ją wywołać samodzielnie:

```
<?php

$tablica = array( => 5, 3, 8, 7, 9, 24, 15, 2, 19);

echo $tablica;

?>
```

Skrypt ten pokaże nam tylko jeden napis: *Array*, czyli... nazwę typu. Właśnie z tego powodu ostrzegaliśmy przed wprowadzaniem danych do funkcji jako *funkcja*("*$zmienna*");. Gdyby w zmiennej była tablica, do funkcji zamiast niej dotarłby napis *Array* i całość przestałaby działać. Osobną kwestią jest wydajność takiego zmuszania do konwersji do tekstu, a potem z powrotem na tekst właściwy.

Upewnij się, że rozumiesz już istotę działania tablic, gdyż bardzo przydadzą się nam one w następnym rozdziale.

Przypisy

[1] Tak naprawdę komputer nie potrafi losować liczb. Za całą tą zasłonką kryją się różne skomplikowane wzory matematyczne inicjowane najczęściej aktualnym czasem, dające wrażenie losowości wyników.