PHP5. Programowanie

Autorzy: Rasmus Lerdorf, Kevin Tatroe, Peter MacIntyre
Tłumaczenie: Małgorzata Czart, Łukasz Piwko, Konrad Woś
ISBN: 978-83-246-0613-9
Tytuł oryginału: Programming PHP, Second Edition
Format: B5, stron: 496

Dostępny nieodpłatnie PHP to jeden z najpopularniejszych języków programowania, dzierżący jednocześnie palmę pierwszeństwa wśród technologii wykorzystywanych do tworzenia dynamicznych witryn WWW i aplikacji internetowych. Według oficjalnych statystyk użyto go do napisania ponad 40% wszystkich aplikacji internetowych i zainstalowano na ponad 22 milionach serwerów. Źródłem jego popularności jest przede wszystkim jasna i czytelna składnia, ogromne możliwości, szybkość i wydajność oraz mnogość potencjalnych zastosowań. Najnowsze wcielenie PHP, oznaczone cyfrą 5, to w pełni obiektowy język programowania, który nadal zachował swoją prostotę i czytelność.

„PHP. Programowanie" to podręcznik przedstawiający wszystkie tajniki języka PHP5. W jasny i zwięzły sposób opisuje jego składnię oraz techniki programistyczne wykorzystywane przy tworzeniu aplikacji internetowych. Czytając go, poznasz zasady programowania obiektowego w PHP5, dowiesz się, w jaki sposób korzystać z informacji zapisanych w bazach danych, generować z poziomu skryptów pliki graficzne i dokumenty PDF oraz przetwarzać pliki XML. Każde z zagadnień zostało zilustrowane przykładami i wskazówkami pochodzącymi z najlepszego źródła, jest nim twórca języka PHP, Rasmus Lerdorf. Cennym dodatkiem do książki jest alfabetyczne zestawienie wszystkich funkcji PHP.

W książce poruszono między innymi:

- Strukturę leksykalną PHP
- Przetwarzanie łańcuchów znakowych i korzystanie z wyrażeń regularnych
- Programowanie obiektowe
- Zarządzanie sesjami i połączeniami HTTP
- Komunikacja z bazami danych
- Przetwarzanie grafiki
- Korzystanie z plików XML
- Zabezpieczanie aplikacji
- Tworzenie rozszerzeń dla PHP
- Programowanie w PHP dla Windows

Przekonaj się, dlaczego PHP jest tak niezwykle popularny i dołącz do grona jego użytkowników

Wydawnictwo Helion
ul. Kościuszki 1c
44-100 Gliwice
tel. 032 230 98 63
e-mail: helion@helion.pl

Spis treści

ROZDZIAŁ 3.

Funkcje

Funkcja to nazwany blok kodu wykonujący określone zadanie, najprawdopodobniej działający na zestawie przekazanych mu wartości zwanych *parametrami* i najprawdopodobniej zwracający jedną wartość. Funkcje zmniejszają ilość czasu potrzebną na wykonanie kompilacji, ponieważ niezależnie od tego, ile razy są wywoływane, są kompilowane tylko raz dla danej strony. Zwiększają one również niezawodność, gdyż pozwalają na korektę dowolnego błędu w pojedynczym miejscu zamiast we wszystkich, w których wykonywane jest dane zadanie. Zwiększają również czytelność poprzez odizolowanie kodu, który wykonuje określone zadania.

W tym rozdziale zostaną przedstawione składnia wywołań funkcji i definicje funkcji, jak również omówione będą sposoby wykorzystywania zmiennych w funkcjach i przekazywania wartości do funkcji (zarówno przekazywanie przez wartość, jak i przekazywanie przez referencję). Znajdziemy tu również opis zmiennych funkcji oraz funkcji anonimowych.

Wywoływanie funkcji

Funkcje w programie PHP mogą być wbudowane (lub efektywnie wbudowane poprzez umieszczenie w rozszerzeniu) lub definiowane przez użytkownika. Niezależnie od ich pochodzenia wszystkie są używane jednakowo:

```
$jakaś_wartość = nazwa_funkcji( [ parametr, ... ] );
```

Liczba parametrów wymaganych przez funkcję jest różna w zależności od konkretnej funkcji (a może nawet różnić się dla tej samej funkcji). Parametrami dostarczanymi funkcji mogą być dowolne poprawne wyrażenia, ale powinny one występować w konkretnej, oczekiwanej przez funkcję kolejności. Informacje na temat parametrów wymaganych przez funkcję oraz wartości, które mogą zostać zwrócone, znajdują się w dokumentacji funkcji.

Poniżej znajduje się kilka przykładów funkcji:

```
// strlen() to wbudowana funkcja zwracająca długość łańcucha
$length = strlen("PHP"); // $length wynosi obecnie 3

// sin() i asin() to funkcje matematyczne sinus i arcus sinus
$result = sin(asin(1)); // $result to sinus z arcsin(1), lub 1.0

// unlink() kasuje plik
$result = unlink("functions.txt"); // false jeżeli zakończona niepowodzeniem
```

W pierwszym przykładzie jest przekazywany argument `"PHP"` do funkcji `strlen()`, która podaje nam liczbę znaków w danym łańcuchu. W opisywanym tutaj przypadku zwracana jest wartość 3, która zostaje przypisana do zmiennej `$length`. Jest to najprostszy i najczęściej spotykany sposób używania funkcji.

W drugim przykładzie rezultat `asin(1)` jest przekazywany do funkcji `sin()`. Z uwagi na to, że funkcje sinus i arcus sinus są dla siebie odwrotnymi, wyliczanie sinusa z arcusa sinusa dla jakiejkolwiek wartości zawsze zwróci tę samą wartość. W cytowanym przykładzie widzimy, że funkcja może być wywoływana wewnątrz innej funkcji. Zwrócona wartość wewnętrznego wywołania jest następnie wysyłana do funkcji zewnętrznej, po czym rezultat końcowy zostaje zwrócony i zachowany w zmiennej `$result`.

W trzecim przykładzie funkcji `unlink()` podawana jest nazwa pliku, który ma zostać skasowany. Podobnie jak wiele innych funkcji, w przypadku niepowodzenia zwraca ona `false`. Pozwala to na użycie innej wbudowanej funkcji o nazwie `die()` oraz uproszczonego przetwarzania przez PHP zapisu operatorów logicznych. W takim wypadku można byłoby zapisać omawiany przykład jako:

```
$result = unlink("functions.txt") or die("Operacja zakończyła się niepowodzeniem!");
```

Funkcja `unlink()` w przeciwieństwie do dwóch pierwszych przykładów ma wpływ nie tylko na przekazywane jej parametry. W tym przypadku kasuje ona plik z systemu plików. Wszystkie tego rodzaju efekty uboczne funkcji powinny być starannie dokumentowane.

PHP ma obszerną listę funkcji, które są już zdefiniowane i gotowe do wykorzystania w pisanych przez nas programach. W licznych rozszerzeniach PHP można znaleźć dosłownie wszystko, od obsługi bazy danych po tworzenie grafiki, odczytywanie i zapisywanie plików XML czy pobieranie plików z systemów zdalnych. Dodawanie nowych rozszerzeń do PHP zostało opisane w rozdziale 14., wbudowane funkcje są szczegółowo omówione w dodatku A, a przegląd rozszerzeń PHP znajduje się w dodatku B.

Definiowanie funkcji

Aby zdefiniować funkcję używa się poniższej składni:

```
function [&] nazwa_funkcji ( [ parametr [, ... ] ] )
{
   lista instrukcji
}
```

Lista instrukcji może zawierać kod HTML. Można zadeklarować funkcję PHP, która nie zawiera kodu PHP. Na przykład funkcja `column()` po prostu nadaje krótką nazwę fragmentowi kodu HTML, którego będziemy potrzebować kilkakrotnie na jednej stronie:

```
<? function column() { ?>
</td><td>
<? } ?>
```

Nazwą funkcji może być dowolny łańcuch zaczynający się literą lub podkreśleniem, po których występuje zero lub inne litery, podkreślenia albo cyfry. W nazwach funkcji nie rozróżnia się małych i dużych liter, tzn. można wywołać funkcję `sin()` jako `sin(1)`, `SIN(1)`, `SiN(1)` itd., ponieważ wszystkie te nazwy dotyczą tej samej funkcji.

Funkcje zwracają zazwyczaj jakąś wartość. Aby funkcja zwróciła wartość, należy użyć instrukcji `return`, tzn. umieścić wewnątrz funkcji `return` *wyrażenie*. Gdy podczas wykonywania programu napotkana zostanie instrukcja `return`, kontrola zostanie oddana instrukcji wywołującej,

a rezultaty uzyskane dla *wyrażenie* zostaną zwrócone jako wartość funkcji. Mimo że taki kod może wydawać nam się chaotyczny, w uzasadnionych wypadkach można zamieścić wewnątrz funkcji wiele instrukcji `return` (na przykład, jeżeli mamy instrukcję `switch`, w celu określenia, która z kilku wartości ma zostać zwrócona).

Jeżeli funkcja zostanie zdefiniowana przy użyciu opcjonalnego znaku `&`, zwróci ona raczej wskaźnik do wyniku funkcji niż kopię danych.

Przyjrzyjmy się prostej funkcji. Na listingu 3.1 mamy do czynienia z dwoma łańcuchami, które są łączone, a następnie zwracane jako wynik (w tym przypadku utworzyliśmy odrobinę wolniejszy równoważnik operatora łączenia łańcuchów, ale dla dobra przykładu prosimy o chwilę cierpliwości).

Listing 3.1. Łączenie łańcuchów

```
function strcat($left, $right) {
   $combined_string = $left . $right;
   return $combined_string;
}
```

Funkcja przyjmuje dwa argumenty, `$left` i `$right`. Korzystając z operatora łączenia, funkcja tworzy połączony łańcuch wewnątrz zmiennej `$combined_string`. Na końcu zwracana jest wartość zmiennej `$combined_string`, aby funkcja posiadała wartość w sytuacji, gdy będzie wyliczana z użyciem podanych argumentów.

Z uwagi na to, że instrukcja `return` przyjmuje dowolne wyrażenie, nawet złożone, można uprościć program, tak jak widać na listingu 3.2.

Listing 3.2. Łączenie łańcuchów bis

```
function strcat($left, $right) {
   return $left . $right;
}
```

Jeżeli umieścimy tę funkcję w dokumencie PHP, możemy wywołać ją z dowolnego miejsca tej strony. Spójrzmy na listing 3.3.

Listing 3.3. Używanie funkcji łączenia

```
<?php
   function strcat($left, $right) {
      return $left . $right;
   }
   $first = "To jest";
   $second = " pełne zdanie!";

   echo strcat($first, $second);
?>
```

Gdy ta strona jest wyświetlana, pokazywane jest pełne zdanie.

W poniższym przykładzie funkcja pobiera liczbę całkowitą, podwaja ją i zwraca wynik:

```
function doubler($value) {
   return $value << 1;
}
```

Po zdefiniowaniu funkcji można użyć jej w dowolnym miejscu na stronie. Na przykład:

```
<?= 'Dwa razy 13 wynosi ' . doubler(13); ?>
```

Można zagnieżdżać deklaracje funkcji, ale ma to ograniczone efekty. Zagnieżdżone deklaracje nie ograniczają widoczności funkcji zdefiniowanej wewnątrz. Może ona zostać wywołana z każdego miejsca w programie. Funkcja wewnętrzna nie uzyskuje dostępu do argumentów funkcji zewnętrznej automatycznie. Poza tym funkcja wewnętrzna nie może zostać wywołana, dopóki nie zostanie wywołana funkcja zewnętrzna.

```
function outer ($a) {
   function inner ($b) {
      echo "cię $b";
   }
   echo "$a, witamy ";
}
outer("hej");
inner("Czytelniku");
```

W wyniku działania powyższego kodu otrzymamy:

```
hej, witamy cię Czytelniku
```

Zasięg zmiennych

Jeżeli funkcje nie są wykorzystywane, każda utworzona przez nas zmienna może zostać użyta w dowolnym miejscu na stronie. W przypadku funkcji nie zawsze tak jednak jest. Funkcje mają swój własny zestaw zmiennych, które różnią się od tych na stronie i od tych występujących w innych funkcjach.

Zmienne zdefiniowane w funkcji, łącznie ze swoimi parametrami, nie są dostępne poza funkcją, a zmienne zdefiniowane poza funkcją nie są dostępne wewnątrz funkcji. Jest to ustawienie domyślne. Zostało to pokazane w poniższym przykładzie:

```
$a = 3;

function foo() {
   $a += 2;
}

foo();
echo $a;
```

Zmienna `$a` umieszczona wewnątrz funkcji `foo()` to zupełnie inna zmienna niż zmienna `$a` poza tą funkcją. Mimo że `foo()` korzysta z operatora „dodaj i przypisz”, wartość zewnętrznej zmiennej `$a` będzie wynosić 3 przez cały czas życia strony. Wewnątrz funkcji `$a` ma wartość 2.

Jak to zostało omówione w rozdziale 2., zakres, w jakim zmienna może być widoczna w programie, nazywany jest *zasięgiem* zmiennej. Zmienne utworzone wewnątrz funkcji znajdują się w jej zasięgu (np. mają *zasięg na poziomie funkcji*). Zmienne utworzone poza funkcjami i obiektami mają *zasięg globalny* i istnieją gdziekolwiek poza tymi funkcjami i obiektami. Kilka zmiennych dostarczonych przez PHP ma zarówno zasięg na poziomie funkcji, jak i zasięg globalny.

Na pierwszy rzut oka nawet doświadczonemu programiście może wydawać się, że w cytowanym przed chwilą przykładzie zmienna `$a` będzie miała wartość 5, zanim osiągnięta zostanie instrukcja `echo`. Należy o tym pamiętać przy wybieraniu nazw dla tworzonych przez nas zmiennych.

Zmienne globalne

Jeżeli chcemy, aby zmienna o zasięgu globalnym była dostępna wewnątrz funkcji, należy użyć słowa kluczowego `global`. Jego składnia wygląda następująco:

```
global zmienna1, zmienna2, ...
```

Gdy zmienimy poprzednio cytowany przykład tak, aby włączyć słowo kluczowe `global`, otrzymamy:

```
$a = 3;

function foo() {
   global $a;
   $a += 2;
}

foo();
echo $a;
```

Zamiast utworzyć nową zmienną o nazwie `$a` o zasięgu na poziomie funkcji, PHP korzysta ze zmiennej globalnej `global $a` wewnątrz funkcji. Tym razem wyświetlona wartość `$a` będzie wynosić 5.

Aby uzyskać możliwość korzystania z wybranej przez nas zmiennej lub zmiennych globalnych, należy najpierw umieścić słowo kluczowe `global` wewnątrz funkcji. Z uwagi na to, że parametry funkcji są definiowane wcześniej niż treść funkcji, nie mogą one być nigdy zmiennymi globalnymi.

Użycie `global` jest równoważne z utworzeniem referencji do zmiennej w tablicy `$GLOBALS`. Oznacza to, że obie poniższe deklaracje:

```
global $var;
$var = &$GLOBALS['var'];
```

tworzą zmienną na poziomie danej funkcji, która jest referencją do tej samej wartości co zmienna `$var` w zasięgu globalnym.

Zmienne statyczne

Podobnie jak C, PHP obsługuje deklarowanie zmiennych statycznych. Zmienna statyczna jest przechowywana pomiędzy wszystkimi kolejnymi wywołaniami danej funkcji, a inicjowana jest podczas wykonywania skryptu tylko przy pierwszym wywołaniu funkcji. Aby zadeklarować zmienną statyczną funkcji, należy przy pierwszym użyciu danej zmiennej użyć słowa kluczowego `static`. Zazwyczaj pierwsze użycie zmiennej statystycznej ma na celu przypisanie wartości początkowej:

```
static zmienna [= wartość][,... ];
```

Na listingu 3.4 zmienna `$count` jest zwiększana o jeden przy każdym kolejnym wywołaniu funkcji.

Listing 3.4. Licznik zmiennej statycznej

```
function counter() {
   static $count = 0;
   return $count++;
}
```

```
for ($i = 1; $i <= 5; $i++) {
   print counter();
}
```

Zmiennej statycznej `$count` przy pierwszym wywołaniu funkcji przypisywana jest wartość 0. Gdy wartość zostaje zwrócona, następuje inkrementacja zmiennej `$count`. Kiedy funkcja się kończy, `$count` nie zostaje wymazana, jak się to dzieje w przypadku zmiennych niestatycznych. Jej wartość pozostaje niezmieniona aż do kolejnego wywołania funkcji `counter()`. Pętla `for` wyświetla cyfry od 0 do 4.

Parametry funkcji

Funkcje mogą mieć dowolną liczbę argumentów, które deklaruje się w definicji funkcji. Istnieją dwa sposoby na przekazywanie parametrów do funkcji. Pierwszym sposobem, dużo bardziej popularnym, jest przekazywanie parametrów przez wartość. Drugim — przekazywanie parametrów przez referencję.

Przekazywanie parametrów przez wartość

W większości przypadków parametry przekazuje się przez wartość. Argument to dowolne poprawne wyrażenie. Wyrażenie to jest obliczane, a wyliczona wartość jest przypisywana odpowiedniej zmiennej w funkcji. We wszystkich dotąd cytowanych przykładach argumenty były przekazywane przez wartość.

Przekazywanie parametrów przez referencję

Przekazywanie przez referencję pozwala na zignorowanie normalnych reguł zakresu i danie funkcji bezpośredniego dostępu do zmiennej. Aby argument mógł zostać przekazany przez referencję, musi być zmienną. Wskazany przez nas argument funkcji zostanie przekazany przez referencję, jeżeli przed nazwą zmiennej na liście parametrów wstawimy znak `&`. Na listingu 3.5 widzimy funkcję `doubler()` z kilkoma drobnymi zmianami:

Listing 3.5. Funkcja doubler po raz drugi

```
function doubler(&$value) {
   $value = $value << 1;
}

$a = 3;
doubler($a);
echo($a);
```

Z uwagi na to, że parametr funkcji `$value` jest przekazywany przez referencję, zmodyfikowana przez funkcję zostanie właściwa wartość zmiennej `$a`, a nie kopia tej wartości. Wcześniej podwojona wartość musiała być zwrócona, natomiast teraz przekazywana w wywołaniu zmienna modyfikowana jest tak, by mieć podwojoną wartość.

W tym momencie pojawiają się efekty uboczne: ponieważ zmienna `$a` została przekazana do funkcji `doubler()` przez referencję, wartość `$a` jest zależna od tej funkcji. W tym przypadku funkcja `doubler()` przypisuje jej nową wartość.

Parametr, który zostanie zadeklarowany jako przekazywany przez referencję, musi być zmienną. Z tego względu, gdybyśmy w powyższym przykładzie użyli instrukcji `<?= doubler(7); ?>`, wystąpiłby błąd. Parametrom przekazywanym przez referencję można jednak przypisać domyślną wartość (w ten sam sposób jak w przypadku nadawania domyślnej wartości parametrom przekazywanym przez wartość).

Nawet w przypadkach, w których nasza funkcja nie ma wpływu na daną wartość, możemy chcieć, aby parametr był przekazany przez referencję. Gdy przekazywany jest przez wartość, PHP musi skopiować tę wartość. Szczególnie w przypadku łańcuchów i obiektów taka operacja może być bardzo kosztowna. Przekazywanie przez referencję pozwala na uniknięcie konieczności kopiowania wartości.

Parametry domyślne

Czasami funkcja może potrzebować z góry określonego parametru. Na przykład gdy wywołujemy funkcję, aby przejść do ustawień strony, funkcja może przyjąć parametr o nazwie ustawienia do wyszukania. Zamiast stosowania jakiegoś słowa kluczowego do oznaczenia, że chcemy uzyskać wszystkie ustawienia, można po prostu nie podawać żadnego argumentu. Takie zachowanie jest możliwe dzięki użyciu argumentów domyślnych.

Aby określić parametr domyślny, należy przypisać wartość do parametru przy deklarowaniu funkcji. Wartość przypisana do parametru jako domyślna nie może być wyrażeniem złożonym. Może być jedynie wartością stałą.

```
function get_preferences($which_preference = "all" ) {
   // jeżeli $which_preference równa się "all", zwraca wszystkie ustawienia;
   // w przeciwnym wypadku, podaje żądane ustawienie...
}
```

Kiedy wywołujemy funkcję `get_preferences()`, możemy zdecydować, czy chcemy podać jakiś argument, czy nie. Jeżeli argument zostanie podany, zwrócone zostanie ustawienie odpowiadające podanemu łańcuchowi. Jeżeli nie, zwrócone zostaną wszystkie ustawienia.

Funkcja może mieć dowolną liczbę parametrów o domyślnych wartościach. Jednak muszą być one umieszczone na liście po wszystkich parametrach, które nie mają domyślnych wartości.

Zmienna liczba parametrów

Funkcja może wymagać zmiennej liczby argumentów. Na przykład opisywana w poprzednim punkcie funkcja `get_preferences()` może zwrócić ustawienia dla dowolnej liczby nazw, a nie tylko dla jednej. Aby zadeklarować funkcję ze zmienną liczbą argumentów, należy całkowicie pominąć blok parametrów.

```
function get_preferences() {
   // jakiś kod
}
```

W PHP istnieją trzy funkcje, których można użyć w funkcji, aby pobrać przekazane jej parametry. Funkcja `func_get_args()` zwraca tablicę wszystkich parametrów przekazanych danej funkcji, `func_num_args()` zwraca liczbę parametrów przekazanych funkcji, a `func_get_arg()` zwraca określony argument spośród parametrów.

```
$array = func_get_args();
$count = func_num_args();
$value = func_get_arg(numer_argumentu);
```

Na listingu 3.6 funkcja `count_list()` przyjmuje dowolną liczbę argumentów. Przechodzi ona po wszystkich argumentach i zwraca sumę wszystkich wartości. Jeżeli nie zostały podane żadne parametry, zwraca `false`.

Listing 3.6. Licznik argumentów

```
function count_list() {
   if(func_num_args() == 0) {
      return false;
   }
   else {
      for($i = 0; $i < func_num_args(); $i++) {
         $count += func_get_arg($i);
      }
      return $count;
   }
}

echo count_list(1, 5, 9);
```

Wynik każdej z tych funkcji nie może zostać użyty bezpośrednio jako parametr innej funkcji. Aby można było użyć wyniku jednej z tych funkcji jako parametru, trzeba najpierw przypisać zmiennej wynik funkcji, a następnie użyć tej zmiennej w wywołaniu funkcji. Poniższe wyrażenie nie zadziała:

```
foo(func_num_args());
```

Zamiast niego należy użyć, co następuje:

```
$count = func_num_args();
foo($count);
```

Brakujące parametry

PHP pozwala na bycie leniwym — przy wywoływaniu funkcji można przekazać do niej dowolną liczbę argumentów. Parametry, których funkcja oczekuje, a które nie zostaną jej przekazane, pozostają nieustawione, a dla każdego z nich zostanie wyświetlone ostrzeżenie:

```
function takes_two( $a, $b ) {
   if (isset($a)) { echo " a jest ustawione\n"; }
   if (isset($b)) { echo " b jest ustawione\n"; }
}
echo "Przy dwóch argumentach:\n";
takes_two(1, 2);
echo "Przy jednym argumencie:\n";
takes_two(1);
```

W wyniku działania powyższego kodu otrzymamy:

```
Przy dwóch argumentach:
 a jest ustawione
 b jest ustawione
Przy jednym argumencie:
Warning: Missing argument 2 for takes_two()
 in /path/to/script.php on line 6
 a jest ustawione
```

Wartości zwracane

Przy użyciu słowa kluczowego `return` funkcje PHP mogą zwracać tylko jedną wartość:

```
function return_one() {
   return 42;
}
```

Aby zwrócić kilka wartości, należy skorzystać z tablicy:

```
function return_two() {
   return array("Fred", 35);
}
```

Domyślnie wartości są kopiowane z funkcji. Funkcja zdeklarowana przy użyciu znaku `&` przed jej nazwą zwraca referencję (alias) do wartości zwracanej:

```
$names = array("Fred", "Barney", "Wilma", "Betty");
function & find_one($n) {
   global $names;
   return $names[$n];
}
$person =& find_one(1);             // Barney
$person = "Barnetta";               // zmienia $names[1]
```

W tym kodzie funkcja `find_one()` zwraca alias do zmiennej `$names[1]` zamiast kopii jej wartości. Ponieważ przypisana została referencja, `$person` to alias do `$names[1]`, a drugie przypisanie zmienia wartość w `$names[1]`.

Technika ta jest czasem wykorzystywana do zwracania dużych łańcuchów lub tablic z funkcji. Jednak istniejący mechanizm PHP kopiowania podczas zmiany lub inaczej „płytkiego kopiowania" oznacza zazwyczaj, że nie ma potrzeby zwracania referencji z funkcji. Zwracanie referencji do większych ilości danych nie ma sensu, z wyjątkiem sytuacji, gdy wiemy, że będziemy zmieniać dane, do których odwołamy się przez referencję. Zwracanie referencji jest wolniejsze w porównaniu do zwracania wartości i polega na mechanizmie „płytkiego kopiowania", który upewnia się, iż kopia tych danych nie została utworzona, jeśli nie zostały one zmienione.

Zmienne funkcje

Podobnie jak zmienne zmiennych, można wywołać funkcję w oparciu o wartość zmiennej. Dla przykładu przyjrzyjmy się sytuacji, w której zmienna jest wykorzystywana do określenia, którą z trzech funkcji wywołać:

```
switch($which) {
   case 'first':
      first();
      break;

   case 'second':
      second();
      break;

   case 'third':
      third();
      break;
}
```

W tym przypadku moglibyśmy użyć wywoływania funkcji zmiennej do wywołania odpowiedniej funkcji. Aby wywołać funkcję zmienną, należy umieścić w nawiasach za zmienną parametry dla tej funkcji. Poprzedni przykład należałoby przepisać następująco:

```
$which();  // jeżeli $which ma wartość 'first', wywoływana jest funkcja first() itd
...
```

Jeżeli nie istnieje funkcja dla danej zmiennej, w momencie wykonywania kodu pojawi się błąd wykonania. Aby temu zapobiec, można użyć wbudowanej funkcji `function_exists()`, która przed próbą wywołania funkcji zmienną pozwoli na sprawdzenie, czy ona istnieje:

```
$yes_or_no = function_exists(nazwa_funkcji);
```

Na przykład:

```
if(function_exists($which)) {
   $which();  // jeżeli $which ma wartość "first", wywoływana jest funkcja first() itd
...
}
```

Konstrukcji języka takich jak `echo()` czy `isset()` nie da się wywołać przez funkcje wywoływane zmienną:

```
$f = 'echo';
$f('Witaj świecie!');  // nie działa
```

Funkcje anonimowe

Niektóre funkcje PHP wykorzystują dostarczane im przez nas funkcje do wykonywania części swej pracy. Na przykład funkcja `usort()` korzysta z utworzonej przez nas i przekazanej do niej funkcji, aby określić kolejność sortowania elementów w tablicy.

Mimo że można do takich celów zdefiniować funkcję, tak jak to zostało wcześniej pokazane, funkcje te są zazwyczaj lokalne i tymczasowe. Aby odzwierciedlić przejściową naturę funkcji zwrotnych, tworzone są i wykorzystywane *funkcje anonimowe* (lub funkcje lambda).

Funkcję anonimową można utworzyć, korzystając z `create_function()`. Funkcja ta przyjmuje dwa argumenty — pierwszy opisuje parametry przyjmowane przez funkcję anonimową, a drugi to właściwy kod. Zwracana jest losowo wygenerowana nazwa funkcji:

```
$func_name = create_function(łańcuch_argumentów, łańcuch_kodu);
```

Na listingu 3.7 znajduje się przykładowe użycie `usort()`.

Listing 3.7. Funkcje anonimowe

```
$lambda = create_function('$a,$b', 'return(strlen($a) - strlen($b));');
$array = array('tutaj naprawdę długi łańcuch', 'ten', 'średnia długość', 'większy');
usort($array, $lambda);
print_r($array);
```

Tablica jest sortowana przez funkcję `usort()` z użyciem funkcji anonimowej, według długości łańcucha.