Lekcja 3 - funkcje

Funkcje są to wydzielone bloki kodu przeznaczone do wykonywania konkretnych zadań. W celu utworzenia funkcji należy użyć słowa kluczowego function. Taka konstrukcja ma schematyczną postać:

```
function  nazwa_funkcji()
    {
    //instrukcje wnętrza funkcji
    }
```

Przykład 3.1. Pseudo-kod ilustrujący zastosowanie funkcji

```
<?php
function foo ($arg_1, $arg_2, /* ..., */ $arg_n)
{
   echo "Przykładowa funkcja.\n";
   return $retval;
}
?>
```

Dowolny poprawny kod PHP może się pojawić wewnątrz funkcji, także definicje innych funkcji i klas.

Nazwy funkcji podlegają takim samym ograniczeniom jak wszystkie inne etykiety w PHP. Poprawna nazwa funkcji zaczyna się od litery lub podkreślenia, po których następuje dowolna liczba liter, liczb lub podkreśleń. Powyższa zasada w postaci wyrażenia regularnego przedstawia się następująco: *[a-zA-Z_\x7f-\xff][a-zA-Z0-9_\x7f-\xff]**.

W PHP 3, funkcje muszą być zdefiniowane przed odwołaniem do nich. W PHP 4 nie ma takiego wymagania. ***Wyjątkiem*** jest warunkowe definiowanie funkcji, tak jak pokazano to w dwóch przykładach zamieszczonych poniżej.

Kiedy funkcja jest definiowana w postaci warunkowej, tak jak to przedstawiono na dwóch poniższych przykładach, jej definicja musi następować ***przed*** jej wywołaniem.

Przykład 3.2. Funkcje warunkowe

```
<?php

$makefoo = true;

/* Nie można wywołać foo() z tego miejsca,
  ponieważ funkcja ta jeszcze nie istnieje.
  Można natomiast wywołać bar() */

bar();
```

```
if ($makefoo) {
  function foo()
  {
    echo "Nie istnieję dopóki wykonanie programu nie dojdzie do tego miejsca.\n";
  }
}

/* W tym momencie można spokojnie wywołać foo()
   ponieważ $makefoo ma wartość true */

if ($makefoo) foo();

function bar()
{
  echo "Istnieję od samego początku działania programu.\n";
}

?>
```

Przykład 3.3. Funkcje wewnątrz funkcji

```
<?php
function foo()
{
  function bar()
  {
    echo "Nie istnieję dopóki foo() nie zostanie wywołana.\n";
  }
}

/* Nie można wywołać bar()
   ponieważ nie istnieje. */

foo();

/* Teraz można wywołać bar(),
   ponieważ wywołanie foo() utworzyło
   tą funkcję. */

bar();

?>
```

Wszystkie funkcje i klasy w PHP mają zasięg globalny - można je wywołać spoza ciała funkcji, nawet jeśli zostały w niej zdefiniowane, i vice versa.

PHP nie obsługuje przeciążania funkcji. Nie jest także możliwa od-definiowanie lub przedefiniowanie wcześniej zadeklarowanych funkcji.

Notatka: W nazwach funkcji nie jest istotna wielkość znaków, jednakże dobrze jest wywoływać funkcje tak, jak zostały zdeklarowane.

W PHP możliwe jest rekurencyjne wywoływanie funkcji. Należy jednak unikać rekurencyjnych wywołań funkcji lub metod o głębokości większej niż 100-200 poziomów, gdyż może to zniszczyć stos i spowodować przerwanie działania skryptu.

Przykład 3.4. Funkcje rekurencyjne

```
<?php
function rekurencja($a)
{
   if ($a < 20) {
      echo "$a\n";
      rekurencja($a + 1);
   }
}
?>
```

Argumenty funkcji

Informacje mogą być przekazywane do funkcji przez listę argumentów, która jest separowaną przecinkami listą wyrażeń.

PHP obsługuje przekazywanie argumentów przez wartość (domyślnie), przez referencję , i wartości domyślne argumentów. Listy argumentów o zmiennej długości są obsługiwane tylko w PHP 4 i nowszych; zobacz rozdział Listy argumentów o zmiennej długości i opisy funkcji func_num_args(), func_get_arg(), i func_get_args() aby uzyskać więcej informacji. Podobny efekt może być uzyskany w PHP 3 przez przekazywanie tablicy argumentów do funkcji.

Przykład 3.5. Przekazywanie tablic do funkcji

```
function pobiera_tablice($wejscie)
{
   echo "$wejscie[0] + $wejscie[1] = ", $wejscie[0]+$wejscie[1];
}
```

Przekazywanie argumentów przez referencję

Domyślnie, argumenty funkcji są przekazywane przez wartość (a więc jeśli zmienisz wartość argumentu wewnątrz funkcji, nie zmieni się ona poza funkcją). Jeśli chcesz pozwolić funkcji na modyfikację swoich argumentów, musisz przekazać je przez referencję.

Jeśli chcesz, aby argumenty były zawsze przekazywane przez referencję, przed nazwą zmiennej w definicji funkcji wstaw znak ampersand (&):

Przykład 3.6. Przekazywanie argumentów przez referencję

```
<?php
function dodaj_cos_extra(&$string)
{
   $string .= 'i coś extra.';
}
$str = 'To jest string, ';
dodaj_cos_extra($str);
echo $str;    // wyświetla 'To jest string string, i coś extra.'
```

```
?>
```

Wartości domyślne argumentów

Funkcja może definiować, podobnie jest w C++, wartości domyślne dla argumentów skalarnych.

Przykład 3.7. Użycie domyślnych wartości argumentów

```
<?php
function rob_kawe ($typ = "cappucino")
{
   return "Robię kubek $typ.\n";
}
echo rob_kawe ();
echo rob_kawe ("espresso");
?>
```

PHP pozwala także na korzystanie z tablic i specjalnego typu NULL jako wartości domyślnych.

Przykład 3.8. Używanie nieskalarnych typów jako wartości domyślne

```
<?php
function rob_kawe($typy = array("cappuccino"), $urzadzenie = NULL)
{
   $urzadzenie = is_null($urzadzenie) ? "rece" : $urzadzenie;
   return "Robie kubek ".join(", ", $types)." poprzez $urzadzenie.\n";
}
echo rob_kawe();
echo rob_kawe(array("cappuccino", "lavazza"), "ekspres");
?>
```

Domyślna wartość musi być stałym wyrażeniem, nie (na przykład) zmienną. członkiem klasy lub wywołaniem funkcji.

Zauważ, że używając domyślnych argumentów, argumenty zawierające wartości domyślne powinny być po prawej stronie tych nie zawierających wartości domyślnych; w przeciwnym przypadku funkcja może nie działać tak jak się tego spodziewałeś. Przedstawione to zostało na poniższym przykładzie.

Przykład 3.9. Błędne zastosowanie domyślnych argumentów funkcji

```
<?php
function robjogurt ($typ = "acidophilus", $smak)
{
   return "Robię miskę $typ $smak.\n";
}

echo robjogurt ("malinowy");   // nie działa tak jak się spodziewaliśmy
?>
```

Przykład 17.10. Poprawne użycie domyślnych argumentów funkcji

```
<?php
```

```
function robjogurt ($smak, $typ = "acidophilus")
{
   return "Robię miskę $type $flavour.\n";
}

echo robjogurt ("malinowy");   // działa tak jak się spodziewaliśmy
?>
```

Powyższy kod wyświetli:

Notatka: Od PHP 5, domyślne argumenty funkcji mogą być przekazywane przez referencję.

Listy argumentów o zmiennej długości

PHP 4 i nowsze obsługują listy o zmiennej długości w funkcjach zdefiniowanych przez użytkownika. Jest naprawdę prostę przy użyciu funkcji func_num_args(), func_get_arg() i func_get_args().

Nie wymagana jest żadna specjalna składnia. Listy argumentów mogą być ciągle jawnie podane przy definicji funkcji i będą się zachowywać normalnie.

Zwracane wartości

Wartości są zwracane przy użyciu opcjonalnej instrukcji return. Może być zwracany dowonlny typ, włączając w to tablice i obiekty. Spowoduje to natychmiastowe zakończenie działania funkcji i przekazanie kontroli do linii, z której była wywołana. Więcej informacji w opisie **return()**.

Przykład 3.11. Zastosowanie return()

```
<?php
function kwadrat ($num)
{
   return $num * $num;
}
echo kwadrat (4);   // wyświetla '16'.
?>
```

Nie można zwracać zwracać wielu wartości z funkcji, ale podobne efekty mogą być uzyskane przez zwracanie listy.

Przykład 3.12. Zwracanie tablicy dla uzyskania wielu wyników

```
<?php
function maleLiczby()
{
   return array (0, 1, 2);
}
list ($zero, $jeden, $dwa) = maleLiczby();
?>
```

Aby funkcja zwracała referencję, musisz użyć operatora referencji & i w deklaracji funkcji i przy przypisywaniu zwracanej wartości do zmiennej.

Przykład 3.13. Zwracanie referencji przez funkcję

```
<?php
function &zwrocReferencje()
{
   return $jakasref;
}

$nowaref =& zwrocReferencje();
?>
```

Zmienne funkcje

PHP obsługuje koncepcję zmiennych funkcji. Oznacza to, że jeśli po nazwie zmiennej występują nawiasy, PHP będzie szukało funkcji o nazwie będącej wartością zmiennej i będzie próbowało wywołać ją. Między innymi może być to użyte do implementacji funkcji callback, tablicy funkcji itp.

Zmienne funkcje nie będą działać z elementami składowymi języka takimi jak echo(), print(), unset(), isset(), empty(), **include()**, **require()** i innymi podobnymi. Aby zastosować zmienne funkcje z takimi składowymi, niezbędne jest zastosowanie funkcji obudowujących.

Przykład 3.14. Przykład zmiennej funkcji

```
<?php
function foo() {
   echo "W foo()<br />\n";
}

function bar($arg = '') {
   echo "W bar(); argumentem jest '$arg'.<br />\n";
}

// Funkcja obudowująca składnię echo

function echoIt($string) {
   echo $string;
}

$func = 'foo';
$func(); // wywoła foo()

$func = 'bar';
```

```
$func('test'); // wywoła bar()

$func = 'echoIt';
$func('test'); // wywoła echoIt()
?>
```

Możliwe jest także wywołanie metody obiektu korzystając z mechanizmu zmiennych funkcji. You can also call an object's method by using the variable functions feature.

Przykład 3.15. Przykład zmiennych metod

```
<?php
class Foo
{
   function Zmienna()
   {
      $name = 'Bar';
      $this->$name(); // Wywoła metodę Bar()
   }

   function Bar()
   {
      echo "To jest Bar";
   }
}

$foo = new Foo();
$funkcja = "Zmienna";
$foo->$funkcja();  // Wywoła $foo->Zmienna()

?>
```

Zmienne funkcje

PHP obsługuje koncepcję zmiennych funkcji. Oznacza to, że jeśli po nazwie zmiennej występują nawiasy, PHP będzie szukało funkcji o nazwie będącej wartością zmiennej i będzie próbowało wywołać ją. Między innymi może być to użyte do implementacji funkcji callback, tablicy funkcji itp.

Zmienne funkcje nie będą działać z elementami składowymi języka takimi jak echo(), print(), unset(), isset(), empty(), **include()**, **require()** i innymi podobnymi. Aby zastosować zmienne funkcje z takimi składowymi, niezbędne jest zastosowanie funkcji obudowujących.

Przykład 3.14. Przykład zmiennej funkcji

```
<?php
function foo() {
   echo "W foo()<br />\n";
}

function bar($arg = '') {
```

```
    echo "W bar(); argumentem jest '$arg'.<br />\n";
}

// Funkcja obudowująca składnię echo

function echoIt($string) {
    echo $string;
}

$func = 'foo';
$func(); // wywoła foo()

$func = 'bar';
$func('test'); // wywoła bar()

$func = 'echoIt';
$func('test'); // wywoła echoIt()
?>
```

Możliwe jest także wywołanie metody obiektu korzystając z mechanizmu zmiennych funkcji. You can also call an object's method by using the variable functions feature.

Przykład 3.15. Przykład zmiennych metod

```
<?php
class Foo
{
    function Zmienna()
    {
        $name = 'Bar';
        $this->$name(); // Wywoła metodę Bar()
    }

    function Bar()
    {
        echo "To jest Bar";
    }
}

$foo = new Foo();
$funkcja = "Zmienna";
$foo->$funkcja();  // Wywoła $foo->Zmienna()

?>
```

Zadania

Każdy skrypt powinien zawierać finkcję

1. Wyznaczenie kolejnych liczb ciągu Fibonacciego (do liczby wskazanej przez użytkownika
2. Wyznaczenie NWD dla liczb a, b
3. Skrypt sprawdzający poprawność wprowadzonego numeru pesel, orz wyświetlający datę urodzenia (pełną) wraz z informacją o płci albo komunikat o błędnym PESELU.