# Упражнение 6. Функции. Деклариране и използване. Математически функции

## Функции. Деклариране и използване

Упражнението има за цел да тества някои от примерите, включени в лекцията за създаването на потребителски функции в PHP. Ще припомним, че функция в PHP може да бъде декларира посредством следния синтаксис:

```
function fname ( $arg1, $arg2…)
{
//оператор1; оператор2...
[return израз;]
}
```

Където: fname – име на функцията, $arg1, $arg2…- списък от параметри на функцията, [return израз;] – незадължителен оператор, чрез който функцията връща стойността на израза и се прекратява изпълнението на функцията.

**Пример 1.** Ще започнем с един пример, демонстриращ функция в PHP:

```
<!DOCTYPE html>
<html>
    <head>
   <meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
        <title></title>
    </head>
    <body>
 <?php
function myCompanyMotto()
{
    echo "Ние доставяме количество, не качество!<br />";
}
myCompanyMotto();
//Ние доставяме количество, не качество!
?>
 </body>
</html>
```

**Пример 2:** функция writeName().

```
<?php
function writeName()
{ echo "Tom Taylor"; }
echo "My name is ";
writeName();
?>
```

Извежда:

My name is Tom Taylor

Запомнете:

- Функцията започва с ключова дума function
- Името на функцията е добре да съответства на нейното предназначение
- Имената на функциите не са регистрово чувствителни, но следват същите правила като за другите имена в PHP: започват с буква или _ (но не число), следват цифри, букви или _.
- Кодът е между "{" и "}"

За да се върне стойност от функцията, трябва да се извика return израз във функцията. Ако една функция е декларирана вътре в друга функция, то тя ще бъде достъпна само, ако външната функция се изпълни поне веднъж.

Важна характеристика на функциите е, че може да им изпращате информация (чрез параметри), която те да използват.

Параметрите се поставят (както знаете от езика C++) в скобите "()" и изглеждат като PHP променливи.

Нека да създадем една нова функция, която създава един персонализиран поздрав, зависещ от името на потребителя.

Нашият параметър ще бъде именно името на потребителя и функцията ще извежда поздрава, конкатениран с подаденото име:

**Пример 3:**

```
<?php
function myGreeting($firstName){
    echo "Hello there ". $firstName . "!<br />";
}
myGreeting("Jack");myGreeting("Ahmed");
myGreeting("Julie");myGreeting("Charles");
?>
```

```
Изход:
Hello there Jack!
Hello there Ahmed!
Hello there Julie!
Hello there Charles!
```

**Пример 5:**

```
<?php
function myGreeting($firstName, $lastName){
    echo "Hello there ". $firstName ." ". $lastName ."!<br />";
}
myGreeting("Jack", "Black");
myGreeting("Ahmed", "Zewail");
myGreeting("Julie", "Roberts");
myGreeting("Charles", "Schwab");
?>
```

```
Изход:
Hello there Jack Black!
Hello there Ahmed Zewail!
Hello there Julie Roberts!
Hello there Charles Schwab!
```

Позволена е и рекурсия при функциите на PHP, тоест функцията да извиква сама себе си. Пример за това е една функция за изчисляване на факториел на число. Тоест, ние задаваме число, а функцията връща неговия факториел.

**Пример 6:**

```
<?php
function fact($n)
{ if ($n==0) return 1;
  else return $fact = $n * fact($n-1);
}
echo "fact(3)=".fact(3);
   // може да се напише echo (3*2);
   // но ако числото е по-голямо,
echo "<br>fact(10)=".fact(10);
   // то е много по-удобно да се ползва функцията,
   // вместо да пишем (10*9*8*...*3*2);
?>
```

Резултат:

```
fact(3)=6
fact(10)=3628800
```

Забележка:

- В PHP е възможно рекурсивното извикване на функции.
- Избягвайте рекурсивното извикване на функция/метод с повече от 200 нива на рекурсия, тъй като това може да доведе до проблеми със стека и до прекратяване изпълнението на скрипта.

**Пример 7.** Този пример демонстрира особеностите при деклариране на функция в условен блок: Когато една функция е декларирана в условен блок, то нейното дефиниране трябва да предшества нейното извикване:

```
Пример 8.
<?php
$make = true;
/* тук не бива да извикаме функция Make_event(); това е  фатална грешка!
защото тя още не съществува, но може да извикаме функция Save_info() */
if ($make){
// дефиниране на функция Make_event()
  function Make_event()
  {
    echo "<p>I would like to study Python<br>";
  }
}
/* сега може вече да извикаме Make_event(), защото
Make_event()е вече дефинирана! */
Make_event();
?>
```

Резултат:

```
I would like to study Python
```

**Пример 8.** Ако една функция е декларирана вътре в друга функция, то тя ще бъде достъпна само ако външната функция се изпълни поне веднъж.

Тъй като е дефинирана вътре във функция foo(), функция bar() не съществува, докато не се извика foo(). Иначе функциите и класовете в PHP имат глобална област на видимост - могат да бъдат извиквани и извън функцията в която са дефинирани.

```
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
<title>Functions</title>
</head>
 <body>
<?php
function foo()
{
function bar()
  {
    echo "Аз не съществувам, докато не се извика foo().\n";
  }
}
//bar(); Fatal error: Call to undefined function bar()
//in C:\xampp\htdocs\PhpProject16\index.php on line 17
/* тоест, не можем да извикаме bar(), преди да сме
 * извикали foo(), тъй като bar() все още не съществува. */
```

```
foo();
/* Сега можем да извикаме bar(),
   изпълнението на foo() я
   е направило достъпна. */
bar();
//Резултат: Аз не съществувам, докато не се извика foo().
?>
</body>
</html>
```

С *return* можем да прекъснем изпълнението на функция:
Пример 9.

```
<?php
function hello($who)
{
echo "<br>Hello $who";
if ($who == "Tom") {
return; // Изпълнението се прекъсва, ако параметъра е "Tom"
}
echo ", how are you";
}
hello("World"); // Отпечатва "Hello Tom"
hello("Reader") // Отпечатва "Hello Reader, how are you?"
?>
```

**Предаване на параметри на функции:**

По стойност е подразбиращия се метод, по който фактическите параметри предават своите стойности към функцията, тоест стойностите на фактическите параметри се копират във формалните параметри; измененията на параметрите (формалните параметри) вътре във функцията не се предават извън функцията, тоест фактическите параметри не променят своята стойност;

**Пример 10.1**: Предаване по стойност – изменението на $x се предава и извън функцията;

```
<?php
function f1 ($x){
   $x++;
}
$n=5;
f1($n); //$n се предава по стойност
echo "<br>n=".$n; //n=5
?>
```

Извежда се:

```
n=5
```

**Пример 10.2**: Предаване по адрес – изменението на $x се предава и извън функцията;

```
<?php
function f1 (&$x){
   $x++;
}
$n=5;
f1($n);  //$n се предава по адрес
echo "<br>n=".$n; //n=6
?>
```

Извежда се:

```
n=6
```

**Връщане на резултат от функция:**

**Пример 11.1**: Връщане на резултат по стойност (подразбиращ се начин на връщане на резултат)

```
<?php
function f1 ($x){
   $x++;
        return $x; //връща стойността на $x
}
$n=5;
$y=f1($n);// в $y ще се запише стойността на $x
echo "y=".$y; //y=6
echo "<br>n=".$n; //n=5
?>
```

Извежда се:

```
y=6
n=5
```

Пример 11.2: Връщане на резултат по адрес

В резултат на своята работа функцията може да връща адрес на някаква променлива. За да се върне от функцията адрес, е нужно, при декларирането на функцията, да поставим пред името на функцията знак амперсанд ( & ) и всеки път когато я извикваме – пред името и, също да поставяме амперсанд ( & ). Обикновено, функцията връща адрес на някакава глобална променлива, или адрес на елемент от глобален масив, или адрес на статична променлива или адрес на един от аргументите, ако той се предаде по адрес.

```
<?
function &ref($par){
return $GLOBALS['name'];
}
$name = "Peter";
$var =&ref($name);
/*var и name ще сочат едно и също място от паметта,
 * защото при такова извикване, чрез return при във $var
 *  не се копира стойността на глобалната променлива $name,
 *  а се създава указател към нея */
echo "<br>name is ".$name;  //name is Peter
echo "<br>name is ".$var;  //name is Peter
$var="Mimi";
echo "<br>name is ".$name;  //name is Mimi
   // извежда 10 и 10
?>
```

Извежда се:

```
name is Peter
name is Peter
name is Mimi
```

**Задаване на подразбиращи се стойности на параметри на функция**

В една функция може да се зададат подразбиращи се стойности на формалните параметри. Правилото е, ако такива има, то те да са последни в списъка с параметри. Самата подразбираща се стойност трябва да е константен израз, а не променлива, не представител на клас или извикване на друга функция.

**Пример 12: Задаване на подразбиращи се стойности на параметри**

```
<?
function Message($sign="The Rector of the TU - Varna.")
{
// тук параметър sign има стойност по подразбиране
  echo "Dear students, welcome to the Technical University of Varna,
Computer Science Department!<br>";
```

```
    echo $sign . "<br>";
  }
  Message();
      // Извиква се функцията без параметри.
      // В този случай се извежда: The Rector of the TU - Varna
  Message("Sincerely, Your Dean.");
      // В този случай се извежда: Sincerely, Your Dean
   ?>
```

Резултат:

```
Dear students, welcome to the Technical University of Varna, Computer
Science Department!
   The Rector of the TU - Varna.

Dear students, welcome to the Technical University of Varna, Computer
Science Department: !
   Sincerely, Your Dean.
```

Използване на подразбиращи се стойности на параметрите – ако при извикването на функцията пропуснем да зададем някой параметър, то ще се използва подразбиращата му стойност.

**Пример 13:** От PHP 4 нататък се поддържа списък с аргументи с променлива дължина в потребителски-дефинираните функции. Това, както се вижда и от примера не изисква специален синтаксис.

Чрез функцията **func_num_args()**, се връща броя (int) на аргументите, подадени към функцията**.** Чрез **func_get_arg()** – се извлича специфициран аргумент от списъка с аргументи на потребителси-дефинираната функция.

```
<?php
function foo()
{   $numargs = func_num_args();
     echo "Number of arguments: $numargs<br />\n";
     if ($numargs >= 0) {
         echo "First argument is: " . func_get_arg(0) . "<br />\n";
     }
}
foo (10, 12, 333);
//Number of arguments: 3
//First argument is: 10
?>
```

**Статични променливи във функциите**

**Пример 14:** Статични променливи във функциите – променливи, които запазват стойността си след изпълнение на функцията. Достъпни са само във функцията. Декларират се чрез static.

```
<?php
function fn(){
         static $counter = 0;
         $counter++;
         echo "Call number = $counter<br>";
         }
fn();fn();fn();
?>
```

Резултат:

```
Call number = 1
Call number = 2
Call number = 3
```

Статичните променливи се инициализират само при първото изпълнение на функцията.

**Вградени функции**

Php предоставя на програмиста голям брой вградени функции, с различно предназначение, които могат да се използват навсякъде без да е необходимо да се декларират. Справочник на математическите функции може да намерите на http://www.php.net/manual/bg/ref.math.php.

На http://www.php.net/manual/bg/math.constants.php можете да видите списък на предварително-дефинирани математически константи (те са част от ядрото на PHP).

Ето някои от тях:

abs — абсолютна стойност на число
acos — Аркускосинус
acosh — Хиперболичен арккосинус
asin — Аркуссинус
asinh — Хиперболичен аркуссинус
atan2 — Аркустангенс на две променливи
atan — Аркустангенс
atanh — Хиперболичен аркустангенс
ceil — Закръгляне на дробно число до по-голямо цяло, echo ceil(4.3); //5
cos — Косинус
cosh — Хиперболичен косинус
decbin — Переводит число из десятичной системы счисления в двоичную
deg2rad — Преобразува стойност от градуси в радиани
exp — е на степен подаденото число
floor — Закръгляне на дробно число до по-малко цяло
fmod — Връща дробния остатък при целочислено деление
getrandmax — Връща максимално възможното случайно число
hexdec — Переводит число из шестнадцатиричной системы счисления в десятичную
hypot — Рассчитывает длину гипотенузы прямоугольного треугольника
is_finite — Проверява дали стойността е допустимо крайно число
is_infinite — Проверява дали стойността е безкрайност
is_nan — Проверява дали стойността е "не число"
log10 — Десетичен логаритъм
log — Натурален логаритъм
max — Връща най-голямата стойност, echo max(1, 3, 5, 6, 7); // 7
min — Връща най-малката стойност
mt_getrandmax — Показва максимално възможната стойност на случайно число
mt_rand — Генерира случайно число по метод mt
mt_srand — Инициализира предварително генератора на случайни числа mt
pi — Връща числото Пи, echo pi(); // 3.1415926535898
pow — За степенуване, например echo pow(2,3); //8
rad2deg — Преобразува стойност от радиани в градуси
rand — Генерира случайно число от 0 до getrandmax(), може в диапазон, н.р. echo rand(5, 15);
round — Закръгля число от тип float
sin — Синус
sinh — Хиперноличен синус
sqrt — Корен квадратен, echo sqrt(9); //3
srand — Изменя началното число на генератора на псевдослучайни числа

tan — Тангенс
tanh — Хиперболичен тангенс и др.

**Пример 15: Проверка за съществуване на функция:** function_exists(string f_name) връща TRUE ако функцията с име f_name съществува, а ако не – FALSE;

Да допълним пример 14 по следния начин:

```
<?php
function fn(){
                static $counter = 0;
                $counter++;
                echo "Call number = $counter<br>";
                }
if (function_exists('fn')) {
    echo "fn function is available.<br />\n";
} else {
    echo "fn functions is not available.<br />\n";
}

fn();fn();fn();
?>
```

**Извеждане на масив с декларираните функции**:

get_defined_functions();

**Пример 16:** Функцията get_defined_functions(); връща масив с целочислени индекси, елементите на който съдържат имената на достъпните функции.

```
<PRE>
<?php
print_r(get_defined_functions());
?>
</PRE>
```

Резултат:

```
Array
(    [internal] => Array
        (   [0] => zend_version
            [1] => func_num_args
            [2] => func_get_arg
            [3] => func_get_args
            [4] => strlen
            [5] => strcmp
            [6] => strncmp
            [7] => strcasecmp
            [8] => strncasecmp
            [9] => each
            [10] => error_reporting
            [11] => define
            [12] => defined
            [13] => get_class
            [14] => get_called_class
```

...и т.н.

**Задача за самостоятелна реализация**

Създайте скрипт, съдържащ функция recArea, която получава 2 параметъра, за дължина $l и ширина $w на паралелепипед и изчислява лицето му $area. Извежда следния текст в прозореца на браузъра:

***Rectangle Area Function***

Правоъгълник с дължина 2 и ширина 4 има лице 8.

**Примерна реализация 1:**

```
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
<title>Rectangle Area Function</title>
</head>
<body>
<h2>Rectangle Area Function</h2>
<?php
//Дефиниране на функцията.
function recArea($l, $w)
{
$area = $l * $w;
echo "Правоъгълник с дължина $l и ширина $w има лице $area.";
}

//Извикване на функцията.
recArea(2, 4);
?>
</body>
</html>
```

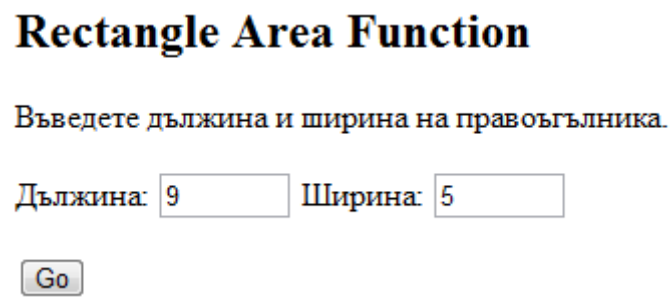
**Или примерна реализация 2:**

```
<?php
//Дефиниране на функцията.
function recArea($l, $w){
$a = $l * $w;
return $a;
}
//Извикване на функцията.
$area=recArea(2, 4);
echo "Правоъгълник с дължина 2 и ширина 4 има лице $area.";
?>
```

**Или примерна реализация 3:**

```
<?php
//Дефиниране на функцията.
function recArea($l, $w){
$a = $l * $w;
return $a;
}
echo "Правоъгълник с дължина 2 и ширина 4 има лице ". recArea(2, 4);
?>
```

**Примерна реализация 4 (Интерактивна):** Създайте нова версия, предоставяща форма на потребителя.




След натискане на бутона “**Go**” се получава отговора:

## *Rectangle Area Function*

Правоъгълник с дължина 9 и ширина 5 има лице 45

Вариант 1.

Функция **index.php:**

```
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
<title>Rectangle Area Function</title>
</head>
<body>
<h2>Rectangle Area Function</h2>
 <form method="post" action="yourfile.php">
<p>Въведете дължина и ширина на правоъгълника.</p>
<p>Дължина:  <input type="text" name="length" size="5" />
Ширина:  <input type="text" name="width" size="5" /></p>
<input type="submit" name="submit" value="Go" />
</form>
 </body>
</html>
```

Функция **yourfile.php:**

```
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
<title>Rectangle Area Function</title>
</head>
<body>
<h2>Rectangle Area Function</h2>
<?php
//Define function.
function recArea($l, $w){
  $area = $l * $w;
  return $area;
}
?>
<?php
if(isset ($_POST['submit'])){
//Retrieve user values.
$l = $_POST['length'];
$w = $_POST['width'];
//Use function with user values in statement.
echo "Правоъгълник с дължина $l и ширина $w има лице ". recArea($l, $w);
}
?>
 </body>
</html>
```

Вариант 2. Само с един файл:

**index.php**

```
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
<title>Rectangle Area Function</title>
</head>
<body>
<h2>Rectangle Area Function</h2>
<?php
//Define function.
```

```
function recArea($l, $w){
  $area = $l * $w;
  return $area;
}
?>
<form method="post" action="<? $_PHP_Self ?>">
<p>Въведете дължина и ширина на правоъгълника.</p>
<p>Дължина:  <input type="text" name="length" size="5" />
Ширина:  <input type="text" name="width" size="5" /></p>
<input type="submit" name="submit" value="Go" />
</form>
<?php
//Retrieve user values.
if(isset ($_POST['submit'])) //без това дава грешка!!!
{$l = $_POST['length'];
$w = $_POST['width'];
//Use function with user values in statement.
echo "Правоъгълник с дължина $l и ширина $w има лице ". recArea($l, $w);}
?>
 </body>
</html>
```