Лабораторно упражнение № 5

Функции. Деклариране и използване. Вградени функции.

1. Функции. Деклариране и използване.

Функциите са неразделна част от всеки език за програмиране от високо ниво.
Функциите в PHP имат следния синтаксис:

```
function fname($arg1, $arg2,  ...,  $argn)
{
        echo "Примерна функция.\n";
        return $value;
}
```

Където:

fname – име на функцията. Имената на функциите в PHP се подчиняват на същите изисквания както имената на променливите. За разлика от променливите обаче при имената на функциите няма значение дали те са зададени с големи или малки букви. т.е. названията fname(), Fname() и FNAME() ще бъдат възприети от PHP като една и съща функция. Понеже не е възможно да имате две или повече функции с еднакви названия, то при създаването на всяка функция трябва да задавате уникално име, което се отличава от другите имена на функции поне по един символ, а не по големите и малките букви.

$arg1, $arg2…- списък от параметри на функцията. Аргументите на функциите се записват в скоби, разделени със запетая и съдържат входни данни на функциите. PHP позволява предаване на аргументи по стойност (по подразбиране) и по адрес (параметри-псевдоними). Ако на даден аргумент на функция се зададе някаква стойност при дефинирането на функцията, тогава се казва, че това е стойността на аргумента по подразбиране. Ако при извикването на функцията не бъде зададена някаква друга стойност на аргумента, се използва стойността по подразбиране. Ако обаче при извикването на функцията се укаже друга стойност на аргумента, тогава стойността по подразбиране се пренебрегва автоматично и се работи с новата стойност.

return израз – незадължителен оператор, чрез който функцията връща стойността на израза и се прекратява изпълнението на функцията.

Пример 1. Функция за изчисляване на факториел на число.

```
<?php
    function fact($n)
    {
        if ($n==0)
            return 1;
        else
            return  $n * fact($n-1);
    }
    $n=3;
    echo "fact($n)=".fact($n);
```

```
?>
```

Резултат:

```
fact(3)=6
```

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

Пример 2. 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.

```
<?php
$make = true;
/* тук не бива да извикаме функция Make_event(); това е
фатална грешка, защото тя още не съществува, но може да
извикаме функция Save_info() */
//Това извикване на функцията ще изведе стойностите по
подразбиране
Save_info();
if ($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();
// дефиниране на функция Save_info
function Save_info($first='Ivan', $last='Ivanov',
$message='I study PHP'){
  echo "<br>$message<br>";
  echo "Name: ". $first . " ". $last . "<br>";
}
//Това извикване на функцията ще изведе стойностите на
параметрите
Save_info("Nina","Nikolova", "I study Lisp");
?>
```

Резултат:

```
I study PHP
Name: Ivan Ivanov
I would like to study Python
I study Lisp
Name: Nina Nikolova
```

Пример 3. !!!Ако една функция е дефинирана вътре в друга функция, то тя ще бъде достъпна само ако външната функция се изпълни поне веднъж.
Тъй като е дефинирана вътре във функция funk1(), функция funk2() не съществува, докато не се извика funk1(). 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 funk1()
{
      function funk2()
      {
            echo "Аз не съществувам, докато не се извика
            func1().";
      }
}
funk1();
funk2();
?>
</body>
</html>
```

Резултат:
Аз не съществувам, докато не се извика func1().
Пример 4. С оператор ***return*** можем да прекъснем изпълнението на функция.
Операторът return прекратява изпълнението на текущо изпълняваната функция, при което управлението се предава на функцията, която я е извикала, и връща своя аргумент като резултат от извикването на тази функция. Ако се използва на глобално ниво операторът return прекратява изпълнението на целия скрипт (т.е. файл с разширение php). Ако файлът е добавен чрез include или require, то управлението се връща обратно на извикващия файл. Освен това ако файлът е добавен чрез include, то стойността, върната от return, е стойност, върната от цялото извикване на файла.

```
<?php
     function hello($who)
     {
      echo "<br>Hello $who";
      if ($who == "World")
            return;
      echo ", how are you";
     }
     hello("World");
```

```
    hello("Reader")
?>
```

Резултат:

Hello World
Hello Reader, how are you?

Пример 5. Функциите не могат да връщат повече от една стойност, но могат да върнат масив от стойности.

```
<?php
    function init_array ()
    {
     return array(10,20,30);
    }
    list($first, $second, $thierd)=init_array();
    echo "Values: $first, $second, $thierd";
?>
```

Резултат:

Values: 10, 20, 30

2. Предаване на параметри на функции

2.1. Предаване на параметри по стойност

Предаването на параметри по стойност е метод, при който фактическите параметри предават своите стойности към функцията, като стойностите на фактическите параметри се копират във формалните параметри. Измененията на параметрите (формалните параметри) вътре във функцията не се предават извън функцията (фактическите параметри не променят своята стойност).

Пример 6. Предаване по стойност – изменението на $x не се предава извън функцията.

```
<?php
    function f1 ($x)
    {
     $x+=10;
     echo "x= $x";
    }
    $n=5;
    f1($n);
    echo "<br>";
    echo "n=".$n;
?>
```

Резултат:

```
x= 15
n=5
```

2.2. Предаване на параметри по адрес

Предаването на параметри по адрес е метод, при който се работи с оригиналните параметри, а не с техните копия.

Пример 7. Предаване по адрес – изменението на $x се предава и извън функцията.

```
<?php
     function f1 (&$x)
     {
      $x+=10;
      echo "x= $x";
     }
     $n=5;
     f1($n);
     echo "<br>";
     echo "n=".$n;
?>
```

Резултат:

```
x= 15
n= 15
```

3. Задаване на подразбиращи се стойности на параметри

Пример 8. Задаване на подразбиращи се стойности на параметри

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

```
<?php
     function Message($sign="The Rector of the TU - Varna.")
     {
          echo "Dear students, welcome to the Technical
          University of Varna!<br>";
          echo $sign . "<br>";
     }
     Message(); // Извиква се функцията без параметри.
     Message("Sincerely, Your Dean.");
?>
```

Резултат:

```
Dear students, welcome to the Technical University of Varna!
The Rector of the TU - Varna.
Dear students, welcome to the Technical University of Varna!
Sincerely, Your Dean.
```

От PHP 4 нататък се поддържа списък с аргументи с променлива дължина в потребителски дефинираните функции. 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 потребителски дефинираната функция.

Пример 9. Демонстрация на функциите **func_num_args()** и **func_get_arg().**

```
<?php
```

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

Резултат:

Number of arguments: 3
First argument is: 10

4. Статични променливи във функция.

Пример 10. Статични променливи във функциите – променливи, които запазват стойността си след изпълнение на функцията. Достъпни са само във функцията. Декларират се чрез static. Статичните променливи се инициализират само при първото изпълнение на функцията.

```
<?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
```

5. 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 функциите може да намерите на:
http://www.php.net/manual/bg/funcref.php
http://php.saparev.com/funcref.html

5.1. function_exist(string f_name) – проверява дали дадена функция съществува. Функцията връща като резултат TRUE ако съществува функция с име f_name, а ако не – FALSE.

Пример 11. Допълнение към Пример 10.

```
<?php
```

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

Резултат:

```
fn function is available.
Call number = 1
```

5.2. Специализирани функции за работа със символни низове

PHP не притежава клас String със съответните му функции за обработка на низове, но предоставя набор от функции за основните символни операции: търсене, заместване, разделяне на части и други.

5.2.1. strlen($niz) – определя дължина на низ.

Пример 12. Демонстрира достъп до елемент от низ и функция за дължина на низ.

```
<PRE>
<?php
    $a = "Hello world";
    for ($i = 0; $i<strlen($a); $i++)
      echo "$a[$i]"."<br>";
    print_r($a);
?>
</PRE>
```

Резултат:

```
H
e
l
l
o

w
o
r
l
d
Hello world
```

5.2.2. int strpos(низ, подниз[, начален индекс за търсене=0]) - връща началния индекс на подниза в низа, ако е открит. В противен случай връща FALSE.

5.2.3. string strstr(низ, подниз) -търси първото появяване на подниз в низ и ако бъде открит връща низ: от началото на откритият низ до края на стринга.

5.2.4. str_split(низ[, дължина на подниза]) - разделя низ на части със зададен размер

5.2.5. explode(разделител, низ[, max брой разделения]) – разделя низ на части по зададен разделител,

Пример 13.

```
<PRE>
<?php
     $a = "Hello world";
     if(strpos($a,"Hello")!=0)
       echo "Sub-string 'Hello'is not found!<br>";
     else
     {
       echo "Sub-string 'Hello'is found!<br>";
       $i=strpos($a,"Hello");
       echo "<br>"."Test for strpos:  ";
       echo "The index of \"Hello\" is $i";
      }
      echo "<br>"."Test for strstr($a, \"lo\"):
      ".strstr($a,"lo");
      $b = explode(" ", $a);
      echo "<br>"."Test for explode:  "."<br>";
      print_r($b);
      $b=str_split($a, 2);
      echo "<br>"."Test for str_split:  "."<br>";
      print_r( $b);
?>
</PRE>
```

Резултат:

```
Sub-string 'Hello'is found!

Test for strpos:  The index of "Hello" is 0
Test for strstr(Hello world, "lo"):lo world
Test for explode:
Array
(
    [0] => Hello
    [1] => world
)

Test for str_split:
Array
```

```
(
    [0] => He
    [1] => ll
    [2] => o
    [3] => wo
    [4] => rl
    [5] => d
)
```

5.2.6. strcmp() – сравняване на низове. Функцията е чувствителна към регистъра.

5.2.7. strcasecmp() - сравняване на низове. Функцията е нечувствителна към регистъра.

Пример 14. Сравняване на низове.

```
<?php
    $string = '123aa';
    if ($string == 123)
       echo "The string equals 123"."<br>";
    $string = "Hello World";
    if (strcmp($string, "hello world") == 0)
       echo "false - функцията е чувствителна към  регистъра"
       ."<br>";
    if (strcasecmp($string, "hello world") == 0)
       echo "true - strcasecmp(), не е чувствителна към
       регистъра"."<br>";
?>
```

Резултат:

The string equals 123
true - strcasecmp(), не е чувствителна към регистъра

5.2.8. str_replace(какво търсим, с какво го заместваме, в кой изходен низ) – търсене и заместване в низ. Функцията е чувствителна към регистъра.

5.2.9. str_ireplace(какво търсим, с какво го заместваме, в кой изходен низ) - търсене и заместване в низ. Функцията не е чувствителна към регистъра.

Пример 15. Премахване на белите полета - trim(), ltrim(), rtrim().Промяна на регистъра - strtolower(), strtoupper(), ucfirst(), ucwords().

```
<?php
    //Премахване на белите полета - trim(), ltrim(), rtrim()
    $title = " Programming PHP ";
    echo "title=$title.<br>";
    $str_1 = ltrim($title); // $str_1 е "Programming PHP "
    $str_2 = rtrim($title); // $str_2 е " Programming PHP"
    $str_3 = trim($title); // $str_3 е "Programming PHP"
    echo "str1=$str_1,str2=$str_2,str3=$str_3.<br>";
    //Промяна на регистъра - strtolower(), strtoupper(),
    ucfirst(), ucwords()
    $string1 = "FRED flintstone";
    $string2 = "barney rubble";
    print(strtolower($string1))."<br>"; // fred flintstone
```

```
    print(strtoupper($string1)."<br>"; // FRED FLINTSTONE
    print(ucfirst($string2)."<br>"; // Barney rubble
    print(ucwords($string2)."<br>"; // Barney Rubble
?>
```

Резултат:

```
title= Programming PHP .
str1=Programming PHP ,str2= Programming PHP,str3=Programming
PHP.
fred flintstone
FRED FLINTSTONE
Barney rubble
Barney Rubble
```