Функции. Деклариране и използване. Функции за операции с файлове

Виолета Божикова

Какво разглежда тази лекция?

- **Създаването на потребителски функции в PHP.**
- **Специализирани функции за работа със символни низове**
- **Функции за операции с файлове**

Деклариране на функция

- Функция може да бъде декларирана посредством следния синтаксис:

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

```
<?
function fname ()
{
$a="hell";
return $a;
}
$c=Fname();
echo $c; //hell
?>
```

Където: fname – име на функцията, case insensitive

- $arg1, $arg2…- списък от параметри на функцията (незадължителен),
- [return израз;] – незадължителен оператор, чрез който функцията връща стойността на израза и се прекратява изпълнението на функцията (виж примера!!!).
- return; // Изпълнението на функцията се прекъсва

За рекурсията

- 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.

Един пример - функция за изчисляване на факториел (рекурсия)

- Ще създадем една функция **fact($n)** за изчисляване на факториел на число: функцията се вика два пъти и отпечатва факториела на 3 и на 10.

```
<?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

Особености при деклариране на функция

- 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 (Пример 2.2).
- **Условно Деклариране на функция:** Когато една функция е декларирана в условен блок, то нейното дефиниране трябва да предшества нейното извикване (Пример 2).
- PHP не поддържа повторното дефиниране на функция, т.е няма функции с еднакви имена!

Особености при деклариране на функция

- Пример 2.2. Тъй като е дефинрана 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.

```
function foo()
{echo "Аз съм foo().\n";
function bar()
  {
    echo "<br>Аз съм bar(). Аз не съществувам, докато не се извика foo().\n";
  }
}
       /* Не можем да извикаме bar() все още,
          понеже не съществува. */
foo();              //Аз съм foo()
       /* Сега можем да извикаме bar(),
          изпълнението на foo() я
          е направило достъпна. */
bar();          //Аз съм bar(). Аз не съществувам, докато не се извика foo().
?>
```

- С **return;** можем да прекъснем изпълнението на функция:

```
…
function hello($who)
{
echo "<br>Hello $who";
if ($who == "World")
return;
  /*Ако аргумента е "World” изпълнението на функцията се
    прекъсва */
echo ", how are you";
}
hello("World"); // Отпечатва "Hello World"
hello("Reader"); // Отпечатва "Hello Reader, how are you?"
…
```

Пример 2.

```php
<?php
$make = true;
/* тук не бива да извикаме функция Make_event();
това е  фатална грешка!
защото тя още не съществува, но може да извикаме
функция Save_info() */
Save_info("Ivan","Ivanov",  "I study PHP");
if ($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, $last, $message){
  echo "<br>$message<br>";
  echo "Name: ". $first . " ". $last . "<br>";
}
Save_info("Nina","Nikolova", "I study Lisp");
// Save_info може да се извика и тук
?>
```

Пример 3:Предаване по стойност

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

Пример 4:Предаване по адрес

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

?>
```

Резултат (Пример 2):

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

Предаване на параметри на функции:

- **по стойност – по подразбиране** това е метода **,** по който фактическите параметри предават своите стойности към функцията (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;
- **по адрес** – изменението на формалния параметър във функцията (например **$y**) се предава и извън функцията (върху фактическия **$x**):

  Извикване на функцията, например: **f($x);**

  Дефиниране на самата функция: **function f(&$y) {…}**
- Примери 3 и 4.

Пример 5:Връщане на резултат по стойност

```php
<?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

?>
```

Връщане на резултат:

пример 6.1

```php
<?php
$name="Peter";
function &get_name()
{return $GLOBALS["name"];}
$new_name=&get_name();
/*сега $new_name и $name ще сочат едно и също
място към паметта, защото при такова извикване,
чрез return в $new_name не копира  стойността
на глобалната променлива $name, а се създава
указател към нея */
$new_name="John";
/*промяната на $new_ name води по промяна и на
$name */
echo "User name is ".$name;
//Извежда  User name is John
?>
```

Връщане на резултат:

- В резултат на своята работа функцията може да връща **адрес на някаква променлива** (пример 6).
- 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 (&).

Връщане на резултат:

пример 6.2

- Обикновено, функцията връща:
  - адрес на някакава глобална променлива (както е в примери 6.1 и 6.2),
  - 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){
global $Odd, $Even;
  if ($par % 2 == 0) return $Even;
  else return $Odd;
}
$Odd = "Odd number";
$Even = "Even number"; //Нечетно, Четно
$a=4;
$var =&ref($a);
/*var и Even ще сочат едно и също място от
паметта,защото при такова извикване, чрез return
при във $var не се копира стойността на
глобалната променлива $Even, а се създава
указател към нея */
echo "<br>$a is an ".$var;  //4 is an Even number
echo "<br>$a is an ".$Even;  //4 is an Even
number
$a=17;
$var =&ref($a);
echo "<br>$a is an ".$var;//17 is an Odd number
echo "<br>$a is an ".$Odd;//17 is an Odd number
   // извежда 10 и 10
?>
```

Задаване на подразбиращи се стойности на параметри на функция

- При функциите в PHP е позволено дефинирането на параметри с подразбиращи се стойности.
- 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 7 демонстрира използване на параметри с подразбиращи се стойности: Така, ако при извикването на функцията пропуснем да зададем стойност на някой параметър с подразбираща се стойност, то ще се използва подразбиращата му стойност.

Пример 7: Задаване на подразбиращи се стойности на параметри

```
<?
function Message($sign="The Rector of the
TU - Varna.")
{
/* тук параметър sign има стойност по
подразбиране*/
echo "Some Information...to the students
of Computer Science Department: !<br>";
echo $sign . "<br>";
}
Message();
// Извиква се функцията без параметри.
/* В този случай – подпис: The Rector of
the TU – Varna*/
Message("Sincerely, Your Dean.");
/* В този случай – подпис: Sincerely,
Your Dean*/
 ?>
```

Резултат:

Some Information...to the students of Computer Science Department: !
The Rector of the TU - Varna.
Some Information...to the students of Computer Science Department: !
Sincerely, Your Dean.

Използване на подразбиращи се стойности на параметри

Списък на аргументи с променлива дължина и статични променливи във функции

- При потребителски-дефинираните функции в PHP (от версия 4 нататък) се поддържа променлив брой параметри (аргументи) (Пример 8).
- Това не изисква специален синтаксис.
  - Посредством функцията func_num_args() се връща броя (int) на аргументите, подадени към функцията, а
  - Чрез func_get_arg() – се извлича специфициран аргумент от списъка с аргументи на потребителси-дефинираната функция.
- **Статични променливи във функциите** – променливи, които запазват стойността си след изпълнение на функцията. **Достъпни са само във функцията**. Декларират се чрез **static** (пример 9).
  - Статичните променливи **се инициализират само** при първото изпълнение на функцията.

Списък на аргументи с променлива дължина и статични променливи във функции

Пример 8: Списък на аргументи с променлива дължина

```
<?php
function DataCheck(){
  $n = func_num_args();
  echo "Number of function's arguments: $n<br>";
}
DataCheck();
DataCheck(1,2,3);
?>
```

Резултат:

Number of function's arguments: 0
Number of function's arguments: 3

Пример 9: Статични променливи във функциите

```
<?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

```
<?php
function foo()
{$numargs = func_num_args();
 echo "Number of arguments: $numargs<br/>\n";
if ($numargs >= 2)
echo "Second argument is: " . func_get_arg(1) .
 "<br/>";
}
foo (1, 2, 3);
?>
```

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. Подробна информация на:

http://www.php.net/manual/bg/funcref.php

http://php.saparev.com/funcref.html

- **Проверка за съществуване на функция:**

bool function_exists(string f_name);

връща TRUE ако функцията с име **f_name** съществува, а ако не – FALSE; Пример: **if (function_exists('f_name')) echo " available";**

- **Извеждане на масив с декларираните функции**(пр.10):

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

Пример 10: Извежда 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] ...и т.н.
```

Пример 11: достъп до елемент от 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
```

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 др.
- **Достъп до елементите на низ и дължина на низа (strlen()):**

```
$a = "Hello world“;
for ($i = 0; $i<strlen($a); $i++)
echo "$a[$i]"."<br>";
```

функции за търсене на подниз

- Php предоставя 2 вида **функции за търсене на подниз**: функции, които връщат индекс на подниз и функции, които връщат самия подниз.
  - функции за търсене, които връщат индекс на подниз в символен низ

int strpos(низ, подниз[, начален индекс за търсене, 0 по подразбиране])

- strpos() връща началния индекс на подниза в низа, ако е открит. В противен случай връща FALSE (пример 12).

функции за търсене на подниз

Още функции за търсене, които връщат индекс на подниз в символен низ:

int stripos(низ, подниз[, начален индекс])

- същата като предната функция, но тази не прави разлика между големи и малки букви (case-insensitive);

int strrpos(…)

- като предната, но започва да търси от дясно наляво, тоест намира индекса на последния подниз;

int strripos(…)

- търси от дясно наляво и не прави разлика между малки и големи букви;

функции за търсене на подниз

– функции за търсене, които връщат подниз от символен низ

int strstr(низ, подниз)

- търси първото появяване на подниз в низ и ако бъде открит връща низ: от началото на открития подниз до края на стринга.

```
$a = "Hello world";
echo "<br>".strstr($a, "lo");//lo world
```

int stristr(низ, подниз)

- също като предната, но не прави разлика между малки и големи букви;

```
$a = "Hello world";
echo "<br>".stristr($a, "LO");//lo world.
```

функции за низове - примери

Пример 12: функции за търсене, които връщат индекс на подниз в символен низ

```
<PRE>
<?php
$a = "Hello world";
if(strpos($a,"Hello")===FALSE)
echo "Sub-string 'Hello'is not
found!<br>";
else
{echo "Sub-string 'Hello'is found!<br>";
$i=strpos($a,"Hello");
echo $i;}
echo "<br>".strstr($a, "lo");
//lo world
?>
</PRE>
```

Изход:

```
Sub-string 'Hello'is found!
0
lo world
```

Пример 13:

```
<?php
/* split the phrase by any number of
commas or space characters,
which include " ", \r, \t, \n and \f */
$keywords = preg_split("/[\s,]+/",
"hypertext language, programming");
print_r($keywords);
?>
</PRE>
```

Резултат:

```
Array
(
    [0] => hypertext
    [1] => language
    [2] => programming

)
```

Функции за разделяне на низ на части

- частите на низа се връщат като елементи на масив:
  - **Части със зададен размер:**

str_split(низ[, дължина на подниза]);

```
$a = "Hello world";
$b=str_split($a, 2);
print_r( $b);
```

Извежда масив:

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

Функции за разделяне на низ на части - по зададен разделител

- **частите се връщат като елементи на масив:**

Функция:

explode(разделител, низ[, max брой върнати части]);

```
$a = "p1 p2 p3";
$b = explode(" ", $a); print_r($b);
$b = explode (" ", $a, 2); print_r($b);
```

Извежда:

Array ([0] => p1 [1] => p2 [2] => p3)

Array ([0] => p1 [1] => p2 p3)

Функции за разделяне на низ на части – по разделител регулярен израз

- частите се връщат като елементи на масив
- Функция explode дава възможност за разделяне на низ при един указан разделител. Понякога се налага разделянето да е по няколко разделителя. За тази цел може да се използва функцията **preg_split** в която разделителя е регулярен израз.

array preg_split(рег. израз, низ[, max брой върнати части [, опции]]);

- Опциите са различни, н.р:

PREG_SPLIT_NO_EMPTY – празните поднизове няма да се записват в резултантния масив;

... И др.

Пример 13: Разделя се един string в компоненти - символи

Функции за разделяне на низ на части - по зададена позиция

- частите се връщат като елементи на масив.

substr (низ, начален индекс[, дължина]);

- ако не се укаже дължина, низът се извежда до края;

```
$rest = substr("abcdef", 2);
echo $rest; // извежда "cdef“
```

търсене и заместване в низ

str_replace(), str_ireplace()

- str_replace("World", "Reader", "Hello World");

какво, с какво заместваме, в кой изходен низ

//чувствителна към регистъра

- str_ireplace("world", "Reader", "Hello World");

//не е чувствителна към регистъра

//Hello Reader е резултата и в двата случая

Указва се какво търсим, с какво заместваме, в кой изходен низ

Премахване на белите полета - trim(), ltrim(), rtrim()

```
$title = " Programming PHP \n";
$str_1 = ltrim($title);
// $str_1 е "Programming PHP \n"
$str_2 = rtrim($title);
// $str_2 е " Programming PHP"
$str_3 = trim($title);
// $str_3 е "Programming PHP”
```

Промяна на регистъра - strtolower(), strtoupper(), ucfirst(), ucwords()

```
$string1 = "FRED flintstone";
$string2 = "barney rubble";
print(strtolower($string1)); // fred flintstone
print(strtoupper($string1)); // FRED
   FLINTSTONE
print(ucfirst($string2)); // Barney rubble
print(ucwords($string2)); // Barney Rubble
```

Функции за четене от файл

Функциите за четене от файл: поддържат **файлов указател**, като четат последователно съдържанието на файла, започвайки от позицията на указателя, премествайки указателя след прочитане на определен брой байтове или символи.

- **Функция за четене на ред от файл: fgets().** Чете ред от файл, от позиция, сочена от файловия указател:

string fgets (resource $handle [, int $length])

Където:

- ***handle*** **-** това е файловият указател (манипулатор), получен от функцията fopen(), а
- ***length*** – брой байтове, които трябва да бъдат прочетени**.**

Функции за четене от файл ()

- Файловият манипулатор ***handle*** трябва да бъде валиден, тоест трябва да сочи към файл, който е отворен успешно.
- Четенето (с **fgets)** завършва, когато се изпълни едно от условията:
  - изчетени са (**length-1)** символи или е
  - срещнат символ за нов ред, който се включва във върнатия низ,
  - или се достигне EOF (край на файла).
- !!!Ако дължина **length** не е подадена, четенето ще продължи до края на реда.
- **Забележка**: До PHP 4.3.0, неподаването на *length* се подразбираше като подаване на 1024.

Функции за четене от файл

Връщани стойности (за fgets)

- Връща низ с дължина до (length – 1) байта, изчетен от файл указан чрез *handle* . При грешка връща **FALSE**.
- Примери:В примерите се прочита и извежда последователно, по редове съдържанието на файла proba.txt, който в случая е в същата папка в която е php файла. Етикетът за преформатиране "<pre>" (primer 15) е необходим за да отработи символа за преминаване на нов ред при извеждане на текста във Web браузъра (без него, прочетеният файл се визуализира на един ред).

Пример 14: четене от файл: fgets

```
<pre>
<?php
$fp = @fopen("proba.txt", "r");
if ($fp) {
    while (!feof($fp)) {
        $buffer = fgets($fp, 4096);
        echo $buffer;
    }
    fclose($fp);
}
?>
</pre>
```

Пример 15: четене от файл: fgets

```
<?php
echo "<pre>";
$fp = fopen("proba.txt", "r") or
die("<br>can't open the file for
reading");
    while (!feof($fp)) {
        $buffer = fgets($fp, 4096);
        echo $buffer;
    }
    fclose($fp);
echo "</pre>";
?>
```

Други Функции за файлове

Функция за проверка за край на файл

- **feof(resource fp)** – връща TRUE ако е достигнат край на файла, например: **while (!feof($fp)) {...четем файла...**

```
Пример 16 (за fgetc())
<?php
$fp = fopen('proba.txt', 'r');
if (!$fp) {
    echo "<br>I can't open the file";
}
while (false !== ($char = fgetc($fp))) {
    echo "$char\n";
}
?>
```

```
Пример 17 (fread)
<?php
// изчита съдържанието на файл в низ
$filename = "proba.txt";
$handle = fopen($filename, "r");
$contents = fread($handle,
filesize($filename));
print_r($contents); //отпечатва
прочетения низ в прозореца на браузъра
fclose($handle);
?>
```

Функции за четене от файл

- **Функция fgetc –** прочита един символ от текущата позиция на файловия указател
- **Описание**

string **fgetc** (resource $handle)

- **Параметри**

$handle - Файловият указател трябва да бъде валиден, и трябва да сочи към файл, който е бил отворен успешно чрез функция fopen() (и все още да не е затворен от fclose()).

Функции за четене от файл

Връщани стойности (от fgetc)

- Връща низ, съдържащ един символ, от файл указан чрез манипулатор **handle**. Символът се взима от текущата позиция, сочена от указателя на файла. Връща **FALSE** в случаите, когато указателят на файла е в неговия край (**EOF**).
- **Предупреждение:**Тази функция може да върне булева стойност **FALSE**, но може също да върне небулева стойност, която се интерпретира като **FALSE**, като 0 или "". Използвайте оператора === за проверка на връщаната стойност от тази функция.

Функции за четене от файл

- **Четене на зададен брой байтове – fread:**

string **fread** (resource $handle , int $length)

- **fread()** изчита до ***length*** символа, започвайки от файловия указател, указан чрез ***handle***. Четенето спира при едно от следните събития (Пример 17):
  - ***length*** **bytes са прочетени**
  - **достигнат е EOF (край на файла)**

Параметри на fread:

- ***handle*** – Указател на файлов ресурс, който обикновено се създава посредством fopen();
- ***length*** - Максимален брой на изчетените байтове.

Функции за четене от файл

- **fread()** чете от текущата позиция на указателя за файла. Използвайте ftell(), за да разберете текущата позиция и rewind(), за да върнете указателя в началото на файла.

Връщани стойности:

- Връща изчетения низ или **FALSE** в случай на грешка.
- **fread** е по-подходяща функция за двоични файлове, понеже не прекратява четенето при символ за край на ред, а **fgets** е по-подходяща за четене на текстови файлове.
- При операционни системи, където има разлика между двоични и текстови файлове (примерно Windows), файлът трябва да бъде отворен с режим 'b', при извикването на fopen().

Функции за четене от файл

file_get_contents()

- Ако просто ви трябва да сложите съдържанието на файл в низ - използвайте file_get_contents(), която има много по-добра производителност от **fread()** – Пр. 17.

string **file_get_contents** (string $filename [, int $flags [, resource $context [, int $offset [, int $maxlen]]]])

- Функцията **file_get_contents()** връща резултата от четене на файла ***filename*** в низ, като започва четенето на ***filename*** от отместване ***offset*** и изчита ***maxlen*** байта. При грешка **file_get_contents()** ще върне **FALSE.**

За file_get_contents()

Предупреждение:

- Параметърът **flags** (за **file_get_contents**) е наличен от версия PHP 6 и по-нови.
- Стойността на параметър **flags** може да бъде комбинация от следните флагове (с някои ограничения):
  - **FILE_BYNARY** (файлът се изчита в двоичен режим) или
  - **FILE_TEXT** (файлът се изчита като текстов).
  - Може да се зададе също и флаг **FILE_USE_INCLUDE_PATH** (търси файла и в include_path).
- **offset** - Отместването от което започва четенето.
- **maxlen** - Максималната дължина на данните, които да бъдат изчетени.

Функции за запис във файл

- Функция **fwrite()** - записва съдържанието на *string* във файловия поток сочен от **$handle** .

int **fwrite** (resource $handle , string $string [, int $length])

- *handle* – Указател на файлов ресурс, който обикновено се създава посредством fopen().
- *string* - Низът, който ще бъде записан.
- *length* - Ако е подаден параметър *length* , записът ще спре след като *length* байта са записани или докато се достигне края на *string*, тоест - което от двете се случи първо.

Функции за запис във файл

- **fwrite()** връща броя на записаните байтове или **FALSE** при грешка.
- **Бележки:**
  1. При операционни системи, които правят разлика между двоични и текстови файлове (като Windows), файлът трябва да е отворен с флаг "b" при извикването на fopen().
  2. Ако запишете два пъти във указател към файл, вторите данни ще бъдат добавени към края на файла, т.е резултатът от следващия пример е следния:

Пример:

```
<?php
   $fp = fopen('data.txt', 'w');
   fwrite($fp, '1');
   fwrite($fp, '23');
   fclose($fp);
   // Съдържанието на 'data.txt' сега е 123, а не 23!
   ?>
```

- **Функция fputs - Псевдоним на fwrite()**

Функция file_put_contents - Записва низ във файл:

int file_put_contents (string $filename , mixed $data [, int $flags [, resource $context]])

- Функцията извършва същото, както и последователността fopen(), fwrite() и fclose() (извикани успешно).
- ***filename*** - Път до файла в който ще бъдат записвани данните. Ако *filename* не съществува ще бъде създаден. Ако съществува ще бъде презаписан, освен ако не е подаден флаг **FILE_APPEND**. Стойността на *flags* може да бъде комбинация от различни флагове, свързани с ИЛИ оператор (|).
- ***data*** - Данните, който ще бъдат записани. Могат да са string, array или stream.

Налични флагове

Флаг	Описание
FILE_USE_INCLUDE_PATH	Търси файла *filename* и в пътищата за включване. Вижте include_path за повече информация.
FILE_APPEND	Ако файлът *filename* съществува, данните ще бъдат добавени към него вместо да бъдат презаписани отгоре.
LOCK_EX	Придобива изключително заключване върху файла докато се записва в него.
FILE_TEXT	Данните от *data* се записват в текстов режим. Ако се използва unicode семантика, кодирането по подразбиране ще е UTF-8. Можете да използвате различно кодиране чрез създаването на context или чрез използването на **stream_default_encoding()**, за да промените подразбиращото се кодиране. Този флаг не може да бъде използван едновременно с

Пример: **Функция file_put_contents**

```
$filename = "proba.txt";
$handle = fopen($filename, "w");
$s="Hello boys and Girls!";
file_put_contents('proba.txt',$s);
//запис във файла на $s
fclose($handle);
```

Други Функции за работа с файл

Функцията fopen — Отваря файл или URL:

resource **fopen** (string $filename, string $mode [, bool $use_include_path [, resource $context]])

Функцията **fopen()** свързва именован ресурс, определен чрез параметър *filename,* с поток. Тук:

- Ако *filename* е локален файл той е във формат "c:\\data\\info.txt" или "c:/data/info.txt ". Пример:

<?php $handle = fopen("c:\\data\\info.txt", "r"); ?>

- **mode** - определя типа на достъп, който искате за потока. Той може да бъде един от следните: **r,r+,w, w+,a, a+, x, x+.** Още:**b** (bynary).

Пример: **$fp = fopen('data.txt', 'wb');**

Списък за възможните режими при работа fopen()

Режим	**Описание**
'r'	Отваря само за четене. Установява указателя на файла в неговото начало.
'r+'	Отваря за четене и запис. Установява указателя на файла в неговото начало.
'w'	Отваря само за запис. Установява указателя на файла в неговото начало и нулира дължината на файла. Ако файлът не съществува се опитва да го създаде.
'w+'	Отваря за четене и запис. Установява указателя на файла в неговото начало и нулира дължината на файла. Ако файлът не съществува се опитва да го създаде.
'a'	Отваря само за запис. Установява указателя на файла в неговия край. Ако файлът не съществува се опитва да го създаде.
'a+'	Отваря за четене и запис. Установява указателя на файла в неговия край. Ако файлът не съществува се опитва да го създаде.
'x'	Отваря само за запис. Установява указателя на файла в неговото начало. Ако файлът съществува, **fopen()** ще върне **FALSE** и ще генерира грешка от тип **E_WARNING**. Ако файлът не съществува се опитва да го създаде. Това е равносилно да подаване на флагове *O_EXCL|O_CREAT* към използваната отдолу системна функция *open(2)*. Този режим се поддържа от PHP 4.3.2 и по-късните и работи единствено за локални файлове.
'x+'	Създава и отваря за четене и запис. Установява указателя на файла в неговото начало. Ако файлът съществува, **fopen()** ще върне **FALSE** и ще генерира грешка от типе **E_WARNING**. Ако файлът не съществува се опитва да го създаде. Това е равносилно да подаване на флагове *O_EXCL|O_CREAT* към използваната отдолу системна функция *open(2)*. Този режим се поддържа от PHP 4.3.2 и по-късните и работи единствено за локални файлове.

Други Функции за работа с файл

Подробен списък може да намерите на http://php.saparev.com/ref.filesystem.html.

Тук ще споменем само още няколко функции:

- **Проверка за съществуване на файл:**

bool file_exists (string $filename),

Проверява дали даден файл съществува._Връща **TRUE** ако файлът или директорията подадени с параметър *filename* съществуват. Ако не - връща **FALSE.**

- **Проверка на размера на файл: filesize.** Връща размера на файл в байтове или FALSE при грешка:
- **int filesize (string $filename)**

Други Функции за работа с файл

Пример filesize :

```
<?php
$filename = 'C:/xampp/htdocs/PhpProject16/proba.txt';
//$filename = 'C:\\xampp\\htdocs\\PhpProject16\\proba.txt';
if (file_exists($filename))  echo "Файлът $filename съществува";
    else echo "Файлът $filename не съществува";
?>
```

Пример filesize :

```
<?php
$fp = fopen('data.txt', 'wb');  $s="Hello boys and Guirls!";
file_put_contents('data.txt',$s);
fclose($fp);
$filename = 'data.txt';
echo $filename . ': ' . filesize($filename) . ' bytes';
?>
```

Резултат: data.txt: 22 bytes

Други Функции за работа с файл

- **Проверка, дали файла е разрешен за четене: True, ако да!**

bool is_readable (string $filename)

- **Проверка, дали файла е разрешен за запис:**

bool is_writable (string $filename)

- Връща **TRUE** ако *filename* съществува и може да се записва в него. Параметърът *filename* може да бъде и директория, като по този начин се проверява дали може да се записва в директорията.

Други Функции за работа с файл

- **Проверка, дали файла е изпълнима програма:**

bool is_executable (string $filename)

- Връща **TRUE** ако файла *filename* е изпълним или **FALSE** ако не е.

- **Изтриване на файл:**

bool unlink (string $filename)

- Изтрива файл указан с *filename* . Подобно на Unix C unlink() функцията.

Други Функции за работа с файл

- **Премахване на директория:**

bool **rmdir** (string $dirname)

- Опитва се да изтрие директория посочена чрез ***dirname*** (път до директорията). Директорията трябва да е празна и да имате необходимите права за изтриване.
- **Връщани стойности: TRUE** при успех или **FALSE** при неуспех.

Други Функции за работа с файл

- **Създаване на директория:**

bool **mkdir** (string $pathname [, int $mode]...)

- *pathname* - Път до директорията.
- *mode – режим* (по подразбиране е **0777**), което е равносилно на възможно най-неограничения достъп.
- Връща **TRUE** при успех или **FALSE** при неуспех.
- Има и допълнителни параметри, за подробности http://php.saparev.com/ref.filesystem.html.