Масивите в PHP

Виолета Божикова

Особености на масивите в PHP

- Масивите в PHP са асоциативни, те са съвкупност от подредени асоциации (двойки ключ-стойност), за разлика от масивите в другите езици, където са индексирани.
- Масив може да бъде създаден посредством езиковата конструкция array(). Тя приема определено количество двойки *ключ => стойност* , разделени със запетаи:

array([ключ=>] стойност [[,ключ=>] стойност ...);

- ключът може да бъде цяло число или низ
- стойността може да бъде всеки PHP тип

Създаване на масив в PHP

- **Пример 1.**

```
<?php
$a=array(3=>"Peter", 'b'=>"John","Horry");
print_r($a); //съвпада с print_r($a,False);
?>
```

- Резултатът е:

```
Array ( [3] => Peter [b] => John [4] => Horry )
```

- Функция **print_r()** разпечатва информацията за една променлива във вид, четим за потребителя.

Общ вид на **print_r**:

mixed **print_r**(mixed $expression
[, bool $return = false]).

Създаване на масив в PHP

- **print_r** с директива **<pre>**, тоест при следната реализация на пример1, изходът ще е в следния вид:

```
<pre>
<?php
$a=array(3=>"Peter", 'b'=>"Johm","Horry");
print_r($a);
?>
</pre>
```

Изходът е:

```
Array
(
    [3] => Peter
    [b] => Johm
    [4] => Horry
)
```

Създаване на масив в PHP

- Указването на TRUE за втори параметър в print_r дава възможност за присвояване на резултата от функцията print_r на променлива, в случая на променлива $result.

```
<?php
$a=array(3=>"Peter", 'b'=>"Johm","Horry");
$result=print_r($a,TRUE);
//$results съдържа изходът от функция print_r
echo $result;
//Array ( [3] => Peter [b] => Johm [4] => Horry )
?>
```

- **Пример 2.**

```
<?php
$arr = array("foo" => "bar", 12 => true, "13" => "Maria", "012"=>"Dean");
//Array ( [foo] => bar [12] => 1 [13] => Maria [012] => Dean )
echo $arr["foo"];       // bar
echo "<br>".$arr[12];    // 1
echo "<br>".$arr["12"];    // 1
echo "<br>".$arr["012"];    // Dean
echo "<br>".$arr["13"];  //Maria
echo "<br>".$arr[13];    // Maria
?>
```

Внимание: **Винаги трябва да заграждате низовите индекси на масиви с апострофи или кавички. Ключът може да бъде или *цяло число*, или *низ*. Ако ключът е представен като обикновено цяло число, той ще бъде интерпретиран като такова (т.е. ключът е *"12"* - ще се интерпретира като *12*, докато *"012"* - като *"012"*). Не можете да използвате масиви или обекти като ключове. Ако се опитате да го направите ще предизвикате предупреждение: *Illegal offset type* (невалиден тип отместване).**

- Плаващите числа в ключовете се съкращават до цели и ако имаме:

```
$arr = array("foo" => "bar", 12 => true, "13" =>
   "Maria", "012"=>"Dean", 13.3=>"Toni");
```

- То изходът е:

```
echo "<br>".$arr["13"];  // Toni
echo "<br>".$arr[13];    // Maria
```

- В PHP не съществуват различни типове - за индексирани и за асоциативни масиви; има само един тип масив, който може да съдържа едновременно и целочислени и низови ключове.

- Стойността както казахме може да бъде от всякакъв тип, в примера по-долу – масив.

Пример 3.1

```
<?php
$arr = array("Article" => array(1 => "Kiwi", 5 => "Apple", 3 =>
    "Orange"), "Price"=>array(1=>2.35, 5=>1.35, 3=>1.70));
echo $arr["Article"][1]." - ". $arr["Price"][1];
echo "<br>".$arr["Article"][5]." - ". $arr["Price"][5];
echo "<br>".$arr["Article"][3]." - ". $arr["Price"][3];
echo "<br>".$arr[2];
?>
```

<u>изходът е:</u>

Kiwi - 2.35

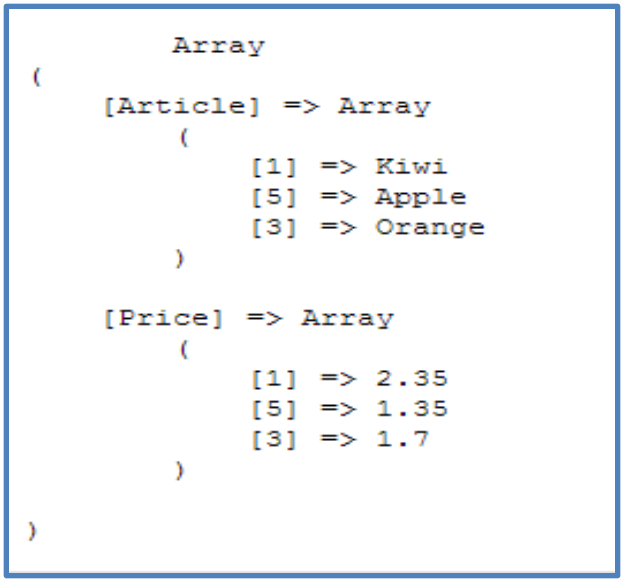
Apple - 1.35

Orange - 1.7

- Разпечатване с print_f.

Пример 3.2

```
<pre>
<?php
$arr = array("Article" => array(1 => "Kiwi", 5 =>
   "Apple", 3 => "Orange"), "Price"=>array(1=>2.35,
   5=>1.35, 3=>1.70));
print_r($arr);
?>
</pre>
```



```
Array
(
    [Article] => Array
        (
            [1] => Kiwi
            [5] => Apple
            [3] => Orange
        )

    [Price] => Array
        (
            [1] => 2.35
            [5] => 1.35
            [3] => 1.7
        )

)
```

- Ако при обявяване на масив не укажете изрично ключ за дадена стойност, то ще се види коя е най-голямата стойност от целочислените индекси до момента, например **s** и ключът на новата стойност ще бъде **s+1**.
- Ако укажете ключ, който вече има присвоена стойност, то тази стойност ще бъде презаписана.

Пример 4.

```
<?php
// Този масив е същият като ...
$arr=array(1 => 10, 20, 30, 3 => 12);
 // ...този:
//array(1 => 10, 2 => 20, 3 => 12);
//всъщност: 3 => 30 се припокрива с 3 => 12
echo "<br>".$arr[1];    // 10
echo "<br>".$arr[2];    // 20
echo "<br>".$arr[3];    // 12
?>
```

- **Предупреждение:** От PHP 4.3.0 насам начинът за генериране на индекси се промени: Сега ако добавяте към масив, в който текущият максимален ключ е отрицателен, следващият създаден ключ ще бъде нула (*0*). Докато преди PHP 4.3.0: новият индекс щеше да бъде установен в най-големия съществуващ ключ + 1, както е случая с положителните индекси.

Пример 5.

```
<?php
$arr=array(-5 => 10, 20, 30, 3 => 12);
echo "<br>".$arr[-5];    // 10
echo "<br>".$arr[0];    // 20
echo "<br>".$arr[1];    // 30
echo "<br>".$arr[3];    // 12
?>
```

- Използването на **TRUE** като ключ ще се изчисли като целочислено *1*. Употребата на **FALSE** като ключ ще се изчисли като *0 (нула)*.
- Използването на *NULL* като ключ ще се счита като празен низ "". Употребата на празен низ като ключ ще създаде (или презапише) ключ с празен низ и съответната му стойност (това не е същото като използването на празни квадратни скоби, тоест не е []):
- **Пример 6.1**

```
<?php
$arr=array(null => 10, 20, 30, 3 => 12);
echo "<br>".$arr[null];    // 10
//аналогично на echo "<br>".$arr[""];   // 10
echo "<br>".$arr[0];    // 20
echo "<br>".$arr[1];    // 30
echo "<br>".$arr[3]; //12
?>
```

```
Array
(
    [] => 10
    [0] => 20
    [1] => 30
    [3] => 12
)
```

- **Пример 6.2**

```
<?php
$arr=array(true => 10, 20, 30, 3 => 12);
echo "<br>".$arr[true];    // 10
echo "<br>".$arr[0];    //
echo "<br>".$arr[1];    // 10
echo "<br>".$arr[3]; //12
?>
```

```
Array
(
    [1] => 10
    [2] => 20
    [3] => 12
)
```

- **Пример 6.2**

```
<?php
$arr=array(false => 10, 20, 30, 3 => 12);
echo "<br>".$arr[false];    // 10
echo "<br>".$arr[0];    // 10
echo "<br>".$arr[1];    // 20  или: echo "<br>".$arr[true];
echo "<br>".$arr[2];  //30
echo "<br>".$arr[3]; //12
?>
```

```
Array
(
    [0] => 10
    [1] => 20
    [2] => 30
    [3] => 12
)
```

Създаване/променяне на масив

- Можете също да променяте съществуващ масив чрез изрично установяване на стойностите в него. Това се осъществява чрез присвояване на стойностите в масива, като ключовете се указват в квадратни скоби, пример: **$arr[0]=14;**
- Можете също да пропуснете ключа като добавите празна двойка квадратни скоби (**[]**) след името на променливата: **$arr[]=100;**
- **Пример 7.**

```
Array
(
    [0] => 10
    [1] => 20
    [2] => 30
    [3] => 12
)
```

```
<?php
$arr=array(false => 10, 20, 30, 3 => 12);
$arr[0]=14;  //Променя се съдържането на елемент с индекс 0.
$arr[]=100;  //Генерира се елемент с индекс 4, тоест $arr[4] = 100;
    echo "<br>".$arr[false];    // 14
echo "<br>".$arr[0];    // 14
echo "<br>".$arr[1];    // 20  или: echo "<br>".$arr[true];
echo "<br>".$arr[2];  //30
echo "<br>".$arr[3]; //12
echo "<br>".$arr[4]; //100
?>
```

```
Array
(
    [0] => 14
    [1] => 20
    [2] => 30
    [3] => 12
    [4] => 100
)
```

Създаване/променяне на масив

- Ако масивът *$arr* (Пример 7) все още не съществува, той ще бъде създаден. Така че това също е и алтернативен начин за указване на масив. За да промените дадена стойност, просто присвоете нова стойност на елемента, указан чрез ключа му. Ако искате да премахнете някоя двойка ключ/стойност, трябва да я унищожите посредством unset().
- **Пример 8.**

```
<?php
$arr[0]=14;
$arr[]=100;
echo "<br>".$arr[false];    // 14
echo "<br>".$arr[0];    // 14
echo "<br>".$arr[1];    // 100
unset($arr[0]); // Това премахва елемент от масива
unset($arr);    // Това изтрива целия масив
?>
```

Създаване/променяне на масив

- **Забележка**: Както беше споменато по-горе, ако предоставите квадратните скоби без указан ключ, тогава ще бъде взет максималния съществуващ целочислен индекс и новият ключ ще бъде тази стойност + 1 .
- Ако все още няма целочислени индекси, ключът ще бъде *0* (нула).
- Ако укажете ключ, който вече има присвоена стойност, то тази стойност ще бъде презаписана.
- Забележете, че максималната целочислена стойност, използвана за целта, *не е задължително да съществува в масива в момента*. Тя просто трябва да е съществувала някога в масива от последния път, когато масивът е бил повторно индексиран. Следният пример пояснява казаното:

Създаване/ променяне на масив

Пример 9.

```
<?php
// Създаване на обикновен масив.
$arr = array(1, 2, 3, 4, 5);
echo "1.";
print_r($arr);
echo "<br>";
// Сега изтриваме всеки елемент, но
оставяме самия масив неизтрит:
foreach ($arr as $i => $value) {
    unset($arr[$i]);
}
echo "2.";
print_r($arr);
echo "<br>";
// Добавяне на елемент (забележете, че
новият ключ е 5, а не 0, както
// вероятно бихте очаквали).
$arr[] = 6;
echo "3.";
print_r($arr);
echo "<br>";
// Повторно индексиране:
$arr = array_values($arr); //извършва се
пре-индексиране!!!
$arr[] = 7;
echo "4.";
print_r($arr);
?>
```

Примерът по-горе ще изведе:

1. Array ([0] => 1 [1] => 2 [2] => 3 [3] => 4 [4] => 5)
2. Array ( )
3. Array ([5] => 6)
4. Array ([0] => 6 [1] => 7)

Полезни функции за работа с масиви

- Има голямо количество полезни функции за работа с масиви.Ние ще се ограничим до няколко.
- **Забележка**: Функцията unset() позволява унищожаването на ключове от масив, при което масивът НЯМА да бъде повторно индексиран (както се вижда от пример 9). Ако използвате единствено "обичайните целочислени индекси" (започващи от нула, увеличаващи се с едно), можете да постигнете ефекта на повторно индексиране като използвате функция array_values().
- Синтаксис: **array array_values (array $input)** – връща всички стойности на един масив и го пре-индексира с обичайните целочислени индекси.

Полезни функции за работа с масиви

- **Пример 10.**

```
<PRE>
<?php
$a = array(1 => 'one', 2 => 'two', 3 => 'three');
print_r($a);
unset($a[2]);
/* ще доведе до масив, който би бил дефиниран като
   $a = array(1 => 'one', 3 => 'three');
   Тоест – няма пре-индексиране   */
print_r($a);
$a = array_values($a);   // пре-индексиране!!!
print_r($a);
// Сега $a е array(0 => 'one', 1 =>'three')
?>
</PRE>
```

```
Array
(
    [1] => one
    [2] => two
    [3] => three
)
Array
(
    [1] => one
    [3] => three
)
Array
(
    [0] => one
    [1] => three
)
```

Полезни функции за работа с масиви

- **Пример 11: Извличане на стойностите на масив - array_values($array)**

```
<PRE>
<?php
$array = array('man' => "m", 'woman' => "w",
   "child"=>'c');
print_r(array_values($array));
?>
</PRE>
```

Резултат:

```
Array
(
    [0] => m
    [1] => w
    [2] => c
)
```

Полезни функции за работа с масиви

- **Пример 12.**

```
<PRE>
<?php
$ar = array('man' => "m", "woman"=>"w", "child"=>'c');
echo "<br>The Initial array is:<br>";
print_r($ar);
if (array_key_exists('man', $ar)) {
   echo "Index 'man' is in the array! ";}
if (in_array("m", $ar))
   echo "Value 'm' is  in the array!";
$br=count($ar);
echo "<br>Count elements: $br<br>"; // output 3
echo "<br>The array_values function returns:<br>";
print_r(array_values($ar));
echo "<br>But the Initial array arr is the same:<br>";
print_r($ar);
echo "<br>The array_keys function returns:<br>";
$c=array_keys($ar); //Формира се масив $c - от ключовете на $ar
print_r($c);
$key = array_search('w', $ar);
//Търсим ключа, по зададена стойност в масива
print_r($key); //$key =woman
?>
</PRE>
```

Резултат:

```
The Initial array is:
Array
(
    [man] => m
    [woman] => w
    [child] => c
)
Index 'man' is in the array! Value 'm' is  in the
array!
Count elements: 3

The array_values function returns:
Array
(
    [0] => m
    [1] => w
    [2] => c
)

But the Initial array arr is the same:
Array
(
    [man] => m
    [woman] => w
    [child] => c
)

The array_keys function returns:
Array
(
    [0] => man
    [1] => woman
    [2] => child
)
woman
```

Полезни функции за работа с масиви

- Функция **array_keys** — връща всички ключове или подмножество от ключове на масива.
- Общ синтаксис:
  array array_keys (array $input [, mixed $search_value [, bool $strict = false]]),
- където: **input** е масива, чиито ключове ще извличаме; **search_value** е незадължителен елемент: ако е зададен се извличат ключовете само на елементите, които имат тази стойност; **bool $strict** (по подразбиране **false**) е незадължителен елемент: ако е зададен (**true**) сравнението е за еквивалентност (===), иначе за равенство (==).
- Пример:

```
$c=array_keys($ar, "w");
```

извлича ключовете само на елементи със стойност "w". Резултатът ще е [0] => woman.

Полезни функции за работа с масиви

- Функция **array_key_exists()** — проверява дали даден ключ или индекс съществува (True/False) в масива.
- Общ синтаксис:
  bool array_key_exists (mixed $key , array $search),
- където **$key** е търсения ключ, а **$search** – масива в който търсим указания ключ.
- В пример 12 по горе:

```
if (array_key_exists('man', $ar))
{    echo "Index 'man' is in the array! ";}
```

Функция **in_array()** — проверява дали дадена стойност е в масива.

В пример 12 по горе:

```
if (in_array("m", $ar))
echo "Value 'm' is  in the array!";
```

Полезни функции за работа с масиви

- Функция: **array_search()** — претърсва масива за дадена стойност (value) и връща съответния ключ, на първата намерена стойност (иначе – FALSE, 0, "", или Null).
- Ще демонстрираме функцията със следния пример:

```
$key = array_search('w', $ar);
//Търсим ключа, по зададена стойност в масива
print_r($key);          //$key =woman
```

- Функция **count()** – преброява елементите на масива или на обекта:

```
int count ( mixed $var [, int $mode = COUNT_NORMAL ] ),
```

- където **$var** е масива или обекта, а **$mode** е параметър за рекурсивност, по подразбиране - **COUNT_NORMAL** .
- Ако **$mode =**COUNT_RECURSIVE и масивът е многомерен, то функцията ще преброи всички негови елементи.
- В пример 12 по-горе:

```
$br=count($ar); echo "<br>Count elements: $br<br>"; // 3
```

Полезни функции за работа с масиви

- Функция
  array array_fill (int *$start_index* , int *$num* , mixed *$value*),
- където **$start_index** е целочислен индекс на първия елемент от масива, **int $num** е броя на елементите на масива, **$value** е стойността, която получават всички елементи на масива.
- Дава възможност да създадем масив, в който елементите имат последователни целочислени индекси и една и съща зададена стойност.
- Пример:

```
$a = array_fill(5, 3, 'banana');
print_r($a);
```

- Изход: **Array([5] => banana [6] => banana [7] => banana)**

Полезни функции за работа с масиви

- Функция sort: Сортира един масив във възходящ ред. Общ вид:

bool sort (array &$array [, int $sort_flags = SORT_REGULAR]).

- Тук първия параметър **$array** е масивът, който ще се сортира, а вторият параметър **$sort_flags** определя начина на сортиране:
- SORT_REGULAR – нормално, сравнява стойностите на елементите в съответствие с техния тип.
- SORT_NUMERIC – сравнява стойностите като числа.
- SORT_STRING – сравнява стойностите като символни низове.
- SORT_LOCALE_STRING – сравнява стойностите в съответствие с локалните настройки - current locale. Добавено в PHP 4.4.0 и 5.0.2. Настройките могат да се сменят чрез setlocale().

Полезни функции за работа с масиви

Пример:

```
<?php
$fruits = array("lemon", "orange", "banana", "apple");
echo "<br>Before Sorting:<br>";
foreach ($fruits as $key => $val) {
    echo "fruits[" . $key . "] = " . $val . "<br>";
}
sort($fruits);
echo "<br>After Sorting:<br>";
foreach ($fruits as $k => $v) {
    echo "fruits[" . $k . "] = " . $v . "<br>";
}
?>
```

Резултат:

```
Before Sorting:
fruits[0] = lemon
fruits[1] = orange
fruits[2] = banana
fruits[3] = apple

After Sorting:
fruits[0] = apple
fruits[1] = banana
fruits[2] = lemon
fruits[3] = orange
```

Полезни функции за работа с масиви

Да направим малка промяна в предния пример:

```
<?php
$fruits = array(1=>"lemon", 2=>"orange", 3=>"banana",123=>
    "apple");
echo "<br>Before Sorting:<br>";
foreach ($fruits as $key => $val) {
   echo "fruits[" . $key . "] = " . $val . "<br>";
}
sort($fruits);
echo "<br>After Sorting:<br>";
foreach ($fruits as $k => $v) {
   echo "fruits[" . $k . "] = " . $v . "<br>";
}
?>
```

```
Before Sorting:
fruits[1] = lemon
fruits[2] = orange
fruits[3] = banana
fruits[123] = apple

After Sorting:
fruits[0] = apple
fruits[1] = banana
fruits[2] = lemon
fruits[3] = orange
```

Вижда се, че след сортиране със sort()

сме загубили оригиналните индекси!!!

Използвайте asort() за да се справите с този проблем!

Полезни функции за работа с масиви

- Забележка: Операторът за цикъл foreach е създаден специално за масиви. Той предоставя лесен начин за обхождане на масив. Форми:

foreach (array_expression as $val) { блок инструкции } и

foreach (array_expression as $key => $val) { блок инструкции }

- При изпълнение на цикъла, указателят на масива се установява на първия елемент. При всяко изпълнение на блока, променливата **$val** получава стойността на поредния елемент на масива, а променливата **$key** - на ключа:
- Ако в предния пример може да напишем:

```
$fruits = array("lemon", "orange", "banana", "apple");
echo "<br>Before Sorting:<br>";
foreach ($fruits as $fruct) {
    echo "Do you like $fruct? <br>";...
```

Дава като резултат:

```
Before Sorting:
Do you like lemon?
Do you like orange?
Do you like banana?
Do you like apple?
```

Полезни функции за работа с масиви

- Ако укажем **$sort_flags = SORT_NUMERIC** ще забележим промяната на резултата:

```
sort($fruits, SORT_NUMERIC);
echo "<br>After Sorting:<br>";
foreach ($fruits as $k => $v)
 echo "fruits[" . $k . "] = " . $v . "<br>";
```

- Дава като резултат:

After Sorting:
fruits[0] = apple
fruits[1] = banana
fruits[2] = orange
fruits[3] = lemon

Полезни функции за работа с масиви

Сортиране на елементите на масив с функция:

bool array multisort($масив [,option1[,option2],…).

- Това е една много по-мощна функция от sort(). Основен недостатък на sort() е, че може да сортира само във възходящ ред.

Функция multisort() дава възможност за сортиране на няколко масива едновременно, като за всеки масив се задават 2 параметъра:

- за тип на сравнение (SORT_STRING – като низ “10”<”4”, SORT_NUMERIC – като числа 10>4) и
- за посока на сортиране (SORT_ASC – в нарастващ ред (по подразбиране), SORT_DESC – като горното, но в намаляващ ред).

Полезни функции за работа с масиви

- **Вариант 1:**

```
<PRE>
<?php
$ar1 = array(10, 100, 100, 0);
$ar2 = array(1, 3, 2, 4);
array_multisort($ar1, $ar2);
echo "<br>Print the array ar1:<br>";
print_r($ar1);
echo "<br>Print the array ar2:<br>";
print_r($ar2);
?>
</PRE>
```

Резултат от изпълнение:

```
Print the array ar1:
Array
(
    [0] => 0
    [1] => 10
    [2] => 100
    [3] => 100
)

Print the array ar2:
Array
(
    [0] => 4
    [1] => 1
    [2] => 2
    [3] => 3
)
```

Полезни функции за работа с масиви

- **Вариант 1:**

```
<PRE>
<?php
$ar1 = array(10, 100, 100, 0);
$ar2 = array(1, 3, 2, 4);
array_multisort($ar1, SORT_DESC, $ar2);
echo "<br>Print the array ar1:<br>";
print_r($ar1);
echo "<br>Print the array ar2:<br>";
print_r($ar2);
?>
</PRE>
```

```
Print the array ar1:
Array
(
    [0] => 100
    [1] => 100
    [2] => 10
    [3] => 0
)

Print the array ar2:
Array
(
    [0] => 2
    [1] => 3
    [2] => 1
    [3] => 4
)
```

Полезни функции за работа с масиви

- **Вариант 3:** вариант 1 с var_dump

```
<PRE>
<?php
$ar1 = array(10, 100, 100, 0);
$ar2 = array(1, 3, 2, 4);
array_multisort($ar1, $ar2);
echo "<br>Print the array ar1 with var_dump :<br>";
var_dump($ar1);
echo "<br>Print the array ar2 with var_dump:<br>";
var_dump($ar2);
?>
</PRE>
```

Резултат от изпълнение:

```
Print the array ar1 with var_dump :
array(4) {
  [0]=>
  int(0)
  [1]=>
  int(10)
  [2]=>
  int(100)
  [3]=>
  int(100)
}

Print the array ar2 with var_dump:
array(4) {
  [0]=>
  int(4)
  [1]=>
  int(1)
  [2]=>
  int(2)
  [3]=>
  int(3)
}
```

Полезни функции за работа с масиви

- Във вариант 3 е използвана функция:

void var_dump (mixed $expression [, mixed $...]),

която разпечатва структурирана информация за един или повече изрази, включваща техния тип и стойност.

Масивите и обектите се разгъват рекурсивно. От резултата се вижда и разликата с функция print_r().

- В **array_multisort** бихме могли да зададем и допълнителни параметри, например:

array_multisort($ar1,SORT_NUMERIC, SORT_DESC,$ar2);

print_r($ar1) …… var_dump($ar1)

Резултат от изпълнение:

```
Print the array ar1:
Array
(
    [0] => 0
    [1] => 10
    [2] => 100
    [3] => 100
)

Print the array ar2:
Array
(
    [0] => 4
    [1] => 1
    [2] => 2
    [3] => 3
)
```

Резултат от изпълнение:

```
Print the array ar1 with var_dump :
array(4) {
  [0]=>
  int(0)
  [1]=>
  int(10)
  [2]=>
  int(100)
  [3]=>
  int(100)
}

Print the array ar2 with var_dump:
array(4) {
  [0]=>
  int(4)
  [1]=>
  int(1)
  [2]=>
  int(2)
  [3]=>
  int(3)
}
```

- **!!!Присвояването на масиви в PHP винаги става по стойност**. За да копие на масив по референция, трябва да използвате референтния оператор &.

```
<PRE>
<?php
$arr1 = array(2, 3);   echo "<br>1. Print the array ar1:<br>";
print_r($arr1); // [0] => 2, [1] => 3
$arr2 = $arr1;  //по стойност
echo "<br>2. Print the array ar2:<br>";
print_r($arr2);
$arr2[] = 4; // $arr2 е променен, добавен е нов елемент, но $arr1 е все още array(2, 3)
echo "<br>3. Print the array ar2 again:<br>";
print_r($arr2);   //2,3,4
$arr3 = &$arr1;  //по референция
echo "<br>4. Print the array ar3=&ar1:<br>";
print_r($arr3); //2,3
$arr3[] = 4; // сега $arr1 и $arr3 са едно и също
echo "<br>5. Print the array ar3:<br>";
print_r($arr3); //2,3,4
echo "<br>6. Print the array ar1:<br>";
print_r($arr1); //2,3,4
?>
</PRE>
```

Итерация на елементите на масив: функции each, list и reset

- Функция **array each(array &$array)** – връща текущата двойка **ключ-стойност** в един 4-елементен масив с ключове **1, value, 0, key**.
- Функцията премества вътрешния указател на масива на следващия елемент. Така, ако преди изпълнението на each вътрешния указател е бил след последния елемент, функцията връща false.

```
<PRE>
<?php
$arr1 = array('name'=>"Ivan", "age"=>23);
 echo "<br>1. Print the array ar1:<br>";
print_r($arr1); // [name]=>Ivan [age]=>23
$arr2 = each($arr1);
echo "<br>2. Print the array ar2:<br>";
print_r($arr2);
$arr2 = each($arr1);
echo "<br>3. Print the array ar2:<br>";
print_r($arr2);
?>
</PRE>
```

Резултат:

```
1. Print the array ar1:
Array
(
    [name] => Ivan
    [age] => 23
)

2. Print the array ar2:
Array
(
    [1] => Ivan
    [value] => Ivan
    [0] => name
    [key] => name
)

3. Print the array ar2:
Array
(
    [1] => 23
    [value] => 23
    [0] => age
    [key] => age
)
```

Още полезни функции за масиви...

- В предния пример се вижда се, че четири-елементния масив, който връща функцията each съдържа 2 за извлечената стойност и 2 елемента за извлечения ключ .
- Функция **array list** (mixed *$varname* [, mixed *$...*]) се използва за извличане на стойности на елементи на масив и присвояването им на променливи, с една операция.

```
<?php
$fruits = array('oranges', 'apples', 'kiwi');
// Listing all the variables
list($a, $b, $c) = $fruits;
echo "I like $a,$b and $c.<br>";
//I like oranges,apples and kiwi.
list($a,,$c) = $fruits;
//извличат се стойностите на 2 елемента, първия и третия
echo "My favorite fruits are $a and $c.<br>";
//I like oranges,apples and kiwi.
//My favorite fruits are oranges and kiwi.
?>
```

Още полезни функции за масиви...

- Функция mixed **reset** (array *&$array*). Функцията reset() установява вътрешния указател на масива на първия елемент.

Пример:

```
<PRE>
<?php
$arr = array('one', 'two', 'three', 'four');
echo "<br>1. Print the array arr:<br>";
print_r($arr); //
echo "<br>2. Print the current element of arr:<br>";
// by default, the pointer is on the first element
echo current($arr) . "<br /><br>"; // "one"

// skip two elements
next($arr);
next($arr);
echo "<br>3. Two elements were skipped:<br>";
echo current($arr) . "<br /><br>"; // "three"

// reset pointer, start again on step one
reset($arr);
echo "<br>4. After reset():<br>";
echo current($arr) . "<br /><br>"; // "one"
?>
</PRE>
```

```
1. Print the array arr:
Array
(
    [0] => one
    [1] => two
    [2] => three
    [3] => four
)
2. Print the current element of arr:
one
3. Two elements were skipped:
three
4. After reset():
one
```

Още полезни функции за масиви...

- Други функции:

current() – Връща текущия елемент в масив

end() – Установява вътрешния указател на масив на последния елемент: end($arr); // four

next() – Премества с 1 стъпка напред указателя на масива

prev() – Връща с една стъпка указателя на масива.

Пример: prev($arr);