Лабораторно упражнение № 4

Масиви. Функции за масиви. Оператор foreach.

1. Масиви

Типа array представлява променлива, на която могат да се зададат едновременно множество стойности. Тя се ползва обикновено когато е нужно да се съхранят няколко близки по значение стойности, обединени под общо название. Масивите до голяма степен приличат на списък със записи. Записването на смислово свързани данни като масив - една променлива, предоставя гъвкавост и удобство при обработката на тези данни.

1.1. Видове масиви

В PHP се използват следните видове масиви:

- Числови (numeric) - за индекси на отделните стойности се използват числа.
- Асоциативни (associative) - за индекси на отделните стойности се използват думи.
- Многомерни (multidimensional) - масиви, които съдържат в себе си други масиви.

Създаването на масив става чрез конструкцията array(), като отделните стойности се отделят чрез запетаи.

1.1.1. Числови масиви

Съществуват два начина за задаване на числови масиви в PHP:

- ✓ **Без задаване на индекси, с директно изброяване на стойностите.**
  $променлива = array("стойност1", "стойност2", "стойност3");

Пример 1.

```
<?php
    $automobile = array("Мерцедес", "BMW", "Форд");
    echo  $automobile [0];
?>
```

Тъй като на една и съща променлива са присвоени едновременно няколко стойности, всяка от тези стойности автоматично получава пореден номер.
Индексирането винаги започва от 0, така че индексът, отговарящ на „Мерцедес" ще бъде 0, а за „Форд" - 2. Стойностите, съхранявани в един числов масив, са достъпни чрез името на масива и съответния индекс.

- ✓ **Със задаване на индекси.**

Друг начин на дефиниране на променлива от тип масив е като всяка стойност се зададе поотделно на променливата. Всяка от стойностите на променливата е достъпна чрез името на променливата, съчетано с номера на стойността. Началния номер започва от нула. При всяко присвояване на стойност на променливата може за да се укаже и идентификационния номер.

Пример 2.

```
<?php
    $automobile [0] = "Мерцедес";
    $automobile[1] = "BMW";
    $automobile[2] = "Форд";
    echo $automobile [0];
?>
```

Ако не бъдат указани изрично номера та на стойностите те ще им бъдат присвоени по подразбиране.

Пример3.

```
<?php
    $automobile[] = "Мерцедес";
    $automobile[] = "BMW";
    $automobile[] = "Форд";
    echo $automobile [0];
?>
```

1.1.2. Асоциативни масиви

При създаване на масиви понякога е по-удобно за индекси да се използват думи вместо числа. Освен чрез цифри стойностите на масивите могат да бъдат идентифицирани и чрез буквени наименования, например думи или съкращения. Всяка от думите с които са идентифицирани различните стойности от масива се превръща в "ключ" за съответната стойност.

Асоциативните масиви в PHP са съвкупност от подредени асоциации (двойки ключ-стойност). 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 всякаква
```

Съществуват два начина за задаване на асоциативни масиви в PHP:

- ✓ **С непосредствено изброяване на двойките индекс-стойност.**

Пример 4.

```
<?php
     $signature = array("FirstName" => "Иван",
     "LastName"=>"Петров");
     echo $signature ["LastName"];
     echo "<br>";
     echo "Документът е подписан от г-н
     ".$signature["LastName"].".";
?>
```

- ✓ **С отделно изброяване на индексите и съответстващите им стойности.**

Пример 5.

```
<?php
     $signature ["FirstName"] = "Иван";
     $signature ["LastName"] = "Петров";
     echo $signature ["LastName"];
     echo "<br>";
     echo "Документът е подписан от г-н
     ".$signature["LastName"].".";
?>
```

Обръщането към дадена стойност става като се изпише името на масива, следвано от ключа за стойността, който също трябва да е затворен в квадратни скоби.

Пример 6. Дефиниране на масив $a и извеждане с print_r(). Обърнете внимание, че за третия елемент е зададена само стойност. Вижте какъв е ключа (индекса) на този елемент (с 1 по-голям от последния целочислен индекс). Функция **print_r()** разпечатва информацията за една променлива във вид, удобен за четене от потребителя.

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

Резултат:

```
Array
(   [3] => Peter
    [b] => Jon
    [4] => Harry
)
```

Вижда се, че ако не укажете изрично ключ за дадена стойност, то ще се вземе най-голямата стойност от целочислените индекси и новият ключ ще бъде тази стойност + 1

Пример 7. Дефиниране на масив $arr и извеждане поелементно с езиковата конструкция echo. Обърнете внимание на резултата, който виждате.

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

Ще работи ли коректно скрипта, ако допълним масива с един елемент, например:

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

<?php
    $arr = array("foo" => "bar", 12 => true, "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
?>
```

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. "12" ще се интерпретира като 12, докато "012" - като "012"). Ако укажете ключ, който вече има присвоена стойност, то тази стойност ще бъде презаписана.

<u>Плаващите числа в ключовете се съкращават до цели.</u>

Пример:

```
    $arr = array("foo" => "bar", 12 => true, "012"=>"Dean",
    13.3=>"Toni");
    То изходът е:
```

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

В PHP не съществуват различни типове за индексирани и асоциативни масиви. Има само един тип масив, който може да съдържа едновременно целочислени и низови индекси. Стойността може да бъде от всякакъв тип.

Пример 8. Дефиниране на масив $arr, със ключове "Article" и "Price" и стойност за всеки един от ключовете – масив.

```
<?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];
?>
```

Резултат:

Kiwi - 2.35
Apple - 1.35
Orange - 1.7

Пример 9. Използването на **`TRUE`** като ключ ще се изчисли като целочислено *1*. Употребата на **`FALSE`** като ключ ще се изчисли като целочислено *0*. Използването на *NULL* като ключ ще се изчисли като празен низ. Употребата на празен низ като ключ ще създаде (или презапише) ключ с празен низ и съответната му стойност.

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

ИЛИ

```
<?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
?>
```

!!!Не можете да използвате масиви или обекти като ключове.

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.
Можете също да пропуснете ключа като добавите празна двойка квадратни скоби ("*[]*") към името на променливата.
Пример 10.

```
<?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
    $arr[4] = 100;
    echo "<br>".$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
?>
```

Ако масивът *$arr* все още не съществува, той ще бъде създаден. Това е 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 11.

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

2. Функции за масиви.

2.1.array_values() - повторно индексиране на целочислен масив. Функцията връща като резултат всички стойности на един масив и го преиндексира с целочислени индекси.

Синтаксис:

array array_values(array $input)

(Вижте раздел функции за масиви.)

Функцията **unset()** позволява унищожаването на ключове от масив, при което масивът НЯМА да бъде повторно преиндексиран (както се вижда от примера). Ако използвате единствено целочислени индекси (започващи от нула, увеличаващи се

с едно), можете да постигнете ефекта на повторно индексиране като използвате функция **array_values().**

Пример 12. Преиндексиране с array_values().

```
<PRE>
<?php
   $a = array(1 => 'one', 2 => 'two', 3 => 'three');
   print_r($a);
   unset($a[2]);
   print_r($a);
   /* ще доведе до масив, който би бил дефиниран като
      $a = array(1 => 'one', 3 => 'three');
      а НЕ като $a = array(1 => 'one', 2 =>'three');   */
   $b = array_values($a);
   print_r($b);
?>
</PRE>
```

Резултат:

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

Пример 13. Преиндексиране с array_values(), ако индексите са символни низове (с апострофи или с кавички).

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

Резултат:

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

2.2. array_keys() — връща всички ключове или подмножество от ключове на масив.

Синтаксис:

array array_keys (array $input [,mixed $search_value [,bool $strict= false]])
input - 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:

```
$c=array_keys($array, "w"); //извлича ключовете само на
елементи със стойност "w".
```

Резултат:
[0] => woman

2.3. **array_key_exists()** - проверява дали даден ключ или индекс съществува в масива. Функцията връща като резултат TRUE ако даденият ключ съществува и false ако не съществува.

Синтаксис:
bool array_key_exists (mixed $key, array $search)
$key - търсения ключ
$search – масива в който търсим указания ключ
Пример:

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

2.4. **in_array()** - проверява дали дадена стойност е в масива. Функцията връща като резултат TRUE ако дадената съществува и FALSE ако не съществува.

Пример:

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

2.5. **array_search()** - претърсва масива за дадена стойност (value) и връща съответния ключ, на първата намерена стойност (иначе – FALSE, 0, "", или Null).

Пример:

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

2.6. **count()** – преброява елементите на масива или на обекта. Функцията връща като резултат броя на елементите на даден масив или броя на свойствата на даден обект.

Синтаксис:
int count (mixed $var [, int $mode = COUNT_NORMAL])
$var - масив или обект
$mode - параметър за рекурсивност, по подразбиране - COUNT_NORMAL. Ако втория параметър е COUNT_RECURSIVE и масивът е многомерен, то функцията ще преброи всички негови елементи.
Пример:

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

Пример 14. Демонстрира функции array_keys(), array_key_exists(), in_array(), array_search(), count(),array_values()

```
<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
```

2.7. array_fill() - запълва масив с дадени стойности. Функцията връща като резултат запълнения масив.

Синтаксис:

array array_fill (int *$start_index*, int *$num*, mixed *$value*)

$start_index - целочислен индекс на първия елемент от масива

$num - броя на елементите на масива

$value - 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
)
```

2.8. sort() - сортира един масив във възходящ ред. Връща TRUE при успех или FALSE при неуспех.

Синтаксис:

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

$array - масивът, който ще се сортира.

$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().

3. Оператор foreach

В PHP4 е въведен операторът foreach. Той предоставя удобен начин за обхождане на масиви и е приложим само върху тях. При опит за прилагане на foreach върху данни от друг тип се издава съобщение за грешка. Цикъл foreach се използва, за да се приложи една и съща поредица от команди към всеки от елементите на даден масив. Този оператор предоставя лесен начин за обхождане на масив.

Този оператор има два варианта:

1. **foreach (array_expression as $val){**
   блок инструкции }

array_expression – име на масив

Цикъл foreach работи като взема елемент на масива (започвайки от първия), съхранява стойността му като променлива $val, която се използва за обработка от операторите в тялото на цикъла и след завършване на всички команди от цикъла, започва отначало, но вече със следващия елемент - докато се обработят всички елементи на масива.

Първият оператор обхожда масива, като на всяка итерация стойността на променливата параметър на цикъла получава стойността на текущия елемент и вътрешният указател на масива се увеличава с 1 т.е. на следващата итерация ще съдържа указател към следващия елемент.

Пример:

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

Резултат:

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

2. **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.

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

Резултат:

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

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)
- за посока на сортиране (SORT_ASC – в нарастващ ред, SORT_DESC –в намаляващ ред).

Пример 15. Сортиране с функцията array_multisort()

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:

```
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
)
```

Вариант 2:

```
<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:

```
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)
}
```

Бихме могли да зададем и допълнителни параметри, например:

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

!!! Трябва да се подчертае, че присвояването на масиви **винаги става по стойност**. За да копирате масив по референция, трябва да използвате референтния оператор &.

Пример 16. Копиране на масив по референция.

```
<PRE>
<?php
    $arr1 = array(2, 3);
    echo "<br>1. Print the array ar1:<br>";
    print_r($arr1); // [0] => 2, [1] => 3
    $arr2 = $arr1;
    echo "<br>2. Print the array ar2:<br>";
    print_r($arr2);
    $arr2[] = 4; // $arr2 е променен,
                 // $arr1 е все още array(2, 3)
    echo "<br>3. Print the array ar2 again:<br>";
    print_r($arr2);   //2,3,4
    $arr3 = &$arr1;
```

```
    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>
```

Резултат:

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

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

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

4. Print the array ar3=&ar1:
Array
(
    [0] => 2
    [1] => 3
)

5. Print the array ar3:
Array
(
    [0] => 2
    [1] => 3
    [2] => 4
)

6. Print the array ar1:
Array
(
    [0] => 2
    [1] => 3
    [2] => 4
)
```

4. Итерация на елементите на масив - функции each(), list() и reset()

4.1. each() – връща текущата двойка ключ-стойност в 4-елементен масив с ключове 0,1,key,value. 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.

Синтаксис:

array each(array *&$array*)

Пример 15. Пример с each()

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

Вижда се, че четири-елементния масив, който връща функцията each съдържа 2 елемента за извлечения ключ и 2 за извлечената стойност.

4.2. list() – езикова конструкция, която се използва за присвояване на стойности на променливи с една операция. Функцията не връща стойност.

Синтаксис:

void list(mixed *$varname*[, mixed *$*..])

Пример 16. Пример с list()

```
<?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>";
    //My favorite fruits are oranges and kiwi.
?>
```

Резултат:

I like oranges,apples and kiwi.

My favorite fruits are oranges and kiwi.

4.3. reset()- установява вътрешния указател на масива на първия елемент. Функцията връща като резултат стойността на първия елемент на масива или FALSE, ако масивът е празен.

Синтаксис:

mixed reset(array &*$array*)

Пример 17. Пример с reset()

```
<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
```

- current() – Връща текущия елемент в масив.
- end() – Установява вътрешния указател на масив на последния елемент.
- next() – Премества с една стъпка напред указателя на масива
- prev() – Връща с една стъпка указателя на масива.