

Лекция 5

Работа с массивами в РНР.

Операции над массивами

Формирование массива

Функция `list()`

```
$names[0]="Александр";  
$names[1]="Николай";  
$names[2]="Яков";
```

```
$alex = $names[0];  
$nick = $names[1];  
$yakov = $names[2];
```

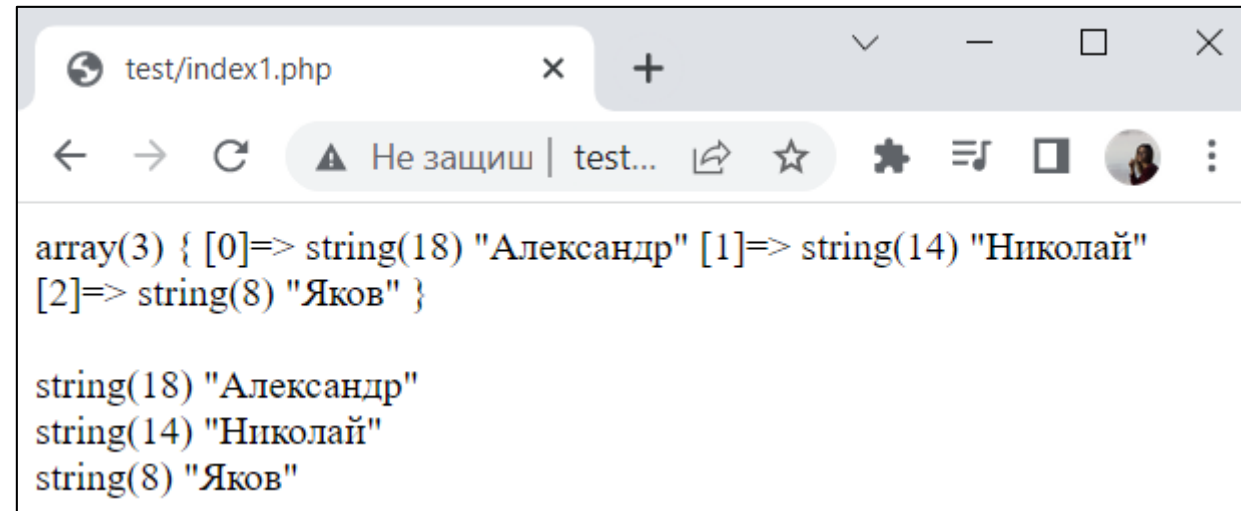


```
list ($alex, $nick, $yakov) = $names;
```

Формирование массива

```
<?php
    $names[0]="Александр";
    $names[1]="Николай";
    $names[2]="Яков";
    var_dump ($names);

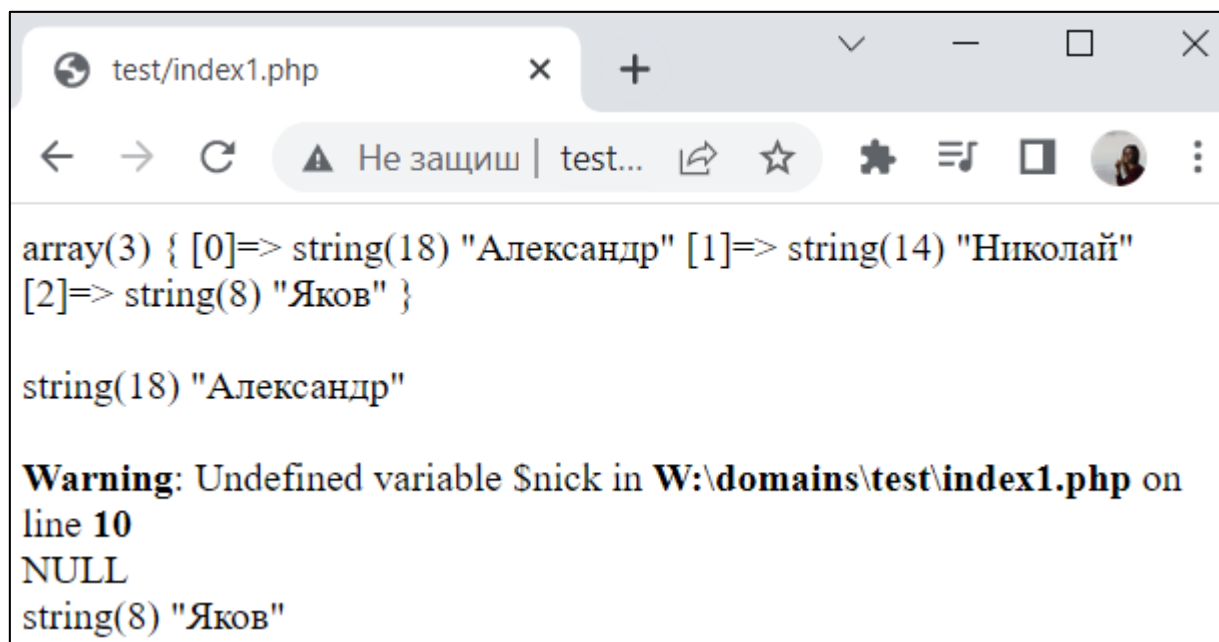
    list($alex, $nick, $yakov)=$names;
    var_dump ($alex);
    var_dump ($nick);
    var_dump ($yakov);
?>
```



Формирование массива

```
<?php
    $names[0]="Александр";
    $names[1]="Николай";
    $names[2]="Яков";
    var_dump ($names);

    list($alex, , $yakov)=$names;
    var_dump ($alex);
    var_dump ($nick);
    var_dump ($yakov);
?>
```



test/index1.php

← → ↻ ⚠ Не защищ | test... ☆ ⚙ ≡ 📺 👤 ⋮

```
array(3) { [0]=> string(18) "Александр" [1]=> string(14) "Николай"
[2]=> string(8) "Яков" }

string(18) "Александр"

Warning: Undefined variable $nick in W:\domains\test\index1.php on
line 10
NULL
string(8) "Яков"
```

Формирование массива

Функция `array()`

```
<?php
// Создает пустой массив:
$arr = array();
// Создает список с тремя элементами. Индексы начинаются с нуля:
$arr2 = array("Иванов", "Петров", "Сидоров");
// Создает ассоциативный массив с тремя элементами:
$arr3 = array("Иванов"=>"Иван", "Петров"=>"Петр", "Сидоров"=>"Сидор");
var_dump ($arr);
var_dump ($arr2);
var_dump ($arr3);
?>
```

```
test/index1.php
array(0) { }

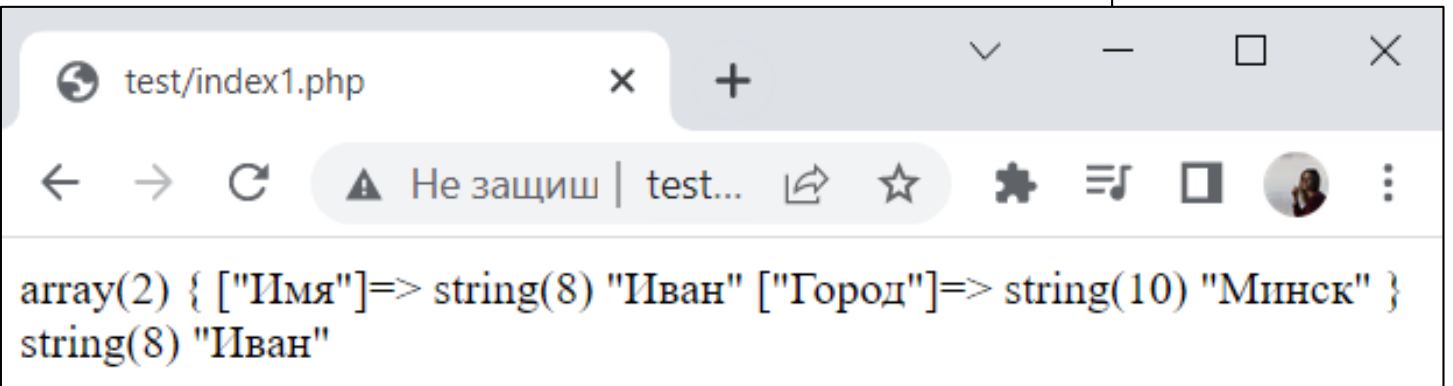
array(3) { [0]=> string(12) "Иванов" [1]=> string(12) "Петров" [2]=>
string(14) "Сидоров" }

array(3) { ["Иванов"]=> string(8) "Иван" ["Петров"]=> string(8) "Петр"
["Сидоров"]=> string(10) "Сидор" }
```

Формирование массива

Функция `array()`

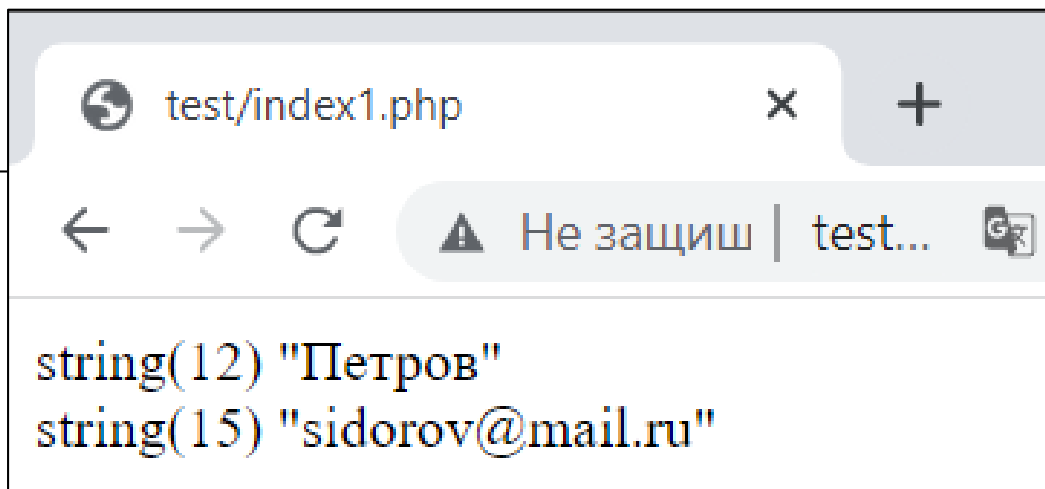
```
<?php
// Создает многомерный ассоциативный массив:
$arr = array(
    "Иванов" => array("Имя"=>"Иван", "Город"=>"Минск"),
    "Петров" => array("Имя"=>"Саша", "Город"=>"Гомель"),
    "Сидоров" => array("Имя"=>"Максим", "Город"=>"Брест"),
);
var_dump($arr["Иванов"]);
var_dump($arr["Иванов"]["Имя"]);
?>
```



Формирование массива

Функция `array()`

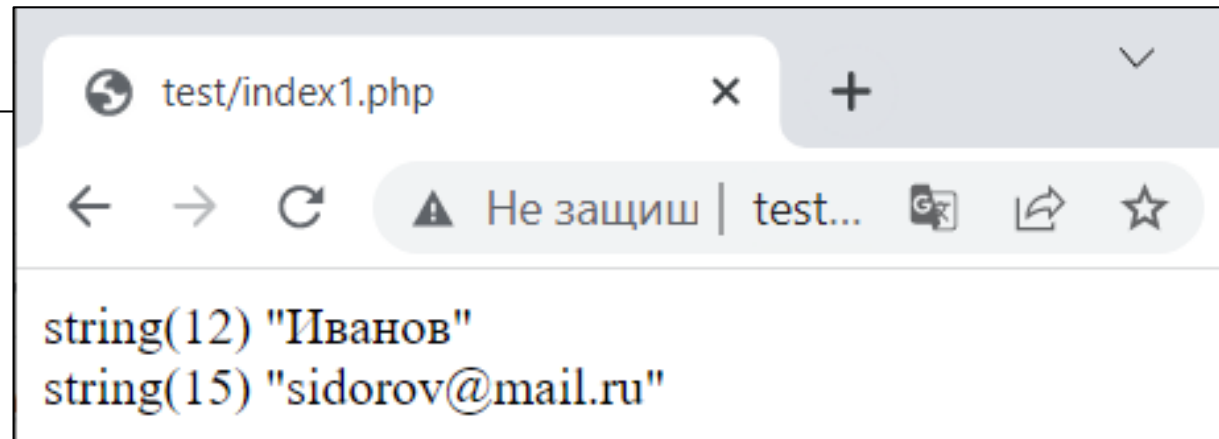
```
<?php
$arr5 = array(
    1=>array("name"=>"Иванов", "age"=>"24", "email"=>"ivanov@mail.ru"),
    2=>array("name"=>"Петров", "age"=>"34", "email"=>"petrov@mail.ru"),
    array("name"=>"Сидоров", "age"=>"47", "email"=>"sidorov@mail.ru")
);
var_dump ($arr5[2]["name"]);
echo "<br>";
var_dump ($arr5[3]["email"]);
?>
```



Формирование массива

Функция `array()`

```
<?php
$arr5 = array(
    "студент1"=>array("name"=>"Иванов", "age"=>"24", "email"=>"ivanov@mail.ru"),
    array("name"=>"Петров", "age"=>"34", "email"=>"petrov@mail.ru"),
    array("name"=>"Сидоров", "age"=>"47", "email"=>"sidorov@mail.ru")
);
var_dump ($arr5["студент1"]["name"]);
echo "<br>";
var_dump ($arr5[1]["email"]);
?>
```



Сортировка массивов

*Сортировка массива по значениям с помощью функций **asort()** и **arsort()***

Функция **asort()** сортирует массив, указанный в ее параметре, так, чтобы его значения шли в алфавитном (если это строки) или в возрастающем (для чисел) порядке.

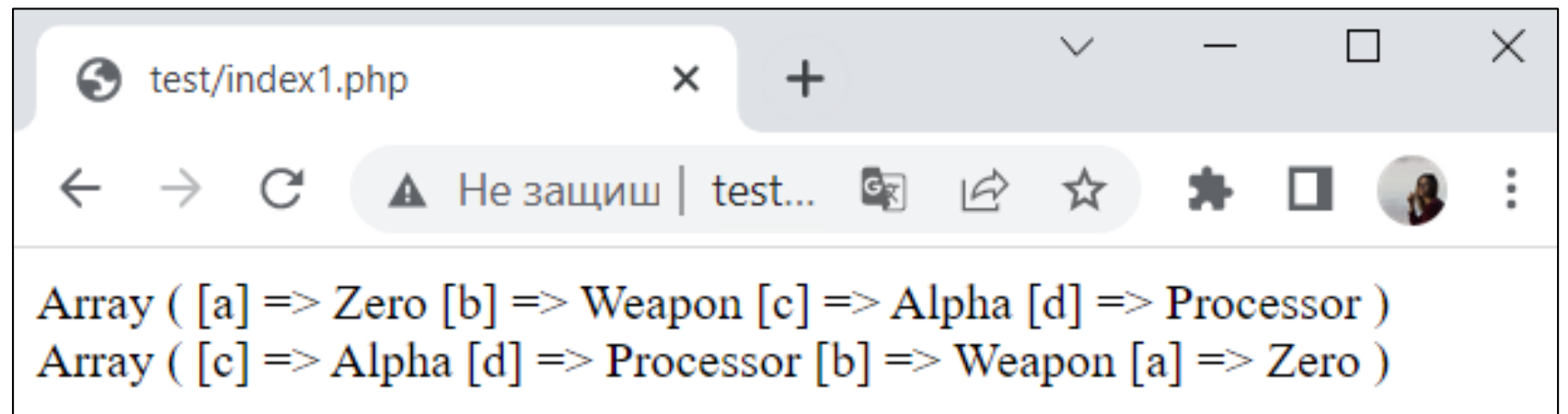
При этом сохраняются связи между ключами и соответствующими им значениями, т. е. некоторые пары ключ=>значение просто "всплывают" наверх, а некоторые — наоборот, "опускаются".

Функция **arsort()** выполняет то же самое, за одним исключением: она упорядочивает массив не по возрастанию, а по убыванию.

Сортировка массивов

*Сортировка массива по значениям с помощью функций **asort()** и **arsort()***

```
<?php
$A = array("a"=>"Zero", "b"=>"Weapon", "c"=>"Alpha", "d"=>"Processor");
print_r($A);
asort($A);
print_r($A);
?>
```



```
Array ( [a] => Zero [b] => Weapon [c] => Alpha [d] => Processor )
Array ( [c] => Alpha [d] => Processor [b] => Weapon [a] => Zero )
```

Сортировка массивов

*Сортировка массива по значениям с помощью функций **asort()** и **arsort()***

```
<?php
$A = array("a"=>"Zero", "b"=>"Weapon", "c"=>"Alpha", "d"=>"Processor");
print_r($A);
arsort($A);
print_r($A);
?>
```

```
Array ( [a] => Zero [b] => Weapon [c] => Alpha [d] => Processor )
Array ( [a] => Zero [b] => Weapon [d] => Processor [c] => Alpha )
```

Сортировка массивов

*Сортировка по ключам с помощью функций **ksort()** и **krsort()***

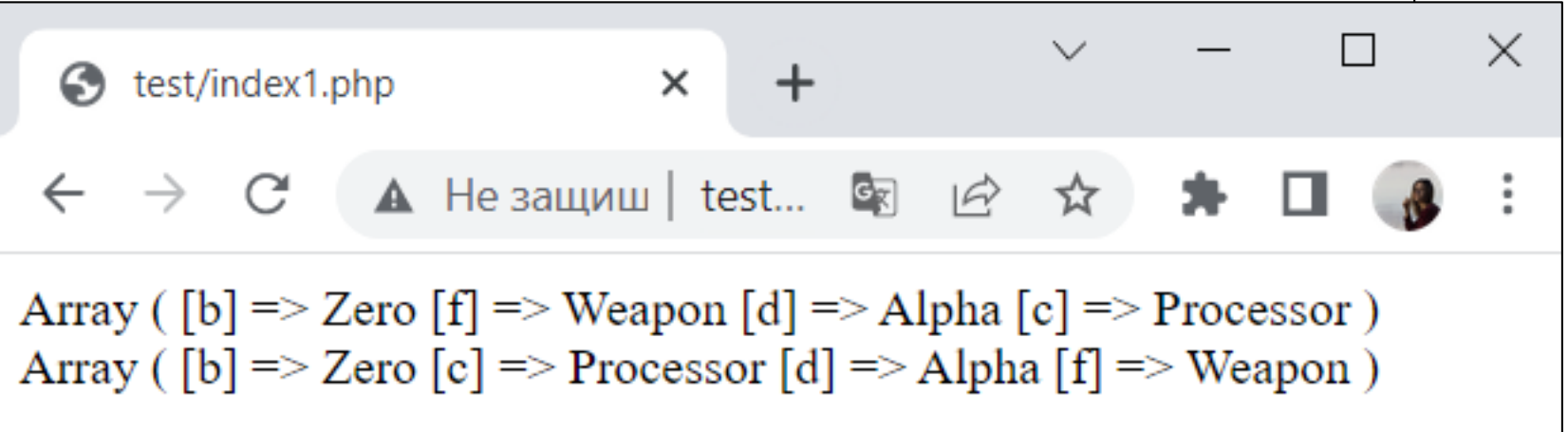
Функция **ksort()** практически идентична функции `asort()`, с тем различием, что сортировка осуществляется не по значениями, а по ключам (в порядке возрастания).

Функция для сортировки по ключам в обратном порядке называется **krsort()** и применяется точно в таком же контексте, что и `ksort()`.

Сортировка массивов

Сортировка по ключам с помощью функций `ksort()` и `krsort()`

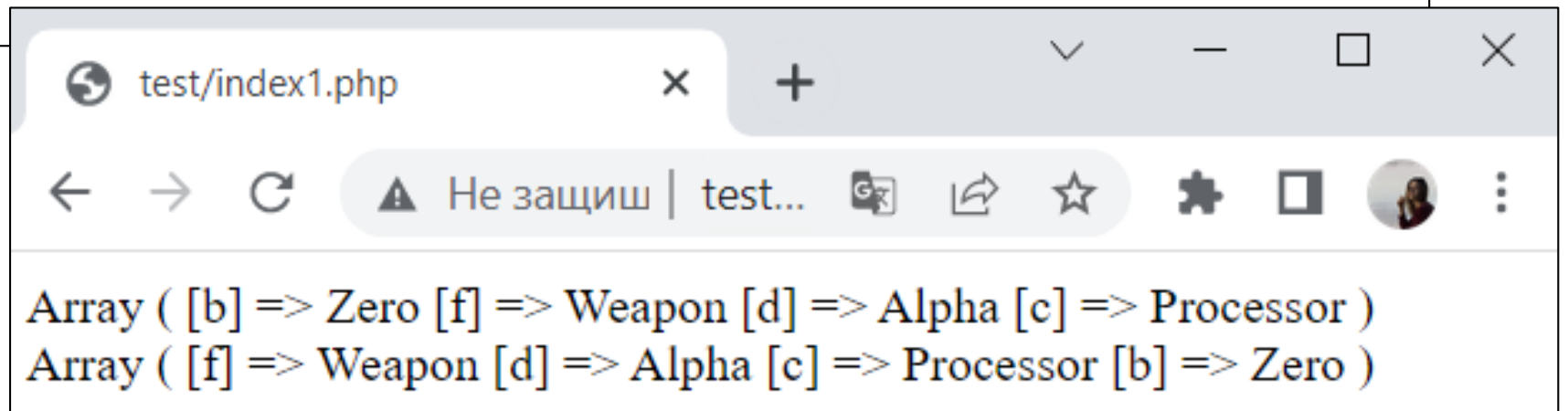
```
<?php
$A = array("b"=>"Zero", "f"=>"Weapon", "d"=>"Alpha", "c"=>"Processor");
print_r($A);
ksort($A);
print_r($A);
?>
```



Сортировка массивов

Сортировка по ключам с помощью функций `ksort()` и `krsort()`

```
<?php
$A = array("b"=>"Zero", "f"=>"Weapon", "d"=>"Alpha", "c"=>"Processor");
print_r($A);
krsort($A);
print_r($A);
?>
```



Сортировка массивов

*Сортировка по ключам при помощи функции **uksort()***

Сортирует массив, используя для сравнения его ключей функцию, предоставленную пользователем. Используйте эту функцию, если массив должен быть отсортирован по какому-нибудь необычному признаку.

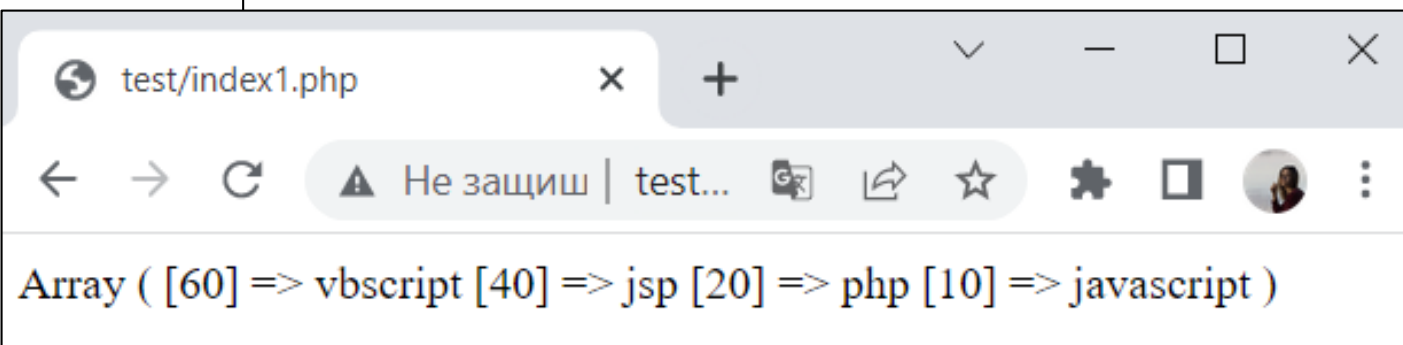
Например, пусть в **\$Files** хранится список имен файлов и подкаталогов в текущем каталоге. Возможно, мы захотим вывести этот список не только в лексикографическом порядке, но также и чтобы все каталоги предшествовали файлам. В этом случае нам стоит воспользоваться функцией **uksort()**, написав предварительно функцию сравнения с двумя параметрами, как того требует **uksort()**.

Сортировка массивов

*Сортировка по ключам при помощи функции **uksort()***

```
<?php
function my_sort($x, $y)
{
    if ($x == $y)
        return 0;
    return ($x > $y) ? -1 : 1;
}
$names = array(
    "10" => "javascript",
    "20" => "php",
    "60" => "vbscript",
    "40" => "jsp"
);

uksort($names, "my_sort");
print_r ($names);
?>
```



Сортировка массивов

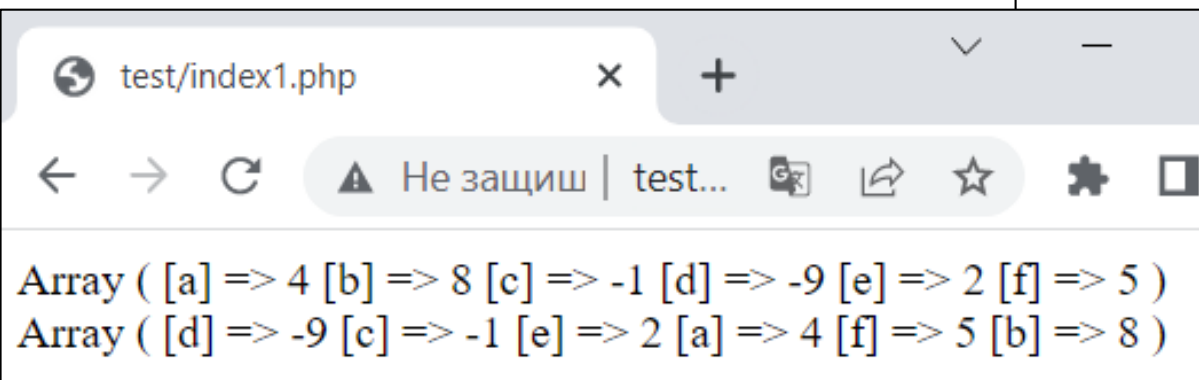
*Сортировка по значениям при помощи функции **uasort()***

Функция **uasort()** очень похожа на `uksort()`, с той разницей, что сменной (пользовательской) функции сортировки "подсовываются" не ключи, а очередные значения из массива. При этом также сохраняются связи в парах ключ=>значение.

Сортировка массивов

Сортировка по значениям при помощи функции `uasort()`

```
<?php
// Функция сравнения
function cmp($a, $b) {
    if ($a == $b) {
        return 0;
    }
    return ($a < $b) ? -1 : 1;
}
// Сортируемый массив
$array = array('a' => 4, 'b' => 8, 'c' => -1, 'd' => -9, 'e' => 2, 'f' => 5);
print_r($array);
// Сортируем и выводим получившийся массив
uasort($array, 'cmp');
print_r($array);
?>
```



Сортировка массивов

*Сортировка массивов с помощью функции **array_multisort()***

Функция `array_multisort()`. Может быть использована для сортировки сразу нескольких массивов или одного многомерного массива в соответствии с одной или несколькими размерностями.

При этом ассоциативные (string) ключи будут сохранены, но числовые ключи будут переиндексированы.

Сортировка массивов

*Сортировка массивов с помощью функции **array_multisort()***

```
array_multisort(  
    array &$array1,  
    mixed $array1_sort_order = SORT_ASC,  
    mixed $array1_sort_flags = SORT_REGULAR,  
    mixed ...$rest  
)
```

Список параметров:

array1 - сортируемый массив (array).

array1_sort_order - порядок для сортировки вышеуказанного аргумента типа array (**SORT_ASC** для сортировки по возрастанию, **SORT_DESC** для сортировки по убыванию).

Этот аргумент может меняться местами с **array1_sort_flags** или вообще быть пропущенным.

В этом случае подразумевается значение **SORT_ASC**.

array1_sort_flags - настройки сортировки для вышеуказанного аргумента array.

Сортировка массивов

*Сортировка массивов с помощью функции **array_multisort()***

Флаг способа сортировки:

SORT_REGULAR - обычное сравнение элементов (без изменения типов).

SORT_NUMERIC - сравнение элементов как чисел.

SORT_STRING - сравнение элементов как строк.

SORT_LOCALE_STRING - сравнение элементов как строк, учитывая текущую локаль.

SORT_NATURAL - сравнение элементов как строк с использованием алгоритма "natural order".

SORT_FLAG_CASE - для сортировки без учёта регистра.

Этот аргумент может меняться местами с **array1_sort_order** или вообще быть пропущенным. В этом случае подразумевается значение **SORT_REGULAR**.

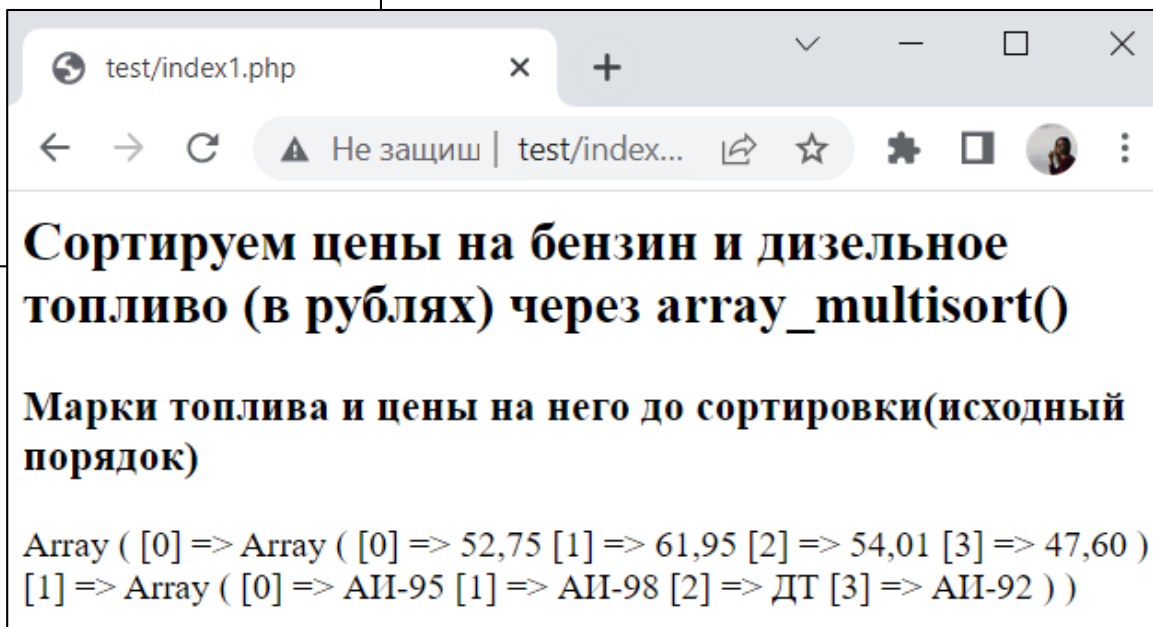
Сортировка массивов

Сортировка массивов с помощью функции `array_multisort()`

```
<?php
echo "<h2>Сортируем цены на бензин и дизельное топливо
(в рублях) через array_multisort()</h2>";

$petrol = array(
    array("52,75", "61,95", "54,01", "47,60"),
    array("АИ-95", "АИ-98", "ДТ", "АИ-92")
);

echo "<h3>Марки топлива и цены на него до
сортировки(исходный порядок)</h3>";
print_r($petrol);
?>
```



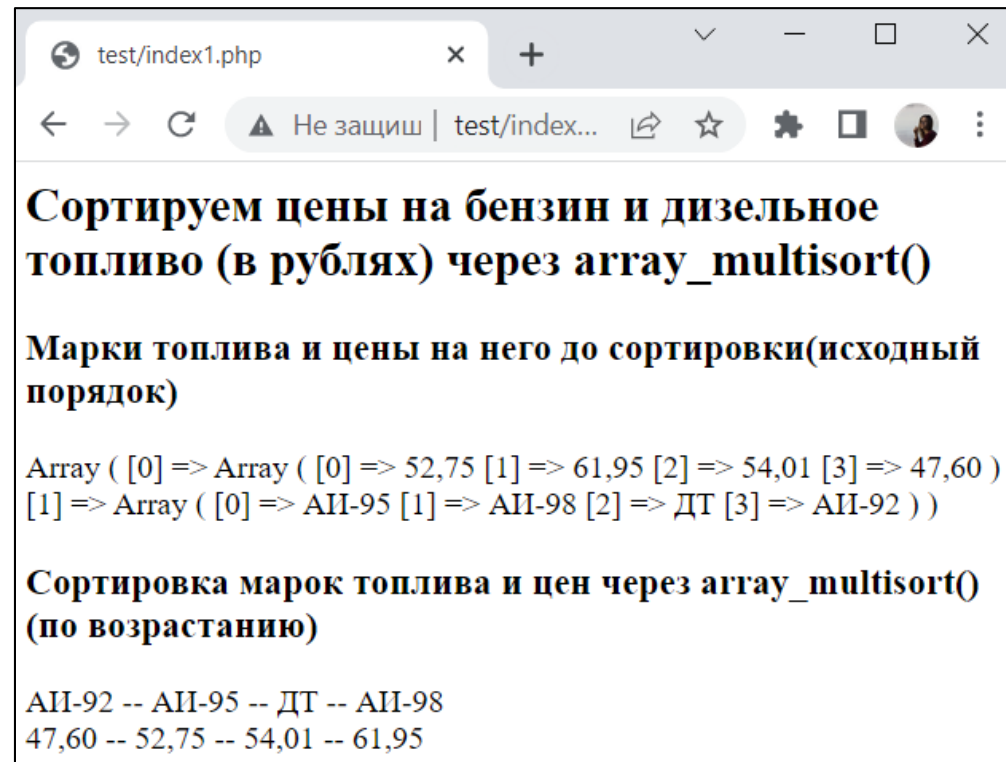
Сортировка массивов

Сортировка массивов с помощью функции `array_multisort()`

```
<?php
echo "<h2>Сортируем цены на бензин и дизельное
топливо (в рублях) через array_multisort()</h2>";
$petrol = array(
    array("52,75", "61,95", "54,01", "47,60"),
    array("АИ-95", "АИ-98", "ДТ", "АИ-92")
);
echo "<h3>Марки топлива и цены на него до
сортировки(исходный порядок)</h3>";
print_r($petrol);

echo("<h3>Сортировка марок топлива и цен через
array_multisort() (по возрастанию)</h3>");
array_multisort($petrol[0], $petrol[1]);
echo implode(" -- ", $petrol[1]) . "<br>";
echo implode(" -- ", $petrol[0]) . "<br>";

?>
```



Сортировка массивов

*Переворачивание массива с помощью функции **array_reverse()***

Функция **array_reverse()** возвращает массив, элементы которого следуют в обратном порядке относительно массива, переданного в параметре.

array_reverse(array \$array, bool \$preserve_keys = false)

Список параметров:

array - входной массив.

preserve_keys – если установлено в true, то числовые ключи будут сохранены.

Нечисловые ключи не подвержены этой опции и всегда сохраняются.

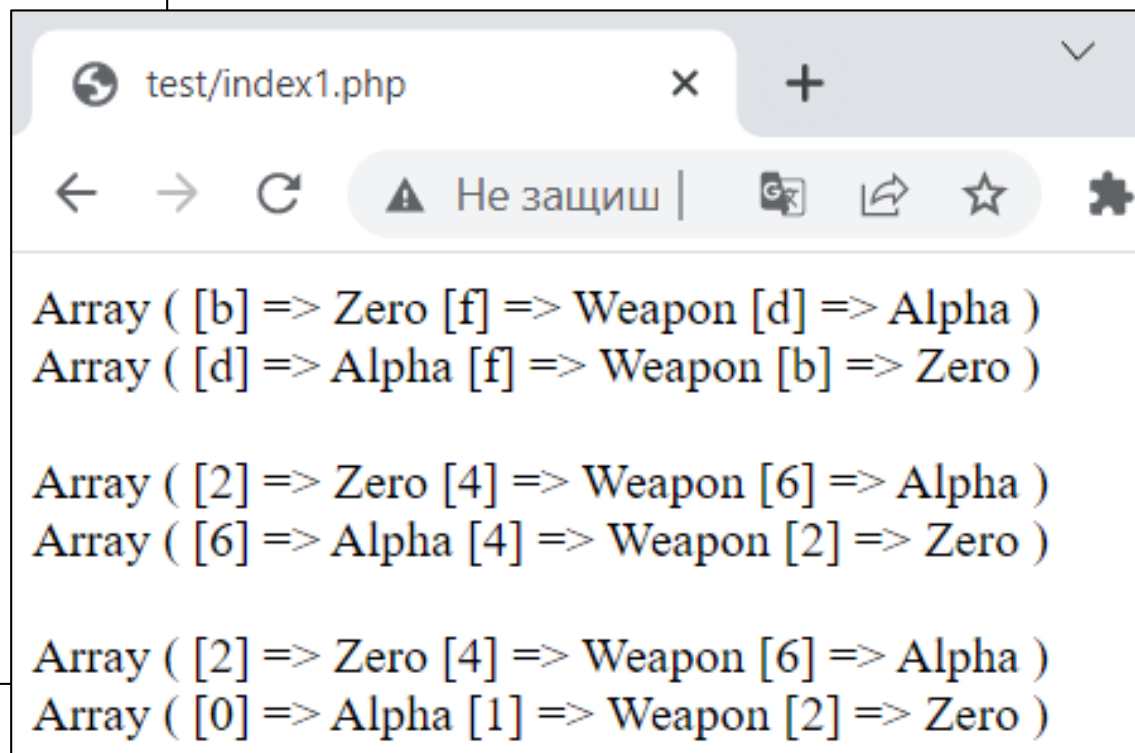
Сортировка массивов

Переворачивание массива с помощью функции `array_reverse()`

```
<?php
$A=array("b"=>"Zero","f"=>"Weapon","d"=>"Alpha");
print_r($A);
$B=array_reverse($A);
print_r($B);

$C=array(2=>"Zero",4=>"Weapon",6=>"Alpha");
print_r($C);
$D=array_reverse($C, true);
print_r($D);

$F=array(2=>"Zero",4=>"Weapon",6=>"Alpha");
print_r($F); echo "<br>";
$E=array_reverse($F);
print_r($E);
?>
```



```
test/index1.php
← → ↻ ⚠ Не защищ | 🔍 📄 ☆ ⚙

Array ( [b] => Zero [f] => Weapon [d] => Alpha )
Array ( [d] => Alpha [f] => Weapon [b] => Zero )

Array ( [2] => Zero [4] => Weapon [6] => Alpha )
Array ( [6] => Alpha [4] => Weapon [2] => Zero )

Array ( [2] => Zero [4] => Weapon [6] => Alpha )
Array ( [0] => Alpha [1] => Weapon [2] => Zero )
```

Сортировка массивов

*Сортировка списка при помощи функций **sort()** и **rsort()***

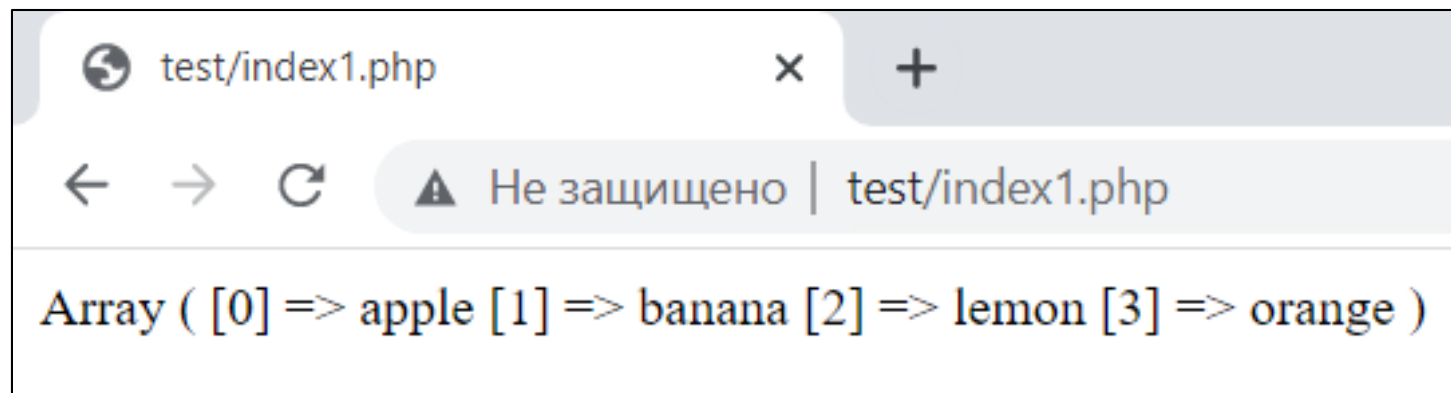
Функция **sort()** сортирует список (разумеется, по значениям) в порядке возрастания, а **rsort()** — в порядке убывания.

Эта функция присваивает новые ключи элементам array. Она удалит все существующие ключи, а не просто переупорядочит их.

Сортировка массивов

*Сортировка списка при помощи функций **sort()** и **rsort()***

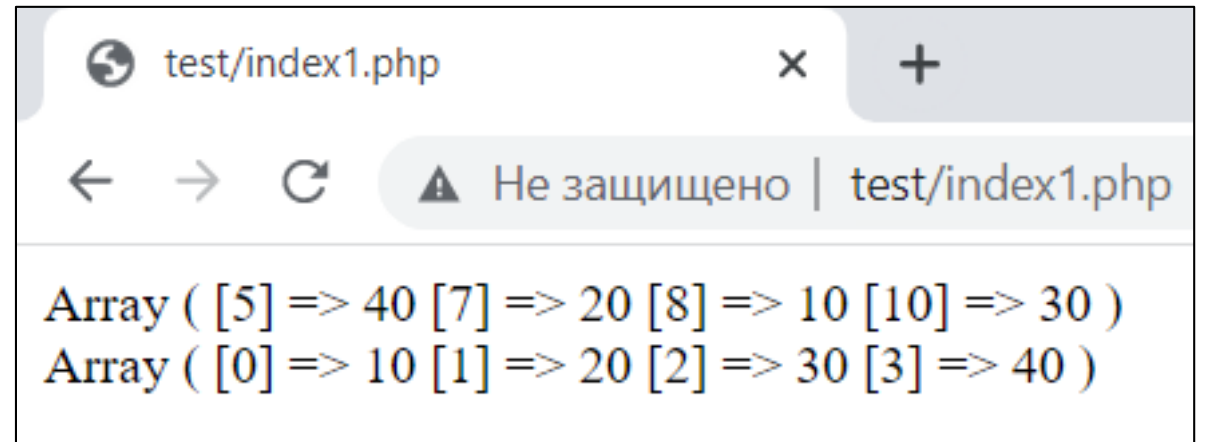
```
<?php
$fruits = array("lemon", "orange", "banana", "apple");
sort($fruits);
print_r($fruits);
?>
```



Сортировка массивов

*Сортировка списка при помощи функций **sort()** и **rsort()***

```
<?php
$A=array(5=>"40", 7=>"20", 8=>"10", 10=>"30");
print_r($A);
sort($A);
print_r($A);
?>
```



```
test/index1.php x +
← → ↻ ⚠ Не защищено | test/index1.php
Array ( [5] => 40 [7] => 20 [8] => 10 [10] => 30 )
Array ( [0] => 10 [1] => 20 [2] => 30 [3] => 40 )
```

Сортировка массивов

*Перемешивание списка с помощью функции **shuffle()***

Функция **shuffle()** "перемешивает" список, переданный ей первым параметром, так, чтобы его значения распределялись случайным образом. Обратите внимание, что, во-первых, изменяется сам массив, а во вторых, ассоциативные массивы воспринимаются как списки.

Эта функция присваивает новые ключи элементам array. Она удалит все существующие ключи, а не просто переупорядочит их.

Сортировка массивов

*Перемешивание списка с помощью функции **shuffle()***

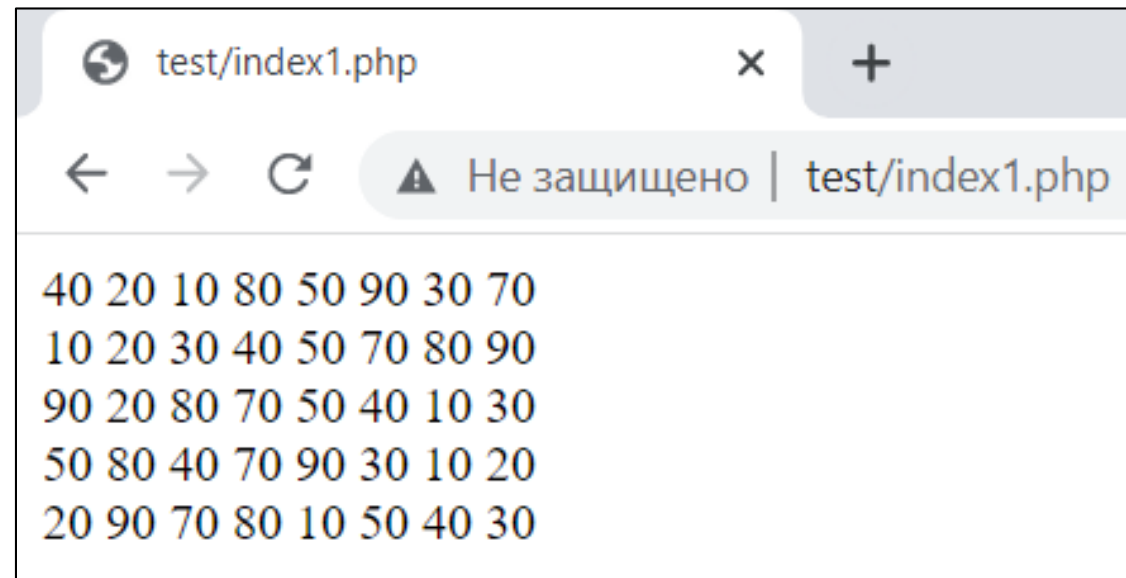
```
<?php
$A=array(40,20,10,80,50,90,30,70);
foreach($A as $V) echo $V." ";

sort($A);
foreach($A as $V) echo $V." ";

shuffle($A);
foreach($A as $V) echo $V." ";

shuffle($A);
foreach($A as $V) echo $V." ";

shuffle($A);
foreach($A as $V) echo $V." ";
?>
```



```
test/index1.php
← → ↻ ⚠ Не защищено | test/index1.php

40 20 10 80 50 90 30 70
10 20 30 40 50 70 80 90
90 20 80 70 50 40 10 30
50 80 40 70 90 30 10 20
20 90 70 80 10 50 40 30
```

Операции с ключами и значениями массива

`array_flip(array $arr)`

Функция **`array_flip()`** "пробегаёт" по массиву и меняет местами его ключи и значения. Исходный массив **`$arr`** не изменяется, а результирующий массив просто возвращается. Конечно, если в массиве присутствовали несколько элементов с одинаковыми значениями, учитываться будет только последний из них.

Операции с ключами и значениями массива

`array_flip(array $arr)`

```
<?php
$A=array("40", "20", "10", "80", "50");
var_dump($A);
$A=array_flip($A);
var_dump($A);
?>
```

```
test/index1.php
array(5) { [0]=> string(2) "40" [1]=> string(2) "20" [2]=> string(2) "10"
[3]=> string(2) "80" [4]=> string(2) "50" }

array(5) { [40]=> int(0) [20]=> int(1) [10]=> int(2) [80]=> int(3) [50]=>
int(4) }
```

```
<?php
$input = array("oranges", "apples", "pears");
print_r($input);
$flipped = array_flip($input);
print_r($flipped);
?>
```

```
test/index1.php
Array ( [0] => oranges [1] => apples [2] => pears )

Array ( [oranges] => 0 [apples] => 1 [pears] => 2 )
```


Операции с ключами и значениями массива

```
array_keys(array $array, mixed $search_value, bool $strict = false)
```

Возвращает все или некоторое подмножество ключей массива.

Список параметров:

array - массив, содержащий возвращаемые ключи.

search_value - если указано, будут возвращены только ключи, у которых значения элементов массива совпадают с этим параметром.

strict - определяет использование строгой проверки на равенство (===) при поиске.

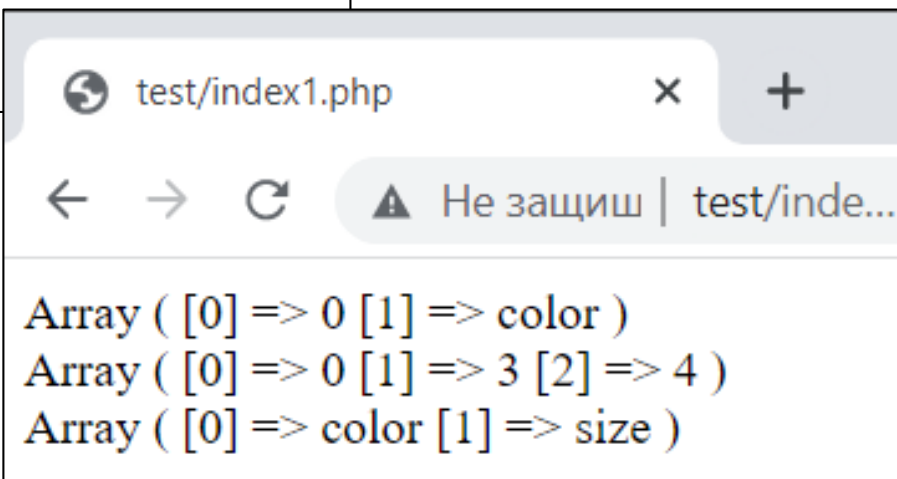
Операции с ключами и значениями массива

`array_keys(array $array, mixed $search_value, bool $strict = false)`

```
<?php
$array = array(0 => 100, "color" => "red");
print_r(array_keys($array));

$array = array("blue", "red", "green", "blue", "blue");
print_r(array_keys($array, "blue"));

$array = array("color" => array("blue", "red", "green"),
              "size"   => array("small", "medium", "large"));
print_r(array_keys($array));
?>
```

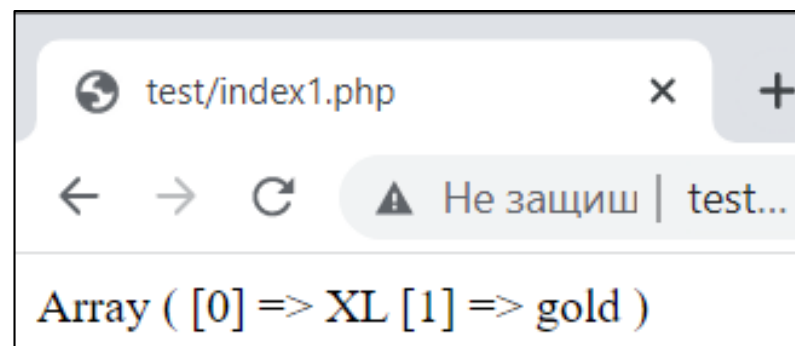


Операции с ключами и значениями массива

`array_values(array $arr)`

Возвращает массив со всеми значениями массива array. Она также заново индексирует возвращаемый массив числовыми индексами.

```
<?php
$array = array("size" => "XL", "color" => "gold");
print_r(array_values($array));
?>
```

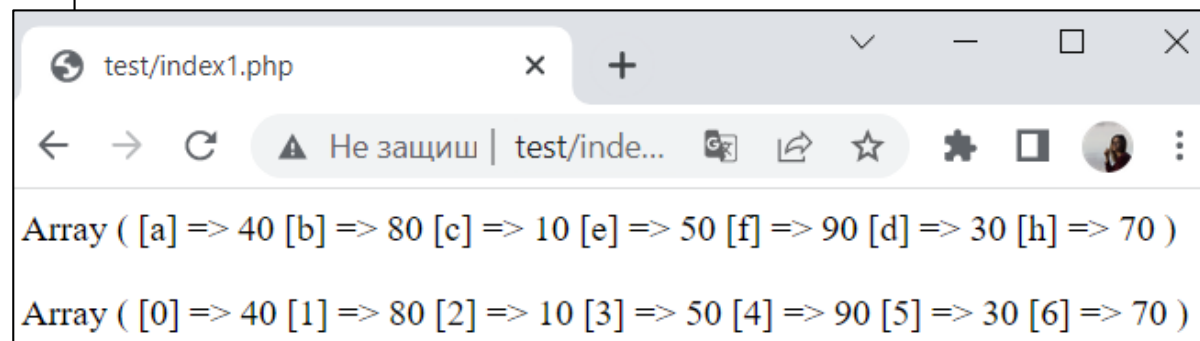


test/index1.php

← → ↻ Не защищ | test...

Array ([0] => XL [1] => gold)

```
<?php
$A=array("a"=>"40", "b"=>"20","c"=> "10",
"b"=>"80", "e"=>"50", "f"=>"90",
"d"=>"30", "h"=>"70");
print_r($A);
$B=array_values($A);
print_r($B);
?>
```



test/index1.php

← → ↻ Не защищ | test/inde... ☆ ⚙ □ 👤 ⋮

Array ([a] => 40 [b] => 80 [c] => 10 [e] => 50 [f] => 90 [d] => 30 [h] => 70)

Array ([0] => 40 [1] => 80 [2] => 10 [3] => 50 [4] => 90 [5] => 30 [6] => 70)

Операции с ключами и значениями массива

`in_array(mixed $val, array $arr)`

Функция `in_array()` возвращает `true`, если элемент со значением `$val` присутствует в массиве `$arr`.

```
<?php
$A=array("a"=>"40", "b"=>"20","c"=> "10", "b"=>"80", "e"=>"50");
print_r($A);
$B=in_array(50,$A);
var_dump($B);
$B=in_array(55,$A);
var_dump($B);
?>
```

test/index1.php



Не защиш | test/inde...



```
Array ( [a] => 40 [b] => 80 [c] => 10 [e] => 50 )
bool(true)
bool(false)
```

Операции с ключами и значениями массива

`array_count_values(list $List)`

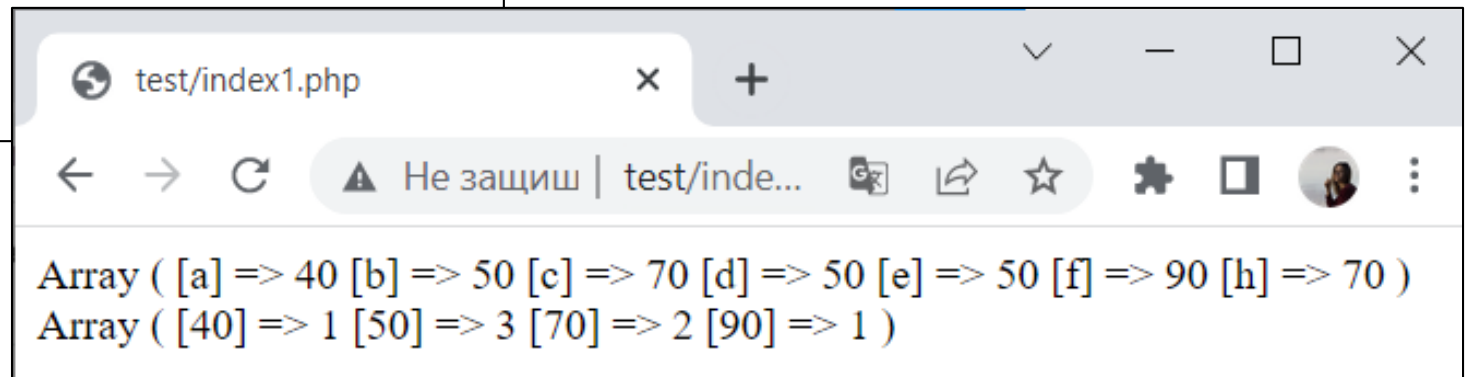
Функция **`array_count_values()`** подсчитывает, сколько раз каждое значение встречается в списке **`$List`**, и возвращает ассоциативный массив с ключами — элементами списка и значениями — количеством повторов этих элементов. Иными словами, функция **`array_count_values()`** подсчитывает частоту появления значений в списке **`$List`**.

Операции с ключами и значениями массива

array_count_values(list \$List)

```
<?php
$A = array("a"=>"40", "b"=>"50", "c"=>"70",
"d"=>"50", "e"=>"50", "f"=>"90", "h"=>"70");

print_r($A); echo "<br>";
$B = array_count_values($A);
print_r($B);
?>
```



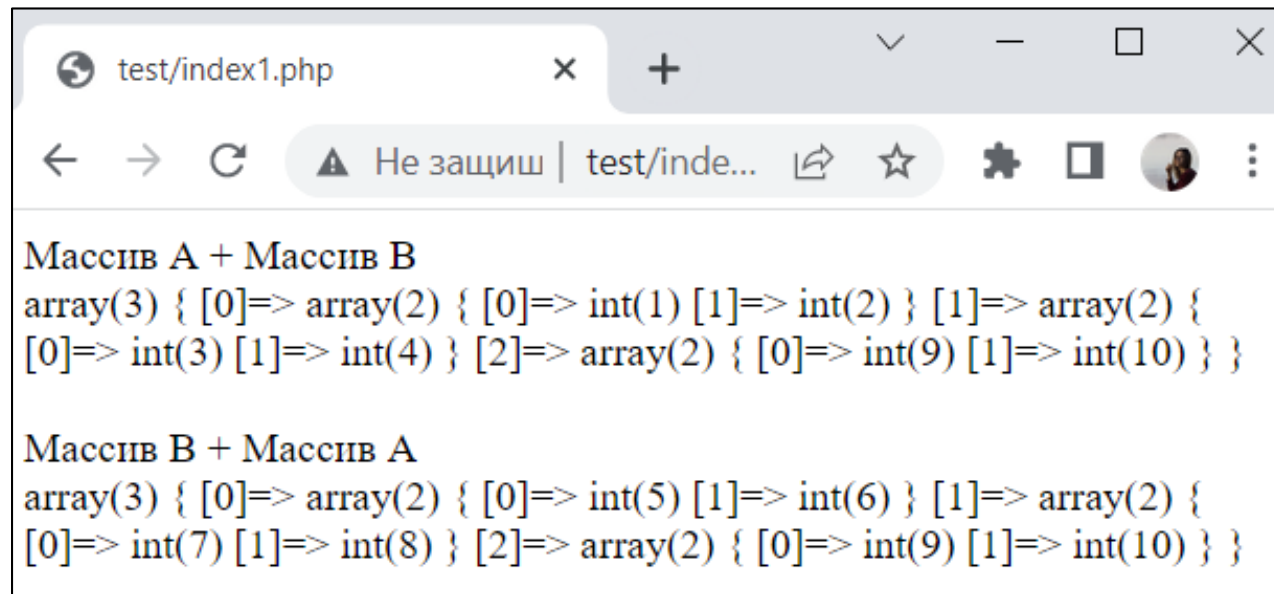
Слияние массивов

Слияние (конкатенация) массивов – это операция создания массива, состоящего из элементов нескольких других массивов. Слияние массивов - это очень опасная операция, поскольку результат слияния подчиняется своей логике, забыв о которой можно потерять данные. Слияние массивов реализуется при помощи оператора "+" или с помощью функции `array_merge()`.

Слияние массивов

```
<?php
$A[] = array(0=>1, 1=>2);
$A[] = array(0=>3, 1=>4);
$B[] = array(0=>5, 1=>6);
$B[] = array(0=>7, 1=>8);
$B[] = array(0=>9, 1=>10);

echo "Массив A + Массив B", "<br>";
$C = $A + $B;
var_dump($C);
echo "Массив B + Массив A", "<br>";
$C=$B+$A;
var_dump($C);
?>
```



Слияние массивов

Функция `array_merge()`

Функция `array_merge()` призвана устранить все недостатки, присущие оператору "+" для слияния массивов. А именно, она сливает массивы, перечисленные в ее аргументах, в один большой массив и возвращает результат.

Если входные массивы имеют одинаковые строковые ключи, тогда каждое последующее значение будет заменять предыдущее. Однако, если массивы имеют одинаковые числовые ключи, значение, упомянутое последним, *не заменит* исходное значение, а будет добавлено в конец массива.

В результирующем массиве значения исходного массива с числовыми ключами будут перенумерованы в возрастающем порядке, начиная с нуля.

Слияние массивов

Функция `array_merge()`

```
<?php
$A= array("1"=>"первый", "2"=>"второй");
$B = array("1"=>"третий",
"2"=>"четвертый", "3"=>"пятый");
var_dump($A);
var_dump($B);
echo "Массив A + Массив B", "<br>";
$C = $A + $B;
var_dump($C);
echo "Массив B + Массив A", "<br>";
$C=$B+$A;
var_dump($C);
echo 'array_merge($A,$B) ', "<br>";
$C=array_merge($A,$B);
var_dump($C);
?>
```

test/index1.php

← → ↻ ⚠ Не защищ | test/inde... ☆ ⚙ □ 👤 ⋮

array(2) { [1]=> string(12) "первый" [2]=> string(12) "второй" }

array(3) { [1]=> string(12) "третий" [2]=> string(18) "четвертый" [3]=> string(10) "пятый" }

Массив A + Массив B

array(3) { [1]=> string(12) "первый" [2]=> string(12) "второй" [3]=> string(10) "пятый" }

Массив B + Массив A

array(3) { [1]=> string(12) "третий" [2]=> string(18) "четвертый" [3]=> string(10) "пятый" }

array_merge(\$A,\$B)

array(5) { [0]=> string(12) "первый" [1]=> string(12) "второй" [2]=> string(12) "третий" [3]=> string(18) "четвертый" [4]=> string(10) "пятый" }

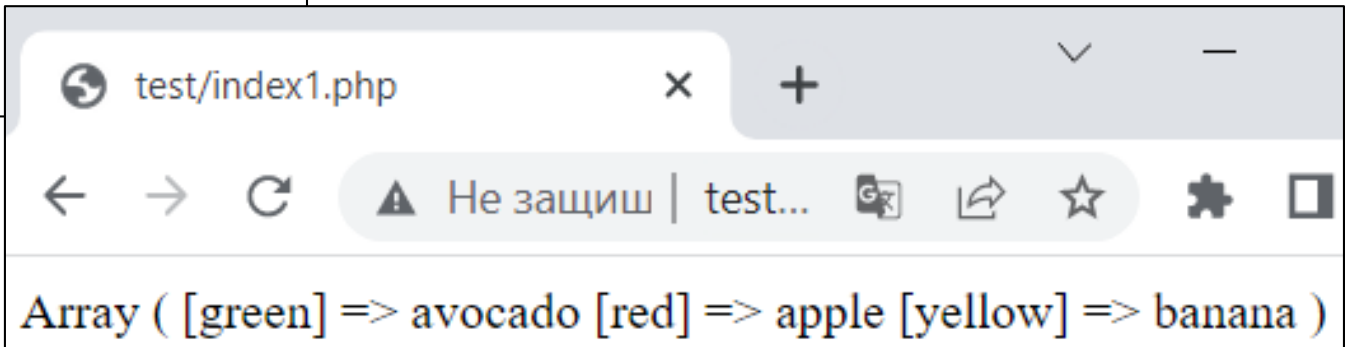
Слияние массивов

Функция `array_combine()`

Создаёт новый массив, используя один массив в качестве ключей, а другой для его значений.

```
<?php
$a = array('green', 'red', 'yellow');
$b = array('avocado', 'apple', 'banana');
$c = array_combine($a, $b);

print_r($c);
?>
```



Получение части массива

```
array_slice(array $Arr, int $offset [, int $len])
```

Эта функция возвращает часть массива, начиная с пары ключ => значения со смещением (номером) **\$offset** от начала и длиной **\$len** (если последний параметр не задан – до конца массива). Параметры **\$offset** и **\$len** задаются по точно таким же правилам, как и аналогичные параметры в функции `substr()`. А именно, они могут быть отрицательными (в этом случае отсчет осуществляется от конца массива) и т.д.

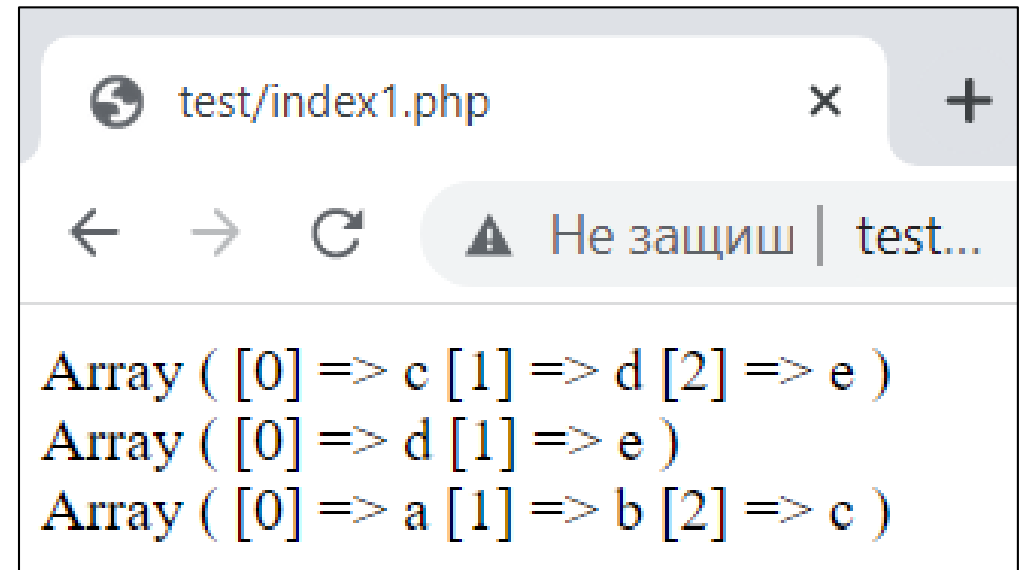
Получение части массива

array_slice(array \$Arr, int \$offset [, int \$len])

```
<?php
$input = array ("a", "b", "c", "d", "e");
$output = array_slice ($input, 2);
// "c", "d", "e"
print_r($output);

$output = array_slice ($input, -2, 2);
// "d", "e"
print_r($output);

$output = array_slice ($input, 0, 3);
// "a", "b", "c"
print_r($output);
?>
```



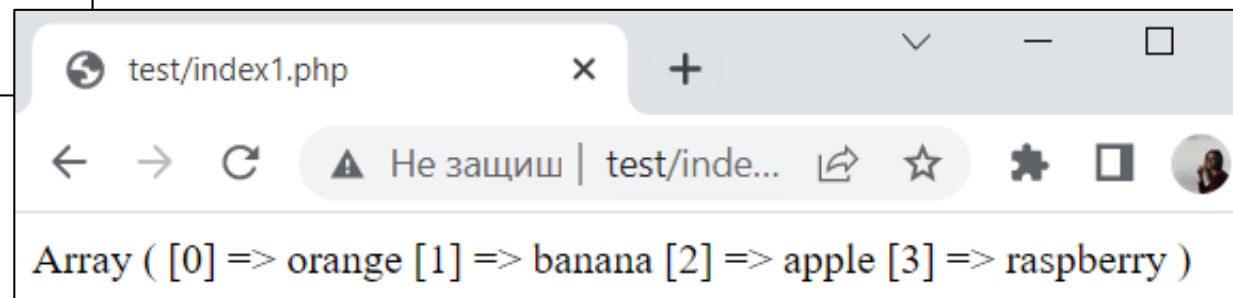
Вставка и удаление элементов массивов

`array_push(aList &$Arr, mixed $var1 [, mixed $var2, ...])`

Эта функция добавляет к списку `$Arr` элементы `$var1`, `$var2` и т. д. Она присваивает им числовые индексы — точно так же, как это происходит для стандартных [].

Вместо использования `array_push()` для добавления одного элемента в массив, лучше использовать `$array[] =`, потому что в этом случае не происходит затрат на вызов функции.

```
<?php
$stack = array("orange", "banana");
array_push($stack, "apple", "raspberry");
print_r($stack);
?>
```

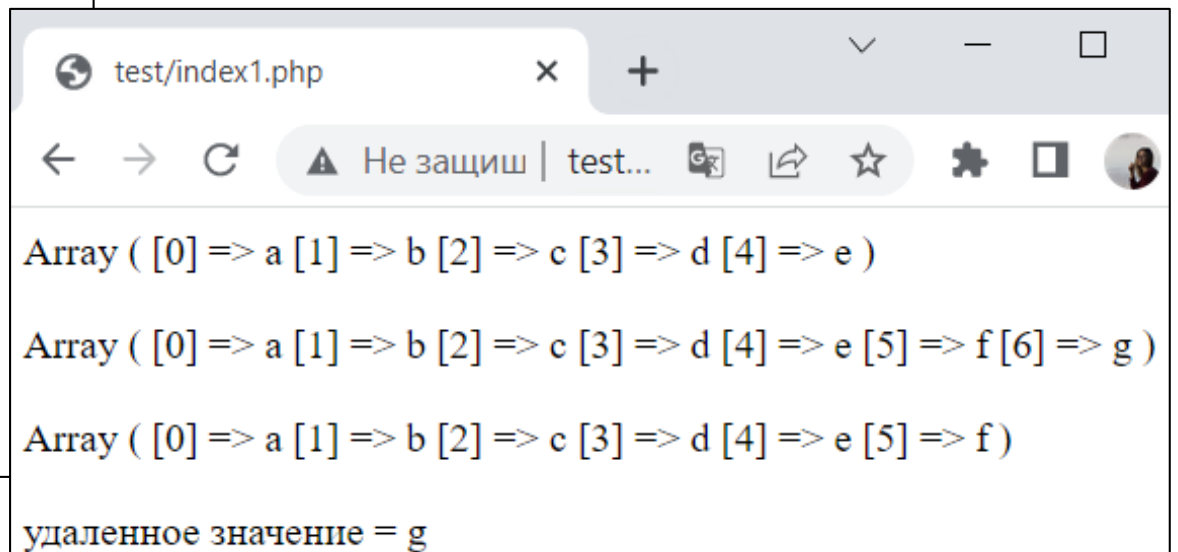


Вставка и удаление элементов массивов

`array_pop(list &$Arr)`

Функция `array_pop()` является противоположностью `array_push()`, снимает элемент с "вершины" стека (то есть берет последний элемент списка) и возвращает его, удалив после этого его из `$Arr`.

```
<?php
$A = array ("a", "b", "c", "d", "e");
print_r($A);
array_push($A, "f", "g");
print_r($A);
$B=array_pop($A);
print_r($A);
echo "удаленное значение = $B";
?>
```



```
test/index1.php
Array ( [0] => a [1] => b [2] => c [3] => d [4] => e )
Array ( [0] => a [1] => b [2] => c [3] => d [4] => e [5] => f [6] => g )
Array ( [0] => a [1] => b [2] => c [3] => d [4] => e [5] => f )
удаленное значение = g
```

Вставка и удаление элементов массивов

```
array_unshift(list &$Arr, mixed $var1 [, mixed $var2, ...])
```

Функция `array_unshift` очень похожа на `array_push()`, но добавляет перечисленные элементы не в конец, а в начало массива. При этом порядок следования `$var1`, `$var2` и т. д. остается тем же, т. е. элементы как бы "вдвигаются" в список слева.

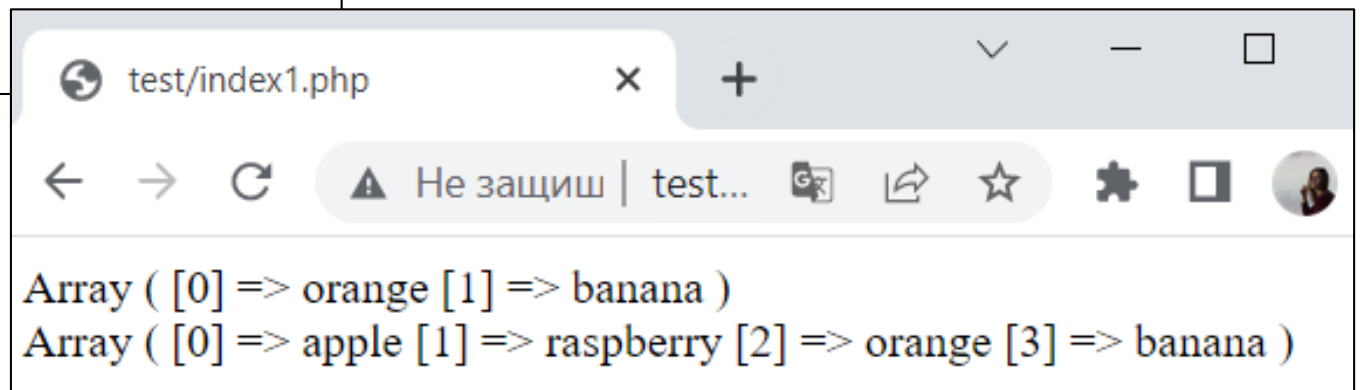
Новым элементам списка, как обычно, назначаются числовые индексы, начиная с 0; при этом все ключи старых элементов массива, которые также были числовыми, изменяются (чаще всего они увеличиваются на число вставляемых значений). Функция возвращает новый размер массива.

Вставка и удаление элементов массивов

`array_unshift(list &$Arr, mixed $var1 [, mixed $var2, ...])`

```
<?php
$queue = array("orange", "banana");
print_r($queue);

array_unshift($queue, "apple", "raspberry");
print_r($queue);
?>
```



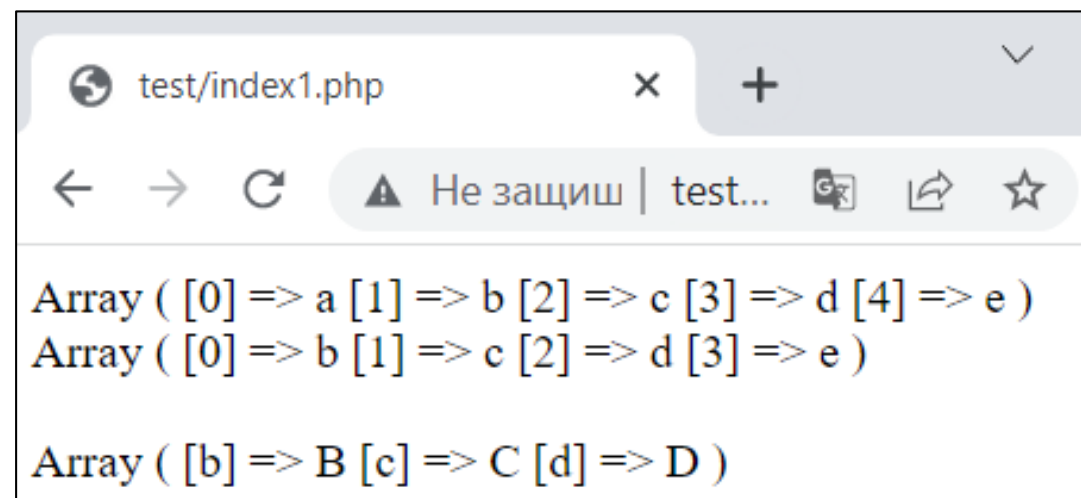
Вставка и удаление элементов массивов

`array_shift(list &$Arr)`

Функция `array_shift()` извлекает первый элемент массива `$Arr` и возвращает его. Она сильно напоминает `array_pop()`, но только получает начальный, а не конечный элемент, а также производит довольно сильную "встряску" всего массива: ведь при извлечении первого элемента приходится корректировать все числовые индексы у всех оставшихся элементов.

```
<?php
$A = array ("a", "b", "c", "d", "e");
print_r($A); echo "<br>";
array_shift($A);
print_r($A);

$A = array ("a"=>"A", "b"=>"B", "c"=>"C",
"d"=>"D");
array_shift($A);
print_r($A);
?>
```



```
Array ( [0] => a [1] => b [2] => c [3] => d [4] => e )
Array ( [0] => b [1] => c [2] => d [3] => e )

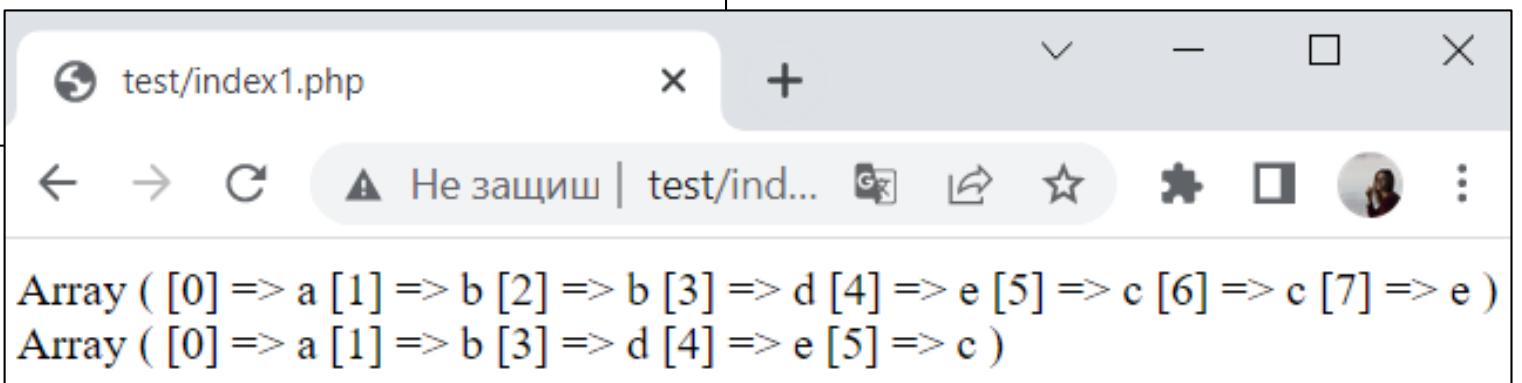
Array ( [b] => B [c] => C [d] => D )
```

Вставка и удаление элементов массивов

`array_unique(array $Arr)`

Функция `array_unique()` возвращает массив, составленный из всех уникальных значений массива `$Arr` вместе с их ключами. В результирующий массив помещаются первые встретившиеся пары ключ=>значение. Обратите внимание, что ключи сохраняются.

```
<?php
$A = array ("a", "b", "b", "d", "e", "c", "c", "e");
print_r($A);
$B=array_unique($A);
print_r($B);
?>
```



```
test/index1.php
Array ( [0] => a [1] => b [2] => b [3] => d [4] => e [5] => c [6] => c [7] => e )
Array ( [0] => a [1] => b [3] => d [4] => e [5] => c )
```

Вставка и удаление элементов массивов

```
array_splice(array &$Arr, int $offset [, int $len] [, int $Repl])$Arr)
```

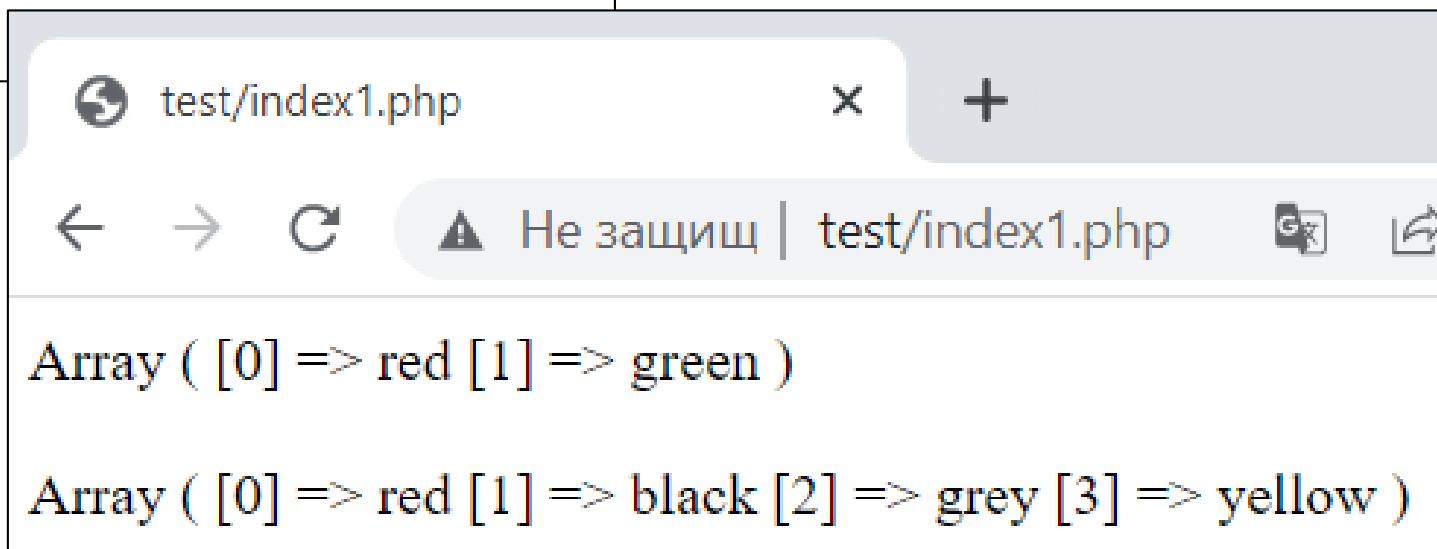
Функция **array_splice**, также как и **array_slice()**, возвращает подмассив **\$Arr**, начиная с индекса **\$offset** максимальной длины **\$len**, но, вместе с тем, она делает и другое полезное действие. А именно, она заменяет только что указанные элементы на то, что находится в массиве **\$Repl** (или просто удаляет, если **\$Repl** не указан).

Параметры **\$offset** и **\$len** задаются так же, как и в функции **substr()** — а именно, они могут быть и отрицательными, в этом случае отсчет начинается от конца массива.

Вставка и удаление элементов массивов

`array_splice(array &$Arr, int $offset [, int $len] [, int $Repl])$Arr)`

```
<?php
$input=array("red", "green", "blue", "yellow");
array_splice($input,2);
print_r($input);
$input=array("red", "green", "blue", "yellow");
array_splice($input, 1, 2, array("black", "grey"));
print_r($input);
?>
```



Вставка и удаление элементов массивов

array_splice(array &\$Arr, int \$offset [, int \$len] [, int \$Repl])\$Arr)

```
<?php
$input=array("red", "green", "blue", "yellow");
array_splice($input, 1, 1, array("black", "grey"));
print_r($input);
$input=array("red", "green", "blue", "yellow");
array_splice($input, 2, 0, array("black", "grey"));
print_r($input);
$input=array("red", "green", "blue", "yellow");
array_splice($input, 1, 2, "black");
print_r($input);
?>
```

Переменные и массивы

```
compact(mixed $vn1 [, mixed $vn2, ...])
```

Функция **compact()** упаковывает в массив переменные из текущего контекста (глобального или контекста функции), заданные своими именами в **\$vn1**, **\$vn2** и т. д. При этом в массиве образуются пары с ключами, равными содержимому **\$vnN**, и значениями соответствующих переменных.

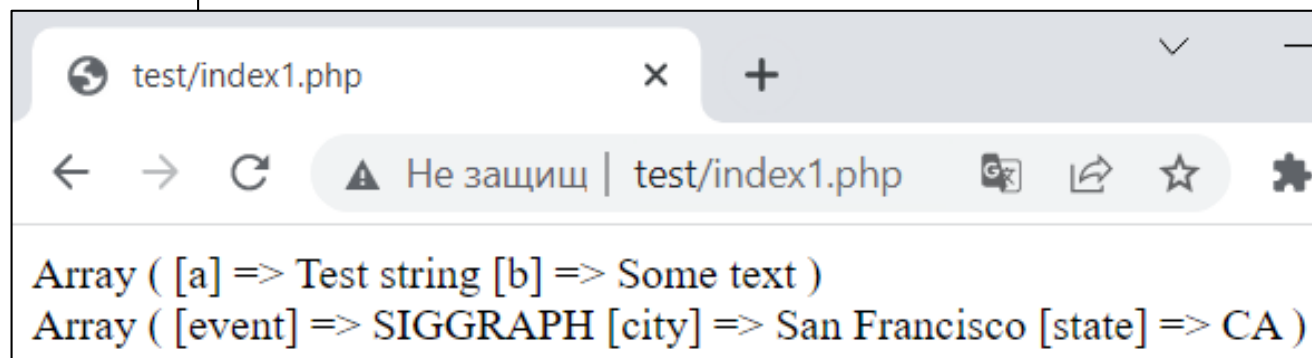
Переменные и массивы

compact(mixed \$vn1 [, mixed \$vn2, ...])

```
<?php
$a="Test string";
$b="Some text";
$A=compact("a","b");
print_r($A);

$city  = "San Francisco";
$state = "CA";
$event = "SIGGRAPH";
$location_vars = array("city", "state");

$result = compact("event", $location_vars);
print_r($result);
?>
```

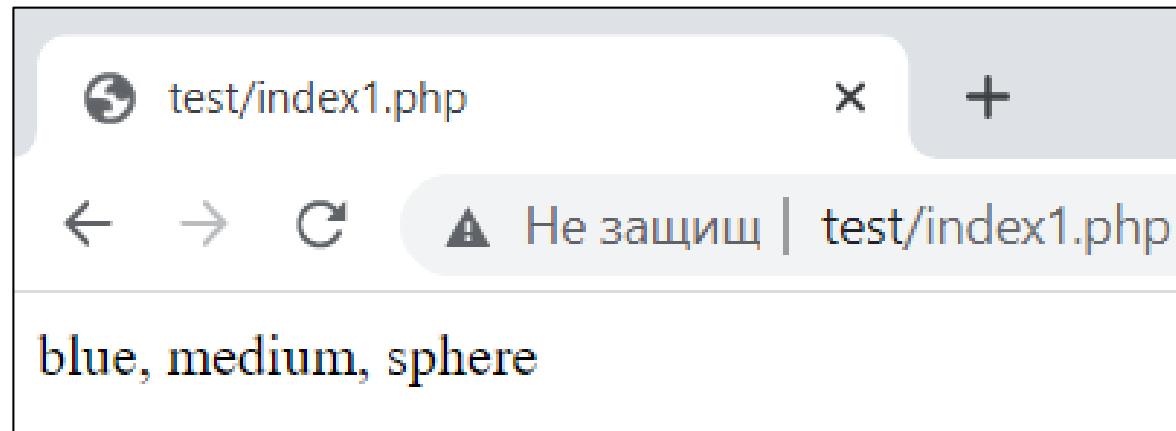


Переменные и массивы

extract(array \$Arr [, int \$type] [, string \$prefix])

Функция **extract()** производит действия, прямо противоположные **compact()**. А именно, она получает в параметрах массив **\$Arr** и превращает каждую его пару ключ=>значение в переменную текущего контекста.

```
<?php
$var_array = array("color" => "blue",
                  "size"  => "medium",
                  "shape" => "sphere");
extract($var_array);
echo "$color, $size, $shape";
?>
```

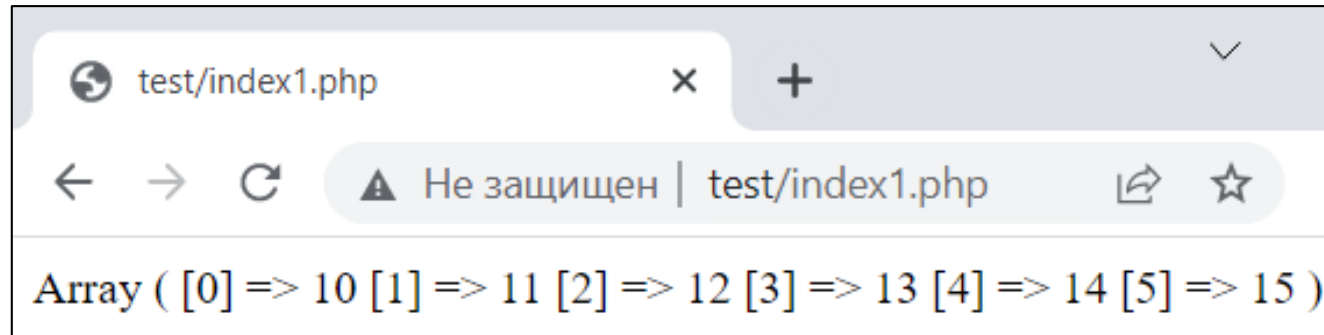


Создание списка, диапазон чисел

`range(int $low, int $high)`

Эта функция очень простая. Она создает список, заполненный целыми числами от `$low` до `$high` включительно.

```
<?php
$low=10;
$high=15;
$A=range($low,$high);
print_r($A);
?>
```

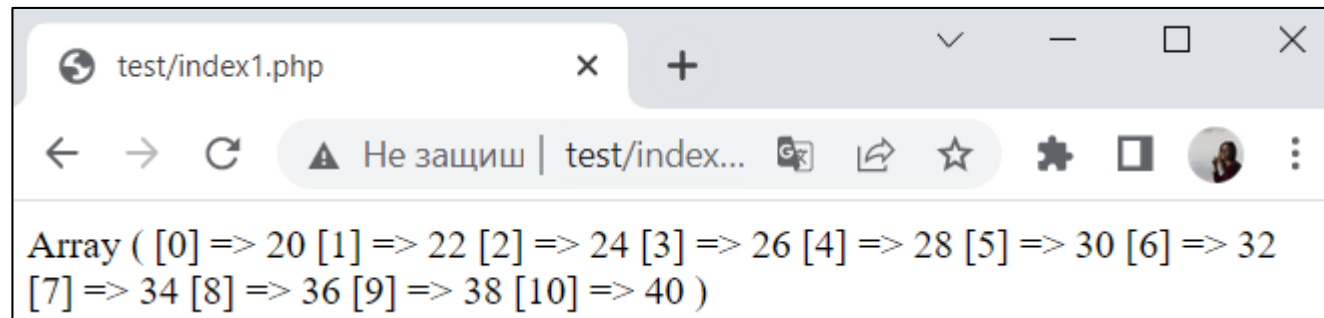


test/index1.php

Не защищен | test/index1.php

Array ([0] => 10 [1] => 11 [2] => 12 [3] => 13 [4] => 14 [5] => 15)

```
<?php
$low=20;
$high=35;
$A=range($low,$high,2);
print_r($A);
?>
```



test/index1.php

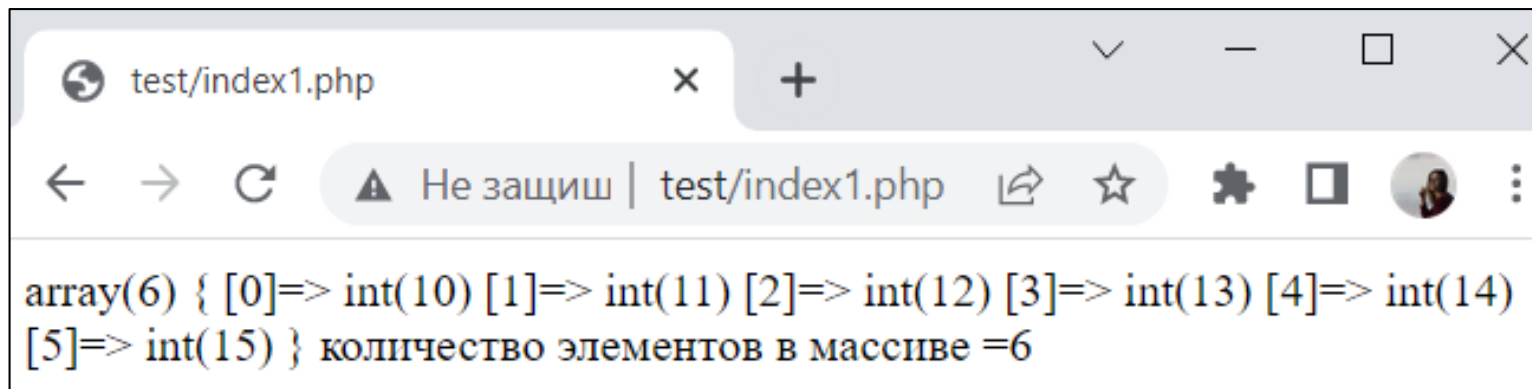
Не защищен | test/index...

Array ([0] => 20 [1] => 22 [2] => 24 [3] => 26 [4] => 28 [5] => 30 [6] => 32 [7] => 34 [8] => 36 [9] => 38 [10] => 40)

Счетчик элементов массива

Для подсчета элементов массива предназначена функция **count()**.

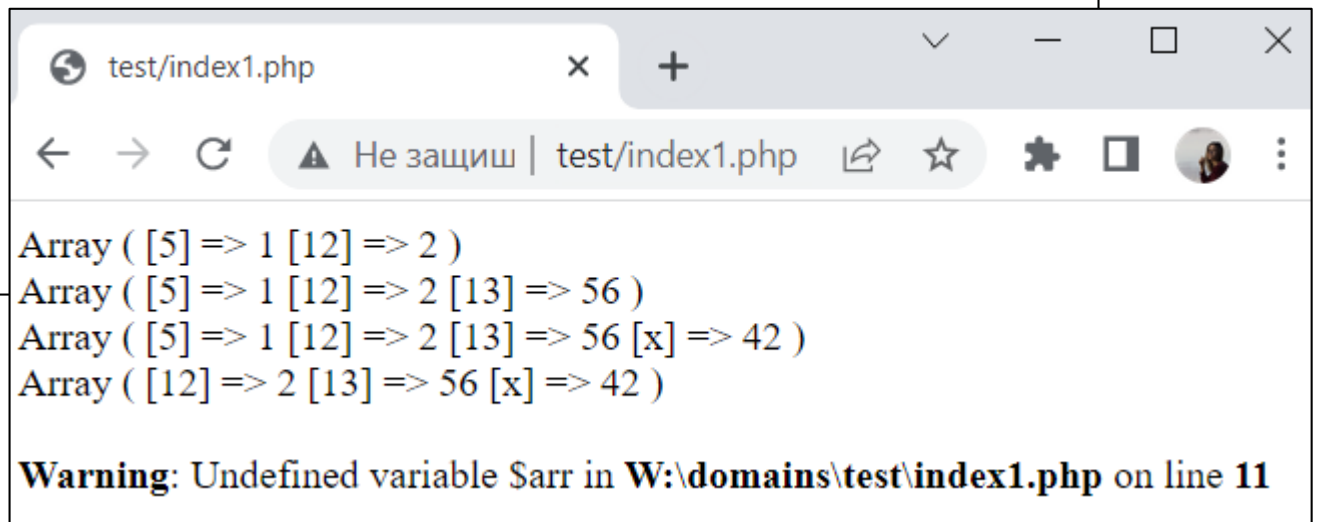
```
<?php
$low=10;
$high=15;
$A=range($low,$high);
var_dump($A);
echo "количество элементов в массиве=".count($A);
?>
```



Удаление массива и его элементов

Если вы хотите удалить массив целиком, воспользуйтесь функцией `unset()`.

```
<?php
$arr = array(5 => 1, 12 => 2);
print_r($arr);
$arr[] = 56;      // В этом месте скрипта это эквивалентно $arr[13] = 56;
print_r($arr);
$arr["x"] = 42; // Это добавляет к массиву новый элемент с ключом "x"
print_r($arr);
unset($arr[5]); // Это удаляет элемент из массива
print_r($arr);
unset($arr);
// Это удаляет массив полностью
print_r($arr);
?>
```

A screenshot of a web browser window. The address bar shows 'test/index1.php'. The page content displays four lines of PHP array output: 'Array ([5] => 1 [12] => 2)', 'Array ([5] => 1 [12] => 2 [13] => 56)', 'Array ([5] => 1 [12] => 2 [13] => 56 [x] => 42)', and 'Array ([12] => 2 [13] => 56 [x] => 42)'. Below the output, a warning message is displayed: 'Warning: Undefined variable \$arr in W:\domains\test\index1.php on line 11'.

```
test/index1.php x + - □ ×
← → ↻ ⚠ Не защищ | test/index1.php 🔗 ☆ ⚙ □ 👤 ⋮
Array ( [5] => 1 [12] => 2 )
Array ( [5] => 1 [12] => 2 [13] => 56 )
Array ( [5] => 1 [12] => 2 [13] => 56 [x] => 42 )
Array ( [12] => 2 [13] => 56 [x] => 42 )

Warning: Undefined variable $arr in W:\domains\test\index1.php on line 11
```

Список функций для работы с массивами

[array_change_key_case](#) — Меняет регистр всех ключей в массиве

[array_chunk](#) — Разбивает массив на части

[array_column](#) — Возвращает массив из значений одного столбца входного массива

[array_combine](#) — Создаёт новый массив, используя один массив в качестве ключей, а другой для его значений

[array_count_values](#) — Подсчитывает количество всех значений массива

[array_diff_assoc](#) — Вычисляет расхождение массивов с дополнительной проверкой индекса

[array_diff_key](#) — Вычисляет расхождение массивов, сравнивая ключи

[array_diff_uassoc](#) — Вычисляет расхождение массивов с дополнительной проверкой индекса, осуществляемой при помощи callback-функции

[array_diff_ukey](#) — Вычисляет расхождение массивов, используя callback-функцию для сравнения ключей

[array_diff](#) — Вычислить расхождение массивов

[array_fill_keys](#) — Создаёт массив и заполняет его значениями с определёнными ключами

[array_fill](#) — Заполняет массив значениями

[array_filter](#) — Фильтрует элементы массива с помощью callback-функции

[array_flip](#) — Меняет местами ключи с их значениями в массиве

[array_intersect_assoc](#) — Вычисляет схождение массивов с дополнительной проверкой индекса

[array_intersect_key](#) — Вычислить пересечение массивов, сравнивая ключи

[array_intersect_uassoc](#) — Вычисляет схождение массивов с дополнительной проверкой индекса, осуществляемой при помощи callback-функции

Список функций для работы с массивами

[array_intersect_ukey](#) — Вычисляет схождение массивов, используя callback-функцию для сравнения ключей

[array_intersect](#) — Вычисляет схождение массивов

[array_is_list](#) — Проверяет, является ли данный array списком

[array_key_exists](#) — Проверяет, присутствует ли в массиве указанный ключ или индекс

[array_key_first](#) — Получает первый ключ массива

[array_key_last](#) — Получает последний ключ массива

[array_keys](#) — Возвращает все или некоторое подмножество ключей массива

[array_map](#) — Применяет callback-функцию ко всем элементам указанных массивов

[array_merge_recursive](#) — Рекурсивное слияние одного или более массивов

[array_merge](#) — Сликает один или большее количество массивов

[array_multisort](#) — Сортирует несколько массивов или многомерные массивы

[array_pad](#) — Дополнить массив определённым значением до указанной длины

[array_pop](#) — Извлекает последний элемент массива

[array_product](#) — Вычислить произведение значений массива

[array_push](#) — Добавляет один или несколько элементов в конец массива

[array_rand](#) — Выбирает один или несколько случайных ключей из массива

[array_reduce](#) — Итеративно уменьшает массив к единственному значению, используя callback-функцию

[array_replace_recursive](#) — Рекурсивно заменяет элементы первого массива элементами переданных массивов

Список функций для работы с массивами

[array_replace](#) — Заменяет элементы массива элементами других переданных массивов

[array_reverse](#) — Возвращает массив с элементами в обратном порядке

[array_search](#) — Осуществляет поиск данного значения в массиве и возвращает ключ первого найденного элемента в случае успешного выполнения

[array_shift](#) — Извлекает первый элемент массива

[array_slice](#) — Выбирает срез массива

[array_splice](#) — Удаляет часть массива и заменяет её чем-нибудь ещё

[array_sum](#) — Вычисляет сумму значений массива

[array_udiff_assoc](#) — Вычисляет расхождение в массивах с дополнительной проверкой индексов, используя для сравнения значений callback-функцию

[array_udiff_uassoc](#) — Вычисляет расхождение в массивах с дополнительной проверкой индексов, используя для сравнения значений и индексов callback-функцию

[array_udiff](#) — Вычисляет расхождение массивов, используя для сравнения callback-функцию

[array_uintersect_assoc](#) — Вычисляет пересечение массивов с дополнительной проверкой индексов, используя для сравнения значений callback-функцию

[array_uintersect_uassoc](#) — Вычисляет пересечение массивов с дополнительной проверкой индекса, используя для сравнения индексов и значений индивидуальные callback-функции

[array_uintersect](#) — Вычисляет пересечение массивов, используя для сравнения значений callback-функцию

Список функций для работы с массивами

[array_unique](#) — Убирает повторяющиеся значения из массива

[array_unshift](#) — Добавляет один или несколько элементов в начало массива

[array_values](#) — Выбирает все значения массива

[array_walk_recursive](#) — Рекурсивно применяет пользовательскую функцию к каждому элементу массива

[array_walk](#) — Применяет заданную пользователем функцию к каждому элементу массива

[array](#) — Создает массив

[arsort](#) — Сортирует массив в порядке убывания и поддерживает ассоциацию индексов

[asort](#) — Сортирует массив в порядке возрастания и поддерживает ассоциацию индексов

[compact](#) — Создает массив, содержащий названия переменных и их значения

[count](#) — Подсчитывает количество элементов массива или Countable объекте

[current](#) — Возвращает текущий элемент массива

[each](#) — Возвращает текущую пару ключ/значение из массива и смещает его указатель

[end](#) — Устанавливает внутренний указатель массива на его последний элемент

[extract](#) — Импортирует переменные из массива в текущую таблицу символов

[in_array](#) — Проверяет, присутствует ли в массиве значение

[key_exists](#) — Псевдоним array_key_exists

[key](#) — Выбирает ключ из массива

[krsort](#) — Сортирует массив по ключу в порядке убывания

[ksort](#) — Сортирует массив по ключу в порядке возрастания

Список функций для работы с массивами

[list](#) — Присваивает переменным из списка значения подобно массиву

[natcasesort](#) — Сортирует массив, используя алгоритм "natural order" без учёта регистра символов

[natsort](#) — Сортирует массив, используя алгоритм "natural order"

[next](#) — Перемещает указатель массива вперёд на один элемент

[pos](#) — Псевдоним current

[prev](#) — Передвигает внутренний указатель массива на одну позицию назад

[range](#) — Создаёт массив, содержащий диапазон элементов

[reset](#) — Устанавливает внутренний указатель массива на его первый элемент

[rsort](#) — Сортирует массив в порядке убывания

[shuffle](#) — Перемешивает массив

[sizeof](#) — Псевдоним count

[sort](#) — Сортирует массив по возрастанию

[uasort](#) — Сортирует массив, используя пользовательскую функцию для сравнения элементов с сохранением ключей

[uksort](#) — Сортирует массив по ключам, используя пользовательскую функцию для сравнения ключей

[usort](#) — Сортирует массив по значениям используя пользовательскую функцию для сравнения элементов