# **Лабораторная работа №6-7. Работа с массивами в PHP. Операции над массивами**

Рассмотрим некоторые часто используемые функции для работы с массивами.

## **Функция list()**

Предположим, у нас есть массив, состоящий из трех элементов:

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

Допустим, в какой-то момент нам нужно передать значения всех трех элементов массива, соответственно трем переменным: ***$alex***, ***$nick***, ***$yakov***. Это можно сделать так:

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

Если массив большой, то такой способ присвоения элементов массива переменным не очень удобен.

Есть более рациональный подход – использование функции **list()**:

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

Если нам нужны только "Николай" и "Яков", то мы можем сделать так:

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

## **Функция array()**

Функция **Array()** используется специально для создания массивов. При этом она позволяет создавать пустые массивы. Вот методы использования функции **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");  
*// Создает многомерный ассоциативный массив:*  
$arr4 = array("name"=>"Иванов", "age"=>"24", "email"=>"ivanov@mail.ru");  
$arr4 = array("name"=>"Петров", "age"=>"34", "email"=>"petrov@mail.ru");  
$arr4 = array("name"=>"Сидоров", "age"=>"47", "email"=>"sidorov@mail.ru");  
?>

## **Операции над массивами**

### **Сортировка массивов**

Начнем с самого простого – сортировки массивов. В **PHP** для этого существует очень много функций. С их помощью можно сортировать ассоциативные массивы и списки в порядке возрастания или убывания, а также в том порядке, в каком вам необходимо – посредством пользовательской функции сортировки.

### **Сортировка массива по значениям c 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». Например:

$A=array("a"=>"Zero","b"=>"Weapon","c"=>"Alpha","d"=>"Processor");  
asort($A);  
foreach($A as $k=>$v) echo "$k=>$v ";

*// выводит "c=>Alpha d=>Processor b=>Weapon a=>Zero"  
// как видим, поменялся только порядок пар ключ=>значение*

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

### **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).

Например:

$A = array("d"=>"Zero", "c"=>"Weapon", "b"=>"Alpha", "a"=>"Processor");  
ksort($A);

for(Reset($A); list($k,$v)=each($A);)

echo "$k=>$v ";  
*// выводит "a=>Processor b=>Alpha c=>Weapon d=>Zero"*

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

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

Довольно часто нам приходится сортировать что-то по более сложному критерию, чем просто по алфавиту. 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()**.

<?php  
*// Эта функция должна сравнивать значения $f1 и $f2 и возвращать:  
// -1, если $f1<$f2,  
// 0, если $f1==$f2  
// 1, если $f1>$f2  
// Под < и > понимается следование этих имен в выводимом списке*function FCmp($f1,$f2) {

*// Каталог всегда предшествует файлу*

if(is\_dir($f1) && !is\_dir($f2)) return -1;  
*// Файл всегда идет после каталога*

if(!is\_dir($f1) && is\_dir($f2)) return 1;  
*// Иначе сравниваем лексикографически*

if($f1<$f2) return -1; elseif($f1>$f2) return 1; else return 0;  
}  
*// Пусть $Files содержит массив с ключами — именами файлов  
// в текущем каталоге. Отсортируем его.*  
uksort($Files,"FCmp"); *// передаем функцию сортировки "по ссылке"*  
?>

Конечно, связи между ключами и значениями функцией **uksort()** сохраняются, т. е., опять же, некоторые пары просто «всплывают» наверх, а другие – «оседают».

### **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***.

Пример:

<?php  
*// Функция сравнения*  
function cmp($a, $b) {  
    if ($a == $b) {  
        return 0;  
    }  
    return ($a < $b) ? -1 : 1;  
}  
  
*// Сортируемый массив*  
$array = array('a' => 4, 'b' => 8, 'c' => -1, 'd' => -9, 'e' => 2, 'f' => 5, 'g' => 3, 'h' => -4);  
print\_r($array);  
  
*// Сортируем и выводим получившийся массив*  
uasort($array, 'cmp');  
print\_r($array);  
?>

Результат выполнения данного примера:

Array

(

[a] => 4

[b] => 8

[c] => -1

[d] => -9

[e] => 2

[f] => 5

[g] => 3

[h] => -4

)

Array

(

[d] => -9

[h] => -4

[c] => -1

[e] => 2

[g] => 3

[a] => 4

[f] => 5

[b] => 8

)

### **Переворачивание массива с помощью функции *array\_reverce()***

Функция **array\_reverse()** возвращает массив, элементы которого следуют в обратном порядке относительно массива, переданного в параметре. При этом связи между ключами и значениями, конечно, не теряются.

Первым параметром передается массив, а вторым – сохранять ключи при перестановке элементов или нет (***true*** – да, ***false*** – нет).

Второй параметр указывать необязательно. В таком случае по умолчанию вторым параметром является ***false***.

**array\_reverse**(массив, [сохранять ли ключи]);

Например, вместо того, чтобы ранжировать массив в обратном порядке при помощи **arsort()**, мы можем отсортировать его в прямом порядке, а затем перевернуть:

$A = array("a"=>"Zero","b"=>"Weapon","c"=>"Alpha","d"=>"Processor");  
asort($A);  
$A = array\_reverse($A);

<?php

$arr = [1, 2, 3, 4, 5];

$arr = array\_reverse($arr);

var\_dump($arr);

?>

*Результат выполнения кода:*

[5, 4, 3, 2, 1]

Конечно, указанная последовательность работает дольше, чем один-единственный вызов **arsort()**.

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

Эти две функции предназначены, в первую очередь, для сортировки списков.

Функция **sort()** сортирует список (разумеется, по значениям) в порядке возрастания, а **rsort()** – в порядке убывания. Пример для функции **sort()**:

<?php  
$A = array("40", "20", "10", "30");  
sort($A);  
for($i=0; $i<count($A); $i++) echo "$A[$i]"."<br>";  
*// выводит 10 20 30 40*  
?>

### **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. Пример:

$A=array(10,20,30,40,50);  
shuffle($A);  
foreach($A as $v) echo "$v ";

Приведенный фрагмент кода выводит числа **10, 20, 30, 40 и 50** в случайном порядке.

### **Операции с ключами и значениями массива**

**array\_flip(array *$arr*)**

Функция **array\_flip()** «пробегает» 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:

$A = array("a"=>"aaa", "b"=>"aaa", "c"=>"ccc");  
$A = array\_flip($A);  
*// теперь $A===array("aaa"=>"b", "ccc"=>"c");*

**array\_keys(array *$arr* [,mixed *$SearchVal*])**

Функция **array\_keys()** возвращает список, содержащий все ключи массива ***$arr***. Если задан необязательный параметр ***$SearchVal***, то она вернет только те ключи, которым соответствуют значения ***$SearchVal***.

Фактически, эта функция с заданным вторым параметром является обратной по отношению к оператору **[]** – извлечению значения по его ключу.

$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));

*Результатом выполнения вышеприведенной программы будет:*

Array

(

[0] => 0

[1] => color

)

Array

(

[0] => 0

[1] => 3

[2] => 4

)

Array

(

[0] => color

[1] => size

)

**array\_values(array *$arr*)**

Функция **array\_values()** возвращает список всех значений в ассоциативном массиве ***$arr***. Очевидно, такое действие бесполезно для списков, но иногда оправдано для хэшей.

<?php  
$array = array("size" => "XL", "color" => "gold");  
print\_r(array\_values($array));  
?>

*Результатом выполнения будет:*

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

**in\_array(mixed *$val*, array *$arr*)**

Функция **in\_array()** возвращает **true**, если элемент со значением ***$val*** присутствует в массиве ***$arr***.

Впрочем, если вам часто приходится проделывать эту операцию, подумайте: не лучше ли будет воспользоваться ассоциативным массивом и хранить данные в его ключах, а не в значениях? На этом вы можете сильно выиграть в быстродействии.

<?php  
$os = array("Mac", "NT", "Irix", "Linux");  
if (in\_array("Irix", $os)) {  
    echo "Нашел Irix";  
}  
if (in\_array("mac", $os)) {  
    echo "Нашел mac";  
}  
?>

*Второго совпадения не будет, потому что****in\_array()****регистрозависима, таким образом, программа выведет:*

Нашел Irix

**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***.

Пример:

$List=array(1, "hello", 1, "world", "hello");  
array\_count\_values($array);

*// возвращает array(1=>2, "hello"=>2, "world"=>1)*

### **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()**[.](http://www.php.su/learnphp/datatypes/?array3#array_merge) Слияние списков может осуществляться только с помощью функции **array\_merge()**.

Предположим, мы имеем два массива:

$A = array("1"=>"первый", "2"=>"второй");  
$B = array("3"=>"третий", "4"=>"четвертый");

Теперь сольем данные два массива в один массив **$C**:

$C = $A + $B;

Оператор "**+**" для массивов не коммутативен. Это означает, что **$A + $B** не равно **$B + $A**.

В результате рассмотренного примера мы получим массив **$C** следующего вида:

"1"=>"Первый", "2"=>"Второй", "3"=>"Третий", "4"=>"Четвертый"

А в результате **$B + $A** мы получим такой массив:

"3"=>"Третий", "4"=>"Четвертый", "1"=>"Первый", "2"=>"Второй"

При слиянии списков такой метод не работает. Поясним данный факт на примере:

Предположим, у нас есть два массива:

$A = array(10,11,12);   
$B = array(13,14,15);

В результате слияния списков **$A** и **$B** (**$A + $B**) мы получим: 10,11,12. А это совсем не тот результат, который мы хотели получить... Связано это с тем, что при слиянии списков с одинаковыми индексами в результирующем массиве остается элемент первого массива, причем на том же месте. В таком случае нам необходимо использовать функцию **array\_merge()**

**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 в массивах встречаются одинаковые ключи, в результат помещается пара ***ключ=>значение*** из того массива, который расположен правее в списке аргументов. Однако это не затрагивает числовые ключи: элементы с такими ключами помещаются в конец результирующего массива в любом случае.  
Таким образом, с помощью **array\_merge()** мы можем избавиться от всех недостатков оператора "**+**" для массивов. Вот пример, сливающий два списка в один:

$L1=array(100,200,300);  
$L2=array(400,500,600);  
$L=array\_merge($L1,$L2);

*// теперь $L===array(100,200,300,400,500,600);*

Всегда используйте эту функцию, если вам нужно работать именно со списками, а не с обычными ассоциативными массивами.

### **Получение части массива**

Для получения части массива можно использовать функцию **array\_slice().**

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

Эта функция возвращает часть ассоциативного 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.

Вот несколько примеров:

$input = array ("a", "b", "c", "d", "e");  
$output = array\_slice ($input, 2); *// "c", "d", "e"*  
$output = array\_slice ($input, 2, -1); *// "c", "d"*  
$output = array\_slice ($input, -2, 1); *// "d"*  
$output = array\_slice ($input, 0, 3); *// "a", "b", "c"*

### **Вставка и удаление элементов массивов**

Мы уже знаем несколько операторов, которые отвечают за вставку и удаление элементов. Например, оператор **[]** (пустые квадратные скобки) добавляет элемент в конец массива, присваивая ему числовой ключ, а оператор **Unset()** вместе с извлечением по ключу удаляет нужный элемент. Язык PHP поддерживает и многие другие функции, которые иногда бывает удобно использовать.

**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 стандарных **[]**. Если вам нужно добавить всего один элемент, наверное, проще и будет воспользоваться этим оператором:

array\_push($Arr,1000); *// вызываем функцию*  
$Arr[]=1000; *// то же самое, но короче*

<?php  
$stack = array("orange", "banana");  
array\_push($stack, "apple", "raspberry");  
print\_r($stack);  
?>

*В результате выполнения этого примера, $stack содержит следующие элементы:*

Array  
(  
    [0] => orange  
    [1] => banana  
    [2] => apple  
    [3] => raspberry  
)

Обратите внимание, что функция **array\_push()** воспринимает массив, как стек, и добавляет элементы всегда в его конец. Она возвращает новое число элементов в массиве.

**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 ***$Arr***. С помощью этой функции мы можем строить конструкции, напоминающие стек. Если список ***$Arr*** был пуст, функция возвращает пустую строку.

<?php  
$stack = array("orange", "banana", "apple", "raspberry");  
$fruit = array\_pop($stack);  
print\_r($stack);  
?>

*После этого, в $stack будет только 3 элемента:*

Array

(

[0] => orange

[1] => banana

[2] => apple

)

*и raspberry будет присвоено переменной $fruit.*

**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. Вот пример ее применения:

$A=array(10,"a"=>20,30);  
array\_unshift($A,"!","?");

*// теперь $A===array(0=>"!", 1=>"?", 2=>10, a=>20, 3=>30)*

**mixed array\_shift(list &*$Arr*)**

Функция **mixed 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  
$stack = array("orange", "banana", "apple", "raspberry");  
$fruit = array\_shift($stack);  
print\_r($stack);  
?>

*В результате в массиве $stack останется 3 элемента:*

Array

(

[0] => banana

[1] => apple

[2] => raspberry

)

**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***:

$input=array("a" => "green", "red", "b" => "green", "blue", "red");  
$result=array\_unique($input);

*// теперь $result===array("a"=>"green", "red", "blue");*

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

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 также, как и в функции [**substr()**](http://www.php.su/functions/?substr) – а именно, они могут быть и отрицательными, в этом случае отсчет начинается от конца массива. Вот некоторые примеры:

<?php  
$input=array("red", "green", "blue", "yellow");  
array\_splice($input,2);  
*// Теперь $input===array("red", "green")*  
array\_splice($input,1,-1);  
*// Теперь $input===array("red", "yellow")*array\_splice($input, -1, 1, array("black", "maroon"));  
*// Теперь $input===array("red", "green", "blue", "black", "maroon")*array\_splice($input, 1, count($input), "orange");  
*// Теперь $input===array("red", "orange")*  
?>

Последний пример показывает, что в качестве параметра **$Repl** мы можем указать и обычное, строковое значение, а не массив из одного элемента.

## **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. Вот пример использования этой функции:

$a = "Test string";  
$b = "Some text";  
$A = compact("a","b");

*// теперь $A===array("a"=>"Test string", "b"=>"Some text")*

Почему же тогда параметры функции обозначены как **mixed**? Дело в том, что они могут быть не только строками, но и списками строк. В этом случае функция последовательно перебирает все элементы этого списка, и упаковывает те переменные из текущего контекста, имена которых она встретила. Более того, эти списки могут, в свою очередь, также содержать списки строк, и т. д. Правда, последнее используется сравнительно редко, но все же вот пример:

$a = "Test";  
$b = "Text";  
$c = "CCC";  
$d = "DDD";  
$Lst = array("b",array("c","d"));  
$A = compact("a",$Lst);

*// теперь $A===array("a"=>"Test", "b"=>"Text", "c"=>"CCC", "d"=>"DDD")*

**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.

## **Создание списка – диапазона чисел**

**array range (**[**mixed**](https://php.ru/manual/language.pseudo-types.html#language.types.mixed)***$low*,**[**mixed**](https://php.ru/manual/language.pseudo-types.html#language.types.mixed)***$hig* [,**[**number**](https://php.ru/manual/language.pseudo-types.html#language.types.number)***$step* = 1 ] )**

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

Если указан параметр ***$step***, то он будет использоваться как инкремент между элементами последовательности. ***$step*** должен быть положительным числом. Если ***$step*** не указан, он принимает значение по умолчанию 1.

<?php  
*// array(0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12)*  
foreach (range(0, 12) as $number) {  
    echo $number;  
}  
  
*// Параметр step  
// array(0, 10, 20, 30, 40, 50, 60, 70, 80, 90, 100)*  
foreach (range(0, 100, 10) as $number) {  
    echo $number;  
}  
  
*// Использование последовательности знаков  
// array('a', 'b', 'c', 'd', 'e', 'f', 'g', 'h', 'i');*foreach (range('a', 'i') as $letter) {  
    echo $letter;  
}

*// array('c', 'b', 'a');*  
foreach (range('c', 'a') as $letter) {  
    echo $letter;  
}  
?>

## **Счетчик элементов массива**

Для подсчета элементов массива предназначена функция [**count()**](http://www.php.su/functions/?count).

Пример использования функции **count()**:

<?php  
$arr[]=5;  
$arr[]=4;  
$arr[]=8;  
$arr[]=3;  
$arr[]=8;  
echo "<h2>Число элементов массива: ".count($arr)."</h2>";  
*// Выводит: Число элементов массива: 5*  
?>

## **Удаление массива и его элементов**

Если вы хотите удалить массив целиком, воспользуйтесь функцией [**unset()**](http://www.php.su/functions/?unset).

Если вы хотите удалить пару ***ключ/значение***, вы также можете использовать функцию **unset()**. Приведем конкретные примеры:

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

## **Некоторые особенности работы с массивами**

### **Преобразование в массив (тип array)**

Для любого из типов: **integer**, **float**, **string**, **boolean** и **resource**, если вы преобразуете значение в массив, вы получите массив с одним элементом (с индексом 0), являющимся скалярным значением, с которого вы начали.

Если вы преобразуете в массив объект ([**object**](http://www.php.su/learnphp/datatypes/?object)), вы получите в качестве элементов массива свойства (переменные-члены) этого объекта. Ключами будут имена переменных-членов.

Если вы преобразуете в массив значение [**NULL**](http://www.php.su/learnphp/datatypes/?null), вы получите пустой массив.

## **Сравнение массивов**

Массивы можно сравнивать при помощи функции **array\_diff()** и **операторов массивов:**

**Операторы, работающие с массивами:**

| **Пример** | **Название** | **Результат** |
| --- | --- | --- |
| $a + $b | Объединение | Объединение массива $a и массива $b. |
| $a == $b | Равно | **TRUE**в случае, если $a и $b содержат одни и те же элементы. |
| $a === $b | Тождественно равно | **TRUE** в случае, если $a и $b содержат одни и те же элементы в том же самом порядке. |
| $a != $b | Не равно | **TRUE** если массив $a не равен массиву $b. |
| $a <> $b | Не равно | **TRUE** если массив $a не равен массиву $b. |
| $a !== $b | Тождественно не равно | **TRUE** если массив $a не равен тождественно массиву $b. |

Пример сравнения массивов:

<?php  
$a = array("apple", "banana");  
$b = array(1 => "banana", "0" => "apple");  
  
var\_dump($a == $b); *// bool(true)*  
var\_dump($a === $b); *// bool(false)*  
?>

## **Некоторые полезные практические примеры по работе с массивами**

<?php  
*// это*  
$a = array( 'color' => 'red',  
             'taste' => 'sweet',  
             'shape' => 'round',  
             'name'  => 'apple',  
                        4        *// ключом будет 0*           );  
  
*// полностью соответствует*  
$a['color'] = 'red';  
$a['taste'] = 'sweet';  
$a['shape'] = 'round';  
$a['name']  = 'apple';  
$a[]        = 4;        *// ключом будет 0*  
  
$b[] = 'a';  
$b[] = 'b';  
$b[] = 'c';  
*// создаст массив array(0 => 'a' , 1 => 'b' , 2 => 'c'),  
// или просто array('a', 'b', 'c')*  
?>

Еще один практический пример:

<?php  
*// Массив как карта (свойств)*$map = array( 'version'    => 4,  
               'OS'         => 'Linux',  
               'lang'       => 'english',  
               'short\_tags' => true  
             );  
               
*// исключительно числовые ключи*  
$array = array( 7,  
                 8,  
                 0,  
                 156,  
                 -10  
               );  
*// это то же самое, что и array(0 => 7, 1 => 8, ...)*  
  
$switching = array(         10, *// ключ = 0*  
                     5    =>  6,  
                     3    =>  7,   
                     'a'  =>  4,  
                           11, *// ключ = 6 (максимальным числовым индексом был 5)*  
                     '8'  =>  2, *// ключ = 8 (число!)*  
                     '02' => 77, *// ключ = '02'*  
                     0    => 12  *// значение 10 будет перезаписано на 12*  
                   );  
                     
*// пустой массив*  
$empty = array();  
?>

Коллекция:

<?php  
$colors = array('красный', 'синий', 'зеленый', 'желтый');  
  
foreach ($colors as $color) {  
     echo "Вам нравится $color?\n";  
}  
?>

Результат работы рассмотренного скрипта:

*Вам нравится красный?*

*Вам нравится синий?*

*Вам нравится зеленый?*

*Вам нравится желтый?*

Следующий пример создает начинающийся с единицы массив:

<?php  
$firstquarter  = array(1 => 'Январь', 'Февраль', 'Март');  
print\_r($firstquarter);  
?>

Результат работы приведенного скрипта будет следующий:

*Array*

*(*

*[1] => 'Январь'*

*[2] => 'Февраль'*

*[3] => 'Март'*

*)*

Пример заполнения массива:

<?php  
*// заполняет массив всеми элементами директории*  
$handle = opendir('.');  
while (false !== ($file = readdir($handle))) {  
     $files[] = $file;  
}  
closedir($handle);  
?>

Массивы упорядочены. Вы можете изменять порядок элементов, используя различные функции сортировки. Вы можете подсчитать количество элементов в массиве, используя функцию **count()**.

Рекурсивные и многомерные массивы:

<?php  
$fruits = array ( "фрукты" => array ( "a" => "апельсин",  
                                       "b" => "банан",  
                                       "c" => "яблоко"  
                                     ),  
                   "числа"  => array ( 1,  
                                       2,  
                                       3,  
                                       4,  
                                       5,  
                                       6  
                                     ),  
                   "дырки"  => array (      "первая",  
                                       5 => "вторая",  
                                            "третья"  
                                     )  
                 );  
  
*// Несколько примеров доступа к значениям предыдущего массива*  
echo $fruits["дырки"][5];    *// напечатает "вторая"*echo $fruits["фрукты"]["a"]; *// напечатает "апельсин"*  
unset($fruits["дырки"][0]);  *// удалит "первая"*  
  
*// Создаст новый многомерный массив*  
$juices["яблоко"]["зеленое"] = "хорошее";  
?>

Обратите внимание, что при присваивании массива всегда происходит копирование значения. Чтобы копировать массив по ссылке, вам нужно использовать оператор ссылки:

<?php  
$arr1 = array(2, 3);  
$arr2 = $arr1;  
$arr2[] = 4; *// $arr2 изменился,*  
             *// $arr1 по прежнему array(2,3)*  
                
$arr3 = &$arr1;  
$arr3[] = 4; *// теперь $arr1 и $arr3 эквивалентны*  
?>