



- [Downloads](#)
- [Documentation](#)
- [Get Involved](#)
- [Help](#)
- 

[Dutch PHP Conference 2024](#)

## [Getting Started](#)

[Introduction](#)

[A simple tutorial](#)

## [Language Reference](#)

[Basic syntax](#)

[Types](#)

[Variables](#)

[Constants](#)

[Expressions](#)

[Operators](#)

[Control Structures](#)

[Functions](#)

[Classes and Objects](#)

[Namespaces](#)

[Enumerations](#)

[Errors](#)

[Exceptions](#)

[Fibers](#)

[Generators](#)

[Attributes](#)

[References Explained](#)

[Predefined Variables](#)

[Predefined Exceptions](#)

[Predefined Interfaces and Classes](#)

[Predefined Attributes](#)

[Context options and parameters](#)

[Supported Protocols and Wrappers](#)

## [Security](#)

[Introduction](#)

[General considerations](#)

[Installed as CGI binary](#)

[Installed as an Apache module](#)

[Session Security](#)

[Filesystem Security](#)

[Database Security](#)

[Error Reporting](#)

[User Submitted Data](#)

[Hiding PHP](#)

[Keeping Current](#)

## [Features](#)

[HTTP authentication with PHP](#)

[Cookies](#)

[Sessions](#)

[Dealing with XForms](#)

[Handling file uploads](#)

[Using remote files](#)

[Connection handling](#)

[Persistent Database Connections](#)

[Command line usage](#)

[Garbage Collection](#)  
[DTrace Dynamic Tracing](#)

## [Function Reference](#)

[Affecting PHP's Behaviour](#)  
[Audio Formats Manipulation](#)  
[Authentication Services](#)  
[Command Line Specific Extensions](#)  
[Compression and Archive Extensions](#)  
[Cryptography Extensions](#)  
[Database Extensions](#)  
[Date and Time Related Extensions](#)  
[File System Related Extensions](#)  
[Human Language and Character Encoding Support](#)  
[Image Processing and Generation](#)  
[Mail Related Extensions](#)  
[Mathematical Extensions](#)  
[Non-Text MIME Output](#)  
[Process Control Extensions](#)  
[Other Basic Extensions](#)  
[Other Services](#)  
[Search Engine Extensions](#)  
[Server Specific Extensions](#)  
[Session Extensions](#)  
[Text Processing](#)  
[Variable and Type Related Extensions](#)  
[Web Services](#)  
[Windows Only Extensions](#)  
[XML Manipulation](#)  
[GUI Extensions](#)

## Keyboard Shortcuts

?	This help
j	Next menu item
k	Previous menu item
g p	Previous man page
g n	Next man page
G	Scroll to bottom
g g	Scroll to top
g h	Goto homepage
g s	Goto search (current page)
/	Focus search box

[arsort »](#)  
[« array\\_walk](#)

- [Руководство по PHP](#)
- [Справочник функций](#)
- [Модули, относящиеся к переменным и типам](#)
- [Массивы](#)
- [Функции для работы с массивами](#)

# array

(PHP 4, PHP 5, PHP 7, PHP 8)

array — Создаёт массив

## Описание

**array**([mixed](#) ...\$values): array

Создаёт массив. Подробнее о массивах рассказано в разделе «[Массивы](#)».

## Список параметров

values

Синтаксис — разделённые запятыми «индекс => значения», определяет индексы и значения. Индекс разрешено указывать строкой или целым числом. Если индекс не указан, будет автоматически сгенерирован целочисленный индекс начиная с 0. Если индекс — целое число, следующим сгенерированным индексом будет — наибольший целочисленный индекс + 1. Обратите внимание, что если определены два одинаковых индекса, следующий перезапишет предыдущий.

Замыкающая (висящая) запятая после последнего элемента массива хотя и необычна, но не нарушает синтаксиса.

## Возвращаемые значения

Возвращает массив параметров. Задать параметрам индекс можно оператором =>. Подробнее о массивах рассказано в разделе «[Массивы](#)».

## Примеры

Следующие примеры показывают, как создавать двумерный массив, как определять ключи ассоциативных массивов и как пропустить и продолжить числовые индексы в обычных массивах, если нумерация начинается с произвольного числа.

### Пример #1 Пример использования языковой конструкции array()

```
<?php

$fruits = array (
    "fruits" => array("a" => "апельсин", "b" => "банан", "c" => "яблоко"),
    "numbers" => array(1, 2, 3, 4, 5, 6),
    "holes" => array("первый", 5 => "второй", "третий")
);
?>
```

### Пример #2 Автоматическая индексация при вызове конструкции array()

```
<?php

$array = array(1, 1, 1, 1, 1, 8 => 1, 4 => 1, 19, 3 => 13);
print_r($array);
?>
```

Результат выполнения приведённого примера:

```
Array
(
    [0] => 1
    [1] => 1
    [2] => 1
```

```
[3] => 13
[4] => 1
[8] => 1
[9] => 19
)
```

Обратите внимание, что индекс «3» определён дважды, и содержит последнее значение — 13. Индекс 4 определён после индекса 8, а следующий сгенерированный индекс (значение 19) — 9, поскольку самым большим индексом был 8.

Этот пример создаёт массив, нумерация которого начинается с 1.

### Пример #3 Пример создания конструкцией `array()` массива, нумерация которого начинается с 1

```
<?php
```

```
$firstquarter = array(1 => 'January', 'February', 'March');
print_r($firstquarter);
?>
```

Результат выполнения приведённого примера:

```
Array
(
    [1] => January
    [2] => February
    [3] => March
)
```

Как и в Perl, разработчику доступны значения массива внутри двойных кавычек. Однако в PHP нужно заключить массив в фигурные скобки.

### Пример #4 Доступ к массиву внутри двойных кавычек

```
<?php
```

```
$foo = array('bar' => 'baz');
echo "Hello {$foo['bar']}!"; // Hello baz!

?>
```

## Примечания

#### Замечание:

**`array()`** — конструкция языка для представления литеральных массивов, а не обычная функция.

## Смотрите также

- [array\\_pad\(\)](#) - Дополнить массив определённым значением до указанной длины
- [list\(\)](#) - Присваивает переменным значения похожим на массивы синтаксисом
- [count\(\)](#) - Подсчитывает элементы в массиве или в объекте Countable
- [range\(\)](#) - Создаёт массив, содержащий диапазон элементов
- [foreach](#)
- Тип [массив](#)

[+add a note](#)

## User Contributed Notes 38 notes

[up](#)

[down](#)

117

[ole dot aanensen at gmail dot com ¶](#)

9 years ago

As of PHP 5.4.x you can now use 'short syntax arrays' which eliminates the need of this function.

Example #1 'short syntax array'

```
<?php
$a = [1, 2, 3, 4];
print_r($a);
?>
```

The above example will output:

```
Array
(
    [0] => 1
    [1] => 2
    [2] => 3
    [3] => 4
)
```

Example #2 'short syntax associative array'

```
<?php
$a = ['one' => 1, 'two' => 2, 'three' => 3, 'four' => 4];
print_r($a);
?>
```

The above example will output:

```
Array
(
    [one] => 1
    [two] => 2
    [three] => 3
    [four] => 4
)
```

[up](#)

[down](#)

8

[jonberglund at gmail dot com ¶](#)

**15 years ago**

The following function (similar to one above) will render an array as a series of HTML select options (i.e. "<option>...</option>"). The problem with the one before is that there was no way to handle <optgroup>, so this function solves that issue.

```
function arrayToSelect($option, $selected = '', $optgroup = NULL)
{
    $returnStatement = '';

    if ($selected == '') {
        $returnStatement .= '<option value="" selected="selected">Select one...</option>';
    }

    if (isset($optgroup)) {
        foreach ($optgroup as $optgroupKey => $optgroupValue) {
            $returnStatement .= '<optgroup label="' . $optgroupValue . '">';

            foreach ($option[$optgroupKey] as $optionKey => $optionValue) {
                if ($optionKey == $selected) {
                    $returnStatement .= '<option selected="selected" value="' . $optionKey . '">' . $optionValue . '</option>';
                } else {
                    $returnStatement .= '<option value="' . $optionKey . '">' . $optionValue . '</option>';
                }
            }

            $returnStatement .= '</optgroup>';
        }
    } else {
        foreach ($option as $key => $value) {
            if ($key == $selected) {
```

```

$returnStatement .= '<option selected="selected" value="' . $key . '">' . $value . '</option>';
} else {
$returnStatement .= '<option value="' . $key . '">' . $value . '</option>';
}
}
}

return $returnStatement;
}

```

So, for example, I needed to render a list of states/provinces for various countries in a select field, and I wanted to use each country name as an <optgroup> label. So, with this function, if only a single array is passed to the function (i.e. "arrayToSelect(\$stateList)") then it will simply spit out a bunch of "<option>" elements. On the other hand, if two arrays are passed to it, the second array becomes a "key" for translating the first array.

Here's a further example:

```

$countryList = array(
'CA' => 'Canada',
'US' => 'United States');

$stateList['CA'] = array(
'AB' => 'Alberta',
'BC' => 'British Columbia',
'AB' => 'Alberta',
'BC' => 'British Columbia',
'MB' => 'Manitoba',
'NB' => 'New Brunswick',
'NL' => 'Newfoundland/Labrador',
'NS' => 'Nova Scotia',
'NT' => 'Northwest Territories',
'NU' => 'Nunavut',
'ON' => 'Ontario',
'PE' => 'Prince Edward Island',
'QC' => 'Quebec',
'SK' => 'Saskatchewan',
'YT' => 'Yukon');

$stateList['US'] = array(
'AL' => 'Alabama',
'AK' => 'Alaska',
'AZ' => 'Arizona',
'AR' => 'Arkansas',
'CA' => 'California',
'CO' => 'Colorado',
'CT' => 'Connecticut',
'DE' => 'Delaware',
'DC' => 'District of Columbia',
'FL' => 'Florida',
'GA' => 'Georgia',
'HI' => 'Hawaii',
>ID' => 'Idaho',
'IL' => 'Illinois',
'IN' => 'Indiana',
'IA' => 'Iowa',
'KS' => 'Kansas',
'KY' => 'Kentucky',
'LA' => 'Louisiana',
'ME' => 'Maine',
'MD' => 'Maryland',
'MA' => 'Massachusetts',
'MI' => 'Michigan',

```

```
'MN' => 'Minnesota',
'MS' => 'Mississippi',
'MO' => 'Missouri',
'MT' => 'Montana',
'NE' => 'Nebraska',
'NV' => 'Nevada',
'NH' => 'New Hampshire',
'NJ' => 'New Jersey',
'NM' => 'New Mexico',
'NY' => 'New York',
'NC' => 'North Carolina',
'ND' => 'North Dakota',
'OH' => 'Ohio',
'OK' => 'Oklahoma',
'OR' => 'Oregon',
'PA' => 'Pennsylvania',
'RI' => 'Rhode Island',
'SC' => 'South Carolina',
'SD' => 'South Dakota',
'TN' => 'Tennessee',
'TX' => 'Texas',
'UT' => 'Utah',
'VT' => 'Vermont',
'VA' => 'Virginia',
'WA' => 'Washington',
'WV' => 'West Virginia',
'WI' => 'Wisconsin',
'WY' => 'Wyoming');
```

...

```
<select name="state" id="state"><?php echo arrayToSelect($stateList,'',$countryList) ?></select>
<select name="country" id="country"><?php echo arrayToSelect($countryList,'US') ?></select>
```

[up](#)

[down](#)

0

[Marcel G](#)

14 years ago

I encountered the following but didn't see it documented/reported anywhere so I'll just mentioned it here.

As written in the manual:

"When index is omitted, an integer index is automatically generated, starting at 0. If index is an integer, next generated index will be the biggest integer index + 1".

This generated index has always the largest integer used as a key so far. So adding \$a[5] = 'foo'; after an \$a[10] = 'bar'; will not force the next generated index to be 6 but to be 11 as 10 was the highest index encountered until here.

Anyway, the following can happen:

```
<?php
$max_int = 2147483647; // Max value for integer on a 32-bit system
$arr = array();

$arr[1] = 'foo'; // New generated index will be 2
$arr[ $max_int ] = 'bar'; // Caution: Next generated index will be -2147483648 due to the integer overflow!
$arr[0] = 'bar'; // The highest value should be 2147483648 but due to the i-overflow it is -2147483648 so current index 0
is larger. The new generated index therefore is 1!
$arr[] = 'failure'; // Warning: Cannot add element to the array as the next element is already occupied.
?>
```

[up](#)

[down](#)

0

[jon hohle](#)

## 16 years ago

Here are shorter versions of sam barrow's functions. From a PHP perspective these are O(1) (without getting into what's going on in the interpreter), instead of O(n).

```
<?php
function randomKey(array $array)
{
    return randomElement(array_keys($array));
}

function randomElement(array $array)
{
    if (count($array) === 0)
    {
        trigger_error('Array is empty.', E_USER_WARNING);
        return null;
    }

    $rand = mt_rand(0, count($array) - 1);
    $array_keys = array_keys($array);

    return $array[$array_keys[$rand]];
}
?>
```

[up](#)

[down](#)

0

[jupiter at nospam dot com ¶](#)

## 17 years ago

```
<?php

// changes any combination of multiarray elements and subarrays
// into a consistent 2nd level multiarray, tries to preserves keys
function changeMultiarrayStructure($multiarray, $asc = 1) {
    if ($asc == 1) { // use first subarrays for new keys of arrays
        $multiarraykeys = array_reverse($multiarray, true);
    } else { // use the last array keys
        $multiarraykeys = $multiarray; // use last subarray keys
    } // end array reordering
    $newarraykeys = array(); // establish array
    foreach ($multiarraykeys as $arrayvalue) { // build new array keys
        if (is_array($arrayvalue)) { // is subarray an array
            $newarraykeys = array_keys($arrayvalue) + $newarraykeys;
        } // if count(prevsubarray)>count(currentarray), extras survive
    } // end key building loop
    foreach ($multiarray as $newsubarraykey => $arrayvalue) {
        if (is_array($arrayvalue)) { // multiarray element is an array
            $i = 0; // start counter for subarray key
            foreach ($arrayvalue as $subarrayvalue) { // access subarray
                $newmultiarray[$newarraykeys[$i]][$newsubarraykey] = $subarrayvalue;
                $i++; // increase counter
            } // end subarray loop
        } else { // multiarray element is a value
            foreach ($newarraykeys as $newarraykey) { // new subarray keys
                $newmultiarray[$newarraykey][$newsubarraykey] = $arrayvalue;
            } // end loop for array variables
        } // end conditional
    } // end new multiarray building loop
    return $newmultiarray;
}

// will change
```



```
$old = array('a'=>1,'b'=>array('e'=>2,'f'=>3),'c'=>array('g'=>4),'d'=>5);
// to
$new = array('e'=>array('a'=>1,'b'=>2,'c'=>4,'d'=>5),
'f'=>array('a'=>1,'b'=>3,'d'=>5));

// note: if $asc parameter isn't default, last subarray keys used

?>
```

The new key/value assignment pattern is clearer with bigger arrays.  
I use this to manipulate input/output data from my db. Enjoy.

[up](#)  
[down](#)

-2

[Mike Mackintosh ¶](#)

**16 years ago**

If you need to add an object as an array key, for example an object from Simple XML Parser, you can use the following.

```
File.XML
<?xml version="1.0" encoding="UTF-8"?>
<Settings type="General">
<Setting name="SettingName">This This This</Setting>
</Settings>
```

The script:

```
<?php
$raw = $xml = new SimpleXMLElement('File.XML');
foreach($raw->Setting as $A => $B)
{
// Set Array From XML
$Setting[(string) $B['name']] = (string) $B[0];
}
?>
```

By telling the key to read the object as a string, it will let you set it.

Hope this helps someone out!

[up](#)  
[down](#)

-2

[brian at blueeye dot us ¶](#)

**18 years ago**

If you need, for some reason, to create variable Multi-Dimensional Arrays, here's a quick function that will allow you to have any number of sub elements without knowing how many elements there will be ahead of time. Note that this will overwrite an existing array value of the same path.

```
<?php
// set_element(array path, mixed value)
function set_element(&$path, $data) {
return ($key = array_pop($path)) ? set_element($path, array($key=>$data)) : $data;
}
?>
```

For example:

```
<?php
echo "<pre>";
$path = array('base', 'category', 'subcategory', 'item');
$array = set_element($path, 'item_value');
print_r($array);
echo "</pre>";
?>
```

Will display:

Array

```
(
[base] => Array
(
[category] => Array
(
[subcategory] => Array
(
[item] => item_value
)
)
)
)
```

[up](#)

[down](#)

-2

[razvan\\_bc at yahoo dot com ¶](#)

**3 years ago**

an array can execute code.

i try this on php 7.4 (xampp version)

```
<?php
```

```
[
$msg='HI GUYS !'
,print ($msg)
,(function()use ($msg){
print 'here is php';
})();
,print ' I`m just an array ,alive one'
];
```

```
?>
```

[up](#)

[down](#)

-3

[r0h4rd at gmail dot com ¶](#)

**13 years ago**

"Quick and dirty" class to get an offset of multidimensional array by given path (sorry, that without comments).

```
<?php
```

```
class ArrayAsPathException extends Exception {}
```

```
class ArrayAsPath {
protected
$data = array(),
$separator = '.';
```

```
public function __construct (array $data = array()) {
$this->data = $data;
}
```

```
public function set ($value, $path = null) {
if (!isset($path)) {
$this->data = $value;
}
```

```
$separator = $this->separator;
$pathtoken = strtok($path, $separator);
```

```

$code = '';
$pices = '['.$pathtoken.'\']';
while ($pathtoken !== false) {
    if (($pathtoken = strtok($separator)) !== false) {
        $code .= 'if (!isset($this->data'.$pices.')) $this->data'.$pices.' = array(); ';
        $pices .= '['.$pathtoken.'\']';
    } else {
        $code .= 'return $this->data'.$pices.' = $value;';
    }
}

return eval($code);
}

public function get ($path = '', $default = null) {
    $result = $this->data;
    $separator = $this->separator;
    $pathtoken = strtok($path, $separator);

    while ($pathtoken !== false) {
        if (!isset($result[$pathtoken]) || is_string($result)) {
            if (isset($default)) {
                return $default;
            }

            throw new ArrayAsPathException ('Can\'t find "'.$pathtoken.'" in "'.$path.'"');
        }

        $result = $result[$pathtoken];
        $pathtoken = strtok($separator);
    }

    return $result ? $result : $default;
}

public function has ($path) {
    $result = $this->data;
    $separator = $this->separator;
    $pathtoken = strtok($path, $separator);

    while ($pathtoken !== false) {
        if (!isset($result[$pathtoken]) || is_string($result)) {
            return false;
        }

        $result = $result[$pathtoken];
        $pathtoken = strtok($separator);
    }

    return true;
}

public function setSeparator ($separator) {
    $this->separator = $separator;
}

public function getSeparator ($separator) {
    return $this->separator;
}
}
?>

```

Code:

```
<?php
```

```
$params = new ArrayAsPath;
$params->set(array(
    'foo' => array(
        'bar' => array(
            'item' => 'Value'
        )
    )
));

try {
$params->set('test', 'foo.bar.far.new');
printf(
    'Array:<pre>%s</pre>
foo.bar.item:<pre>%s</pre>
foo.bar.far:<pre>%s</pre>
foo.bar.far.new<pre>%s</pre>',
    var_export($params->get(), true),
    var_export($params->get('foo.bar.item'), true),
    var_export($params->get('foo.bar.far'), true),
    var_export($params->get('foo.bar.far.new'), true)
);
} catch (ArrayAsPathException $e) {
echo 'Oops! It seems that something is wrong. '.$e->getMessage();
}

?>
```

Will display:

```
Array:
array (
    'foo' =>
        array (
            'bar' =>
                array (
                    'item' => 'Value',
                    'far' =>
                        array (
                            'new' => 'test',
                        ),
                    ),
                ),
            ),
        )
```

```
foo.bar.item:
'Value'
```

```
foo.bar.far:
array (
    'new' => 'test',
)
```

```
foo.bar.far.new:
'test'
```

[up](#)

[down](#)

-3

[slicky at newshelix dot com ¶](#)

22 years ago

Notice that you can also add arrays to other arrays with the `$array[]` "operator" while the dimension doesn't matter.

Here's an example:

```
$x[w][x] = $y[y][z];
```

this will give you a 4dimensional associative array.

```
$x[][] = $y[][];
```

this will give you a 4dimensional non associative array.

So let me come to the point. This get interessting for shortening things up. For instance:

```
<?php
foreach ($lines as $line){
if(!trim($line)) continue;
$tds[] = explode("$delimiter",$line);
}
?>
```

[up](#)

[down](#)

-2

[jjm152 at hotmail dot com ¶](#)

**21 years ago**

The easiest way to "list" the values of either a normal 1 list array or a multi dimensional array is to use a `foreach()` clause.

Example for 1 dim array:

```
<?php
$arr = array( 1, 2, 3, 4, 5 );
foreach ( $arr as $val ) {
echo "Value: $Val\n";
}
?>
```

For multi dim array:

```
<?php
$arr = array( 1 => 'one', 2 => 'two', 3 => 'three', 4 => 'four', 5 => 'five');
foreach ( $arr as $key => $value ) {
echo "Key: $key, Value: $value\n";
}
?>
```

This is quite possibly the easiest way i've found to iterate through an array.

[up](#)

[down](#)

-5

[majkel ¶](#)

**8 years ago**

During initialization php parses every value as expression. One can declare variables, do calculations, or even manipulate "current" array

```
<?php

$arr['A'] = [
    &$arr, // circular reference
    'key' => 'val',
    $arr['B'] = [ // declare array, insert key and then value
        'a' => 'b',
    ],
    ucfirst(strtolower('SOME TEXT')),
    true ? 'TRUE' : 'FALSE', // evaluate expression
    $x = 3, // declare variable
```

```
$x *= 3, // perform calculations  
];
```

```
var_dump($arr);
```

```
// Output  
/*  
array(2) {  
  'B' => array(1) {  
    'a' => string(1) "b"  
  }  
  'A' => array(7) {  
    [0] => array(2) {  
      'B' => array(1) {  
        ...  
      }  
      'A' => &array  
    }  
    'key' => string(3) "val"  
    [1] => array(1) {  
      'a' => string(1) "b"  
    }  
    [2] => string(9) "Some text"  
    [3] => string(4) "TRUE"  
    [4] => int(3)  
    [5] => int(9)  
  }  
}  
*/  
?>
```

[up](#)

[down](#)

-4

[rubein at earthlink dot net ¶](#)

**23 years ago**

Multidimensional arrays are actually single-dimensional arrays nested inside other single-dimensional arrays.

`$array[0]` refers to element 0 of `$array`

`$array[0][2]` refers to element 2 of element 0 of `$array`.

If an array was initialized like this:

```
$array[0] = "foo";  
$array[1][0] = "bar";  
$array[1][1] = "baz";  
$array[1][2] = "bam";
```

then:

```
is_array($array) = TRUE  
is_array($array[0]) = FALSE  
is_array($array[1]) = TRUE  
count($array) = 2 (elements 0 and 1)  
count($array[1]) = 3 (elements 0 thru 2)
```

This can be really useful if you want to return a list of arrays that were stored in a file or something:

```
$array[0] = unserialize($somedata);  
$array[1] = unserialize($someotherdata);
```

if `$somedata["foo"] = 42` before it was serialized previously, you'd now have this:

```
$array[0]["foo"] = 42
```

[up](#)

[down](#)

-4

[php ¶](#)

**18 years ago**

This function converts chunks of a string in an array:

```
function array_str($str, $len) {
    $newstr = '';
    for($i = 0; $i < strlen($str); $i++) {
        $newstr .= substr($str, $i, $len);
    }
    return $newstr;
}
```

use it as:

```
$str = "abcdefghilmn";
echo "<table width=\"100%\">\n";
foreach(array_str($str, 4) as $chunk) {
    echo "<tr><td>".$chunk."</td></tr>\n";
}
echo "</table>";
```

this prints:

```
-----
abcd
-----
efgh
-----
ilmn
-----
```

It don't use regular expressions. Please add this function to php :)

[up](#)

[down](#)

-4

[bill at carneyco dot com ¶](#)

**18 years ago**

I wanted to be able to control the flow of data in a loop instead of just building tables with it or having to write 500 select statements for single line items. This is what I came up with thanks to the help of my PHP brother in FL. Hope someone else gets some use out it.

```
<?

//set array variable
$results = array();

//talk to the db
$query = "SELECT * FROM yourtable";
$result = mysql_query($query) or die(mysql_error());

//count the rows and fields
$totalRows = mysql_num_rows($result);
$totalFields = mysql_num_fields($result);

//start the loop
for ( $i = 0; $i < $totalRows; ++$i ) {

//make it 2 dim in case you change your order
$results[$i] = mysql_fetch_array($result);

//call data at will controlling the loop with the array
```

```
echo $results[your_row_id]['your_field_name']; }
```

```
//print the entire array to see what lives where  
print_r($results); ?>
```

[up](#)

[down](#)

-3

[Anonymous ¶](#)

**20 years ago**

Similarly to a comment by stlawson at sbcglobal dot net on this page:

<http://www.php.net/basic-syntax.instruction-separation>

It is usually advisable to define your arrays like this:

```
$array = array(  
'foo',  
'bar',  
);
```

Note the comma after the last element - this is perfectly legal. Moreover, it's best to add that last comma so that when you add new elements to the array, you don't have to worry about adding a comma after what used to be the last element.

```
<?php  
$array = array(  
'foo',  
'bar',  
'baz',  
);  
?>
```

[up](#)

[down](#)

-8

[aissatya at yahoo dot com ¶](#)

**18 years ago**

```
<?php
```

```
$foo = array('bar' => 'baz');  
echo "Hello {$foo['bar']}!"; // Hello baz!
```

```
?>  
  
<?php  
$firstquarter = array(1 => 'January', 'February', 'March');  
print_r($firstquarter);  
?>
```

```
<?php  
$fruits = array (  
"fruits" => array("a" => "orange", "b" => "banana", "c" => "apple"),  
"numbers" => array(1, 2, 3, 4, 5, 6),  
"holes" => array("first", 5 => "second", "third")  
);  
?>
```

[up](#)

[down](#)

-7

[mortoray at ecircle-ag dot com ¶](#)

**18 years ago**

Be careful if you need to use mixed types with a key of 0 in an array, as several distinct forms end up being the same key:

```
$a = array();  
$a[null] = 1;
```



```
$a[0] = 2;
$a['0'] = 3;
$a["0"] = 4;
$a[false] = 5;
$a[0.0] = 6;
$a[''] = 7;
$a[] = 8;
```

```
print_r( $a );
```

This will print out only 3 values: 6, 7, 8.

[up](#)

[down](#)

-5

[\*rdude at fuzzelish dot com ¶\*](#)

**18 years ago**

If you are creating an array with a large number of static items, you will find serious performance differences between using the array() function and the \$array[] construct. For example:

```
<?
```

```
// Slower method
```

```
$my_array = array(1, 2, 3, ? 500);
```

```
// Faster method
```

```
$my_array[] = 1;
```

```
$my_array[] = 2;
```

```
$my_array[] = 3;
```

```
?
```

```
$my_array[] = 500;
```

```
?>
```

[up](#)

[down](#)

-4

[\*jay at ezlasvegas dot net ¶\*](#)

**21 years ago**

If you want to create an array of a set size and you have PHP4, use array\_pad(array(), \$SIZE, \$INITIAL\_VALUE); This can be handy if you wish to initialize a bunch of variables at once:

```
list($Var1, $Var2, etc) = array_pad(array(), $NUMBER_OF_VARS,
$INITIAL_VALUE);
```

Jay Walker

Las Vegas Hotel Associate

<http://www.ezlasvegas.net>

[up](#)

[down](#)

-7

[\*MadLogic at Paradise dot net dot nz ¶\*](#)

**21 years ago**

Heres a simple yet intelligent way of setting an array, grabbing the values from the array using a loop.

```
<?php
```

```
$ary = array("1"=>'One','Two',"3"=>'Three');
```

```
$a = '0'; $b = count($ary);
```

```
while ($a <= $b) {
```

```
$pr = $ary[$a];
```

```
print "$pr<br>";
```

```
$a++;
```

```
}
```

```
?>
```

[up](#)

[down](#)

-6

[eugene at ultimatecms dot co dot za](#)

13 years ago

This is a useful way to get foreign key variables from a specific sql table.

This function can be used to include all relevant data from all relating tables:

```
<?php
function get_string_between($string, $start, $end){
$string = " ".$string;
$ini = strpos($string,$start);
if($ini == 0)
return $tbl;
$ini += strlen($start);
$len = strpos($string,$end,$ini) - $ini;
return substr($string,$ini,$len);
}

function get_foreign_keys($tbl) {
$query = query_getrow("SHOW CREATE TABLE ".$mysql_escape_string($tbl));

$dat = explode('CONSTRAINT',$query[1]);
foreach($dat as $d => $a) {
if(strpos($a,"FOREIGN KEY"))
$data['keys'][] = array($tbl,get_string_between($a,"` FOREIGN KEY (`","`) REFERENCES"));
}

foreach($dat as $d => $a) {
if(strpos($a,"REFERENCE"))
$data['references'][] = explode("` (`",get_string_between($a,"REFERENCES `","`) ON"));
}
return $data;
}

//Example code:

$data = get_foreign_keys('task_table');
echo '<pre>';
print_r($data);
echo '</pre>';

?>

// $query[1] outputs:
```

```
CREATE TABLE `task_table` (
`task_id` int(64) NOT NULL auto_increment,
`ticket_id` int(64) NOT NULL,
`task_type` varchar(64) NOT NULL,
`comment` text,
`assigned_to` int(11) default NULL,
`dependant` int(64) default NULL,
`resolved` int(1) default NULL,
PRIMARY KEY (`task_id`),
KEY `ticket_id` (`ticket_id`,`dependant`),
KEY `assigned_to` (`assigned_to`),
KEY `task_dependant` (`dependant`),
CONSTRAINT `task_table_ibfk_1` FOREIGN KEY (`ticket_id`) REFERENCES `tickets_table` (`ticket_id`) ON DELETE CASCADE ON
UPDATE CASCADE,
CONSTRAINT `task_table_ibfk_2` FOREIGN KEY (`assigned_to`) REFERENCES `contact_table` (`contact_id`) ON DELETE CASCADE ON
UPDATE CASCADE,
CONSTRAINT `task_table_ibfk_3` FOREIGN KEY (`dependant`) REFERENCES `task_table` (`task_id`) ON DELETE CASCADE ON UPDATE
CASCADE
```

```
) ENGINE=InnoDB DEFAULT CHARSET=latin1
```

```
// $data outputs:
```

```
Array (
  [keys] => Array
  (
    [0] => Array
    (
      [0] => task_table
      [1] => ticket_id
    )

    [1] => Array
    (
      [0] => task_table
      [1] => assigned_to
    )

    [2] => Array
    (
      [0] => task_table
      [1] => dependant
    )
  )

  [references] => Array
  (
    [0] => Array
    (
      [0] => tickets_table
      [1] => ticket_id
    )

    [1] => Array
    (
      [0] => contact_table
      [1] => contact_id
    )

    [2] => Array
    (
      [0] => task_table
      [1] => task_id
    )
  )
)
```

[up](#)

[down](#)

-12

[ayyappan dot ashok at gmail dot com](#) ¶

7 years ago

The code bellow by dave@wp.pl:

```
$var_name='var';
$var=array(0,1,2,3,4);
echo $$var_name[2];
```

```
//prints NOTHING instead 2.
```

```
// prints 2 with PHP version 7.01 onwards
```

Note :

This results 2 with PHP version 7.01 onwards. Other versions throws a notice error

[up](#)

[down](#)

-8

[jeremy](#)

**11 years ago**

Some tricky functions;

```
<?php
function is_array_assoc($arr) {
    if (is_array($arr)) {
        foreach ($arr as $k => $v) {
            if (is_string($k) || (is_int($k) && $k < 0)) {
                return 1;
            }
        }
        return 0;
    }
    return -1;
}

function is_array_multi($arr) {
    return is_array($arr)
        ? (count($arr) != count($arr, COUNT_RECURSIVE) ? 1 : 0)
        : -1;
}

$arr[] = array("foo", "bar", 1.09);
$arr[] = array("red", "yellow", 1 => "foo");
$arr[] = array("red", "yellow", -1 => "foo");
$arr[] = array("x" => array("red", "yellow"), "y" => array("one", "two"));
$arr[] = array();
$arr[] = "s";

foreach ($arr as $a) {
    echo is_array_assoc($a) . "\n";
}
echo "\n";
foreach ($arr as $a) {
    echo is_array_multi($a) . "\n";
}
?>
```

0  
0  
1  
1  
0  
-1

0  
0  
0  
1  
0  
-1

```
<?php
function array_count($arr) {
    $r = 0;
    foreach ($arr as $k => $v) {
        if (is_array($v)) {
            $r++;
        }
    }
}
```

```

}
}
return $r;
}

function array_count_all($arr, $r = 0) {
foreach ($arr as $k => $v) {
if (is_array($v)) {
$r = array_count_all($v, $r);
$r++;
}
}
return $r;
}

$a = array("foo", "bar", 1.09, array(1,2,3), array("a" => "aaa"));
echo array_count($a); // 2

$a = array(
"foo", "bar", 1.09,
"x" => array(1,2,3),
array("a" => "aaa", "b" => array(), "c" => array(""), "d" => array("")),
"e" => array(""),
"z" => array(""),
"q" => array("", array()),
"w" => array("lorem", 1 => "ipsum", 18, "dolor", array("last as 11th array"))
);
echo array_count_all($a); // 11
?>

```

[up](#)  
[down](#)  
-7  
[\*webmaster at phpemailformprocessor dot com\*](#) ¶  
**17 years ago**

When using an array to create a list of keys and values for a select box generator which will consist of states I found using "NULL" as an index and ""(empty value) as a value to be useful:

```

<?php

$states = array(
0 => 'Select a State',
NULL => '',
1 => 'AL - Alabama',
2 => 'AK - Alaska',
# And so on ...
);

$select = '<select name="state" id="state" size="1">'. "\r\n";

foreach($states as $key => $value){
$select .= "\t". '<option value="'. $key. '">' . $value. '</option>'. "\r\n";
}

$select .= '</select>';

echo $select;

?>

```

This will print out:

```

<select name="state" id="state" size="1">

```

```

<option value="0">Select a State</option>
<option value=""></option>
<option value="1">AL - Alabama</option>
<option value="2">AK - Alaska</option>
# And so on ...

</select>

```

Now a user has a blank value to select if they later decide to not provide their address in the form. The first two options will return TRUE when checked against the php function - EMPTY() after the form is submitted when processing the form

[up](#)

[down](#)

-11

[tobiasquinteiro at ig dot com dot br ¶](#)

**22 years ago**

```

<?
// This is a small script that shows how to use an multiple array
for($x = 0;$x < 10;$x++){
for($y = 0;$y < 10;$y++){
$mat[$x][$y] = "$x,$y";
}
}

for($x = 0;$x < count($mat);$x++){
for($y = 0;$y < count($mat[$x]);$y++){
echo "mat[$x][$y]: " .
$mat[$x][$y] . " ";
}
echo "\n";
}
?>

```

[up](#)

[down](#)

-13

[baZz ¶](#)

**20 years ago**

Chek this out!!!. Suppose that you want to create an array like the following:

```

<?php
$arr1 = (
0 => array ("customer"=>"Client 1","Item a"),
1 => array ("customer"=>"Client 2","Item b")
);
?>

```

Seems prety easy, but what if you want to generate it dinamically woops!!!. Imagine that you have a file with thousands of lines and each line is a purchase order from diferent clients:

```

<?php
/*function to add elements*/
function addArray(&$array, $id, $var)
{
$tempArray = array( $var => $id);
$array = array_merge ($array, $tempArray);
}
/*The same as above but the element is an array*/
function addArrayArr(&$array, $var, &$array1)
{
$tempArray = array($var => $array1);
$array = array_merge ($array, $tempArray);
}
/*labels of our array or heders of the file*/
$keyarr = array("customer","item");
/*info that may you read from a file line 1 and 2*/

```

```

$valarr0 = array("Client 1","Item a");
$valarr1 = array("Client 2","Item b");

$numofrows = 2;/*In our case is just two lines*/
$tmpArray = array();
for($i = 0; $i < $numofrows; $i++){
    $tmp = "valarr$i";
    $tmpvar = ${$tmp};/*Using var of vars tricky tricky*/
    foreach( $keyarr as $key=>$value){
        addArray($tmparr,$tmpvar[$key],$value);
    }
    addArrayArr($finalarr,$i,$tmparr);
} /*voila all it's perfectly ordered on finalarr*/

/*Here we just print the info but you can insert it into a database*/
echo "Customer: ".$finalarr[0]["customer"]."<br>";
echo "Item: ".$finalarr[0]["item"]."<br>";
echo "Customer: ".$finalarr[1]["customer"]."<br>";
echo "Item: ".$finalarr[1]["item"]."<br>";
?>

```

The lines above should print something like:

Customer: Client 1

Item: Item a

Customer: Client 2

Item: Item b

I hope someone find this useful.

[up](#)

[down](#)

-6

[marcel at labor-club dot de ¶](#)

**20 years ago**

i tried to find a way to create BIG multidimensional-arrays. but the notes below only show the usage of it, or the creation of small arrays like \$matrix=array('birne', 'apfel', 'beere');

for an online game, i use a big array (50x80) elements.

it's no fun, to write the declaration of it in the ordinary way.

here's my solution, to create an 2d-array, filled for example with raising numbers.

```

<?php
$matrix=array();
$sx=30;
$sy=40;
$i=1;
for ($y=0; $y<$sy; $y++)
{
    array_push($matrix,array());
    for ($x=0; $x<$sx; $x++)
    {
        array_push($matrix[$y],array());
        $matrix[$x][$y]=$i;
        $i++;
    }
}
?>

```

if there is a better way, plz send an email. i always want to learn more php!

[up](#)

[down](#)

-5

[mads at nospam westermann dot dk ¶](#)

## 21 years ago

In PHP 4.2.3 (and maybe earlier versions) arrays with numeric indexes may be initialized to start at a specific index and then automatically increment the index. This will save you having to write the index in front of every element for arrays that are not zero-based.

The code:

```
<?php
$a = array
(
21 => 1,
2,
3,
);
print '<pre>';
print_r($a);
print '</pre>';
?>
```

will print:

```
<?php
Array
(
[21] => 1
[22] => 2
[23] => 3
)
?>
```

[up](#)

[down](#)

-6

[matthiasDELETETHIS at ansorgs dot de ¶](#)

## 18 years ago

How to use array() to create an array of references rather than of copies? (Especially needed when dealing with objects.) I played around somewhat and found a solution: place & before the parameters of array() that shall be references. My PHP version is 4.3.10.

Demonstration:

```
<?php
$ref1 = 'unchanged';
$ref2 = & $ref1;

$array_of_copies = array($ref1, $ref2);
print_r($array_of_copies); // prints: Array ( [0] => unchanged [1] => unchanged )
$array_of_copies[0] = 'changed'; // $ref1 = 'changed'; is not equivalent, as it was _copied_ to the array
print_r($array_of_copies); // prints: Array ( [0] => changed [1] => unchanged )
```

```
$array_of_refs = array(& $ref1, & $ref2); // the difference: place & before arguments
print_r($array_of_refs); // prints: Array ( [0] => unchanged [1] => unchanged )
$array_of_refs[0] = 'changed'; // $ref1 = 'changed'; is equivalent as $array_of_refs[0] references $ref1
print_r($array_of_refs, true); // prints: Array ( [0] => changed [1] => changed )
?>
```

[up](#)

[down](#)

-8

[TCross1 at hotmail dot com ¶](#)

## 20 years ago

here is the sort of "textbook" way to output the contents of an array which avoids using foreach() and allows you to index & iterate through the array as you see fit:



```
<?php

$arrayName = array("apples", "bananas", "oranges", "pears");
$arrayLength = count($arrayName);

for ($i = 0; $i < $arrayLength; $i++){
echo "arrayName at[" . $i . "] is: [" . $arrayName[$i] . "]<br>\n";
}

?>
```

enjoy!

-tim  
[up](#)  
[down](#)  
-7

[sergei dot solomonov at gmail dot com ¶](#)

**10 years ago**

Consider:

file inc1.php

-----

```
<?php
return 'key';
```

file inc2.php

-----

```
<?php
return 'value';
```

Test:

```
<?php
$a = [
include 'inc1.php' => include 'inc2.php'
];
var_dump($a);
```

```
/* It works!!!
array(1) {
'key' =>
string(5) "value"
}
*/
```

[up](#)  
[down](#)  
-11

[kamil at navikam dot pl ¶](#)

**14 years ago**

Easy function to unarray an array :-)

It will make \$array['something'] => \$something.

Usefull for making code more clear.

example of use:

```
<?
function unarray($row) {
foreach($row as $key => $value) {
global $$key;
$$key = $value;
}
}
```

```
$sql = mysql_query("SELECT * FROM `pracownicy`");
```

```
while ($row = mysql_fetch_assoc($sql)) {  
    unarray($row);  
    echo $idpracownika.'<br>'; //instead of $row['idpracownika']  
}  
?>
```

[up](#)

[down](#)

-7

[dave at wp dot pl ¶](#)

**8 years ago**

The code bellow:

```
$var_name='var';  
$var=array(0,1,2,3,4);  
echo $$var_name[2];
```

prints NOTHING instead 2.

[up](#)

[down](#)

-10

[joshua dot e at usa dot net ¶](#)

**22 years ago**

Here's a cool tip for working with associative arrays-

Here's what I was trying to accomplish:

I wanted to hit a DB, and load the results into an associative array, since I only had key/value pairs returned. I loaded them into an array, because I wanted to manipulate the data further after the DB select, but I didn't want to hit the DB more than necessary.

Here's how I did it:

```
<?php  
//assume db connectivity  
//load it all into the associative array  
$sql = "SELECT key,value FROM table";  
$result = mysql_query($sql);  
while($row = mysql_fetch_row($result)) {  
    $myArray[$row[0]] = $row[1];  
}  
//now we expand it  
while(list($key,$value) = each($myArray)) {  
    echo "$key : $value";  
}  
?>
```

I found this to be super efficient, and extremely cool.

[up](#)

[down](#)

-12

[sebasg37 at gmail dot com ¶](#)

**16 years ago**

Recursive function similar to print\_r for describing anidated arrays in html <ol>. Maybe it's useful for someone.

```
<?php  
function describeAnidatedArray($array)  
{  
    $buf = '';  
    foreach($array as $key => $value)  
    {  
        if(is_array($value))  
        {  
            $buf .= '<ol>' . describeAnidatedArray($value) . '</ol>';  
        }  
    }  
}
```

```

}
else
$buf .= "<li>$value</li>";
}
return $buf;
}

// Example:
$array = array("a", "b", "c", array("1", "2", array("A", "B")), array("3", "4"), "d");

echo describeAnidatedArray($array);
?>

```

output:

```

# a
# b
# c

```

```

1. 1
2. 2
1. A
2. B

```

```

1. 3
2. 4

```

```

# d

```

[up](#)

[down](#)

-16

[phpm at nreynolds dot me dot uk ¶](#)

**19 years ago**

This helper function creates a multi-dimensional array. For example, creating a three dimensional array measuring 10x20x30: `<?php $my_array = multi_dim(10, 20, 30); ?>`

```

<?php

```

```

function multi_dim()
{
$fill_value = null;

for ($arg_index = func_num_args() - 1; $arg_index >= 0; $arg_index--) {
$dim_size = func_get_arg($arg_index);
$fill_value = array_fill(0, $dim_size, $fill_value);
}

return $fill_value;
}

```

```

?>

```

[up](#)

[down](#)

-12

[John ¶](#)

**21 years ago**

Be careful not to create an array on top of an already existing variable:

```

<?php
$name = "John";
$name['last'] = "Doe";
?>

```

\$name becomes "Dohn" since 'last' evaluates to the 0th position of \$name.

Same is true for multi-arrays.

[+add a note](#)

- [Функции для работы с массивами](#)

- [array change key case](#)
- [array chunk](#)
- [array column](#)
- [array combine](#)
- [array count values](#)
- [array diff assoc](#)
- [array diff key](#)
- [array diff uassoc](#)
- [array diff ukey](#)
- [array diff](#)
- [array fill keys](#)
- [array fill](#)
- [array filter](#)
- [array flip](#)
- [array intersect assoc](#)
- [array intersect key](#)
- [array intersect uassoc](#)
- [array intersect ukey](#)
- [array intersect](#)
- [array is list](#)
- [array key exists](#)
- [array key first](#)
- [array key last](#)
- [array keys](#)
- [array map](#)
- [array merge recursive](#)
- [array merge](#)
- [array multisort](#)
- [array pad](#)
- [array pop](#)
- [array product](#)
- [array push](#)
- [array rand](#)
- [array reduce](#)
- [array replace recursive](#)
- [array replace](#)
- [array reverse](#)
- [array search](#)
- [array shift](#)
- [array slice](#)
- [array splice](#)
- [array sum](#)
- [array udiff assoc](#)
- [array udiff uassoc](#)
- [array udiff](#)
- [array uintersect assoc](#)
- [array uintersect uassoc](#)
- [array uintersect](#)
- [array unique](#)
- [array unshift](#)
- [array values](#)
- [array walk recursive](#)
- [array walk](#)
- [array](#)
- [arsort](#)
- [asort](#)

- [compact](#)
- [count](#)
- [current](#)
- [end](#)
- [extract](#)
- [in array](#)
- [key\\_exists](#)
- [key](#)
- [krsort](#)
- [ksort](#)
- [list](#)
- [natcasesort](#)
- [natsort](#)
- [next](#)
- [pos](#)
- [prev](#)
- [range](#)
- [reset](#)
- [rsort](#)
- [shuffle](#)
- [sizeof](#)
- [sort](#)
- [uasort](#)
- [uksort](#)
- [usort](#)
- Deprecated
  - [each](#)
- [Copyright © 2001-2024 The PHP Group](#)
- [My PHP.net](#)
- [Contact](#)
- [Other PHP.net sites](#)
- [Privacy policy](#)

