# Arrays in PHP

## What is an Array?

A variable is a storage area holding a number or text. The problem is, a variable will hold only one value.

An array is a special variable, which can store multiple values in one single variable.

If you have a list of items (a list of car names, for example), storing the cars in single variables could look like this:

```
$vehicle1="car";
$vehicle2="van";
$vehicle3="bus"
```

However, what if you want to loop through the vehicle and find a specific one? And what if you had not 3 vehicle, but 100?

The best solution here is to use an array!

An array can hold all your variable values under a single name. And you can access the values by referring to the array name.

Each element in the array has its own index so that it can be easily accessed.

In PHP, there are three kind of arrays:

- **Numeric array** - An array with a numeric index
- **Associative array** - An array where each ID key is associated with a value
- **Multidimensional array** - An array containing one or more arrays

## Numeric Arrays (Indexed array)

A numeric array stores each array element with a numeric index. There are two methods to create a numeric array.

Method 1.To assign index  automatically (the index starts at 0):

```
$vehicle=array("car","van","bus","cycle");
```

When you print this you can see that they are assigned as follows

```
$vehicle[0]="car"
$vehicle[1]="van"
```

```
$vehicle[2]=”bus”
$vehicle[3]=”cycle”
```

Method 2: To  assign the index manually:

In PHP code write as such

```
$vehicle[0]=”car”
$vehicle[1]=”van”
$vehicle[2]=”bus”
$vehicle[3]=”cycle”
```

### Example

In the following example you access the variable values by referring to the array name and index:

```
<?php
$vehicle[0]=”car”
$vehicle[1]=”van”
$vehicle[2]=”bus”
$vehicle[3]=”cycle”

echo “I have “, $vehicle[0] . " and " . $vehicle[1] . " .";
?>
```

Output

```
I have car and van.
```

## Associative Arrays

An associative array, each ID key is associated with a value. When storing data about specific named values, a numerical array is not always the best way to do it.

With associative arrays we can use the values as keys and assign values to them.

Example 1: In this example an associative array is used to assign colors to different cars.

```
$carscolor = array("maruthi800"=>”blue”, "swift"=>”black”, "vaganR"=>”green”);
```

Example 2:

```
$carscolor[‘maruthi800’]=”blue”;
$carscolor[‘swift’]=”black”;
$carscolor[‘vaganR’]=”green”;
```

The values can be accessed using the ID keys in a script like the following:

```
<?php
$carscolor[‘maruthi800’]=”blue”;
$carscolor[‘swift’]=”black”;
$carscolor[‘vaganR’]=”green”;

echo "Swift color is " . $ages['swift']  ;
?>
```

Output

```
Swift color is black.
```

## Create an Empty array

To construct an empty array, pass no arguments to `array( )`:

```
$addresses = array(  );
```

### *Adding Values to the End of an Array*

To insert more values into the end of an existing indexed array, use the `[]` syntax:

```
$carscolor = array('black', 'green');
$carscolor[] = 'red';        // $carscolor[2] is ‘red’
```

This construct assumes the array's indices are numbers and assigns elements into the next available numeric index, starting from 0. Attempting to append to an associative array is almost always a programmer mistake, but PHP will give the new elements numeric indices without issuing a warning:

```
$person = array('name' => 'Fred');
$person[] = 'Wilma';                   // $person[0] is now 'Wilma'
```

### *Assigning a Range of Values*

The `range( )` function creates an array of consecutive integer or character values between the two values you pass to it as arguments. For example:

```
$numbers = range(1, 3);                // $numbers = array(1, 2, 3);
$letters = range('a', 'z');            // $letters holds the alphabet
$reverse = range(7, 4);               // $reverse = array(7,6,5,4);
```

### *Getting the Size of an Array*

The `count( )` and `sizeof( )` functions are identical in use and effect. They return the number of elements in the array. There is no stylistic preference about which function you use. Here's an example:

```
    $carscolor = array('red', 'green', 'blue');
    $size  = count($carscolor);              // $size is 3
```

These functions do not consult any numeric indices that might be present:

```
    $tens = array( 10 => 'ten', 20 => 'twenty', 30 => 'thirty');
    $size = count($tens);      // $size is 3
```

Handling Arrays with Loops

You can traverse and manipulate values in an array using the loops available in PHP.

## **The for loop**

**Count function returns the number of elements in an array. The following program is an example for this:**

```
<html>
  <head>
    <title>
      Using a for loop to loop over an array
    </title>
  </head>
  <body>
    <h1>
      Using a for loop to loop over an array
    </h1>
   <?php
      $cars[0] = "bmw";
      $cars[1] = "Maruthi";
      $cars[2] = "honda";

      for ($i = 0; $i < count($actors); $i++)
     {
        echo "\$cars[$i] = ", $cars[$i], "<br>";
      }
    ?>
  </body>
</html>
```

The print_r function

This is a simple function used to print the array contents

Syntax

print_r ($array [, bool return])

In the above program you can use print_r like this

```
<html>
  <head>
    <title>
      Using a for loop to loop over an array
    </title>
  </head>
  <body>
    <h1>
      Using a for loop to loop over an array
    </h1>
   <?php
      $cars[0] = "bmw";
      $cars[1] = "Maruthi";
      $cars[2] = "honda";

      print_r($cars);
    ?>
  </body>
</html>
```

```
Output :
Array ( [0]=>bmw [1]=>Maruthi [2]=>Honda)
```

Note that this gives you the result in key=>value form.

**print_r in associate arrays**

```
<?php
      $age[‘ram’] = 25;
      $age[‘rani’] = 26;
      $age[‘ragav’] = 30;

      print_r($age);
?>
```

```
Output
Array
(
[ram]=>25
[rani]=>26
[ragav]=>30
)
```

## The while loop

```
<html>
  <head>
    <title>
      Using a while loop to loop over an array
    </title>
  </head>
  <body>
```

```
    <h1>
      Using a while loop to loop over an array
    </h1>
   <?php
      $age['ram'] = 25;
      $age['rani'] = 26;
      $age['ragav'] = 30;

      while (list($key,$value)=each($age))
    {
        echo "Key:$key ; value : $value <br>";
      }
    ?>
  </body>
</html>

Output
Key: ram value : 25
Key: rani value : 26
Key: ragav value : 30
```

### *The foreach Construct*

The most common way to loop over elements of an array is to use the `foreach` construct:

```
$addresses = array('spam@cyberpromo.net', 'abuse@example.com');
foreach ($addresses as $value) {
  echo "Processing $value\n";
}

Output:

Processing spam@cyberpromo.net
Processing abuse@example.com
```

PHP executes the body of the loop (the `echo` statement) once for each element of `$addresses` in turn, with `$value` set to the current element. Elements are processed by their internal order.

An alternative form of `foreach` gives you access to the current key:

```
 $person = array('name' => 'Fred', 'age' => 35, 'wife' => 'Wilma');
 foreach ($person as $key => $value)
  {
   echo "Fred's $key is $value\n";
  }

Output:

 Fred's name is Fred
 Fred's age is 35
 Fred's wife is Wilma
```

In this case, the key for each element is placed in `$key` and the corresponding value is placed in `$value`.

The `foreach` construct does not operate on the array itself, but rather on a copy of it. You can insert or delete elements in the body of a `foreach` loop, safe in the knowledge that the loop won't attempt to process the deleted or inserted elements.

## *The Iterator Functions*

Every PHP array keeps track of the current element you're working with; the pointer to the current element is known as the iterator. PHP has functions to set, move, and reset this iterator. The iterator functions are:

current( )

Returns the element currently pointed at by the iterator

reset( )

Moves the iterator to the first element in the array and returns it

next( )

Moves the iterator to the next element in the array and returns it

prev( )

Moves the iterator to the previous element in the array and returns it

end( )

Moves the iterator to the last element in the array and returns it

each( )

Returns the key and value of the current element as an array and moves the iterator to the next element in the array

key( )

Returns the key of the current element

The `each( )` function is used to loop over the elements of an array. It processes elements according to their internal order:

```
    reset($addresses);
    while (list($key, $value) = each($addresses)) {
      echo "$key is $value<BR>\n";
    }
```

Output:

```
    0 is spam@cyberpromo.net
    1 is abuse@example.com
```

This approach does not make a copy of the array, as `foreach` does. This is useful for very large arrays when you want to conserve memory.

The iterator functions are useful when you need to consider some parts of the array separately from others.

## Extracting data from arrays

**Extract function is used to extract data from arrays and store it in variables.**

**Example :**

```
<?php
      $actor['rajini'] = "sivaji";
      $actor['surya'] = "gajini";
      $age['kamal'] = "dasavadharam";

      Extract($actor);
      echo "\$rajini=$rajini <br>";
      echo "\$surya =$surya <br>";
      echo "\$kamal =$kamal <br>";


    ?>
```

**Output**

```
$rajini = sivaji
$surya = gajini
$kamal = dasavadharam
```

**List function**

**List function is used to extract variables from an array. The difference is we can assign the name for the variable as we like.**

**Example**

```
<?php
      $actor['rajini'] = "sivaji";
      $actor['surya'] = "gajini";
      $age['kamal'] = "dasavadharam";

      list($one,$two)=$actor;
      echo $one, "<br>";
      echo $two, "<br>";
    ?>

Output:
rajini
surya
```

## Sorting

Sorting changes the internal order of elements in an array and optionally rewrites the keys to reflect this new order. For example, you might use sorting to arrange a list of scores from biggest to smallest, to alphabetise a list of names or to order a set of users based on how many messages they posted.

PHP provides three ways to sort arrays sorting by keys, sorting by values without changing the keys, or sorting by values and then changing the keys. Each kind of sort can be done in ascending order, descending order, or an order defined by a user-defined function.

PHP functions for sorting an array

| **Effect** | **Ascending** | **Descending** | **User-defined order** |
|---|---|---|---|
| Sort array by values, then reassign indices starting with 0 | `sort( )` | `rsort( )` | `usort( )` |
| Sort array by values | `asort( )` | `arsort( )` | `uasort( )` |
| Sort array by keys | `ksort( )` | `krsort( )` | `uksort( )` |

To sort names into ascending alphabetical order, you'd use this:

```
    $names = array('cath', 'angela', 'brad', 'dave');
    sort($names);        // $names is now 'angela', 'brad', 'cath', 'dave'
```

To get them in reverse alphabetic order, simply call `rsort( )` instead of `sort( )`.

### *Natural-Order Sorting*

PHP's built-in sort functions correctly sort strings and numbers, but they don't correctly sort strings that contain numbers. For example, if you have the filenames *ex10.php*, *ex5.php*, and *ex1.php*, the normal sort functions will rearrange them in this order: *ex1.php*, *ex10.php*, *ex5.php*. To correctly sort strings that contain numbers, use the `natsort( )` and `natcasesort( )` functions:

```
    $output = natsort(input);
    $output = natcasesort(input);
```

## *Reversing Arrays*

The `array_reverse( )` function reverses the internal order of elements in an array:

```
$reversed = array_reverse(array);
```

Numeric keys are renumbered starting at 0, while string indices are unaffected. In general, it's better to use the reverse-order sorting functions instead of sorting and then reversing the order of an array.

The `array_flip( )` function returns an array that reverses the order of each original element's key-value pair:

```
$flipped = array_flip(array);
```

That is, for each element of the array whose value is a valid key, the element's value becomes its key and the element's key becomes its value. For example, if you have an array mapping usernames to home directories, you can use `array_flip( )` to create an array mapping home directories to usernames:

```
<?php
      $actor['rajini'] = "sivaji";
      $actor['surya'] = "gajini";
      $age['kamal'] = "dasavadharam";

      print_r(array_flip($actor));

      ?>
```

Output:

```
Array([sivaji] => "rajini" [gajini]=>"surya" [dasavadharam] => "kamal")
```

Elements whose original values are neither strings nor integers are left alone in the resulting array. The new array lets you discover the key in the original array given its value, but this technique works effectively only when the original array has unique values.

PHP built-in array functions

| **Function** | **Description** |
|---|---|
| array() | Creates an array |
| array_change_key_case() | Returns an array with all keys in lowercase or uppercase |
| array_chunk() | Splits an array into chunks of arrays |
| array_combine() | Creates an array by using one array for keys and another for its values |
| array_count_values() | Returns an array with the number of occurrences for each value |
| array_diff() | Compares array values, and returns the differences |
| array_diff_assoc() | Compares array keys and values, and returns the differences |
| array_diff_key() | Compares array keys, and returns the differences |
| array_diff_uassoc() | Compares array keys and values, with an additional user-made function check, and returns the differences |
| array_diff_ukey() | Compares array keys, with an additional user-made function check, and returns the differences |
| array_fill() | Fills an array with values |
| array_filter() | Filters elements of an array using a user-made function |
| array_flip() | Exchanges all keys with their associated values in an array |
| array_intersect() | Compares array values, and returns the matches |
| array_intersect_assoc() | Compares array keys and values, and returns the matches |
| array_intersect_key() | Compares array keys, and returns the matches |
| array_intersect_uassoc() | Compares array keys and values, with an additional user-made function check, and returns the matches |
| array_intersect_ukey() | Compares array keys, with an additional user-made function check, and returns the matches |
| array_key_exists() | Checks if the specified key exists in the array |
| array_keys() | Returns all the keys of an array |
| array_map() | Sends each value of an array to a user-made function, which returns new values |
| array_merge() | Merges one or more arrays into one array |
| array_merge_recursive() | Merges one or more arrays into one array |
| array_multisort() | Sorts multiple or multi-dimensional arrays |
| array_pad() | Inserts a specified number of items, with a specified |

| | |
|---|---|
| | value, to an array |
| array_pop() | Deletes the last element of an array |
| array_product() | Calculates the product of the values in an array |
| array_push() | Inserts one or more elements to the end of an array |
| array_rand() | Returns one or more random keys from an array |
| array_reduce() | Returns an array as a string, using a user-defined function |
| array_reverse() | Returns an array in the reverse order |
| array_search() | Searches an array for a given value and returns the key |
| array_shift() | Removes the first element from an array, and returns the value of the removed element |
| array_slice() | Returns selected parts of an array |
| array_splice() | Removes and replaces specified elements of an array |
| array_sum() | Returns the sum of the values in an array |
| array_udiff() | Compares array values in a user-made function and returns an array |
| array_udiff_assoc() | Compares array keys, and compares array values in a user-made function, and returns an array |
| array_udiff_uassoc() | Compares array keys and array values in user-made functions, and returns an array |
| array_uintersect() | Compares array values in a user-made function and returns an array |
| array_uintersect_assoc() | Compares array keys, and compares array values in a user-made function, and returns an array |
| array_uintersect_uassoc() | Compares array keys and array values in user-made functions, and returns an array |
| array_unique() | Removes duplicate values from an array |
| array_unshift() | Adds one or more elements to the beginning of an array |
| array_values() | Returns all the values of an array |
| array_walk() | Applies a user function to every member of an array |
| array_walk_recursive() | Applies a user function recursively to every member of an array |
| arsort() | Sorts an array in reverse order and maintain index association |
| asort() | Sorts an array and maintain index association |
| compact() | Create array containing variables and their values |
| count() | Counts elements in an array, or properties in an object |
| current() | Returns the current element in an array |

| | |
|---|---|
| each() | Returns the current key and value pair from an array |
| end() | Sets the internal pointer of an array to its last element |
| extract() | Imports variables into the current symbol table from an array |
| in_array() | Checks if a specified value exists in an array |
| key() | Fetches a key from an array |
| krsort() | Sorts an array by key in reverse order |
| ksort() | Sorts an array by key |
| list() | Assigns variables as if they were an array |
| natcasesort() | Sorts an array using a case insensitive "natural order" algorithm |
| natsort() | Sorts an array using a "natural order" algorithm |
| next() | Advance the internal array pointer of an array |
| pos() | Alias of current() |
| prev() | Rewinds the internal array pointer |
| range() | Creates an array containing a range of elements |
| reset() | Sets the internal pointer of an array to its first element |
| rsort() | Sorts an array in reverse order |
| shuffle() | Shuffles an array |
| sizeof() | Alias of count() |
| sort() | Sorts an array |
| uasort() | Sorts an array with a user-defined function and maintain index association |
| uksort() | Sorts an array by keys using a user-defined function |
| usort() | Sorts an array by values using a user-defined function |

## Definition and Usage

**array()** creates an array, with keys and values. If you skip the keys when you specify an array, an integer key is generated, starting at 0 and increases by 1 for each value.

## Syntax

array(key => value)

| **Parameter** | **Description** |
|---|---|
| key | Optional. Specifies the key, of type numeric or string. If not set, an integer key is generated, starting at 0 |

| | |
|---|---|
| value | Required. Specifies the value |

## Example 1

```
<?php
$a=array("a"=>"Dog","b"=>"Cat","c"=>"Horse");
print_r($a);
?>
```

The output of the code above will be:

```
Array ( [a] => Dog [b] => Cat [c] => Horse )
```

## Example 2

```
<?php
$a=array("Dog","Cat","Horse");
print_r($a);
?>
```

The output of the code above will be:

```
Array ( [0] => Dog [1] => Cat [2] => Horse )
```

## Definition and Usage

The array_change_key_case() function returns an array with all array KEYS in lower case or upper case.

## Syntax

array_change_key_case(array,case)

| **Parameter** | **Description** |
|---|---|
| array | Required. Specifies the array to use |
| case | Optional. Possible values:<br>• CASE_LOWER - Default value. Returns the array key values in lower case.<br>• CASE_UPPER - Returns the array key values in upper case. |

## Tips and Notes

**Note:** If two or more array keys will be the same after running this function, the last array will override the others. (See example 2)

## Example 1

```
<?php
$a=array("a"=>"Cat","b"=>"Dog","c"=>"Horse");
print_r(array_change_key_case($a,CASE_UPPER));
?>
```

The output of the code above will be:

```
Array ( [A] => Cat [B] => Dog [C] => Horse )
```

## Example 2

```
<?php
$a=array("a"=>"Cat","b"=>"Dog","c"=>"Horse","B"=>"Bird");
print_r(array_change_key_case($a,CASE_UPPER));
?>
```

The output of the code above will be:

```
Array ( [A] => Cat [B] => Bird [C] => Horse )
```

```
$data1 = array('10','22','30')
$data2 = array('2','11','3');
$data3 = array();

foreach($data1 as $key => $value) {
   echo "$key =",$value;
}
```

```
Output
[1]=10;
[2]=22;
[3]=30;
```