Array Functions

- count()

The count() function counts the elements of an array, or the properties of an object.

count(array,mode)

Parameter	Description
Array	Required. Specifies the array or object to count.
Mode	Optional. Specifies the mode of the function. Possible values: • 0 - Default. Does not detect multidimensional arrays (arrays within arrays) • 1 - Detects multidimensional arrays

Note: This function may return 0 if a variable isn't set, but it may also return 0 if a variable contains an empty array.

Example

```
<?php
$people = array("Peter", "Joe", "Glenn", "Cleveland");
$result = count($people);
echo $result;
?>
```

The output of the code above will be: 4

- list()

The list() function is used to assign values to a list of variables in one operation.

list(var1,var2...)

Parameter	Description
Var1	Required. The first variable to assign a value to
Var2	Optional. More variables to assign values to

Note: This function only works on numerical arrays.

Example 1

```
<?php
$my_array = array("Dog","Cat","Horse");

list($a, $b, $c) = $my_array;
echo "I have several animals, a $a, a $b and a $c.";
?>
```

The output of the code above will be:

I have several animals, a Dog, a Cat and a Horse.

Example 2

```
<?php
$my_array = array("Dog","Cat","Horse");

list($a, , $c) = $my_array;
echo "Here I only use the $a and $c variables.";
?>
```

The output of the code above will be:

Here I only use the Dog and Horse variables.

- in_array()

The in_array() function searches an array for a specific value. This function returns TRUE if the value is found in the array, or FALSE otherwise.

in_array(search,array,type)

Parameter	Description
Search	Required. Specifies the what to search for
Array	Required. Specifies the array to search
Type	Optional. If this parameter is set, the in_array() function searches for the search-string and specific type in the array

Note: If the search parameter is a string and the type parameter is set to TRUE, the search is case-sensitive.

Example 1

```
<?php
$people = array("Peter", "Joe", "Glenn", "Cleveland");

if (in_array("Glenn",$people))
  {
  echo "Match found";
  }
else
  {
  echo "Match not found";
  }
?>
```

The output of the code above will be:

Match found

Example 2

```
<?php
$people = array("Peter", "Joe", "Glenn", "Cleveland", 23);

if (in_array("23",$people, TRUE))
  {
  echo "Match found<br />";
  }
else
  {
  echo "Match not found<br />";
  }
if (in_array("Glenn",$people, TRUE))
  {
  echo "Match found<br />";
  }
else
  {
  echo "Match not found<br />";
  }
if (in_array(23,$people, TRUE))
  {
  echo "Match found<br />";
  }
else
  {
  echo "Match not found<br />";
  }
?>
```

The output of the code above will be:

Match not found Match found Match found

- current()

The current() function returns the value of the current element in an array.

current(array)

Parameter	Description
Array	Required. Specifies the array to use

Note: This function returns FALSE on empty elements or elements with no value.

Tip: This function does not move the arrays internal pointer. To do this, use the next() and prev() functions.

Example 1

```
<?php
$people = array("Peter", "Joe", "Glenn", "Cleveland");
```

```
echo current($people) . "<br />";
?>
```

The output of the code above will be:

Peter

- next()

The next() function moves the internal pointer to, and outputs, the next element in the array. This function returns the value of the next element in the array on success, or FALSE if there are no more elements.

next(array)

Parameter	Description
Array	Required. Specifies the array to use

Note: This function returns FALSE on empty elements or elements with no value.

Example

```
<?php
$people = array("Peter", "Joe", "Glenn", "Cleveland");
echo current($people) . "<br />";
echo next($people);
?>
```

The output of the code above will be:

Peter
Joe

- prev()

The prev() function moves the internal pointer to, and outputs, the previous element in the array. This function returns the value of the previous element in the array on success, or FALSE if there are no more elements.

prev(array)

Parameter	Description
array	Required. Specifies the array to use

Note: This function returns FALSE on empty elements or elements with no value.

Example

```
<?php
$people = array("Peter", "Joe", "Glenn", "Cleveland");
echo current($people) . "<br />";
echo next($people) . "<br />";
echo prev($people);
?>
```

The output of the code above will be:

Peter
Joe
Peter

- end()

The end() function moves the internal pointer to, and outputs, the last element in the array. This function returns the value of the last element in the array on success.

end(array)

Parameter	Description
array	Required. Specifies the array to use

Example

```
<?php
$people = array("Peter", "Joe", "Glenn", "Cleveland");

echo current($people) . "<br />";
echo end($people);
?>
```

The output of the code above will be:

Peter
Cleveland

- each()

The each() function returns the current element key and value, and moves the internal pointer forward. This element key an value is returned in an array with four elements. Two elements (1 and Value) for the element value, and two elements (0 and Key) for the element key.

This function returns FALSE if there are no more array elements.

each(array)

Parameter	Description
array	Required. Specifies the array to use

Note: This function returns FALSE on empty elements or elements with no value.

Example 1

```
<?php
$people = array("Peter", "Joe", "Glenn", "Cleveland");
print_r (each($people));
?>
```

The output of the code above will be:

Array ([1] => Peter [value] => Peter [0] => 0 [key] => 0)

Example 2

Same example as above, but with a loop to output the whole array:

```
<?php
$people = array("Peter", "Joe", "Glenn", "Cleveland");

reset($people);

while (list($key, $val) = each($people))
  {
  echo "$key => $val<br />";
  }
?>
```

The output of the code above will be:

0 => Peter
1 => Joe
2 => Glenn
3 => Cleveland

- sort()

The sort() function sorts an array by the values. This function assigns new keys for the elements in the array. Existing keys will be removed. This function returns TRUE on success, or FALSE on failure.

sort(array,sorttype)

Parameter	Description
array	Required. Specifies the array to sort
sorttype	Optional. Specifies how to sort the array values. Possible values: • SORT_REGULAR - Default. Treat values as they are (don't change types) • SORT_NUMERIC - Treat values numerically • SORT_STRING - Treat values as strings • SORT_LOCALE_STRING - Treat values as strings, based on local settings

Example

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

sort($my_array);
```

```
print_r($my_array);
?>
```

The output of the code above will be:

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

- rsort()

The rsort() function sorts an array by the values in reverse order. This function assigns new keys for the elements in the array. Existing keys will be removed. This function returns TRUE on success, or FALSE on failure.

rsort(array,sorttype)

Parameter	Description
Array	Required. Specifies the array to sort
Sorttype	Optional. Specifies how to sort the array values. Possible values: • SORT_REGULAR - Default. Treat values as they are (don't change types) • SORT_NUMERIC - Treat values numerically • SORT_STRING - Treat values as strings • SORT_LOCALE_STRING - Treat values as strings, based on local settings

Example

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

rsort($my_array);
print_r($my_array);
?>
```

The output of the code above will be:

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

- asort()

The asort() function sorts an array by the values. The values keep their original keys. This function returns TRUE on success, or FALSE on failure.

asort(array,sorttype)

Parameter	Description
array	Required. Specifying an array
sorttype	Optional. Specifies how to sort the array values. Possible values: • SORT_REGULAR - Default. Treat values as they are (don't change types) • SORT_NUMERIC - Treat values numerically • SORT_STRING - Treat values as strings • SORT_LOCALE_STRING - Treat values as strings, based on local settings

Example

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

asort($my_array);
print_r($my_array);
?>
```

The output of the code above will be:

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

- arsort()

The arsort() function sorts an array by the values in reverse order. The values keep their original keys. This function returns TRUE on success, or FALSE on failure.

arsort(array,sorttype)

Parameter	Description
array	Required. Specifying an array
sorttype	Optional. Specifies how to sort the array values. Possible values: • SORT_REGULAR - Default. Treat values as they are (don't change types) • SORT_NUMERIC - Treat values numerically • SORT_STRING - Treat values as strings

	• SORT_LOCALE_STRING - Treat values as strings, based on local settings

Example

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

arsort($my_array);
print_r($my_array);
?>
```

The output of the code above will be:

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

- reset()

The reset() function moves the internal pointer to the first element of the array.

This function returns the value of the first element in the array on success, or FALSE on failure.

reset(array)

Parameter	Description
Array	Required. Specifies the array to use

Example

```
<?php
$people = array("Peter", "Joe", "Glenn", "Cleveland");

echo current($people) . "<br />";
echo next($people) . "<br />";

echo reset($people);
?>
```

The output of the code above will be:

```
Peter
Joe
Peter
```

- array_merge()

The array_merge() function merges one ore more arrays into one array.

array_merge(array1,array2,array3...)

Parameter	Description
Array1	Required. Specifies an array
Array2	Optional. Specifies an array
Array3	Optional. Specifies an array

Tip: You can assign one array to the function, or as many as you like.

Note: If two or more array elements have the same key, the last one overrides the others.

Note: If you assign only one array to the array_merge() function, and the keys are integers, the function returns a new array with integer keys starting at 0 and increases by 1 for each value. (See example 2)

Example 1

```
<?php
$a1=array("a"=>"Horse","b"=>"Dog");
$a2=array("c"=>"Cow","b"=>"Cat");
print_r(array_merge($a1,$a2));
?>
```

The output of the code above will be:

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

Example 2

Using only one array parameter.

```
<?php
$a=array(3=>"Horse",4=>"Dog");
print_r(array_merge($a));
?>
```

The output of the code above will be:

Array ([0] => Horse [1] => Dog)

- array_reverse()

The array_reverse() function returns an array in the reverse order.

array_reverse(array,preserve)

Parameter	Description

array	Required. Specifies an array
preserve	Optional. Possible values: • true • false Specifies if the function should preserve the array's keys or not.

Example

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

The output of the code above will be:

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

- Array_diff()

The array_diff() function compares two or more arrays, and returns an array with the keys and values from the first array, only if the value is not present in any of the other arrays.

array_diff(array1,array2,array3...)

Parameter	Description
array1	Required. The first array is the array that the others will be compared with
array2	Required. An array to be compared with the first array
array3	Optional. An array to be compared with the first array

Tip: You can compare the first array with one array, or as many as you like.

Note: Only the value is used in the comparison.

Example

```
<?php
$a1=array(0=>"Cat",1=>"Dog",2=>"Horse");
$a2=array(3=>"Horse",4=>"Dog",5=>"Fish");
print_r(array_diff($a1,$a2));
?>
```

The output of the code above will be:

Array ([0] => Cat)

Miscellaneous Function

- define()

The define() function defines a constant.

Constants are much like variables, except for the following differences:

- A constant's value cannot be changed after it is set
- Constant names do not need a leading dollar sign ($)
- Constants can be accessed regardless of scope
- Constant values can only be strings and numbers

define(name,value,case_insensitive)

Parameter	Description
Name	Required. Specifies the name of the constant
Value	Required. Specifies the value of the constant
case_insensitive	Optional. Specifies whether the constant name should be case-insensitive. If set to TRUE, the constant will be case-insensitive. Default is FALSE (case-sensitive)

Example 1

Define a case-sensitive constant:

```
<?php
define("GREETING","Hello you! How are you today?");
echo constant("GREETING");
?>
```

The output of the code above will be:

Hello you! How are you today?

Example 2

Define a case-insensitive constant:

```
<?php
define("GREETING","Hello you! How are you today?",TRUE);
echo constant("greeting");
?>
```

The output of the code above will be:

Hello you! How are you today?

- constant()

The constant() function returns the value of a constant.

constant(constant)

Parameter	Description
constant	Required. Specifies the name of the constant to check

Note: This function also works with class constants.

Example

```
<?php
//define a constant
define("GREETING","Hello you! How are you today?");

echo constant("GREETING");
?>
```

The output of the code above will be:

Hello you! How are you today?

- die()

The die() function prints a message and exits the current script.

This function is an alias of the exit() function.

die(message)

Parameter	Description
message	Required. Specifies the message or status number to write before exiting the script. The status number will not be written to the output.

Example

```
<?php
$site = "http://www.w3schools.com/";
fopen($site,"r")
or die("Unable to connect to $site");
?>
```

- sleep()

The sleep() function delays execution of the current script for a specified number of seconds.

sleep(seconds)

Parameter	Description
seconds	Required. Specifies the number of seconds to delay the script

Example

```
<?php
echo date('h:i:s') . "<br />";

//sleep for 10 seconds
sleep(10);

//start again
echo date('h:i:s');

?>
```

The output of the code above will be something like this:

```
11:00:07
11:00:17
```

- usleep()

The usleep() function delays execution of the current script for a specified number of microseconds (a microsecond equals one millionth of a second).

usleep(microseconds)

Parameter	Description
microseconds	Required. Specifies the number of microseconds to delay the script

Note: This function did not work on Windows platforms until PHP 5.

Example

```
<?php
echo date('h:i:s') . "<br />";

//sleep for 10 seconds
usleep(10000000);

//start again
echo date('h:i:s');
?>
```

The output of the code above will be something like this:

```
09:23:14
09:23:24
```

- eval()

The eval() function evaluates a string as PHP code. The string must be valid PHP code and must end with semicolon.

This function returns NULL unless a return statement is called in the code string. Then the value passed to return is returned. If there is a parse error in the code string, eval() returns FALSE.

eval(phpcode)

Parameter	Description
Phpcode	Required. Specifies the PHP code to be evaluated

Note: A return statement will terminate the evaluation of the string immediately.

Tip: This function can be useful for storing PHP code in a database.

Example

```
<?php
$string = "beautiful";
$time = "winter";

$str = 'This is a $string $time morning!';
echo $str. "<br />";

eval("\$str = \"$str\";");
echo $str;
?>
```

The output of the code above will be:

This is a $string $time morning!
This is a beautiful winter morning!

Server Side Includes (SSI)

You can insert the content of one PHP file into another PHP file before the server executes it, with the include() or require() function.

The two functions are identical in every way, except how they handle errors:

- include() generates a warning, but the script will continue execution
- require() generates a fatal error, and the script will stop

These two functions are used to create functions, headers, footers, or elements that will be reused on multiple pages.

Server side includes saves a lot of work. This means that you can create a standard header, footer, or menu file for all your web pages. When the header needs to be updated, you can only update the include file, or when you add a new page to your site,

you can simply change the menu file (instead of updating the links on all your web pages).

- include()

The include() function takes all the content in a specified file and includes it in the current file. If an error occurs, the include() function generates a warning, but the script will continue execution.

Example 1

Assume that you have a standard header file, called "header.php". To include the header file in a page, use the include() function:

```
<html>
<body>
<?php include("header.php"); ?>
<h1>Welcome to my home page!</h1>
<p>Some text.</p>
</body>
</html>
```

Example 2

Assume we have a standard menu file, called "menu.php", that should be used on all pages:

```
<a href="/default.php">Home</a>
<a href="/tutorials.php">Tutorials</a>
<a href="/references.php">References</a>
<a href="/examples.php">Examples</a>
<a href="/about.php">About Us</a>
<a href="/contact.php">Contact Us</a>
```

All pages in the Web site should include this menu file. Here is how it can be done:

```
<html>
<body>
<div class="leftmenu">
<?php include("menu.php"); ?>
</div>
<h1>Welcome to my home page.</h1>
<p>Some text.</p>
</body>
</html>
```

- require()

The require() function is identical to include(), except that it handles errors differently.

If an error occurs, the include() function generates a warning, but the script will continue execution. The require() generates a fatal error, and the script will stop.

Error Example include() Function

```
<html>
<body>

<?php
include("wrongFile.php");
echo "Hello World!";
?>

</body>
</html>
```

Error message:

```
Warning: include(wrongFile.php) [function.include]:
failed to open stream:
No such file or directory in C:\home\website\test.php on line 5

Warning: include() [function.include]:
Failed opening 'wrongFile.php' for inclusion
(include_path='.;C:\php5\pear')
in C:\home\website\test.php on line 5

Hello World!
```

Error Example require() Function

Now, let's run the same example with the require() function.

```
<html>
<body>

<?php
require("wrongFile.php");
echo "Hello World!";
?>

</body>
</html>
```

Error message:

```
Warning: require(wrongFile.php) [function.require]:
failed to open stream:
No such file or directory in C:\home\website\test.php on line 5

Fatal error: require() [function.require]:
Failed opening required 'wrongFile.php'
(include_path='.;C:\php5\pear')
in C:\home\website\test.php on line 5
```