

# **Working with Arrays**

## **1.1 Unit Structure**

2. Learning Objectives
3. Introduction
4. Storing data in arrays using PHP
5. Numeric/Indexed Arrays
6. PHP Associative Array
7. PHP Multi-dimensional arrays
8. PHP Array operators
9. Manipulating arrays.
10. PHP Array Constants
11. List of Functions

## **1.2 Learning Objectives**

After studying this unit student should be able to:

- Understand one dimensional and multi-dimensional array.
- Understand manipulation of array.
- Understand array constant, operator and function.

## 1.3 Introduction

### 1.3.1 What is a PHP Array?

An array is a data structure that stores one or more similar type of values in a single value. For example, if you want to store 100 numbers then instead of defining 100 variables it's easy to define an array of 100 length.

A PHP array is a variable that stores more than one piece of related data in a single variable. Think of an array as a box of chocolates with slots inside. The box represents the array itself while the spaces containing chocolates represent the values stored in the arrays.

### 1.3.2 Storing data in arrays using PHP

An array is a special variable, which can hold more than one value at a time. If you have a list of items (a list of course names, for example), storing the course in single variables could look like this:

```
$course1 = "B.Sc. Computer Science";  
$course2 = " B.Sc. AI and Data Science ";  
$course3 = "B.Sc. Information Technology";
```

The above example showing variable name is very common, along with numeric index. If we want to store 100+ course name of Open University, then it is strongly recommended to use arrays rather than individual variable.

```
<?php  
$course = ["B.Sc. Computer Science", "B.Sc. AI and Data Science", "B.Sc. Information Technology"];  
print_r($course);  
?>
```

Output:

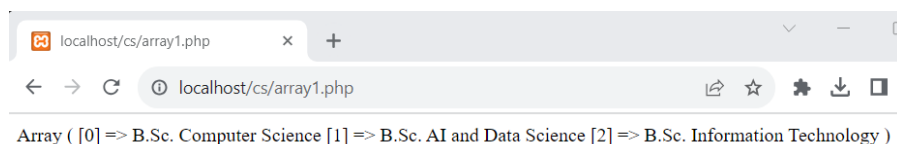


Figure 1: Basic Array

This is the simplest way to define array and store value on it.

`print_r` is an in-built function showing array out on screen in structural manner.

## 1.4 Numeric/Indexed Array

These arrays can store numbers, strings and any object but their index will be represented by numbers. By default, array index starts from zero.

- Numeric arrays use number as access keys.
- An access key is a reference to a memory slot in an array variable.
- The access key is used whenever we want to read or assign a new value an array element.
- Numeric array is also known as Indexed array.

Below is the syntax for creating numeric array in PHP. Array Example:

```
<?php
$variable_name[n] = value;
?>

Or

<?php
$variable_name = array(n => value, ...);
?>
```

### Explanation:

- “\$variable\_name...” is the name of the variable.
- “[n]” is the access index number of the element.
- “value” is the value assigned to the array element.

Suppose we have to create 5 (1,2, ...) numeric value base array that we want to store in array variables or 5 numeric as text (One, Two, ...) value.

We can use the example shown below to do as per above assumption. We have used array() function to create array.

```
<?php
/* First method to create array. */
echo " <p>First Method <br />";
$numbers = array(1, 2, 3, 4, 5);
foreach($numbers as $value ) {
echo "Array Value is $value <br />";
}
```

```

/* Second method to create array. */
echo "<p> Second Method <br /> ";
$numbers[0] = "One";
$numbers[1] = "Two";
$numbers[2] = "Three";
$numbers[3] = "Four";
$numbers[4] = "Five";
foreach($numbers as $value ) {
echo "Array (As text) Value is $value <br />";
}
?>

```

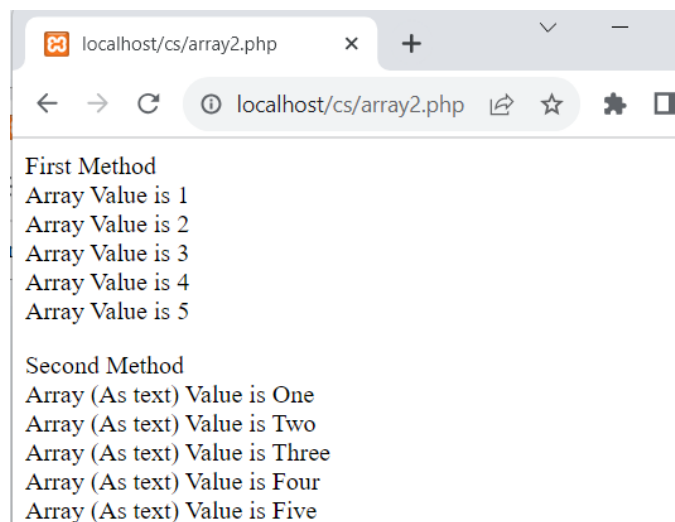


Figure 2: Output of numeric array

If we want separate index value form numeric array, then the following method can be used:  
 Let's assume array contain is text (Zero, One, Two, ...,Five).

```

<?php
$numbers = array ("ZERO","ONE", "TWO","THREE","FOUR","FIVE");
echo $numbers[2];
echo "<br>";
echo $numbers[5];
?>

```

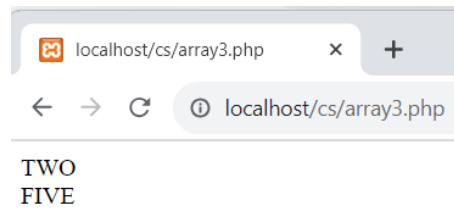


Figure 3: Output of the above code

This is the simplest way to implement numeric/indexed array in PHP.

**Note:** Array indices usually starts at 0.

## 1.5 Associative Array

The associative arrays are very similar to numeric arrays in terms of functionality, but they are different in terms of their index. Associative array will have their index as string so that you can establish a strong association between key and values.

Below is the syntax for creating associative array in php.

```
<?php
$variable_name['key_name'] = value;
$variable_name = array('keyname' => value);
?>
```

Explanation:

- “\$variable\_name...” is the name of the variable.
- “[key\_name]” is the access index number of the element.
- “value” is the value assigned to the array element.

Let’s suppose that we have a group of persons, and we want to assign the gender of each person against their names.

We can use an associative array to do that. The code below helps us to do that.

```
<?php
$persons = array("Lina" => " Female ", "Ashish" => "Male", "Riya" => "Female");
echo "<PRE>";
print_r($persons);
echo "</PRE>";
echo "Riya is a “. $persons["Riya"];
?>
```

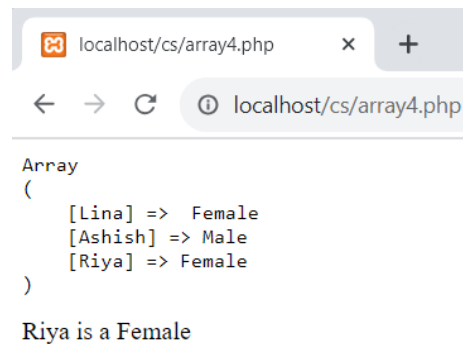


Figure 4: Output of the above code

Associative array is also very useful when retrieving data from the database. The field names are used as id keys.

## 1.6 Multi-Dimensional Arrays

In a multi-dimensional array, each element in the main array can also be an array. And each element in the sub-array can be an array, and so on. Values in the multi-dimensional array are accessed using multiple indices. Multi-dimensional contains other nested arrays.

The advantage of multi-dimensional arrays is that they allow us to group related data together.

Let's now look at a practical example that implements a PHP multi-dimensional array. Suppose we want to create a two-dimensional array to store marks of three students in three subjects. This example is an associative array. You can create numeric array in the same fashion.

```
<?php
$marks = array(
    "ASHISH" => array ("HTML" => 98, "JAVA" => 78, "PHP" => 96),
    "LINA" => array ( "HTML" => 75, "JAVA" => 86, "PHP" => 72 ),
    "RIYA" => array ( "HTML" =>rand(1,100), "JAVA" =>rand(1,100), "PHP" =>rand(1,100))
);
// In nested array of Riya's all marks are random range is 1 to 100
/* Accessing multi-dimensional array values */
echo "Marks for ASHISH in PHP: ";
echo $marks["ASHISH"]["PHP"] . "<br />";
echo "Marks for LINA in JAVA: ";
echo $marks["LINA"]["JAVA"] . "<br />";
echo "Marks for RIYA in HTML: ";
echo $marks["RIYA"]["HTML"] . "<br />";
?>
```

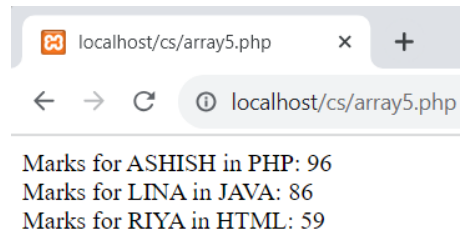


Figure 5 Output of the above code

Using this method data can be stored in multi-dimensional arrays, PHP understands multi-dimensional arrays that are two, three, four, five, or more levels deep. However, arrays more than three levels deep are hard to manage for most people.

## 1.7PHP Array Operators

The PHP array operators are used to compare arrays.

Operator	Name	Example	Result
+	Union	$\$x + \$y$	Union of $\$x$ and $\$y$
==	Equality	$\$x == \$y$	Returns true if $\$x$ and $\$y$ have the same key/value pairs
===	Identity	$\$x === \$y$	Returns true if $\$x$ and $\$y$ have the same key/value pairs in the same order and of the same types
!=	Inequality	$\$x != \$y$	Returns true if $\$x$ is not equal to $\$y$
<>	Inequality	$\$x <> \$y$	Returns true if $\$x$ is not equal to $\$y$
!==	Non-identity	$\$x !== \$y$	Returns true if $\$x$ is not identical to $\$y$

## 1.8Manipulating Arrays

### 1.8.1 Joining the Arrays

The best way to merge two or more arrays in PHP is to use the `array_merge()` function. Items of arrays will be merged together, and values with the same string keys will be overwritten with the last value:

```
<?php
$array1 = ['a' => 'a', 'b' => 'b', 'c' => 'c'];
$array2 = ['m' => 'M', 'n' => 'N', 'o' => 'O'];

echo "Array 1:-";
print_r($array1);
```

```

echo "<br>Array 2:-";
print_r($array2);

$merge = array_merge($array1, $array2);

echo "<br><hr> Result after mearge of Array1 & Array2";

echo "<pre>";
print_r($merge);
echo "</pre>";
?>

```

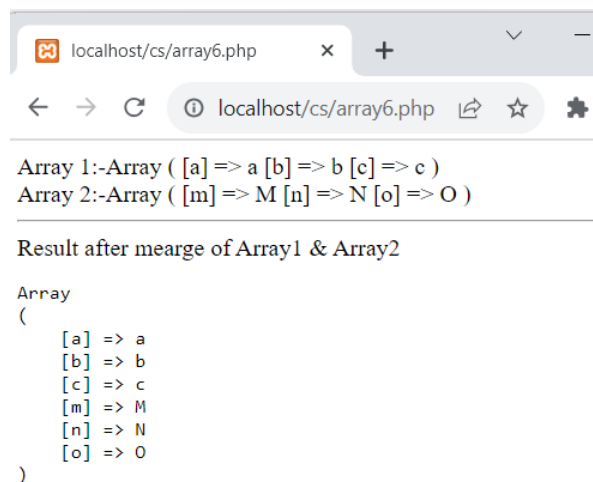


Figure 6: output of the above code

To remove array values from another array (or arrays), use `array_diff()`. This function compares the values of two (or more) arrays, and return an array that contains the entries from array1 that are not present in array2 or array3, etc.

To get values which are present in given arrays, use `array_intersect()`. This function compares the values of two arrays and returns the matches.

The next examples will show how it works:



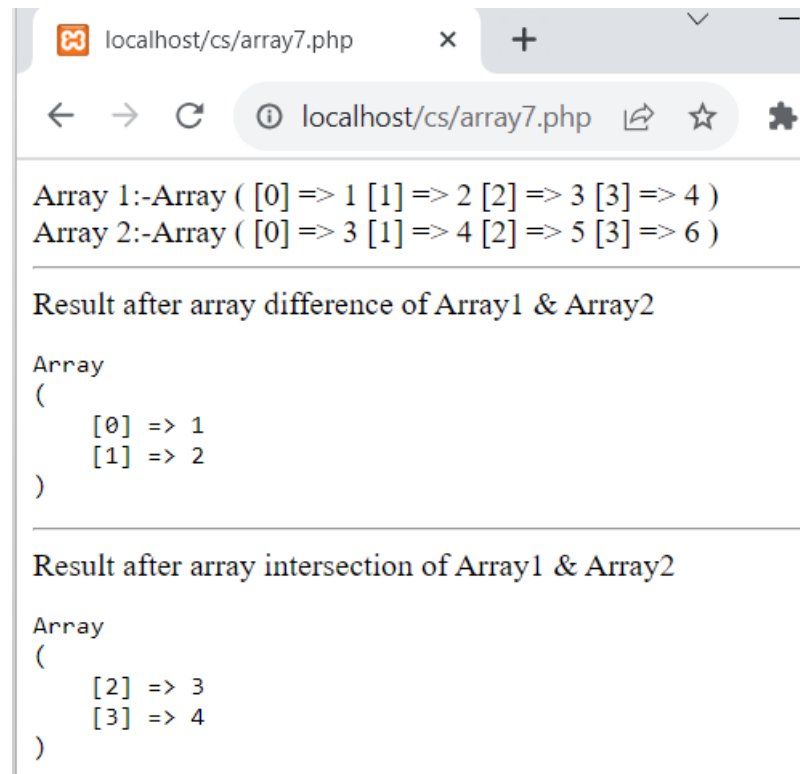
```

<?php
$array1 = [1, 2, 3, 4];
$array2 = [3, 4, 5, 6];
echo "Array 1:-";
print_r($array1);
echo "<br>Array 2:-";
print_r($array2);

echo "<br><hr> Result after array difference of Array1 & Array2";
$diff = array_diff($array1, $array2);
echo "<pre>";
print_r($diff); // [0 => 1, 1 => 2]
echo "</pre>";

echo "<hr> Result after array intersection of Array1 & Array2";
$intersect = array_intersect($array1, $array2);
echo "<pre>";
print_r($intersect); // [2 => 3, 3 => 4]
echo "</pre>";
?>

```



```

localhost/cs/array7.php
Array 1:-Array ( [0] => 1 [1] => 2 [2] => 3 [3] => 4 )
Array 2:-Array ( [0] => 3 [1] => 4 [2] => 5 [3] => 6 )

Result after array difference of Array1 & Array2

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

Result after array intersection of Array1 & Array2

Array
(
    [2] => 3
    [3] => 4
)

```

Figure 7: Output for the above code

### **sizeof(\$arr):**

```
<?php
$data = array("red", "green", "blue");
echo "Array has " . sizeof($data) . " elements";
?>
```

Output: Array has 3 elements

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

This function accepts a PHP array and returns a new array containing only its values (not its keys). Its counterpart is the array\_keys() function.

Use this function to retrieve all the values from an associative array.

```
<?php
$data = array("hero" => "Holmes", "villain" => "Moriarty");
print_r(array_values($data));
?>
```

Output:

Array ([0] => Holmes [1] => Moriarty)

### **array\_keys(\$arr):**

This function accepts a PHP array and returns a new array containing only its keys (not its values). Its counterpart is the array\_values() function.

Use this function to retrieve all the keys from an associative array.

```
<?php
$data = array("hero" => "Holmes", "villain" => "Moriarty");
print_r(array_keys($data));
?>
```

Output:

Array ([0]=>hero [1]=>villain)

**array\_pop(\$arr):**

This function removes an element from the end of an array.

```
<?php
$data = array("Donald", "Jim", "Tom");
array_pop($data);
print_r($data);
?>
```

Output:

```
Array ([0] => Donald [1] => Jim)
```

**array\_push(\$arr, \$val):**

This function adds an element to the end of an array.

```
<?php
$data = array("Donald", "Jim", "Tom");
array_push($data, "Harry");
print_r($data);
?>
```

Output:

```
Array ([0] => Donald [1] => Jim [2] => Tom [3] => Harry)
```

**array\_shift(\$arr):**

This function removes an element from the beginning of an array.

```
<?php
$data = array("Donald", "Jim", "Tom");
array_shift($data);
print_r($data);
?>
```

Output:

```
Array ([0] => Jim [1] => Tom)
```

**array\_unshift(\$arr, \$val):**

This function adds an element to the beginning of an array.

```
<?php
$data = array("Donald", "Jim", "Tom");
array_unshift($data, "Sarah");
print_r($data);
?>
```

Output:

```
Array ( [0] => Sarah [1] => Donald [2] => Jim [3] => Tom )
```

**sort(\$arr):**

This function sorts the elements of an array in ascending order. String values will be arranged in ascending alphabetical order.

Note: Other sorting functions include `asort()`, `arsort()`, `ksort()`, `krsort()` and `rsort()`.

- `sort()` - sort arrays in ascending order
- `rsort()` - sort arrays in descending order
- `asort()` - sort associative arrays in ascending order, according to the value
- `ksort()` - sort associative arrays in ascending order, according to the key
- `arsort()` - sort associative arrays in descending order, according to the value
- `krsort()` - sort associative arrays in descending order, according to the key

Example:

```
<?php
$data = array("g", "t", "a", "s");
sort($data);
print_r($data);
?>
```

Output:

```
Array ([0] => a [1] => g [2] => s [3] => t )
```

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

The function exchanges the keys and values of a PHP associative array.

Use this function if you have a tabular (rows and columns) structure in an array, and you want to interchange the rows and columns.

```
<?php
$data = array("a" => "apple", "b" => "ball");
print_r(array_flip($data));
?>
```

#### **Output:**

Array ([apple] => a [ball] => b)

### **array\_reverse(\$arr):**

The function reverses the order of elements in an array.

Use this function to re-order a sorted list of values in reverse for easier processing— for example, when you're trying to begin with the minimum or maximum of a set of ordered values.

Example:

```
<?php
$data = array(10, 20, 25, 60);
print_r(array_reverse($data));
?>
```

Output:

Array ([0] => 60 [1] => 25 [2] => 20 [3] => 10)

### **array\_merge(\$arr):**

This function merges two or more arrays to create a single composite array. Key collisions are resolved in favor of the latest entry.

Use this function when you need to combine data from two or more arrays into a single structure— for example, records from two different SQL queries.

Example:

```
<?php
$data1 = array("cat", "goat");
$data2 = array("dog", "cow");
print_r(array_merge($data1, $data2));
?>
```

Output:

Array ([0] => cat [1] => goat [2] => dog [3] => cow)

### **array\_rand(\$arr):**

This function selects one or more random elements from an array.

Use this function when you need to randomly select from a collection of discrete values. For example, picking a random color from a list.

```
<?php
$data = array("white", "black", "red");
echo "Today's color is " . $data[array_rand($data)];
?>
```

Output:

Today's color is red

### **array\_search(\$search, \$arr):**

This function searches the values in an array for a match to the search term, and returns the corresponding key if found. If more than one match exists, the key of the first matching value is returned.

Use this function to scan a set of index-value pairs for matches and return the matching index.

```
<?php
$data = array("blue" => "#0000cc", "black" => "#000000", "green" => "#00ff00");
echo "Found " . array_search("#0000cc", $data);
?>
```

Output:

Found blue

### **array\_slice(\$arr, \$offset, \$length):**

This function is useful to extract a subset of the elements of an array, as another array. Extracting begins from array offset \$offset and continues until the array slice is \$length elements long.

Use this function to break a larger array into smaller ones—for example, when segmenting an array by size ("chunking") or type of data.

```
<?php
$data = array("vanilla", "strawberry", "mango", "peaches");
print_r(array_slice($data, 1, 2));
?>
```

Output:

Array ([0] => strawberry [1] => mango)

**array\_unique(\$data):**

This function strips an array of duplicate values.

Use this function when you need to remove non-unique elements from an array—for example, when creating an array to hold values for a table's primary key.

```
<?php
$data = array(1,1,4,6,7,4);
print_r(array_unique($data));
?>
```

Output:

```
Array ([0] => 1 [2] => 4 [3] => 6 [4] => 7)
```

**array\_walk(\$arr, \$func):**

This function "walks" through an array, applying a user-defined function to every element. It returns the changed array.

Use this function if you need to perform custom processing on every element of an array—for example, reducing a series of number by 10%.

```
<?php
function reduceBy10(&$val, $key) {
    $val -= $val * 0.1;
}
$data = array(10,20,30,40);
array_walk($data, 'reduceBy10');
print_r($data);
?>
```

Output:

```
Array ( [0] => 9 [1] => 18 [2] => 27 [3] => 36 )
```

## PHP ARRAY CONSTANTS

These functions allow you to interact with and manipulate arrays in various ways. Arrays are essential for storing, managing, and operating on sets of variables.

Constant	Description
<b>CASE_LOWER</b>	Used with <code>array_change_key_case()</code> to convert arraykeys to lower case
<b>CASE_UPPER</b>	Used with <code>array_change_key_case()</code> to convert array keys to upper case
<b>SORT_ASC</b>	Used with <code>array_multisort()</code> to sort in ascending order
<b>SORT_DESC</b>	Used with <code>array_multisort()</code> to sort in descending order
<b>SORT_REGULAR</b>	Used to compare items normally
<b>SORT_NUMERIC</b>	Used to compare items numerically
<b>SORT_STRING</b>	Used to compare items as strings
<b>SORT_LOCALE_STRING</b>	Used to compare items as strings, based on the current locale

And other normal constants are as below, for more detail and detail example you can refer [php.net](http://php.net)

- `COUNT_NORMAL`
- `COUNT_RECURSIVE`
- `EXTR_OVERWRITE`
- `EXTR_SKIP`
- `EXTR_PREFIX_SAME`
- `EXTR_PREFIX_ALL`
- `EXTR_PREFIX_INVALID`
- `EXTR_PREFIX_IF_EXISTS`
- `EXTR_IF_EXISTS`
- `EXTR_REFS`



## LIST OF ARRAY FUNCTIONS

Function	Description
array()	Creates an array
array_change_key_case()	Changes all keys in an array to lowercase or uppercase
array_chunk()	Splits an array into chunks of arrays
array_column()	Returns the values from a single column in the input array
array_combine()	Creates an array by using the elements from one "keys" array and one "values" array
array_count_values()	Counts all the values of an array
array_diff()	Compare arrays, and returns the differences (compare values only)
array_diff_assoc()	Compare arrays, and returns the differences (compare keys and values)
array_diff_key()	Compare arrays, and returns the differences (compare keys only)
array_diff_uassoc()	Compare arrays, and returns the differences (compare keys and values, using a user-defined key comparison function)
array_diff_ukey()	Compare arrays, and returns the differences (compare keys only, using a user-defined key comparison function)
array_fill()	Fills an array with values
array_fill_keys()	Fills an array with values, specifying keys
array_filter()	Filters the values of an array using a callback function
array_flip()	Flips/Exchanges all keys with their associated values in an array
array_intersect()	Compare arrays, and returns the matches (compare values only)
array_intersect_assoc()	Compare arrays and returns the matches (compare keys and values)
array_intersect_key()	Compare arrays, and returns the matches (compare keys only)
array_intersect_uassoc()	Compare arrays, and returns the matches (compare keys and values, using a user-defined key comparison function)
array_intersect_ukey()	Compare arrays, and returns the matches (compare keys only, using a user-defined key comparison function)
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 recursively
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_replace()	Replaces the values of the first array with the values from following arrays
array_replace_recursive()	Replaces the values of the first array with the values from following arrays recursively
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()	Compare arrays, and returns the differences (compare values only, using a user-defined key comparison function)
array_udiff_assoc()	Compare arrays, and returns the differences (compare keys and values, using a built-in function to compare the keys and a user-defined function to compare the values)
array_udiff_uassoc()	Compare arrays, and returns the differences (compare keys and values, using two user-defined key comparison functions)
array_uintersect()	Compare arrays, and returns the matches (compare values only, using a user-defined key comparison function)
array_uintersect_assoc()	Compare arrays, and returns the matches (compare keys and values, using a built-in function to compare the keys and a user-defined function to compare the values)
array_uintersect_uassoc()	Compare arrays, and returns the matches (compare keys and values, using two user-defined key comparison functions)
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 associative array in descending order, according to the value
asort()	Sorts an associative array in ascending order, according to the value
compact()	Create array containing variables and their values
count()	Returns the number of elements in an array
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 associative array in descending order, according to the key
ksort()	Sorts an associative array in ascending order, according to the 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 indexed array in descending order
shuffle()	Shuffles an array
sizeof()	Alias of count()
sort()	Sorts an indexed array in ascending order
uasort()	Sorts an array by values using a user-defined comparison function
uksort()	Sorts an array by keys using a user-defined comparison function
usort()	Sorts an array using a user-defined comparison function