What is PHP?

* PHP is an acronym for "PHP: Hypertext Preprocessor"
* PHP is a widely-used, open source scripting language
* PHP scripts are executed on the server
* PHP is free to download and use

What is a PHP File?

* PHP files can contain text, HTML, CSS, JavaScript, and PHP code
* PHP code is executed on the server, and the result is returned to the browser as plain HTML
* PHP files have extension ".php"

What Can PHP Do?

* PHP can generate dynamic web page content
* PHP can create, open, read, write, delete, and close files on the server
* PHP can collect form data
* PHP can send and receive cookies
* PHP can add, delete, modify data in your database
* PHP can be used to control user-access
* PHP can encrypt data

## PHP Structure

<?php echo "Hello Everyone"; ?>

### with HTML

<html>

<head>

<title>PHP</title>

</head>

<body>

<?php echo "Hello Everyone"; ?>

</body>

</html>

## PHP Echo/Print

### PHP echo and print Statements

echo and print are more or less the same. They are both used to output data to the screen.

The differences are small: echo has no return value while print has a return value of 1 so it can be used in expressions. echo can take multiple parameters (although such usage is rare) while print can take one argument. echo is marginally faster than print.

### The PHP echo Statement

The echo statement can be used with or without parentheses: echo or echo().

### Display Text

The following example shows how to output text with the echo command (notice that the text can contain HTML markup):

echo "<h2>PHP is Fun!</h2>";

echo "Hello world!<br>";

echo "I'm about to learn PHP!<br>";

echo "This ", "string ", "was ", "made ", "with multiple parameters.";

### Display Variables

The following example shows how to output text and variables with the echo statement:

Example

$txt1 = "Learn PHP";

$txt2 = "W3Schools.com";

$x = 5;

$y = 4;

echo "<h2>" . $txt1 . "</h2>";

echo "Study PHP at " . $txt2 . "<br>";

echo $x + $y;

### The PHP print Statement

The print statement can be used with or without parentheses: print or print().

### Display Text

The following example shows how to output text with the print command (notice that the text can contain HTML markup):

Example

print "<h2>PHP is Fun!</h2>";

print "Hello world!<br>";

print "I'm about to learn PHP!";

### Display Variables

The following example shows how to output text and variables with the print statement:

Example

$txt1 = "Learn PHP";

$txt2 = "W3Schools.com";

$x = 5;

$y = 4;

print "<h2>" . $txt1 . "</h2>";

print "Study PHP at " . $txt2 . "<br>";

print $x + $y;

<?php

echo "LBSTI";

echo 'LBSTI';

echo ('LBSTI');

echo "Yahoo","Baba";

echo "Yahoo"."Baba";

echo "<h1><i>Yahoo"."Baba</i></h1>";

echo 23.58;

print "<h1><i>Yahoo"."Baba</i></h1>";

print 23.58;

?>

## PHP Variables

Variables are "containers" for storing information.

Creating (Declaring) PHP Variables

In PHP, a variable starts with the $ sign, followed by the name of the variable:  
  
Example

$x = 5;

$y = "John"

In the example above, the variable $x will hold the value 5, and the variable $y will hold the value "John".

PHP Variables

A variable can have a short name (like $x and $y) or a more descriptive name ($age, $carname, $total\_volume).

**Rules for PHP variables:**

* A variable starts with the $ sign, followed by the name of the variable
* A variable name must start with a letter or the underscore character
* A variable name cannot start with a number
* A variable name can only contain alpha-numeric characters and underscores (A-z, 0-9, and \_ )
* Variable names are case-sensitive ($age and $AGE are two different variables)

Output Variables

The PHP echo statement is often used to output data to the screen.

The following example will show how to output text and a variable:  
  
Example

$txt = "W3Schools.com";

echo "I love $txt!";

The following example will output the sum of two variables:

Example

$x = 5;

$y = 4;

echo $x + $y;

Assign String to a Variable

Assigning a string to a variable is done with the variable name followed by an equal sign and the string:

Example

$x = "John";

echo $x;

Assign Multiple Values

You can assign the same value to multiple variables in one line:

Example

All three variables get the value "Fruit":

$x = $y = $z = "Fruit";

<?php

$name = "LBSTI<br>";

$num = 2358;

echo $name;

echo "<h1>" . $name . "</h1>";

echo "Hello how are you : " . $name ;

echo $num;

?>

## PHP Data Types

Variables can store data of different types, and different data types can do different things.

PHP supports the following data types:

* String
* Integer
* Float (floating point numbers - also called double)
* Boolean
* Array
* Object
* NULL
* Resource

Getting the Data Type

You can get the data type of any object by using the var\_dump() function.

Example

The var\_dump() function returns the data type and the value:

$x = 5;

var\_dump($x);

PHP String

A string is a sequence of characters, like "Hello world!".

A string can be any text inside quotes. You can use single or double quotes:

Example

$x = "Hello world!";

$y = 'Hello world!';

var\_dump($x);

echo "<br>";

var\_dump($y);

PHP Integer

An integer data type is a non-decimal number between -2,147,483,648 and 2,147,483,647.

Rules for integers:

* An integer must have at least one digit
* An integer must not have a decimal point
* An integer can be either positive or negative
* Integers can be specified in: decimal (base 10), hexadecimal (base 16), octal (base 8), or binary (base 2) notation

In the following example $x is an integer. The PHP var\_dump() function returns the data type and value:

Example

$x = 5985;

var\_dump($x);

PHP Float

A float (floating point number) is a number with a decimal point or a number in exponential form.

In the following example $x is a float. The PHP var\_dump() function returns the data type and value:

Example

$x = 10.365;

var\_dump($x);

PHP Boolean

A Boolean represents two possible states: TRUE or FALSE.

Example

$x = true;

var\_dump($x);

PHP Array

An array stores multiple values in one single variable.

In the following example $cars is an array. The PHP var\_dump() function returns the data type and value:

Example

$cars = array("Volvo","BMW","Toyota");

var\_dump($cars);

PHP NULL Value

Null is a special data type which can have only one value: NULL.

A variable of data type NULL is a variable that has no value assigned to it.

**Tip:** If a variable is created without a value, it is automatically assigned a value of NULL.

Variables can also be emptied by setting the value to NULL:

Example

$x = "Hello world!";

$x = null;

var\_dump($x);

<?php

$x = "LBSTI";

$x = 2500;

$x = 2500.50;

$x = true;

$x = array("html","css","js");

$x = null;

echo $x . "<br>";

echo $x[0] . "<br>";

var\_dump($x);

?>

## PHP Comment

A comment in PHP code is a line that is not executed as a part of the program. Its only purpose is to be read by someone who is looking at the code.

Comments can be used to:

* Let others understand your code
* Remind yourself of what you did - Most programmers have experienced coming
* back to their own work a year or two later and having to re-figure out what they did. Comments can remind you of what you were thinking when you wrote the code
* Leave out some parts of your code

PHP supports several ways of commenting:

Example

Syntax for comments in PHP code:

// This is a single-line comment

# This is also a single-line comment

/\* This is a

multi-line comment \*/

Single Line Comments

Single line comments start with //.

Any text between // and the end of the line will be ignored (will not be executed).

You can also use # for single line comments, but in this tutorial we will use //.

The following examples uses a single-line comment as an explanation:

Example

A comment before the code:

// Outputs a welcome message:

echo "Welcome Home!";

Multi-line Comments

Multi-line comments start with /\* and end with \*/.

Any text between /\* and \*/ will be ignored.

The following example uses a multi-line comment as an explanation:

Example

Multi-line comment as an explanation:

/\*

The next statement will

print a welcome message

\*/

echo "Welcome Home!";

Comments in the Middle of the Code

The multi-line comment syntax can also be used to prevent execution of parts inside a code-line:

Example

The  + 15 part will be ignored in the calculation:

$x = 5 /\* + 15 \*/ + 5;

echo $x;

<?php

/\* Single Line Comment \*/

$x = "LBSTI"; //this is first comment

echo $x;

/\* Multiple Line Comment \*/

/\* $x = "LBSTI";

echo $x; \*/

?>

## PHP Constants

A constant is an identifier (name) for a simple value. The value cannot be changed during the script.

A valid constant name starts with a letter or underscore (no $ sign before the constant name).

**Note:** Unlike variables, constants are automatically global across the entire script.

Create a PHP Constant

To create a constant, use the define() function.

Syntax

define(name, value, case-insensitive);

define(*name*, *value*, *case-insensitive*);

Parameters:

* *name*: Specifies the name of the constant
* *value*: Specifies the value of the constant
* *case-insensitive*: Specifies whether the constant name should be case-insensitive. Default is false. **Note:** Defining case-insensitive constants was deprecated in PHP 7.3. PHP 8.0 accepts only false, the value true will produce a warning.

Example

Create a constant with a **case-sensitive** name:

define("GREETING", "Welcome to W3Schools.com!");

echo GREETING;

Example

Create a constant with a **case-insensitive** name:

define("GREETING", "Welcome to W3Schools.com!", true);

echo greeting;

<?php

define("test",50);

echo test;

define("test1",50,true);

echo TEST1;

$sum = test + 20;

echo $sum;

?>

### PHP Arithmetic Operators

The PHP arithmetic operators are used with numeric values to perform common arithmetical operations, such as addition, subtraction, multiplication etc.

|  |  |  |  |
| --- | --- | --- | --- |
| **Operator** | **Name** | **Example** | **Result** |
| + | Addition | $x + $y | Sum of $x and $y |
| - | Subtraction | $x - $y | Difference of $x and $y |
| \* | Multiplication | $x \* $y | Product of $x and $y |
| / | Division | $x / $y | Quotient of $x and $y |
| % | Modulus | $x % $y | Remainder of $x divided by $y |
| \*\* | Exponentiation | $x \*\* $y | Result of raising $x to the $y'th power |

<?php

$a = 10;

$b = 3;

$c = $a + $b;

$c = $a - $b;

$c = $a \* $b;

$c = $a / $b;

$c = $a % $b;

$c = $a \*\* $b;

$c = $a % $b;

$a++;

++$a;

$a--;

$a;

$c = ($a + $b) \* 2;

echo $c;

?>

### PHP Assignment Operators

The PHP assignment operators are used with numeric values to write a value to a variable.

The basic assignment operator in PHP is "=". It means that the left operand gets set to the value of the assignment expression on the right.

|  |  |  |
| --- | --- | --- |
| **Assignment** | **Same as...** | **Description** |
| x = y | x = y | The left operand gets set to the value of the expression on the right |
| x += y | x = x + y | Addition |
| x -= y | x = x - y | Subtraction |
| x \*= y | x = x \* y | Multiplication |
| x /= y | x = x / y | Division |
| x %= y | x = x % y | Modulus |

<?php

$a = 10;

$b = 3;

$a = $a + $b;

$a += $b;

$a -= $b;

$a \*= $b;

$a /= $b;

$a %= $b;

$a \*\*= $b;

echo $a;

?>

### PHP Comparison Operators

The PHP comparison operators are used to compare two values (number or string):

|  |  |  |  |
| --- | --- | --- | --- |
| **Operator** | **Name** | **Example** | **Result** |
| == | Equal | $x == $y | Returns true if $x is equal to $y |
| === | Identical | $x === $y | Returns true if $x is equal to $y, and they are of the same type |
| != | Not equal | $x != $y | Returns true if $x is not equal to $y |
| <> | Not equal | $x <> $y | Returns true if $x is not equal to $y |
| !== | Not identical | $x !== $y | Returns true if $x is not equal to $y, or they are not of the same type |
| > | Greater than | $x > $y | Returns true if $x is greater than $y |
| < | Less than | $x < $y | Returns true if $x is less than $y |
| >= | Greater than or equal to | $x >= $y | Returns true if $x is greater than or equal to $y |
| <= | Less than or equal to | $x <= $y | Returns true if $x is less than or equal to $y |
| <=> | Spaceship | $x <=> $y | Returns an integer less than, equal to, or greater than zero, depending on if $x is less than, equal to, or greater than $y. Introduced in PHP 7. |

<?php

$a = 10;

$b = 10;

echo $a == $b;

echo $a === $b;

echo $a != $b;

echo $a <> $b;

echo $a !== $b;

echo $a > $b;

echo $a < $b;

echo $a >= $b;

echo $a <= $b;

echo $a <=> $b; //Spaceship (-1,0,1)

?>

## PHP If

### PHP Conditional Statements

Very often when you write code, you want to perform different actions for different conditions. You can use conditional statements in your code to do this.

In PHP we have the following conditional statements:

* if statement - executes some code if one condition is true
* [if...else](https://www.w3schools.com/php/php_if_else_if.asp) statement - executes some code if a condition is true and another code if that condition is false
* [if...elseif...else](https://www.w3schools.com/php/php_if_else_if.asp#php_if_elseif_else) statement - executes different codes for more than two conditions
* [switch](https://www.w3schools.com/php/php_switch.asp) statement - selects one of many blocks of code to be executed

### PHP - The if Statement

The if statement executes some code if one condition is true.

### Syntax

if (condition) {

// code to be executed if condition is true;

}

Example

Output "Have a good day!" if 5 is larger than 3:

if (5 > 3) {

echo "Have a good day!";

}

<?php

$a = 3;

$b = 10;

if($a < $b){

echo "A is Smaller" ;

}

echo "Here is other statement";

if($a == $b){

echo "A is Smaller" ;

}

echo "Here is other statement";

if($a === $b){

echo "A is Smaller" ;

}

echo "Here is other statement";

if($a == $b):

echo "A is Smaller<br>" ;

echo "A is Smaller<br>" ;

endif;

echo "Here is other statement";

?>

### PHP Logical Operators

The PHP logical operators are used to combine conditional statements.

|  |  |  |  |
| --- | --- | --- | --- |
| **Operator** | **Name** | **Example** | **Result** |
| and | And | $x and $y | True if both $x and $y are true |
| or | Or | $x or $y | True if either $x or $y is true |
| xor | Xor | $x xor $y | True if either $x or $y is true, but not both |
| && | And | $x && $y | True if both $x and $y are true |
| || | Or | $x || $y | True if either $x or $y is true |
| ! | Not | !$x | True if $x is not true |

<?php

$age = 20;

/\* Logical And Operator \*/

if($age >= 18 && $age <= 21){

echo "You are eligible.<br>";

}

echo "Here is other statement";

/\* Logical And Operator\*/

if($age >= 18 and $age <= 21){

echo "You are eligible.<br>";

}

echo "Here is other statement";

/\* Logical Or Operator\*/

if($age >= 18 || $age <= 21){

echo "You are eligible.<br>";

}

/\* Logical Not Operator\*/

if(!($age >= 18)){

echo "You are eligible.<br>";

}

/\* Logical xor Operator\*/

if($age >= 18 xor $age <= 21){

echo "You are eligible.<br>";

}

?>

### PHP - The if...else Statement

The if...else statement executes some code if a condition is true and another code if that condition is false.

### Syntax

if (condition) {

// code to be executed if condition is true;

} else {

// code to be executed if condition is false;

}

Example

Output "Have a good day!" if the current time is less than 20, and "Have a good night!" otherwise:

$t = date("H");

if ($t < "20") {

echo "Have a good day!";

} else {

echo "Have a good night!";

}

<?php

$x = 15;

if($x > 30){

echo "X is Greater.";

}else{

echo "X is Smaller.";

}

$x = 100;

if($x == 100){

echo "X is Same.";

}else{

echo "X is not Same.";

}

$name = "LBSTI";

$gender = "male";

if($gender == "male"){

echo "Hello Mr.". $name;

}else{

echo "Hello Miss.". $name;

}

?>

PHP - The if...elseif...else Statement

The if...elseif...else statement executes different codes for more than two conditions.

Syntax

if (condition) {

code to be executed if this condition is true;

} elseif (condition) {

// code to be executed if first condition is false and this condition is true;

} else {

// code to be executed if all conditions are false;

}

<?php

$per = 47;

if($per >= 80 && $per <= 100){

echo "You are in Merit.";

} elseif($per >= 60 && $per < 80){

echo "You are in Ist Division.";

} elseif($per >= 45 && $per < 60){

echo "You are in IInd Division.";

} elseif($per >= 33 && $per < 45){

echo "You are in IIIrd Division.";

} elseif($per < 33){

echo "You are Fail.";

} else{

echo "Please Enter Valid Percentage.";

}

if($per >= 80 && $per <= 100):

echo "You are in Merit.";

elseif($per >= 60 && $per < 80):

echo "You are in Ist Division.";

elseif($per >= 45 && $per < 60):

echo "You are in IInd Division.";

elseif($per >= 33 && $per < 45):

echo "You are in IIIrd Division.";

elseif($per < 33):

echo "You are Fail.";

else:

echo "Please Enter Valid Percentage.";

endif;

?>

### The PHP switch Statement

Use the switch statement to **select one of many blocks of code to be executed**.

### Syntax

switch (expression) {

case label1:

//code block

break;

case label2:

//code block;

break;

case label3:

//code block

break;

default:

//code block

}

This is how it works:

* The expression is evaluated once
* The value of the expression is compared with the values of each case
* If there is a match, the associated block of code is executed
* The break keyword breaks out of the switch block
* The default code block is executed if there is no match

<?php

$weekday = 7;

switch($weekday){

case 1:

echo "Today is Monday";

break;

case 2:

echo "Today is Tuesday";

break;

case 3:

echo "Today is Wednesday";

break;

case 4:

echo "Today is Thursday";

break;

case 5:

echo "Today is Friday";

break;

case 6:

echo "Today is Saturday";

break;

case 7:

echo "Today is Sunday";

break;

default:

echo "Enter the correct weekday.";

}

$weekday = 3;

switch($weekday){

case 1 : case 2 : case 3 :

echo "Today is Monday";

echo "<br>This is just test.";

break;

case 4 :

echo "Today is Thursday";

break;

case 5 :

echo "Today is Friday";

break;

case 6 :

echo "Today is Saturday";

break;

case 7 :

echo "Today is Sunday";

break;

default :

echo "Enter the correct weekday.";

}

$age = 18;

switch($age){

case ($age >= 15 && $age <=20) :

echo "You are eligible.";

break;

case ($age >= 20 && $age <= 30) :

echo "You are not eligible.";

break;

default :

echo "Enter the valid age.";

}

?>

### Short Hand If...Else (Ternary Operator)

There is also a short-hand if else, which is known as the **ternary operator** because it consists of three operands. It can be used to replace multiple lines of code with a single line. It is often used to replace simple if else statements:

### Syntax

variable = (condition) ? expressionTrue : expressionFalse;

<?php

$x = 10;

($x > 20) ? $z = "Greater" : $z = "Smaller";

$z = ($x > 20) ? "Greater" : "Smaller";

$z = "Value is " . ($x > 20 ? "Greater" : "Smaller");

echo $z;

?>

Strings

A string is a sequence of characters, like "Hello world!".

Strings in PHP are surrounded by either double quotation marks, or single quotation marks.

echo "Hello";

echo 'Hello';

Double or Single Quotes?

You can use double or single quotes, but you should be aware of the differences between the two.

Double quoted strings perform action on special characters.

E.g. when there is a variable in the string, it returns the *value* of the variable:

Example

Double quoted string literals perform operations for special characters:

$x = "John";

echo "Hello $x";

<?php

$a = "Hello" ;

$s = $a . " World ";

$a = 200;

$s = $a . " World " . 500;

$s = "Hello ";

$s .= " this is";

$s .= " our world";

$s .= 555;

echo $s;

?>

### PHP while Loop

PHP while Loop

The while loop executes a block of code as long as the specified condition is true.

Example

Print $i as long as $i is less than 6:

$i = 1;

while ($i < 6) {

echo $i;

$i++;

}

The while loop does not run a specific number of times, but checks after each iteration if the condition is still true.

The condition does not have to be a counter, it could be the status of an operation or any condition that evaluates to either true or false.

<?php

$a = 1;

while($a <= 10){

echo "Hello LBSTI<br>";

$a = $a + 1; //Increment Loop

}

while($a <= 20){

echo $a . ") Hello LBSTI<br>";

$a = $a++; //counting

}

$a = 10;

while($a >= 1){

echo $a . ") Hello LBSTI<br>";

$a--; //Decrement Loop

}

echo "<ul>";

while($a >= 1){

echo "<li>".$a . ") Hello LBSTI</li>";

$a = $a - 1;

}

echo "</ul>";

while($a <= 10){

echo "Hello LBSTI<br>";

$a = $a + 2; //increament of 2 or 3

}

?>

### The PHP do...while Loop

The do...while loop will always execute the block of code at least once, it will then check the condition, and repeat the loop while the specified condition is true.

### Example

Print $i as long as $i is less than 6:

$i = 1;

do {

echo $i;

$i++;

} while ($i < 6);

<?php

$a = 1;

do{

echo $a .") Hello LBSTI<br>";

$a++; //Increment Loop

}while($a <= 10)

$a = 10;

do{

echo $a .") Hello LBSTI<br>";

$a--; //Decrement Loop

}while($a <= 1)

?>

### PHP for Loop

The for loop is used when you know how many times the script should run.

### Syntax

for (expression1, expression2, expression3) {

// code block

}

This is how it works:

* expression1 is evaluated once
* expression2 is evaluated before each iterarion
* expression3 is evaluated after each iterarion

<?php

//Increment

for($a = 1; $a <= 10; $a= $a++ ){

echo $a .") Hello LBSTI<br>";

}

//Decrement

for($a = 10; $a >= 1; $a= $a-- ){

echo $a .") Hello LBSTI<br>";

}

?>

### Nested Loop in PHP

The nested loop is the method of using the for loop inside the another for loop. It first performs a single iteration for the parent for loop and executes all the iterations of the inner loop. After that, it again checks the parent iteration and again performs all the iteration of the inner loop. The same process continuously executed until the parent test condition is TRUE.

<?php

for($a = 1; $a <= 100; $a = $a + 10){

for($b = $a; $b < $a + 10; $b++){

echo $b . " ";

}

echo "<br>";

}

?>

## PHP Continue & Break

Continue in For Loops

The continue statement stops the current iteration in the for loop and continue with the next.

Example

Move to next iteration if $x = 4:

for ($x = 0; $x < 10; $x++) {

if ($x == 4) {

continue;

}

echo "The number is: $x <br>";

}

Continue in While Loop

The continue statement stops the current iteration in the while loop and continue with the next.  
  
Example

Move to next iteration if $x = 4:

$x = 0;

while($x < 10) {

if ($x == 4) {

continue;

}

echo "The number is: $x <br>";

$x++;

}

Break in For loop

The break statement can be used to jump out of a for loop.

Example

Jump out of the loop when $x is 4:

for ($x = 0; $x < 10; $x++) {

if ($x == 4) {

break;

}

echo "The number is: $x <br>";

}

Break in While Loop

The break statement can be used to jump out of a while loop.

 Example

$x = 0;

while($x < 10) {

if ($x == 4) {

break;

}

echo "The number is: $x <br>";

$x++;

}

<?php

for ($a = 1; $a < 10; $a++) {

if ($a == 3){

//echo "No. : " . $a . "<br>";

continue;

}

echo "Number : " . $a . "<br>";

}

for ($a = 1; $a < 10; $a++) {

if ($a == 3){

//echo "No. : " . $a . "<br>";

break;

}

echo "Number : " . $a . "<br>";

}

?>

### The goto statement

The **goto** statement is used to jump to another section of a program. It is sometimes referred to as an unconditional jump statement. The goto statement can be used to jump from anywhere to anywhere within a function.

### Syntax:

statement\_1;

if (expr)

goto label;

statement\_2;

statement\_3;

label: statement\_4;

<?php

for($a = 1; $a <= 10; $a++){

if($a == 3){

goto a;

}

echo "Number : " . $a . "<br>";

}

echo "Hello";

echo " World";

a:

echo "Here is label A.";

?>

PHP Functions

The real power of PHP comes from its functions.

PHP has more than 1000 built-in functions, and in addition you can create your own custom functions.

PHP Built-in Functions

PHP has over 1000 built-in functions that can be called directly, from within a script, to perform a specific task.

PHP User Defined Functions

Besides the built-in PHP functions, it is possible to create your own functions.

* A function is a block of statements that can be used repeatedly in a program.
* A function will not execute automatically when a page loads.
* A function will be executed by a call to the function.

Create a Function

A user-defined function declaration starts with the keyword function, followed by the name of the function:

Example

function myMessage() {

echo "Hello world!";

}

Call a Function

To call the function, just write its name followed by parentheses ():

Example

function myMessage() {

echo "Hello world!";

}

myMessage();

<?php

function hello(){

echo "Hello Everybody.<br>";

}

function yahoo(){

echo "Hello Yahoo baba.<br>";

}

hello();

hello();

yahoo();

echo "Hey this is an example.";

hello();

hello();

?>

[Previous](https://www.yahubaba.com/php/php-continue-break)[Next](https://www.yahubaba.com/php/php-functions)

### PHP Function Arguments

Information can be passed to functions through arguments. An argument is just like a variable.

Arguments are specified after the function name, inside the parentheses. You can add as many arguments as you want, just separate them with a comma.

The following example has a function with one argument ($fname). When the familyName() function is called, we also pass along a name, e.g. ("Jani"), and the name is used inside the function, which outputs several different first names, but an equal last name:

Example

function familyName($fname) {

echo "$fname Refsnes.<br>";

}

familyName("Jani");

familyName("Hege");

familyName("Stale");

familyName("Kai Jim");

familyName("Borge");

The following example has a function with two arguments ($fname, $year):

Example

function familyName($fname, $year) {

echo "$fname Refsnes. Born in $year <br>";

}

familyName("Hege", "1975");

familyName("Stale", "1978");

familyName("Kai Jim", "1983");

<?php

function hello($name){

echo "Hello $name.<br>";

}

hello("Yahoo Baba"); /\* ----------- also show error \*/

/\* TWO Argument : \*/

function hello($fname,$lname){

echo "Hello $fname $lname.<br>";

}

hello("Yahoo","Baba");

hello("Bill","Gates");

/\* Default Value : \*/

function hello($fname="First",$lname="Name"){

echo "Hello $fname $lname.<br>";

}

/\* SUM function \*/

function sum($a,$b){

echo $a + $b;

}

sum(10,20.50);

/\* Passing with Variables \*/

$one = 10;

$two = 20.50;

sum($one,$two);

?>

### PHP Functions - Returning values

To let a function return a value, use the return statement:

Example

function sum($x, $y) {

$z = $x + $y;

return $z;

}

echo "5 + 10 = " . sum(5, 10) . "<br>";

echo "7 + 13 = " . sum(7, 13) . "<br>";

echo "2 + 4 = " . sum(2, 4);

<?php

//1)

function hello($fname="First",$lname="Last"){

$v = "$fname $lname";

return $v;

}

echo hello("Yahoo","Baba");

//2)

$name = hello("Yahoo","Baba");

echo "Hello $name";

//3)

function sum($math,$eng,$sc){

$s = $math + $eng + $sc;

return $s;

}

$total = sum(55,65,88);

echo $total;

//4)

function percentage($st){

$per = $st/3;

echo $per . "%";

}

percentage($total);

?>

### Passing Arguments by Reference

In PHP, arguments are usually passed by value, which means that a copy of the value is used in the function and the variable that was passed into the function cannot be changed.

When a function argument is passed by reference, changes to the argument also change the variable that was passed in. To turn a function argument into a reference, the & operator is used:

Example

Use a pass-by-reference argument to update a variable:

function add\_five(&$value) {

$value += 5;

}

$num = 2;

add\_five($num);

echo $num;

<?php

/\* function argument By Value \*/

function testing(&$string)

{

$string .= 'and something extra.';

}

$str = 'This is a string, ';

testing($str);

echo $str;

/\* function argument By Reference\*/

function first($num) {

$num += 5;

}

function second(&$num) {

$num += 6;

}

$number = 10;

first( $number );

echo "Original Value is $number<br />";

second( $number );

echo "Original Value is $number<br />";

?>

## PHP Variable Functions

<?php

function wow($name) {

echo "Hello $name";

}

$func = "wow";

$func('Yahoo Baba');

/\* --- Anonymous function ----\*/

$sayHello = function($name) {

echo "Hello $name!";

};

$sayHello('Yahoo Baba');

?>

### PHP Recursive Function

PHP also supports recursive function call like C/C++. In such case, we call current function within function. It is also known as recursion.

It is recommended to avoid recursive function call over 200 recursion level because it may smash the stack and may cause the termination of script.  
  
Example

<?php

function display($number) {

if($number<=5){

echo "$number <br/>";

display($number+1);

}

}

display(1);

?>

<?php

/\*-----Recursive Function------ \*/

function display($number) {

if($number<=5){

echo "$number <br/>";

display($number+1);

}

}

display(1);

/\* --------Factorial Number--------- \*/

function factorial($n)

{

if ($n < 0){

return -1;

}

if ($n == 0){

return 1;

}else{

return ($n \* factorial ($n -1));

}

}

echo factorial(5);

?>

PHP Variables Scope

In PHP, variables can be declared anywhere in the script.

The scope of a variable is the part of the script where the variable can be referenced/used.

PHP has three different variable scopes:

* local
* global
* static

Global and Local Scope

A variable declared **outside** a function has a GLOBAL SCOPE and can only be accessed outside a function:

Example

Variable with global scope:

$x = 5; // global scope

function myTest() {

// using x inside this function will generate an error

echo "<p>Variable x inside function is: $x</p>";

}

myTest();

echo "<p>Variable x outside function is: $x</p>";

A variable declared **within** a function has a LOCAL SCOPE and can only be accessed within that function:

Example

Variable with local scope:

function myTest() {

$x = 5; // local scope

echo "<p>Variable x inside function is: $x</p>";

}

myTest();

// using x outside the function will generate an error

echo "<p>Variable x outside function is: $x</p>";

<?php

$x = 10;

function test() {

global $x; /\* ------Global Variable------- \*/

echo "Variable x inside function : $x <br>";

}

test();

echo "Variable x outside function : $x";

/\* --------------------- \*/

/\* $x = 5;

$y = 10;

function test() {

global $x, $y;

$x = $x + $y;

}

test();

echo $x; \*/

?>

### What is an Array?

An array is a special variable that can hold many values under a single name, and you can access the values by referring to an index number or name.  
Create Array

You can create arrays by using the array() function:

Example

$cars = array("Volvo", "BMW", "Toyota");

### Multiple Lines

Line breaks are not important, so an array declaration can span multiple lines:

Example

$cars = [

"Volvo",

"BMW",

"Toyota"

];

### Access Array Item

To access an array item, you can refer to the index number for indexed arrays, and the key name for associative arrays.

Example:

Access an item by referring to its index number:

$cars = array("Volvo", "BMW", "Toyota");

echo $cars[2];

<?php

/\* -------Array---------- \*/

$colors = array('red', 'yellow', 'blue', 'green');

echo $colors[0]."<br>";

$colors = ['red', 'yellow', 'blue', 'green']; /\* ----- IInd way \*/

/\* ---- can also use different data types ----- \*/

//2.)

echo "<pre>";

print\_r($colors);

echo "</pre>";

//3.)

$colors[0] = "red"; /\* --------- IIIrd way \*/

$colors[1] = "green"

$colors[2] = "yellow";

$colors[3] = "blue";

//4.)

echo "<ul>"; /\* --------- Loop Method \*/

for($i = 0 ; $i < 4 ; $i++){

echo "<li>$i</li>";

}

echo "</ul>";

?>

### PHP Associative Arrays

Associative arrays are arrays that use named keys that you assign to them.

Example

$car = array("brand"=>"Ford", "model"=>"Mustang", "year"=>1964);

var\_dump($car);

### Access Associative Arrays

To access an array item you can refer to the key name.

Example

Display the model of the car:

$car = array("brand"=>"Ford", "model"=>"Mustang", "year"=>1964);

echo $car["model"];

### Change Value

To change the value of an array item, use the key name:

Example

Change the year item:

$car = array("brand"=>"Ford", "model"=>"Mustang", "year"=>1964);

$car["year"] = 2024;

var\_dump($car);

<?php

$age = array(

"bill" => 25,

"steve" => 28,

"elon" => 22

);

echo $age["bill"] . "<br>";

echo $age["steve"] . "<br>";

echo $age["elon"] . "<br>";

echo '<pre>';

print\_r($age);

echo '</pre>';

echo '<pre>';

var\_dump($age);

echo '</pre>';

//another way to define array

$age1 = [

"bill" => "25",

"steve" => 28,

"elon" => 22

];

echo '<pre>';

print\_r($age1);

echo '</pre>';

echo '<pre>';

var\_dump($age1);

echo '</pre>';

$age["elon"] = 50; /\* --------- reasign value------- \*/

echo '<pre>';

print\_r($age1);

echo '</pre>';

echo '<pre>';

var\_dump($age1);

echo '</pre>';

//array with Numeric key

$age2 = array(

100 => "25",

10 => 28,

13 => 22

);

echo '<pre>';

print\_r($age2);

echo '</pre>';

//array with numeric and string key

$age3 = array(

100 => "25",

"bill" => 28,

13 => 22

);

echo '<pre>';

var\_dump($age3);

echo '</pre>';

?>

### Loop Through an Associative Array

To loop through and print all the values of an associative array, you can use a foreach loop, like this:

Example

Display all array items, keys and values:

$car = array("brand"=>"Ford", "model"=>"Mustang", "year"=>1964);

foreach ($car as $x => $y) {

echo "$x: $y <br>";

}

### Loop Through an Indexed Array

To loop through and print all the values of an indexed array, you can use a foreach loop, like this:

Example

Display all array items:

$cars = array("Volvo", "BMW", "Toyota");

foreach ($cars as $x) {

echo "$x <br>";

}

<?php

$colors = [

"red",

"green",

"blue"

];

foreach($colors as $value){

echo $value . "<br>";

}

/\* Associative Array For Foreach \*/

$age = [

"bill" => 25,

"steve" => 28,

"elon" => 22,

];

foreach($age as $key => $value){

echo "$key = $value <br>";

}

/\* ------work with ul tag------- \*/

echo "<ul>";

foreach($age as $key => $value){

echo "<li>$key = $value </li>";

}

echo "</ul>";

?>

PHP Multidimensional Arrays

A multidimensional array is an array containing one or more arrays.

PHP supports multidimensional 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.  
  
**The dimension of an array indicates the number of indices you need to select an element.**

* For a two-dimensional array you need two indices to select an element
* For a three-dimensional array you need three indices to select an element

PHP - Two-dimensional Arrays

A two-dimensional array is an array of arrays (a three-dimensional array is an array of arrays of arrays.

<?php

$emp = [

[1,"Krishana","Manager",50000],

[2,"Salman","Salesman",20000],

[3,"Mohan","Computer Operator",12000],

[4,"Amir","Driver",5000]

];

echo "<pre>";

print\_r($emp);

echo "</pre>";

/\* TWO dimensional Array row and column \*/

echo $emp[0][0] . " ";

echo $emp[0][1] . " ";

echo $emp[0][2] . " ";

echo $emp[0][3] . " ";

/\* Multidimensional Array For Loop \*/

for ($row = 0; $row < 4; $row++) {

for ($col = 0; $col < 4; $col++) {

echo $emp[$row][$col] . " ";

}

echo "<br>";

}

/\* Multidimensional Array Foreach Loop \*/

foreach ($emp as $v1) {

foreach ($v1 as $v2){

echo $v2 . " ";

}

echo "<br>";

}

/\* Print with Table tag \*/

echo "<table border='2px' cellpadding='5px' cellspacing='0'>";

echo "<tr>

<th>Emp Id</th>

<th>Emp Name</th>

<th>Designation</th>

<th>Salary</th>

</tr>";

foreach ($emp as $v1){

echo "<tr>";

foreach ($v1 as $v2){

echo "<td> $v2 </td>";

}

echo "</tr>";

}

echo "</table>";

?>

### PHP Multidimensional Associative Array

Multidimensional array is used to store an array in contrast to constant values. Associative array stores the data in the form of key and value pairs where the key can be an integer or string. **Multidimensional associative array** is often used to store data in group relation.

<?php

$marks = [

"Krishna" => [

"physics" => 85,

"maths" => 78,

"chemistry" => 89

],

"Salman" => [

"physics" => 68,

"maths" => 73,

"chemistry" => 79

],

"Mohan" => [

"physics" => 62,

"maths" => 67,

"chemistry" => 92

]

];

echo "<pre>";

print\_r($marks);

echo "</pre>";

/\* -------------Foreach Loop Array----------------- \*/

foreach($marks as $key => $v1){

echo $key;

foreach($v1 as $v2){

echo $v2 . " ";

}

echo "</br>";

}

/\* ------------Print with Table tag-------------------- \*/

echo "<table border='2px' cellpadding='5px' cellspacing='0'>

<tr>

<th>Student Name</th>

<th>Physics</th>

<th>Math</th>

<th>Chemistry</th>

</tr>";

foreach($marks as $key => $v1){

echo "<tr>

<td>$key</td>";

foreach($v1 as $v2){

echo "<td> $v2 </td>";

}

echo "</tr>";

}

echo "</table>";

?>

## PHP Foreach Loop with List

<?php

/\* ----------- Index Array----------- \*/

$emp = [

[1,"Krishana","Manager",50000],

[2,"Salman","Salesman",20000],

[3,"Mohan","Computer Operator",12000],

[4,"Amir","Driver",5000]

];

foreach ($emp as list($id, $name,$desg,$salary)) {

echo "$id $name $desg $salary </br>";

}

/\* print with table tag \*/

echo "<table border='1px' cellpadding='5px' cellspacing='0'>

<tr>

<th>Emp Id</th>

<th>Emp Name</th>

<th>Designation</th>

<th>Salary</th>

</tr>";

foreach ($emp as list($id, $name,$desg,$salary)) {

echo "<tr><td>$id</td><td>$name</td><td>$desg</td><td>$salary</td></tr>";

}

echo "</table>";

/\* -----------Multidimensional Associative Array----------- \*/

$emp = [

["id" => 1,"name" => "Krishana","designation" => "Manager","salary" => 50000],

["id" => 2,"name" => "Salman","designation" => "Salesman","salary" => 20000],

["id" => 3,"name" => "Mohan","designation" => "Computer Operator","salary" => 12000],

["id" => 4,"name" => "Amir","designation" => "Driver","salary" => 5000]

];

foreach ($emp as list("id" => $id, "name" => $name,"designation" => $desg,"salary" => $salary)) {

echo "Id: $id; Name: $name; Designation: $desg; Salary: $salary</br>";

}

?>

## PHP Array Count & Sizeof

PHP count() Function

Return the number of elements in an array:

<?php

$cars=array("Volvo","BMW","Toyota");

echo count($cars);

?>

Syntax

count(array, mode)

Parameter Values

|  |  |
| --- | --- |
| **Parameter** | **Description** |
| *array* | Required. Specifies the array |
| *mode* | Optional. Specifies the mode. Possible values:   * 0 - Default. Does not count all elements of multidimensional arrays * 1 - Counts the array recursively (counts all the elements of multidimensional arrays) |

PHP sizeof() Function

Return the number of elements in an array:

<?php

$cars=array("Volvo","BMW","Toyota");

echo sizeof($cars);

?>

Syntax

sizeof(array, mode)

Parameter Values

|  |  |
| --- | --- |
| **Parameter** | **Description** |
| *array* | Required. Specifies the array |
| *mode* | Optional. Specifies the mode. Possible values:   * 0 - Default. Does not count all elements of multidimensional arrays * 1 - Counts the array recursively (counts all the elements of multidimensional arrays) |

<?php

$food = array('orange', 'banana', 'apple');

echo count($food,1).'<br>'; /\* ----- IInd parameter is MODE (0 or 1) \*/

$food1 = array(

'fruits' => array('orange', 'banana', 'apple'),

'veggie' => array('carrot', 'collard', 'pea')

);

/\* (Mode counts all the elements of multidimensional arrays) \*/

echo sizeof($food1,1).'<br>';

echo sizeof($food1['fruits'],1).'<br>';

$len = count($food);

for($i = 0; $i < $len; $i++){

echo $food[$i] . "<br>";

}

$food2 = array('orange', 'banana', 'orange' , 'apple');

// count array values

echo "<pre>";

print\_r(array\_count\_values($food2));

echo "</pre>";

?>

## PHP Array In\_array & Array\_search

Array In\_array

The in\_array() function searches an array for a specific value.

Syntax

in\_array(search, array, type)

Parameter Values

|  |  |
| --- | --- |
| **Parameter** | **Description** |
| *search* | Required. Specifies the what to search for |
| *array* | Required. Specifies the array to search |
| *type* | Optional. If this parameter is set to TRUE, the in\_array() function searches for the search-string and specific type in the array. |

|  |  |
| --- | --- |
| **Return Value:** | Returns TRUE if the value is found in the array, or FALSE otherwise |

array\_search()

The array\_search() function search an array for a value and returns the key.

Syntax

array\_search(value, array, strict)

Parameter Values

|  |  |
| --- | --- |
| **Parameter** | **Description** |
| *value* | Required. Specifies the value to search for |
| *array* | Required. Specifies the array to search in |
| *strict* | Optional. If this parameter is set to TRUE, then this function will search for identical elements in the array. Possible values:   * true * false - Default   When set to true, the number 5 is not the same as the string 5 (See example 2) |

|  |  |
| --- | --- |
| **Return Value:** | Returns the key of a value if it is found in the array, and FALSE otherwise. If the value is found in the array more than once, the first matching key is returned. |

<?php

$food = array('orange', 'banana', 'apple', 'grapes');

echo in\_array("lime", $food);

/\* ---- If Condition InArray------ \*/

if (in\_array("apple", $food)) {

echo "Find Successfully.";

}else{

echo "Can't Find.";

}

echo in\_array(55, $food,true); /\* ---- strict mode------ \*/

/\* --------Multipledimensional Associative Array------------- \*/

$a = array(array('p', 'h'), array('p', 'r'), 'o');

if (in\_array(array('p', 'h'), $a)){}

/\* ----------Array Search------------- \*/

echo array\_search("apple", $food);

/\* -----------Associative Array-------------- \*/

$food = array('a' => 'orange', 'b' => 'banana', 'c' => 'apple', 'd' => 'grapes');

echo array\_search("apple", $food);

echo in\_array("apple", $food);

?>

## PHP Array\_replace & Array\_replace\_recursive

### PHP Array\_replace

The array\_replace() function replaces the values of the first array with the values from following arrays.

### Syntax

array\_replace(array1, array2, array3, ...)

### Parameter Values

|  |  |
| --- | --- |
| **Parameter** | **Description** |
| array1 | Required. Specifies an array |
| array2 | Optional. Specifies an array which will replace the values of array1 |
| array3,... | Optional. Specifies more arrays to replace the values of array1 and array2, etc. Values from later arrays will overwrite the previous ones. |

|  |  |
| --- | --- |
| **Return Value:** | Returns the replaced array, or NULL if an error occurs |

### PHP array\_replace\_recursive() Function

The array\_replace\_recursive() function replaces the values of the first array with the values from following arrays recursively.

### Syntax

array\_replace\_recursive(array1, array2, array3, ...)

### Parameter Values

|  |  |
| --- | --- |
| **Parameter** | **Description** |
| array1 | Required. Specifies an array |
| array2 | Optional. Specifies an array which will replace the values of array1 |
| array3,... | Optional. Specifies more arrays to replace the values of array1 and array2, etc. Values from later arrays will overwrite the previous ones. |

|  |  |
| --- | --- |
| **Return Value:** | Returns the replaced array, or NULL if an error occurs |

<?php

$fruit = ['orange', 'banana', 'apple', 'grapes'];

$veggie = ['carrot', 'pea'];

$newArray = array\_replace($fruit, $veggie);

echo "<pre>";

print\_r($newArray);

echo "</pre>";

/\*----- Index Array -------- \*/

$color = ['red', 'green', 'blue'];

$newArray = array\_replace($fruit, $veggie, $color);

/\* ----------------Associative Array------------------- \*/

$veggie = ['a' => 'carrot', 'b' => 'pea'];

$veggie = ['a' => 'carrot', 1 => 'pea'];

/\* --------------Array Replace Function----------------- \*/

$fruit = ['orange', 'b' => 'banana', 'apple', 'grapes'];

/\* ---- array\_replace\_recursive----------- \*/

$array1 = array("a"=>array("red"),"b"=>array("green","pink"));

$array2 = array("a"=>array("yellow"),"b"=>array("black"));

$newArray = array\_replace\_recursive($array1, $array2);

?>

### PHP array\_pop() Function

The array\_pop() function deletes the last element of an array.

### Syntax

array\_pop(array)

### Parameter Values

|  |  |
| --- | --- |
| **Parameter** | **Description** |
| array | Required. Specifies an array |

|  |  |
| --- | --- |
| **Return Value:** | Returns the last value of array. If array is empty, or is not an array, NULL will be returned. |

### PHP array\_push() Function

The array\_push() function inserts one or more elements to the end of an array.

### Syntax

array\_push(array, value1, value2, ...)

### Parameter Values

|  |  |
| --- | --- |
| **Parameter** | **Description** |
| array | Required. Specifies an array |
| value1 | Optional. Specifies the value to add (Required in PHP versions before 7.3) |
| value2 | Optional. Specifies the value to add |

|  |  |
| --- | --- |
| **Return Value:** | Returns the new number of elements in the array |
|  |  |

<?php

/\* --------- Delete from End------- \*/

$fruit = ["orange", "banana", "apple", "grapes"];

array\_pop($fruit);

echo "<pre>";

print\_r($fruit);

echo "</pre>";

/\* --------- Add on Ending on the Array---------- \*/

$fruit = ["orange", "banana", "grapes"];

array\_push($fruit,"apple","guava","lemon");

echo "<pre>";

print\_r($fruit);

echo "</pre>";

?>

### Array\_shift()

The array\_shift() function removes the first element from an array, and returns the value of the removed element.

### Syntax

array\_shift(array)

### Parameter Values

|  |  |
| --- | --- |
| **Parameter** | **Description** |
| array | Required. Specifies an array |

|  |  |
| --- | --- |
| **Return Value:** | Returns the value of the removed element from an array, or NULL if the array is empty |

### Array\_unshift()

The array\_unshift() function inserts new elements to an array. The new array values will be inserted in the beginning of the array.

### Syntax

array\_unshift(array, value1, value2, value3, ...)

### Parameter Values

|  |  |
| --- | --- |
| **Parameter** | **Description** |
| array | Required. Specifying an array |
| value1 | Optional. Specifies a value to insert (Required in PHP versions before 7.3) |
| value2 | Optional. Specifies a value to insert |
| value3 | Optional. Specifies a value to insert |

|  |  |
| --- | --- |
| **Return Value:** | Returns the new number of elements in the array |

<?php

/\*------- Add on starting on the Array------ \*/

$fruit = ["orange", "banana", "grapes"];

array\_shift($fruit);

echo "<pre>";

print\_r($fruit);

echo "</pre>";

/\*------ Delete from Start------ \*/

$fruit = ["orange", "banana", "grapes"];

array\_unshift($fruit,"Apple", "guava", "lemon");

echo "<pre>";

print\_r($fruit);

echo "</pre>";

?>

### array\_merge() Function

The array\_merge() function merges one or more arrays into one array.

### Syntax

array\_merge(array1, array2, array3, ...)

### Parameter Values

|  |  |
| --- | --- |
| **Parameter** | **Description** |
| array1 | Required. Specifies an array |
| array2 | Optional. Specifies an array |
| array3,... | Optional. Specifies an array |

|  |  |
| --- | --- |
| **Return Value:** | Returns the merged array |

### array\_combine()

The array\_combine() function creates an array by using the elements from one "keys" array and one "values" array.

### Syntax

array\_combine(keys, values)

### Parameter Values

|  |  |
| --- | --- |
| **Parameter** | **Description** |
| keys | Required. Array of keys |
| values | Required. Array of values |

|  |  |
| --- | --- |
| **Return Value:** | Returns the combined array. FALSE if number of elements does not match |

<?php

/\* Multiple Array Merge \*/

$fruit = ["orange", "banana", "grapes"];

$veggie = ['carrot', 'pea'];

$newArray = array\_merge($fruit,$veggie);

echo "<pre>";

print\_r($newArray);

echo "</pre>";

/\* -------------Multiple Array Merge -------------------- \*/

$color = ['red', 'blue'];

array\_merge($fruit,$veggie,$color);

/\* ------------ASSOCIATIVE Index Array------------------ \*/

$fruit = ['a' => "orange", 'b' => "banana", 'c' => "grapes"];

$veggie = ['d' => 'carrot','e' => 'pea']; /\* ---- Also with SAME KEY \*/

/\* -------------Numberic Value Enter----------------- \*/

$veggie = ['b' => 'carrot','e' => 'pea', 55, 68];

/\* ---------if u want the duplicate key entry of first array---------- \*/

$newArray = $fruit + $veggie;

/\* ----------array\_merge\_recursive-------------- \*/

$newArray = array\_merge\_recursive($fruit,$veggie); /\* --- for common key built new array --- \*/

$veggie = ['b' => ['color' => ['red','blue','green']], /\* ---- more complex multidim array ----\*/

'e' => 'pea',

55,

68

];

$newArray = array\_merge\_recursive($fruit,$veggie);

/\* --------array\_combine-------- \*/

$name = array("Ram","Mohan","Salman");

$age = array("35","37","43");

$newArray = array\_combine($name, $age);

echo "<pre>";

print\_r($newArray);

echo "</pre>";

?>

PHP Array\_Slice

The array\_slice() function returns selected parts of an array.

Syntax

array\_slice(array, start, length, preserve)

Parameter Values

|  |  |
| --- | --- |
| **Parameter** | **Description** |
| *array* | Required. Specifies an array |
| *start* | Required. Numeric value. Specifies where the function will start the slice. 0 = the first element. If this value is set to a negative number, the function will start slicing that far from the last element. -2 means start at the second last element of the array. |
| *length* | Optional. Numeric value. Specifies the length of the returned array. If this value is set to a negative number, the function will stop slicing that far from the last element. If this value is not set, the function will return all elements, starting from the position set by the start-parameter. |
| *preserve* | Optional. Specifies if the function should preserve or reset the keys. Possible values:   * true - Preserve keys * false - Default. Reset keys |

|  |  |
| --- | --- |
| **Return Value:** | Returns selected parts of an array |

<?php

/\*------ Array Slice ------\*/

$color=array("red","green","blue","yellow","brown");

$newArray = array\_slice($color, 1, 2);

echo "<pre>";

print\_r($newArray);

echo "</pre>";

/\* -------------with Negative Index---------- \*/

print\_r(array\_slice($color,-2,1));

echo '<br>';

echo '<br>';

/\* ---------Preserve Parameter----------- \*/

print\_r(array\_slice($color,1,2,true));

echo '<br>';

/\* ---------------Preserve Parameter With Associative Array --------------------- \*/

$color = array('a'=>'red', 'b'=>'green', '42'=>'blue', 'd'=>'yellow');

array\_slice($color, 0, 3);

echo '<br>';

array\_slice($color, 0, 3, true);

echo '<br>';

/\* -------------Preserve Parameter - III-------------- \*/

$a=array("0"=>"red","1"=>"green","2"=>"blue","3"=>"yellow","4"=>"brown");

print\_r(array\_slice($a,1,2));

?>

### PHP Array\_Splice()

The array\_splice() function removes selected elements from an array and replaces it with new elements. The function also returns an array with the removed elements.

### Syntax

array\_splice(array, start, length, array)

### Parameter Values

|  |  |
| --- | --- |
| **Parameter** | **Description** |
| array | Required. Specifies an array |
| start | Required. Numeric value. Specifies where the function will start removing elements. 0 = the first element. If this value is set to a negative number, the function will start that far from the last element. -2 means start at the second last element of the array. |
| length | Optional. Numeric value. Specifies how many elements will be removed, and also length of the returned array. If this value is set to a negative number, the function will stop that far from the last element. If this value is not set, the function will remove all elements, starting from the position set by the start-parameter. |
| array | Optional. Specifies an array with the elements that will be inserted to the original array. If it's only one element, it can be a string, and does not have to be an array. |

## 

|  |  |
| --- | --- |
| **Return Value:** | Returns the array consisting of the extracted elements |

<?php

/\*------ Array Splice ------\*/

$color =["red","green","blue","yellow","brown"];

$fruit = ["Orange", "Apple"];

array\_splice($color, 2 ); /\* ----- First Method-------\*/

array\_splice($color, 1, -1); /\* ---- Second Method----\*/

array\_splice($color, 1, -2); /\* ---- Third Method ------- \*/

/\* -------Remove the last element of $color-------- \*/

array\_splice($color, -1);

/\* -------Remove the first element of $color------- \*/

array\_splice($color, 0, 1);

/\* -----------Remove First Two elements and add new elements in $color ------------- \*/

array\_splice($color, 0 , 2, $fruit);

/\* -----------Replace Third and fourth element in $color ------------- \*/

array\_splice($color, 2 , 2, $fruit);

/\* -----------use count method in $color ------------- \*/

array\_splice($color, 2, count($color), $fruit);

/\* -----------add new elements in $color with count method ------------- \*/

array\_splice($color, 2, 0, $fruit);

/\* -----------add new elements in begining of $color with count method ------------- \*/

array\_splice($color, 0, 0, $fruit);

/\* -----------add new elements in the end of $color with count method ------------- \*/

array\_splice($color,count($color),0, $fruit);

echo "<pre>";

print\_r($color);

echo "</pre>";

?>

PHP Array Key

The array\_keys() function returns an array containing the keys.

Syntax

array\_keys(array, value, strict)

Parameter Values

|  |  |
| --- | --- |
| **Parameter** | **Description** |
| *array* | Required. Specifies an array |
| *value* | Optional. You can specify a value, then only the keys with this value are returned |
| *strict* | Optional. Used with the value parameter. Possible values:   * true - Returns the keys with the specified value, depending on type: the number 5 is not the same as the string "5". * false - Default value. Not depending on type, the number 5 is the same as the string "5". |

|  |  |
| --- | --- |
| **Return Value:** | Returns the merged array |

<?php

/\*-------- array\_keys--------\*/

$color =["red","green","blue","yellow"];

$newArray = array\_keys($color);

echo '<pre>';

print\_r($newArray);

echo '</pre>';

/\* --------With Associative Array --------- \*/

$color1 =[

"first" =>"red",

"second" =>"green",

"third" =>"blue",

"fourth" =>"yellow"

];

$newArray1 = array\_keys($color1);

echo '<pre>';

print\_r($newArray1);

echo '</pre>';

/\* ---------Comes in 7.3---------- \*/

$newArray1 = array\_key\_first($color1);

echo '<pre>';

print\_r($newArray1);

echo '</pre>';

/\* ---------Comes in 7.3----------- \*/

$newArray1 = array\_key\_last($color1);

echo '<pre>';

print\_r($newArray1);

echo '</pre>';

/\* ------array\_key\_exists--------- \*/

$newArray1 = array\_key\_exists("third", $color1);

echo '<pre>';

print\_r($newArray1);

echo '</pre>';

$newArray1 = array\_key\_exists("six", $color);

echo '<pre>';

print\_r($newArray1);

echo '</pre>';

/\* -------array\_key\_exists short name key)exists-------- \*/

$newArray1 = key\_exists("second", $color1);

echo '<pre>';

print\_r($newArray1);

echo '</pre>';

/\* ---------check result with if condition----------- \*/

if ($newArray1)

{

echo "Key exists!";

}

else

{

echo "Key does not exist!";

}

?>

### PHP Array Intersect

The array\_intersect() function compares **the values** of two (or more) arrays, and returns the matches.

This function compares the values of two or more arrays, and return an array that contains the entries from array1 that are present in array2, array3, etc.

### Syntax

array\_intersect(array1, array2, array3, ...)

### Parameter Values

|  |  |
| --- | --- |
| **Parameter** | **Description** |
| array1 | Required. The array to compare from |
| array2 | Required. An array to compare against |
| array3,... | Optional. More arrays to compare against |

|  |  |
| --- | --- |
| **Return Value:** | Returns an array containing the entries from array1 that are present in all of the other arrays |

<?php

/\* -----------array\_intersect-------------- \*/

$a1=array("a"=>"red","b"=>"green","c"=>"blue","d"=>"yellow");

$a2=array("a"=>"red","f"=>"green","d"=>"purple");

$newArray = array\_intersect($a1,$a2);

echo "<pre>";

print\_r($newArray);

echo "</pre>";

//show change case

/\* ----------Multiple Array intersect------------- \*/

$a3=array("a"=>"red","b"=>"black","h"=>"yellow");

$newArray11 = array\_intersect($a1,$a2,$a3);

echo "<pre>";

print\_r($newArray11);

echo "</pre>";

/\* ----------- Match only Key-------------\*/

$newArray1 = array\_intersect\_key($a1,$a2);

echo "<pre>";

print\_r($newArray1);

echo "</pre>";

/\* ---------Match Key and Value both---------- \*/

$newArray2 = array\_intersect\_assoc($a1,$a2);

echo "<pre>";

print\_r($newArray2);

echo "</pre>";

/\* ---------Match Key and Value both with callback function----------- \*/

function compare($a,$b){

if ($a===$b){

return 0;

}

return ($a > $b)?1:-1;

}

function compareValue($a,$b){

if ($a===$b){

return 0;

}

return ($a > $b)?1:-1;

}

$newArray3 = array\_intersect\_uassoc($a1,$a2, "compare");

echo "<pre>";

print\_r($newArray3);

echo "</pre>";

$newArray4 = array\_uintersect\_assoc($a1,$a2, "compare"); //this function can write also like this -- Important

echo "<pre>";

print\_r($newArray4);

echo "</pre>";

/\* ----------Match only key with function------------ \*/

$newArray5 = array\_intersect\_ukey($a1,$a2, "compare");

echo "<pre>";

print\_r($newArray5);

echo "</pre>";

/\* --------Match only value with function----------- \*/

$newArray6 = array\_uintersect($a1,$a2, "compare");

echo "<pre>";

print\_r($newArray6);

echo "</pre>";

/\* --------Match both value and key with two different functions----------- \*/

$newArray7=array\_uintersect\_uassoc($a1,$a2,"compare","compareValue");

echo "<pre>";

print\_r($newArray7);

echo "</pre>";

?>

## PHP Array Diff & Udiff

### PHP Array Diff

The array\_diff() function compares the values of two (or more) arrays, and returns the differences.

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.

### Syntax

array\_diff(array1, array2, array3, ...)

### Parameter Values

|  |  |
| --- | --- |
| **Parameter** | **Description** |
| array1 | Required. The array to compare from |
| array2 | Required. An array to compare against |
| array3,... | Optional. More arrays to compare against |

|  |  |
| --- | --- |
| **Return Value:** | Returns an array containing the entries from array1 that are not present in any of the other arrays |

### array\_udiff()

The array\_udiff() function compares the values of two or more arrays, and returns the differences.

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.

### Syntax

array\_udiff(array1, array2, array3, ..., myfunction)

### Parameter Values

|  |  |
| --- | --- |
| **Parameter** | **Description** |
| array1 | Required. The array to compare from |
| array2 | Required. An array to compare against |
| array3,... | Optional. More arrays to compare against |
| myfunction | Required. A string that define a callable comparison function. The comparison function must return an integer <, =, or > than 0 if the first argument is <, =, or > than the second argument |

|  |  |
| --- | --- |
| **Return Value:** | Returns an array containing the entries from array1 that are not present in any of the other arrays |

<?php

/\*--------- array\_diff - compare only values and return differences only form $a1 ---------\*/

$a1 = array("a"=>"red","b"=>"green","c"=>"blue","d"=>"yellow");

$a2 = array("a"=>"red","f"=>"green","d"=>"purple");

$newArray = array\_diff($a1,$a2);

echo "<pre>";

print\_r($newArray);

echo "</pre>";

//show change case

/\* ---------Comapre Three Arrays --------- \*/

$a3 = array("a"=>"red","b"=>"black","h"=>"yellow");

$newArray1 = array\_diff($a1,$a2,$a3);

echo "<pre>";

print\_r($newArray1);

echo "</pre>";

/\*--------- Compare only Key ---------\*/

$newArray2 = array\_diff\_key($a1,$a2);

echo "<pre>";

print\_r($newArray2);

echo "</pre>";

/\* -------Compare Key and Value both ---- Only compare associative array------- \*/

$newArray3 = array\_diff\_assoc($a1,$a2);

echo "<pre>";

print\_r($newArray3);

echo "</pre>";

function compare($a,$b)

{

if ($a===$b){

return 0;

}

return ($a > $b)?1:-1;

}

function compareValue($a,$b)

{

if ($a===$b){

return 0;

}

return ($a > $b)?1:-1;

}

/\* -------Match Key and Value both with callback function------- \*/

$newArray4 = array\_diff\_uassoc($a1,$a2, "compare"); //--- can also use string fun - "strcasecmp"

echo "<pre>";

print\_r($newArray4);

echo "</pre>";

//this function can write also like thid( array\_udiff\_assoc ) -- Important

/\* --------Compare only key with function-------- \*/

$newArray5 = array\_diff\_ukey($a1,$a2, "compare");

echo "<pre>";

print\_r($newArray5);

echo "</pre>";

/\* ------Compare only value with function------ \*/

$newArray6 = array\_udiff($a1,$a2, "compare");

echo "<pre>";

print\_r($newArray6);

echo "</pre>";

/\* --------Compare both value and key with two different functions---------- \*/

$newArray7 = array\_udiff\_uassoc($a1,$a2,"compare","compareValue");

echo "<pre>";

print\_r($newArray7);

echo "</pre>";

?>

## PHP Array\_Values & Array\_Unique

### array\_values()

The array\_values() function returns an array containing all the values of an array.

### Syntax

array\_values(array)

### Parameter Values

|  |  |
| --- | --- |
| **Parameter** | **Description** |
| array | Required. Specifying an array |

|  |  |
| --- | --- |
| **Return Value:** | Returns an array containing all the values of an array |

### array\_unique()

The array\_unique() function removes duplicate values from an array. If two or more array values are the same, the first appearance will be kept and the other will be removed.

### Syntax

array\_unique(array, sorttype)

### Parameter Values

|  |  |
| --- | --- |
| **Parameter** | **Description** |
| array | Required. Specifying an array |
| sorttype | Optional. Specifies how to compare the array elements/items. Possible values:   * SORT\_STRING - Default. Compare items as strings * SORT\_REGULAR - Compare items normally (don't change types) * SORT\_NUMERIC - Compare items numerically * SORT\_LOCALE\_STRING - Compare items as strings, based on current locale |

## 

|  |  |
| --- | --- |
| **Return Value:** | Returns the filtered array |

<?php

/\* ------------Array Values------------ \*/

$a1 = array("a"=>"red","b"=>"green","c"=>"red","d"=>"yellow");

$newArray = array\_values($a1);

echo "<pre>";

print\_r($newArray);

echo "</pre>";

/\* ---------Array Unique---------\*/

$newArray1 = array\_unique($a1);

echo "<pre>";

print\_r($newArray1);

echo "</pre>";

?>

array\_column() Function

The array\_column() function returns the values from a single column in the input array.

Syntax

array\_column(array, column\_key, index\_key)

Parameter Values

|  |  |
| --- | --- |
| **Parameter** | **Description** |
| array | Required. Specifies the multi-dimensional array (record-set) to use. As of PHP 7.0, this can also be an array of objects. |
| column\_key | Required. An integer key or a string key name of the column of values to return. This parameter can also be NULL to return complete arrays (useful together with index\_key to re-index the array) |
| index\_key | Optional. The column to use as the index/keys for the returned array |

|  |  |
| --- | --- |
| **Return Value:** | Returns an array of values that represents a single column from the input array |

Array\_Chunk()

The array\_chunk() function splits an array into chunks of new arrays.

Syntax

array\_chunk(array, size, preserve\_key)

Parameter Values

|  |  |
| --- | --- |
| **Parameter** | **Description** |
| array | Required. Specifies the array to use |
| size | Required. An integer that specifies the size of each chunk |
| preserve\_key | Optional. Possible values:   * true - Preserves the keys * false - Default. Reindexes the chunk numerically |

|  |  |
| --- | --- |
| **Return Value:** | Returns a multidimensional indexed array, starting with zero, with each dimension containing *size* elements |

<?php

/\*------------Array\_Column------------ \*/

$array = array(

array(

'id' => 2201,

'first\_name' => 'Anil',

'last\_name' => 'Kapoor',

),

array(

'id' => 2202,

'first\_name' => 'Salman',

'last\_name' => 'Khan',

),

array(

'id' => 2203,

'first\_name' => 'John',

'last\_name' => 'Abraham',

)

);

$newArray = array\_column($array,'first\_name');

echo "<pre>";

print\_r($newArray);

echo "</pre>";

/\* ------With 3rd Parameter as a key------ \*/

$newArray1 = array\_column($array,'first\_name','id');

echo "<pre>";

print\_r($newArray1);

echo "</pre>";

/\* -------Array chunk------- \*/

$cars = ["Volvo","BMW","Toyota","Honda","Mercedes","Opel"];

$newArray2 = array\_chunk($cars,3);

echo "<pre>";

print\_r($newArray2);

echo "</pre>";

$age = array(

"Mohan" => "35",

"Aman" => "37",

"Ram" => "43",

"Salman" => "25" );

$newArray3 = array\_chunk($age,3,true);

echo "<pre>";

print\_r($newArray3);

echo "</pre>";

?>

## PHP Array\_Flip & Array\_Change\_Key\_Case

PHP array\_flip() Function

The array\_flip() function flips/exchanges all keys with their associated values in an array.

Syntax

array\_flip(array)

Parameter Values

|  |  |
| --- | --- |
| **Parameter** | **Description** |
| *array* | Required. Specifies an array of key/value pairs to be flipped |

|  |  |
| --- | --- |
| **Return Value:** | Returns the flipped array on success. NULL on failure |

array\_chunk() Function

The array\_chunk() function splits an array into chunks of new arrays.

Syntax

array\_chunk(array, size, preserve\_key)

Parameter Values

|  |  |
| --- | --- |
| **Parameter** | **Description** |
| array | Required. Specifies the array to use |
| size | Required. An integer that specifies the size of each chunk |
| preserve\_key | Optional. Possible values:   * true - Preserves the keys * false - Default. Reindexes the chunk numerically |

|  |  |
| --- | --- |
| **Return Value:** | Returns a multidimensional indexed array, starting with zero, with each dimension containing *size* elements |

<?php

/\*--------Array Flip---------\*/

$a = array(

"Bill" =>10,

"Joe" => 20,

"Peter" => 30

);

$newArray = array\_flip($a);

echo "<pre>";

print\_r($newArray);

echo "</pre>";

/\* -----array\_change\_key\_case----- \*/

$newArray2 = array\_change\_key\_case($a);

echo "<pre>";

print\_r($newArray2);

echo "</pre>";

//default is lower case

/\* --------CASE\_UPPER or CASE\_LOWER-------- \*/

$newArray3 = array\_change\_key\_case($a, CASE\_UPPER);

echo "<pre>";

print\_r($newArray3);

echo "</pre>";

?>

### PHP array\_sum() Function

The array\_sum() function returns the sum of all the values in the array.

### Syntax

array\_sum(array)

### Parameter Values

|  |  |
| --- | --- |
| **Parameter** | **Description** |
| array | Required. Specifies an array |

## 

|  |  |
| --- | --- |
| **Return Value:** | Returns the sum of all the values in an array |

### PHP array\_product() Function

The array\_product() function calculates and returns the product of an array.

### Syntax

array\_product(array)

### Parameter Values

|  |  |
| --- | --- |
| **Parameter** | **Description** |
| array | Required. Specifies an array |

|  |  |
| --- | --- |
| **Return Value:** | Returns the product as an integer or float |

<?php

$a1 = array(2, 4, 6, 8);

$a2 = array("a" => 1.2, "b" => 2.3, "c" => 3.4);

/\*------------Array Sum------------ \*/

echo "Sum of a1 = " . array\_sum($a1). '<br><br>';

echo "Sum of a2 = " . array\_sum($a2). '<br><br>';

/\*------------Array Product------------ \*/

echo "Product of a1 = " . array\_product($a1). '<br><br>';

echo "Product of a2 = " . array\_product($a2). '<br><br>';

?>

## PHP Array\_Rand & Shuffle

### PHP array\_rand() Function

The array\_rand() function returns a random key from an array, or it returns an array of random keys if you specify that the function should return more than one key.

### Syntax

array\_rand(array, number)

### Parameter Values

|  |  |
| --- | --- |
| **Parameter** | **Description** |
| array | Required. Specifies an array |
| number | Optional. Specifies how many random keys to return |

|  |  |
| --- | --- |
| **Return Value:** | Returns a random key from an array, or an array of random keys if you specify that the function should return more than one key |

<?php

/\*------------Array Rand------------ \*/

$color = array("red","green","blue","yellow","brown");

$newArray = array\_rand($color);

echo "<pre>";

print\_r($newArray);

echo "</pre>";

echo $color[$newArray]."<br><br>";

/\*------------ IInd Parameter -- as a Number------------ \*/

$newArray1 = array\_rand($color, 2);

echo $color[$newArray1[0]]."<br>";

echo $color[$newArray1[1]]."<br>";

/\*------------Use with Associative Array------------ \*/

$color1 = array("a"=>"red","b"=>"green","c"=>"blue","d"=>"yellow");

$newArray2 = array\_rand($color1, 2);

echo "<pre>";

print\_r($newArray2);

echo "</pre>";

/\*------------Shuffle Array------------ \*/

$color2 = array("red","green","blue","yellow","brown");

shuffle($color2);

echo "<pre>";

print\_r($color2);

echo "</pre>";

/\*------------Shuffle Use With Associative Array------------ \*/

$color3 = array("a"=>"red","b"=>"green","c"=>"blue","d"=>"yellow");

shuffle($color3);

echo "<pre>";

print\_r($color3);

echo "</pre>";

?>

## PHP Array\_Fill & Array\_Fill\_Keys

### PHP array\_fill() Function

The array\_fill() function fills an array with values.

### Syntax

array\_fill(index, number, value)

### Parameter Values

|  |  |
| --- | --- |
| **Parameter** | **Description** |
| index | Required. The first index of the returned array |
| number | Required. Specifies the number of elements to insert |
| value | Required. Specifies the value to use for filling the array |

|  |  |
| --- | --- |
| **Return Value:** | Returns the filled array |

### PHP array\_fill\_keys() Function

The array\_fill\_keys() function fills an array with values, specifying keys.

### Syntax

array\_fill\_keys(keys, value)

### Parameter Values

|  |  |
| --- | --- |
| **Parameter** | **Description** |
| keys | Required. Array of values that will be used as keys |
| value | Required. Specifies the value to use for filling the array |

|  |  |
| --- | --- |
| **Return Value:** | Returns the filled array |

<?php

/\* -----------Array Fill Keys----------- \*/

$a = array("a","b","c","d","e");

$newArray = array\_fill\_keys($a, "Testing");

echo "<pre>";

print\_r($newArray);

echo "</pre>";

/\* -----------Array Fill----------- \*/

$newArray1 = array\_fill(-2, 4, "Testing");

echo "<pre>";

print\_r($newArray1);

echo "</pre>";

?>

### PHP Array\_Walk()

The array\_walk() function runs each array element in a user-defined function. The array's keys and values are parameters in the function.

### Syntax

array\_walk(array, myfunction, parameter...)

### Parameter Values

|  |  |
| --- | --- |
| **Parameter** | **Description** |
| array | Required. Specifying an array |
| myfunction | Required. The name of the user-defined function |
| parameter,... | Optional. Specifies a parameter to the user-defined function. You can assign one parameter to the function, or as many as you like |

|  |  |
| --- | --- |
| **Return Value:** | Returns TRUE on success or FALSE on failure |

### PHP array\_walk\_recursive() Function

The array\_walk\_recursive() function runs each array element in a user-defined function. The array's keys and values are parameters in the function. The difference between this function and the array\_walk() function is that with this function you can work with deeper arrays (an array inside an array).

### Syntax

array\_walk\_recursive(array, myfunction, parameter...)

### Parameter Values

|  |  |
| --- | --- |
| **Parameter** | **Description** |
| array | Required. Specifying an array |
| myfunction | Required. The name of the user-defined function |
| parameter,... | Optional. Specifies a parameter to the user-defined function. You can assign one parameter to the function, or as many as you like. |

|  |  |
| --- | --- |
| **Return Value:** | Returns TRUE on success or FALSE on failure |

<?php

/\* -------Array Walk-------\*/

$fruits = array(

"a" => "Lemon",

"b" => "Orange",

"c" => "Banana",

"d" => "Apple"

);

array\_walk($fruits, "myFunction");

function myFunction($value, $key){

echo "$key : $value <br><br>";

}

/\* -------Third parameter as a value -------\*/

array\_walk($fruits, "myFunction1", "is a key of");

function myFunction1($value, $key , $param){

echo "$key $param $value. <br>";

}

echo '<br><br>';

/\* -------array\_walk\_recursive-------\*/

$veggie = array( "1" => "Carrot", "2" => "Tomatoes");

$fruits1 = array(

$veggie,

"a" => "Lemon",

"b" => "Orange",

"c" => "Banana",

"d" => "Apple"

);

array\_walk\_recursive($fruits1, "myFunction1", "is a key of");

?>

### PHP Array\_Map()

The array\_map() function sends each value of an array to a user-made function, and returns an array with new values, given by the user-made function.

### Syntax

array\_map(myfunction, array1, array2, array3, ...)

### Parameter Values

|  |  |
| --- | --- |
| **Parameter** | **Description** |
| myfunction | Required. The name of the user-made function, or null |
| array1 | Required. Specifies an array |
| array2 | Optional. Specifies an array |
| array3 | Optional. Specifies an array |

|  |  |
| --- | --- |
| **Return Value:** | Returns an array containing the values of array1, after applying the user-made function to each one |

<?php

/\* -------Array Map-------\*/

function square($n){

return $n \* $n;

}

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

$newArray = array\_map('square', $a);

echo "<pre>";

print\_r($newArray);

echo "</pre>";

/\* -------Array Map with two arrays -------\*/

function square1($n ,$m){

return "$n for $m";

}

$b = ['lemon', 'orange', 'banana', 'apple', 'guava'];

$newArray1 = array\_map('square1', $a, $b);

echo "<pre>";

print\_r($newArray1);

echo "</pre>";

/\* -------Return Mutlidimensional array-------\*/

function square2($n ,$m){

return [$n => $m];

}

$newArray2 = array\_map('square2', $a, $b);

echo "<pre>";

print\_r($newArray2);

echo "</pre>";

/\* -------Passing no function-------\*/

$newArray3 = array\_map(null, $a, $b);

echo "<pre>";

print\_r($newArray3);

echo "</pre>";

/\* -------Using Associative Array-------\*/

function square3($n){

return strtoupper($n);

}

$a1 = array("one" => "Apple", "two" => "Banana", "three" => "Orange");

$newArray4 = array\_map("square3", $a1);

echo "<pre>";

print\_r($newArray4);

echo "</pre>";

?>

### PHP Array\_Reduce()

The array\_reduce() function sends the values in an array to a user-defined function, and returns a string.

### Syntax

array\_reduce(array, myfunction, initial)

### Parameter Values

|  |  |
| --- | --- |
| **Parameter** | **Description** |
| array | Required. Specifies an array |
| myfunction | Required. Specifies the name of the function |
| initial | Optional. Specifies the initial value to send to the function |

|  |  |
| --- | --- |
| **Return Value:** | Returns the resulting value |

<?php

/\* -------Array Reduce-------\*/

function myFunction($n,$m){

return $n . "-" . $m;

}

$a = ['orange', 'banana', 'apple'];

$newArray = array\_reduce($a, 'myFunction');

echo "<pre>";

print\_r($newArray);

echo "</pre>";

/\* -------Passing third parameter as a Initial Value -------\*/

$newArray1 = array\_reduce($a, 'myFunction', "lemon");

echo "<pre>";

print\_r($newArray1);

echo "</pre>";

//OR

$newArray2 = array\_reduce($a, 'myFunction', 20);

echo "<pre>";

print\_r($newArray2);

echo "</pre>";

/\* -------Use Numeric Index array-------\*/

$a1 = [1, 2, 3, 4, 5];

$newArray3 = array\_reduce($a1, 'myFunction', 20);

echo "<pre>";

print\_r($newArray3);

echo "</pre>";

//SUM

function myFunction1($n,$m){

return $n + $m;

}

$newArray4 = array\_reduce($a1, 'myFunction1');

echo "<pre>";

print\_r($newArray4);

echo "</pre>";

//Multiplication

function myFunction2($n,$m){

return $n \* $m;

}

$newArray5 = array\_reduce($a1, 'myFunction2');

echo "<pre>";

print\_r($newArray5);

echo "</pre>";

/\* -------Pass third Initial Parameter-------\*/

$newArray6 = array\_reduce($a1, 'myFunction2',10);

echo "<pre>";

print\_r($newArray6);

echo "</pre>";

//Additon

/\* -------Can also right -------\*/

function myFunction3($n,$m){

//$n =$n + $m;

$n += $m;

return $n;

}

$newArray7 = array\_reduce($a1, 'myFunction3',10);

echo "<pre>";

print\_r($newArray7);

echo "</pre>";

?>

## PHP Array Sorting

asort()

The asort() function sorts an associative array in ascending order, according to the value.

Syntax

asort(array, sorttype)

Parameter Values

|  |  |
| --- | --- |
| **Parameter** | **Description** |
| *array* | Required. Specifies the array to sort |
| *sorttype* | Optional. Specifies how to compare the array elements/items. Possible values:   * 0 = SORT\_REGULAR - Default. Compare items normally (don't change types) * 1 = SORT\_NUMERIC - Compare items numerically * 2 = SORT\_STRING - Compare items as strings * 3 = SORT\_LOCALE\_STRING - Compare items as strings, based on current locale * 4 = SORT\_NATURAL - Compare items as strings using natural ordering * 5 = SORT\_FLAG\_CASE - |

|  |  |
| --- | --- |
| **Return Value:** | TRUE on success. FALSE on failure |

<?php

/\*-------Array Sorting Function------- \*/

$food = array('orange', 'banana', 'grapes', 'apple');

sort($food); //Sorting

echo "<pre>";

print\_r($food);

echo "</pre>";

rsort($food); //Reverse

echo "<pre>";

print\_r($food);

echo "</pre>";

/\*-------Numerical Index array------- \*/

$food1 = [22,15,3,64];

rsort($food1); //Reverse

echo "<pre>";

print\_r($food1);

echo "</pre>";

/\*-------Use Associative Array ------- \*/

$food2 = array("d" => "lemon",

"a" => "orange",

"b" => "banana",

"c" => "apple"

);

sort($food2); /// ------ sorting

echo "<pre>";

print\_r($food2);

echo "</pre>";

/\*-------- associative sort -----------\*/

$food3 = array("d" => "lemon",

"a" => "orange",

"b" => "banana",

"c" => "apple"

);

asort($food3); //---- maintain also index

echo "<pre>";

print\_r($food3);

echo "</pre>";

/\*-------- associative reverse sort -----------\*/

arsort($food3);

echo "<pre>";

print\_r($food3);

echo "</pre>";

/\*-------Key Sort------- \*/

ksort($food3);

echo "<pre>";

print\_r($food3);

echo "</pre>";

krsort($food3); // Key Reverse Sort

echo "<pre>";

print\_r($food3);

echo "</pre>";

/\*--------- natsort --- Natural Order Sort --- algorithm ------- \*/

$array1 = array("img12.png", "img10.png", "img2.png", "img1.png");

natsort($array1);

echo "<pre>";

print\_r($array1);

echo "</pre>";

/\*--------- natcasesort() - case incentive "natural order" algorithm ------- \*/

$array2 = array("Img12.png", "Img10.png", "img2.png", "img1.png");

natcasesort($array2);

echo "<pre>";

print\_r($array2);

echo "</pre>";

/\*-------array\_multisort --- not create new array------- \*/

$foods = array("orange","banana");

$veggie = array("lemon","carrot");

array\_multisort($foods,$veggie);

echo "<pre>";

print\_r($foods);

echo "</pre>";

echo "<pre>";

print\_r($veggie);

echo "</pre>";

/\*-------array\_reverse------- \*/

$foods1 = array("orange","banana","apple","grapes");

$newArray = array\_reverse($foods1);

echo "<pre>";

print\_r($newArray);

echo "</pre>";

?>

## PHP Array Traversing

<?php

/\*-------Array Traversing Function------- \*/

$food = array('orange', 'banana', 'apple', 'grapes');

echo "<b>Current : </b>" . current($food) ."<br>";

echo "<b>Key : </b>" . key($food) ."<br>";

echo "<b>Pos : </b>" . pos($food) ."<br><br>";

/\*-------Array Traversing Next Function------- \*/

next($food);

echo "<b>Current : </b>" . current($food) ."<br><br>";

next($food);

echo "<b>Current : </b>" . current($food) ."<br><br>";

/\*-------Array Traversing Prev Function------- \*/

prev($food);

echo "<b>Current : </b>" . current($food) ."<br><br>";

/\*-------Array Traversing End Function------- \*/

end($food);

echo "<b>Current : </b>" . current($food) ."<br>";

echo "<b>Key : </b>" . key($food) ."<br><br>";

/\*-------Array Traversing Each Function------- \*/

echo "<pre>";

print\_r(each($food));

echo "</pre>";

/\*-------Array Traversing Reset Function------- \*/

reset($food);

echo "<b>Current : </b>" . current($food) ."<br>";

?>

## PHP Array List

### PHP list() function

The PHP list( ) function is used to assign values to a list of variables in one operation. This function was introduced in PHP 4.0.Syntax

array list ( mixed $var1 [, mixed $... ] );

### Parameter

|  |  |  |
| --- | --- | --- |
| **Parameter** | **Description** | **Is compulsory** |
| Variable1 | The first variable to assign a value to. | compulsory |
| Variable2... | More variables to assign values to. | Optional |

### ****Return value****

The list( ) function returns the assigned array.

<?php

/\*-------Array List Function------- \*/

$color = array('red', 'green', 'blue');

list($a , $b, $c) = $color;

echo "Value of a : $a <br>";

echo "Value of b : $b <br>";

echo "Value of c : $c <br>";

echo "<br><br>";

/\*-------Array Numberic Value------- \*/

$color1 = array(22,55,33);

list($a , $b, $c) = $color1;

echo "Value of a : $a <br>";

echo "Value of b : $b <br>";

echo "Value of c : $c <br>";

echo "<br><br>";

/\*------- Remove Variable------- \*/

list($a , , $c) = $color1;

echo "Value of a : $a <br>";

echo "Value of c : $c <br>";

echo "<br><br>";

/\*-------Only works with Numeric Index------- \*/

$color2 = array(0 => 'red', 1 => 'green', 2 => 'blue');

list($a , $b, $c) = $color2;

echo "Value of a : $a <br>";

echo "Value of b : $b <br>";

echo "Value of c : $c <br>";

echo "<br><br>";

/\*-------All values in One Array------- \*/

list($d[0], $d[1], $d[2]) = $color2;

echo "Value of a : $d[0] <br>";

echo "Value of b : $d[1] <br>";

echo "Value of c : $d[2] <br>";

echo "<br><br>";

?>

## PHP Array Extract & Compact

### PHP extract() Function

The extract() function imports variables into the local symbol table from an array.

This function uses array keys as variable names and values as variable values. For each element it will create a variable in the current symbol table.

This function returns the number of variables extracted on success.

### Syntax

extract(array, extract\_rules, prefix)

### Parameter Values

|  |  |
| --- | --- |
| **Parameter** | **Description** |
| array | Required. Specifies the array to use |
| extract\_rules | Optional. The extract() function checks for invalid variable names and collisions with existing variable names. This parameter specifies how invalid and colliding names are treated.  Possible values:   * EXTR\_OVERWRITE - Default. On collision, the existing variable is overwritten * EXTR\_SKIP - On collision, the existing variable is not overwritten * EXTR\_PREFIX\_SAME - On collision, the variable name will be given a prefix * EXTR\_PREFIX\_ALL - All variable names will be given a prefix * EXTR\_PREFIX\_INVALID - Only invalid or numeric variable names will be given a prefix * EXTR\_IF\_EXISTS - Only overwrite existing variables in the current symbol table, otherwise do nothing * EXTR\_PREFIX\_IF\_EXISTS - Only add prefix to variables if the same variable exists in the current symbol table * EXTR\_REFS - Extracts variables as references. The imported variables are still referencing the values of the array parameter |
| prefix | Optional. If EXTR\_PREFIX\_SAME, EXTR\_PREFIX\_ALL, EXTR\_PREFIX\_INVALID or EXTR\_PREFIX\_IF\_EXISTS are used in the extract\_rules parameter, a specified prefix is required.  This parameter specifies the prefix. The prefix is automatically separated from the array key by an underscore character. |

## 

|  |  |
| --- | --- |
| **Return Value:** | Returns  the number of variables extracted on success |

### PHP compact() Function

The compact() function creates an array from variables and their values.

### Syntax

compact(var1, var2...)

### Parameter Values

|  |  |
| --- | --- |
| **Parameter** | **Description** |
| var1 | Required. Can be a string with the variable name, or an array of variables |
| var2,... | Optional. Can be a string with the variable name, or an array of variables. Multiple parameters are allowed. |

## 

|  |  |
| --- | --- |
| **Return Value:** | Returns an array with all the variables added to it |

<?php

/\*-------Array Extract Function------- \*/

$a = "orange";

$color = array('a' => 'red', 'b' => 'green', 'c' => 'blue');

extract($color);

echo "Value of a : $a <br>";

echo "Value of b : $b <br>";

echo "Value of c : $c <br>";

echo '<br><br>';

/\*-------Extract\_rules------- \*/

//EXTR\_OVERWRITE

extract($color,EXTR\_OVERWRITE);

echo "Value of a : $a <br>";

echo "Value of b : $b <br>";

echo "Value of c : $c <br>";

echo '<br><br>';

//EXTR\_SKIP

$a1 = "orange";

$color = array('a1' => 'red', 'b1' => 'green', 'c1' => 'blue');

extract($color,EXTR\_SKIP);

echo "Value of a1 : $a1 <br>";

echo "Value of b1 : $b1 <br>";

echo "Value of c1 : $c1 <br>";

echo '<br><br>';

//EXTR\_PREFIX\_SAME ;

extract($color,EXTR\_PREFIX\_SAME,"test");

echo "Value of a1 : $a1 <br>";

echo "Value of a1 : $test\_a1 <br>";

echo "Value of b1 : $b1 <br>";

echo "Value of c1 : $c1 <br>";

echo '<br><br>';

//EXTR\_PREFIX\_ALL

extract($color,EXTR\_PREFIX\_ALL,"test");

echo "Value of a1 : $a1 <br>";

echo "Value of a1 : $test\_a1 <br>";

echo "Value of b1 : $test\_b1 <br>";

echo "Value of c1 : $test\_c1 <br>";

echo '<br><br>';

/\*-------Compact Function------- \*/

$firstname = "Yahoo";

$lastname = "Baba";

$age = "20";

$gender = "Male";

$country = "India";

$newArray = compact("firstname", "lastname", "age");

echo '<pre>';

print\_r($newArray);

echo '</pre>';

/\*-------Use extra array in compact function------- \*/

$extra = ["gender" , "country"];

$newArray1 = compact("firstname", "lastname", "age",$extra );

echo '<pre>';

print\_r($newArray1);

echo '</pre>';

?>

### PHP Array\_Range

The PHP range( ) function is used to create an array containing a range of elements. The PHP range( ) function returns an array of elements from low to high. This function was introduced in PHP 4.0.

### Syntax

array range ( mixed $start , mixed $end [, number $step = 1 ] );

### Parameter

|  |  |  |
| --- | --- | --- |
| **Parameter** | **Description** | **Is compulsory** |
| low | It is a lower range of the array. | compulsory |
| high | It is an upper range of the array. | compulsory |
| step | Steps to increase array element. By default it is 1. | Optional |

### Returns

The range( ) function returns array of elements

<?php

/\*-------Array Range Function------- \*/

$newArray = range(0, 10);

echo "<pre>";

print\_r($newArray);

echo "</pre>";

/\*-------Using step------- \*/

$newArray1 = range(0, 100, 10);

echo "<pre>";

print\_r($newArray1);

echo "</pre>";

/\*-------Using Alphabet------- \*/

$newArray2 = range('a', 'h');

echo "<pre>";

print\_r($newArray2);

echo "</pre>";

//OR

$newArray3 = range('h', 'a');

echo "<pre>";

print\_r($newArray3);

echo "</pre>";

/\*-------Foreach Array Range Function------- \*/

foreach (range('h', 'a') as $letter) {

echo $letter . "<br>";

}

?>

## PHP Explode & Implode

### PHP implode() Function

The implode() function returns a string from the elements of an array.

### Syntax

implode(separator,array)

### Parameter Values

|  |  |
| --- | --- |
| **Parameter** | **Description** |
| separator | Optional. Specifies what to put between the array elements. Default is "" (an empty string) |
| array | Required. The array to join to a string |

|  |  |
| --- | --- |
| **Return Value:** | Returns a string from elements of an array |

### PHP explode() Function

The explode() function breaks a string into an array.

### Syntax

explode(separator,string,limit)

### Parameter Values

|  |  |
| --- | --- |
| **Parameter** | **Description** |
| separator | Required. Specifies where to break the string |
| string | Required. The string to split |
| limit | Optional. Specifies the number of array elements to return.  Possible values:   * Greater than 0 - Returns an array with a maximum of limit element(s) * Less than 0 - Returns an array except for the last -limit elements() * 0 - Returns an array with one element |

## 

|  |  |
| --- | --- |
| **Return Value:** | Returns an array of strings |

<?php

/\*-------Explode Function------- \*/

$str = "Hello world. Its a beautiful day";

$array = explode(" ",$str);

echo "<pre>";

print\_r($array);

echo "</pre>";

/\* ------ \*/

$array1 = explode(".",$str);

echo "<pre>";

print\_r($array1);

echo "</pre>";

/\*-------Using Limit with explode ------- \*/

$array2 = explode(" ",$str, 2);

echo "<pre>";

print\_r($array2);

echo "</pre>";

$array3 = explode(" ",$str, 0); //-- Return in one Array Value

echo "<pre>";

print\_r($array3);

echo "</pre>";

$array4 = explode(" ",$str, -1);

echo "<pre>";

print\_r($array4);

echo "</pre>";

/\*----------- Another example -----------\*/

//EXAMPLE I

$str1 = "red,green,blue,orange";

$array5 = explode(",",$str1);

echo "<pre>";

print\_r($array5);

echo "</pre>";

//EXAMPLE II

$str2 = "red,green-blue-orange";

$array6 = explode(",",$str2);

echo "<pre>";

print\_r($array6);

echo "</pre>";

//EXAMPLE III

$array7 = explode("-",$str2);

echo "<pre>";

print\_r($array7);

echo "</pre>";

/\*-------Implode Function------- \*/

$arr = array("Hello","World!","Beautiful","Day!");

$strr = implode(" ",$arr);

echo $strr;

echo "<br><br>";

//EMAXPLE II

$strr1 = implode("-",$arr);

echo $strr1;

echo "<br><br>";

//EMAXPLE III

$strr2 = implode(",",$arr);

echo $strr2;

echo "<br><br>";

//EMAXPLE III

$strr3 = implode("wow",$arr);

echo $strr3;

echo "<br><br>";

//EMAXPLE III

$strr4 = implode("<br>",$arr);

echo $strr4;

echo "<br><br>";

//You can use (Join) function work same as Implode

$strr5 = join("<br>",$arr);

echo $strr5;

echo "<br><br>";

?>

## PHP String Str\_split & Chunk\_split

### Str\_split() Function

The str\_split() function splits a string into an array.

### Syntax

str\_split(string,length)

### Parameter Values

|  |  |
| --- | --- |
| **Parameter** | **Description** |
| string | Required. Specifies the string to split |
| length | Optional. Specifies the length of each array element. Default is 1 |

|  |  |
| --- | --- |
| **Return Value:** | If length is less than 1, the str\_split() function will return FALSE. If length is larger than the length of string, the entire string will be returned as the only element of the array. |

### chunk\_split()

The chunk\_split() function splits a string into a series of smaller parts.

### Syntax

chunk\_split(string,length,end)

### Parameter Values

|  |  |
| --- | --- |
| **Parameter** | **Description** |
| string | Required. Specifies the string to split |
| length | Optional. A number that defines the length of the chunks. Default is 76 |
| end | Optional. A string that defines what to place at the end of each chunk. Default is \r\n |

|  |  |
| --- | --- |
| **Return Value:** | Returns the split string |

<?php

/\*-------String Split Function------- \*/

$str = "Yahoo Baba";

$array = str\_split($str);

echo "<pre>";

print\_r($array);

echo "</pre>";

/\*-------String Split Length------- \*/

$array2 = str\_split($str, 3);

echo "<pre>";

print\_r($array2);

echo "</pre>";

/\*-------String Chunk Function------- \*/

$newStr = chunk\_split($str,1,".");

echo $newStr."<br><br>";

$newStr = chunk\_split($str,3,"-");

echo $newStr."<br><br>";

$newStr = chunk\_split($str,3,"<br>");

echo $newStr."<br><br>";

?>

## PHP String LowerCase & UpperCase

### strtolower() Function

The strtolower() function converts a string to lowercase.

### Syntax

strtolower(string)

### Parameter Values

|  |  |
| --- | --- |
| **Parameter** | **Description** |
| string | Required. Specifies the string to convert |

|  |  |
| --- | --- |
| **Return Value:** | Returns the the lowercased string |

### strtoupper() function

The strtoupper() function converts a string to uppercase.

### Syntax

strtoupper(string)

### Parameter Values

|  |  |
| --- | --- |
| **Parameter** | **Description** |
| string | Required. Specifies the string to convert |

|  |  |
| --- | --- |
| **Return Value:** | Returns the the uppercased string |

<?php

/\*-------converts a string to Lowercase------- \*/

$str = "Yahoo Baba";

$newStr = strtolower($str);

echo $newStr;

echo "<br><br>";

/\*-------converts a string to Uppercase------- \*/

$newStr1 = strtoupper($str);

echo $newStr1;

echo "<br><br>";

/\*-------Convert the first character of a string to lowercase: ------- \*/

$newStr2 = lcfirst($str);

echo $newStr2;

echo "<br><br>";

/\*-------Converts the first character of a string to uppercase------- \*/

$newStr3 = ucfirst($str);

echo $newStr3;

echo "<br><br>";

/\*-------Converts the first character of each word in a string to uppercase------- \*/

$newStr4 = ucwords($str);

echo $newStr4;

echo "<br><br>";

?>