# Web Development Using php

# Unit – 2

## What is PHP ?

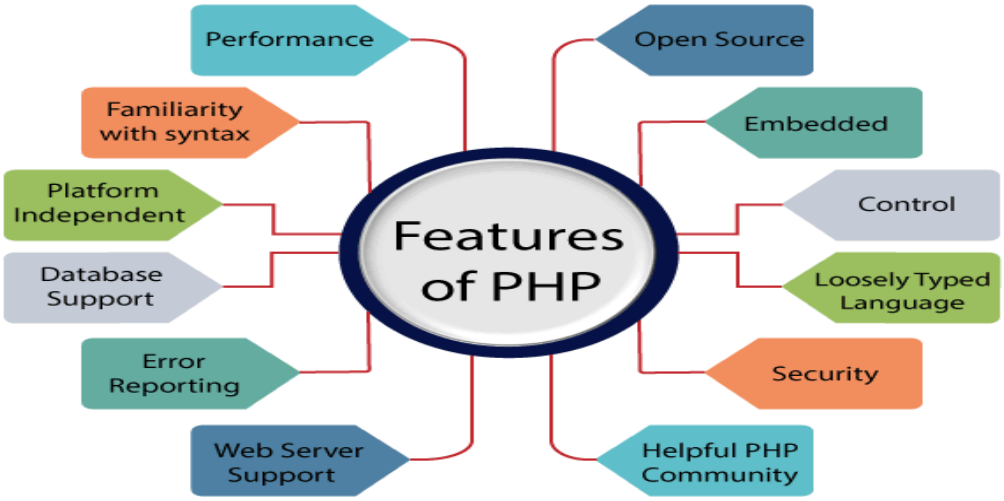
PHP is an **open-source**, **interpreted**, and **object-oriented** scripting language that can be executed at the server-side. PHP is well suited for web development. Therefore, it is used to develop web applications (an application that executes on the server and generates the dynamic page.).

PHP was created by **Rasmus Lerdorf in 1994** but appeared in the market in 1995. **PHP 8.1** is the latest version of PHP, which was released on **25 November 2021**. Some important points need to be noticed about PHP are as followed :

- PHP stands for **Hypertext Preprocessor**.
- PHP is an **interpreted language**, i.e., there is no need for compilation.
- PHP is **faster** than other scripting languages, for example, ASP and JSP.
- PHP is a **server-side** scripting language, which is used to manage the dynamic content of the website.
- PHP can be **embedded into HTML**.
- PHP is an **object-oriented** language.
- PHP is an **open-source** scripting language.
- PHP is **simple** and **easy** to learn language.

## PHP Features :-

- PHP is very popular language because of its simplicity and open source. There are some important features of PHP given below:




- **Performance :-**

PHP script is executed much faster than those scripts which are written in other languages such as JSP and ASP. PHP uses its own memory, so the server workload and loading time is automatically reduced, which results in faster processing speed and better performance .

- **Open Source :-**

PHP source code and software are freely available on the web. You can develop all the versions of PHP according to your requirement without paying any cost. All its components are free to download and use.

- **Familiarity with syntax :-**

PHP has easily understandable syntax. Programmers are comfortable coding with it.

- **Embedded :-** PHP code can be easily embedded within HTML tags and script.
- **Platform Independent :-**

PHP is available for WINDOWS, MAC, LINUX & UNIX operating system. A PHP application developed in one OS can be easily executed in other OS also.

- **Database Support :-**

PHP supports all the leading databases such as MySQL, SQLite, ODBC, etc.

- **Error Reporting :-**

PHP has predefined error reporting constants to generate an error notice or warning at runtime. E.g., E_ERROR, E_WARNING, E_STRICT, E_PARSE.

- **Loosely Typed Language :-**

PHP allows us to use a variable without declaring its datatype. It will be taken automatically at the time of execution based on the type of data it contains on its value.

- **Web servers Support :-**

PHP is compatible with almost all local servers used today like Apache, Netscape, Microsoft IIS, etc.

- **Security :-**

PHP is a secure language to develop the website. It consists of multiple layers of security to prevent threads and malicious attacks.

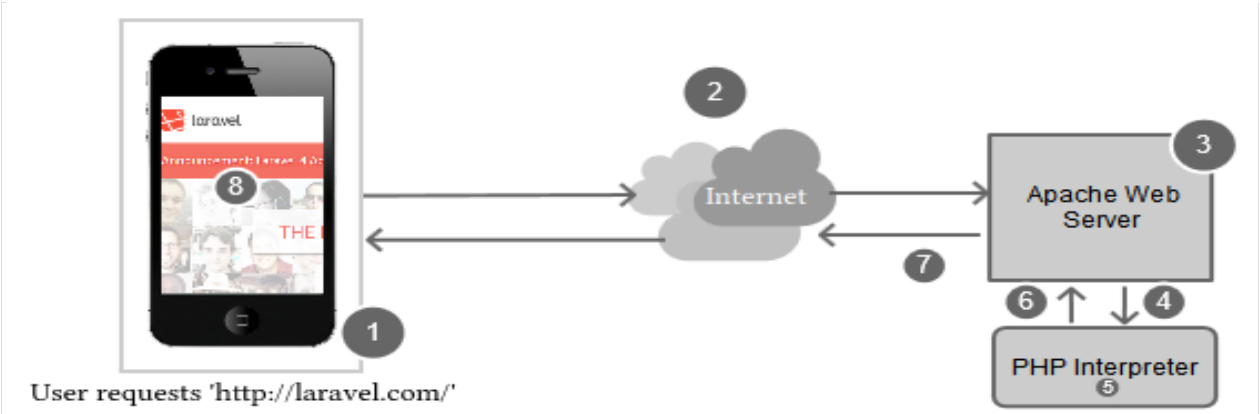
- **Control :-**

Different programming languages require long script or code, whereas PHP can do the same work in a few lines of code. It has maximum control over the websites like you can make changes easily whenever you want.

- **A Helpful PHP Community:**

It has a large community of developers who regularly updates documentation, tutorials, online help, and FAQs. Learning PHP from the communities is one of the significant benefits.

## How PHP Works ?




**Step 1 :-**

The user enters `http://laravel.com` into their browser and taps/hits 'enter'.

**Step 2 :-**

After the user has tapped/hit 'enter', the browser sends the page request over the Internet to the web server.

**Step 3 :-**

The web server *gets* the request and analyzes the request information. Apache realizes that we didn't specify a file, so it looks for a directory index and finds `index.php`.

**Step 4 :-**

Since Apache knows to send files that end with the `.php` file extension to the PHP interpreter, it asks PHP to execute the file.

**Step 5 :-**

In this step, PHP is executing the code contained in the `index.php` file from the request. During this step, PHP may interact with databases, the file system or make external API calls, amongst other things.

**Step 6 :-**

| | |
|---|---|
| | After PHP has finished executing the `index.php` file, it sends the output back to Apache. |
| **Step 7 :-** | |
| | Apache receives the output from PHP and sends it back over the Internet to a user's web browser. This is called the `web response`. |
| **Step 8 :-** | |
| | The user's web browser receives the response from the server, and renders the web page on a computer or device. |

## PHP | php.ini File Configuration :-

### Purpose :-

- It’s a very essential configuration file that controls what a user can or cannot do with the website.
- Each time PHP is initialized, the php.ini file is read by the system.
- Sometimes you need to change the behaviour of PHP at runtime, then this configuration file is to use.
- All the settings related to registering global variables, uploading maximum size, display log errors, resource limits, the maximum time to execute a PHP script and others are written in a file as a set of directives that helps in declaring changes.
- The php.ini file is the default configuration file for running applications that require PHP. It is used to control variables such as upload sizes, file timeouts, and resource limits.
- php.ini file is the configuration file. It is always checked when the server gets started or HTTP is restarted in the module and it configures the website to know what a user can do or can’t do with a website.
- It also helps with the administration of the web server easily.

### Note :-

Whenever some changes are made to the file, you need to restart your web server.

It helps in easy administration of the web with a server using these configuration files. We can also write our own custom configuration files.

## PHP | Basic Syntax :-

The structure which defines PHP computer language is called **PHP syntax**.

PHP files are saved with the ".php" extension. PHP scripts can be written anywhere in the document within PHP tags along with normal HTML.

**Canonical PHP Tags** :- The script starts with **<?php** and ends with **?>**. These tags are also called 'Canonical PHP tags'. Everything outside of a pair of opening and closing tags is ignored by the PHP parser. The open and closing tags are called delimiters. Every PHP command ends with a semi-colon (**;**). Let's look at the *hello world* program in PHP.

```
<?php
# Here echo command is used to print
echo "Hello, world!";
?>
```

# PHP Variables :-

**Variables :-** Variables in a program are used to store some values or data that can be used later in a program. The variables are also like containers that store character values, numeric values, memory addresses, and strings.

In PHP, a variable is declared using a **$ sign** followed by the variable name. Here, some important points to know about variables:

- As PHP is a **loosely typed language**, so we do not need to declare the data types of the variables. It automatically analyses the values and makes conversions to its correct datatype.
- After declaring a variable, it can be **reused** throughout the code.

- **Assignment Operator (=)** is used to assign the value to a variable.

**Syntax of declaring a variable in PHP is given below :-**

```
$ variablename = value ;
```

## Rules for declaring PHP variable:

- A variable must start with a dollar ($) sign, followed by the variable name.
- It can only contain alpha-numeric character and underscore (A-z, 0-9, _).
- A variable name must start with a letter or underscore (_) character.
- A PHP variable name cannot contain spaces.
- One thing to be kept in mind that the variable name cannot start with a number or special symbols.
- PHP variables are case-sensitive, so $name and $NAME both are treated as different variable.

**PHP has a total of eight data types which we use to construct our variables :-**

- **Integers** − are whole numbers, without a decimal point, like 4195.
- **Doubles** − are floating-point numbers, like 3.14159 or 49.1.
- **Booleans** − have only two possible values either true or false.
- **NULL** − is a special type that only has one value: NULL.
- **Strings** − are sequences of characters, like 'PHP supports string operations.'
- **Arrays** − are named and indexed collections of other values

- **Objects** − are instances of programmer-defined classes, which can package up both other kinds of values and functions that are specific to the class.
- **Resources** − are special variables that hold references to resources external to PHP (such as database connections).

# Datatypes in PHP :-

**Data Types** define the type of data a variable can store. PHP allows eight different types of data types. All of them are discussed below. There are pre-defined, user-defined, and special data types.

**The predefined data types are :-**

- Boolean
- Integer
- Double
- String

**The user-defined (compound) data types are :-**

- Array
- Objects

**The special data types are :-**

- NULL
- Resource

**The first five are called simple data types and the last three are compound data types .**

1. **Integer** :- Integers hold only whole numbers including positive and negative numbers, i.e., numbers without fractional part or decimal point. They can be decimal (base 10), octal (base 8), or hexadecimal (base 16). The default base is decimal (base 10). The octal integers

can be declared with leading 0 and the hexadecimal can be declared with leading 0x. The range of integers must lie between -2^31 to 2^31.

**Example :-**

```
<?php
// decimal base integers
$deci1 = 50;
$deci2 = 654;
// octal base integers
$octal1 = 07;
// hexadecimal base integers
$octal = 0x45;
$sum = $deci1 + $deci2;
echo $sum;
echo "\n\n";
//returns data type and value
var_dump($sum)
?>
```

**Output :-**

```
704

int(704)
```

2. **Double** :- It Can hold numbers containing fractional or decimal parts including positive and negative numbers or a number in exponential form. By default, the variables add a minimum number of decimal places. The Double data type is the same as a float as floating-point numbers or real numbers.

   **Example :-**

```
<?php

$val1 = 50.85;

$val2 = 654.26;

$sum = $val1 + $val2;

echo $sum;

echo "\n\n";

 //returns data type and value

var_dump($sum)

?>
```

**Output :-**

```
705.11
float(705.11)
```

**3. String** :- Hold letters or any alphabets, even numbers are included. These are written within double quotes during declaration. The strings can also Be written within single quotes, but they will be treated differently while printing variables. To clarify this look at the example below.

**Example** :-

```
<?php

$name = "Krishna";
```

```
echo "The name of the Geek is $name \n";

echo 'The name of the geek is $name ';

echo "\n\n";

//returns data type, size and value

var_dump($name)

?>
```

**Output :-**
```
The name of the Geek is Krishna
The name of the geek is $name

string(7) "Krishna"
```

**4. Boolean** :- Boolean data types are used in conditional testing. Hold only two values, either TRUE(1) or FALSE(0). Successful events will return *true* and unsuccessful events return *false.* NULL type values are also treated as *false* in Boolean. Apart from NULL, 0 is also considered false in boolean. If a string is empty then it is also considered false in boolean data type.

**Example** :-
```
<?php
if(TRUE)
echo "This condition is TRUE";
if(FALSE)
echo "This condition is not TRUE";
?>
```

**Output :-**
```
This condition is TRUE
```

**5. Array**: Array is a compound data type that can store multiple values of the

same data type. Below is an example of an array of integers. It combines a series of data that are related together.

### Example :-

```
<?php
$intArray = array( 10, 20 , 30);
echo "First Element: $intArray[0]\n";
echo "Second Element: $intArray[1]\n";
echo "Third Element: $intArray[2]\n\n";
//returns data type and value
var_dump($intArray);
?>
```

**Output** :-

```
First Element: 10
Second Element: 20
Third Element: 30
array(3) {
  [0]=>
  int(10)
  [1]=>
  int(20)
  [2]=>
  int(30)
}
```

We will discuss arrays in detail in further articles.

**6. Objects** :- Objects are defined as instances of user-defined classes that can hold both values and functions and information for data processing specific To the class. This is an advanced topic and will be discussed in detail in further

articles. When the objects are created, they inherit all the properties and behaviours from the class, having different values for all the properties. Objects are explicitly declared and created from the *new* keyword.

```
<?php

class gfg {

var $message;

function gfg($message) {

$this->message = $message;

}

function msg() {

return "This is an example of " . $this->message . "!";

} }

// instantiating a object

$newObj = new gfg("Object Data Type");

echo $newObj -> msg();

 ?>
```

**Output :-** **`This is an example of Object Data Type!`**

7. **NULL :-** These are special types of variables that can hold only one value i.e., NULL. We follow the convention of writing it in capital form, but it's case-sensitive. If a variable is created without a value or no value, it is automatically

assigned a value of NULL. It is written in capital letters.

**Example:**

```
<?php

$nm = NULL;

echo $nm;

// this will return no output

// return data type

var_dump($nm);

?>
```

**Output :-** **NULL**

8. **Resources** :- Resources in PHP are not an exact data type. These are basically used to store references to some function call or to external PHP resources. For example, consider a database call. This is an external resource. Resource variables hold special handles for files and database connections. We will discuss resources in detail in further articles.

**Note:**

- To check the type and value of an expression, use the **var_dump()** function which dumps information about a variable.
- PHP allows the developer to cast the data type.

# Variable Scope

**Scope of a variable is defined as its extent in a program within which it**

**can be accessed, i.e. the scope of a variable is the portion of the program within which it is visible or can be accessed.**

PHP has three types of variable scopes:

1. **Local variable**
2. **Global variable**
3. **Static variable**

# Local variable

The variables that are declared within a function are called local variables for that function. These local variables have their scope only in that particular function in which they are declared. This means that these variables cannot be accessed outside the function, as they have local scope.

A variable declaration outside the function with the same name is completely different from the variable declared inside the function. Let's understand the local variables with the help of an example:

*File: local_variable1.php*

```
<?php
    function local_var()
     {
       $num = 45;  //local variable
       echo "Local variable declared inside the function is: ". $num;
     }
     local_var();
```

```
?>
```

**Output :-**

```
Local variable declared inside the function is: 45
```

*File: local_variable2.php*

```
<?php
    function mytest()  {
      $lang = "PHP";
      echo "Web development language: " .$lang;
    }
    mytest();
    //using $lang (local variable) outside the function will generate an error
    echo $lang;
?>
```

**Output:**

```
Web development language: PHP
Notice: Undefined variable: lang in D:\xampp\htdocs\program\p3.php on line 28
```

## Global variable

The global variables are the variables that are declared outside the function. These variables can be accessed anywhere in the program. To access the global variable within a function, use the GLOBAL keyword before the variable. However, these variables can be directly accessed or used outside the function without any keyword. Therefore there is no

need to use any keyword to access a global variable outside the function.

Let's understand the global variables with the help of an example:

## Example:

*File: global_variable1.php*

```
<?php
   $name = "Sanaya Sharma";         //Global Variable
   function global_var()
   {
      global $name;
      echo "Variable inside the function: ". $name;
      echo "</br>";
   }
      global_var();
      echo "Variable outside the function: ". $name;
?>
```

**Output:**

```
Variable inside the function: Sanaya Sharma
Variable outside the function: Sanaya Sharma
```

**Note: Without using the global keyword, if you try to access a global variable inside the function, it will generate an error that the variable is undefined.**

## Example:

*File: global_variable2.php*

```
<?php
$name = "Sanaya Sharma";         //global variable
```

```
    function global_var()
    {
        echo "Variable inside the function: ". $name;
        echo "</br>";
     }
      global_var();
?>
```

**Output:**

```
Notice: Undefined variable: name in D:\xampp\htdocs\program\p3.php on line 6
Variable inside the function:
```

## Using $GLOBALS instead of global

Another way to use the global variable inside the function is predefined $GLOBALS array.

**Example:**

*File: global_variable3.php*

```
<?php
  $num1 = 5;       //global variable
  $num2 = 13;      //global variable
    function global_var()
    {
        $sum = $GLOBALS['num1'] + $GLOBALS['num2'];
        echo "Sum of global variables is: " .$sum;
    }
   global_var();
?>
```

**Output:**

```
Sum of global variables is: 18
```

If two variables, local and global, have the same name, then the local variable has higher priority than the global variable inside the function.

### Example:

*File: global_variable2.php*

```
<?php
$x = 5;
function mytest()
  {
     $x = 7;
     echo "value of x: " .$x;
  }
    mytest();
  ?>
```

### Output:

```
Value of x: 7
```

***Note: local variable has higher priority than the global variable.***

# Static variable

It is a feature of PHP to delete the variable, once it completes its execution and memory is freed. Sometimes we need to store a variable even after completion of function execution. Therefore, another important feature of variable scoping is static variable. We use the static keyword before the variable to define a variable, and this variable is called as **static variable**.

Static variables exist only in a local function, but it does not free its memory after the program execution leaves the scope. Understand it with the help of an example:

## Example:

*File: static_variable.php*

```
<?php
function static_var()
 {
    static $num1 = 3;        //static variable
    $num2 = 6;           //Non-static variable
       //increment in non-static variable
    $num1++;
       //increment in static variable
     $num2++;
       echo "Static: " .$num1 ."</br>";
       echo "Non-static: " .$num2 ."</br>";
  }
     //first function call
    static_var();
    / /second function call
    static_var();
?>
```

## Output:

```
Static: 4
Non-static: 7
Static: 5
Non-static: 7
```

You have to notice that $num1 regularly increments after each function call, whereas $num2 does not. This is why because $num1 is not a static variable, so it freed its memory after the execution of each function call.

## PHP Constants

Constants are identifiers that can be assigned any fixed values. They are similar to a variable except that they can never be changed. They remain constant throughout the program and cannot be altered during execution.

Once a constant is defined, it cannot be undefined or redefined.

Constant identifiers should be written in upper case following the convention. By default, a constant is always case-sensitive, unless mentioned.

A constant name must never start with a number. It always starts with a letter or underscores, followed by letter, numbers or underscore. It should not contain any special characters except underscore, as mentioned.

**PHP constants can be defined by 2 ways:**

1. **Using define() function .**
2. **Using const keyword .**

Constants are similar to the variable except once they defined, they can never be undefined

or changed. They remain constant across the entire program. PHP constants follow the same PHP variable rules. **For example**, it can be started with a letter or underscore only. Conventionally, PHP constants should be defined in uppercase letters.

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

# PHP constant: define()

Use the define() function to create a constant. It defines constant at run time. Let's see the syntax of define() function in PHP.

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

1. **name:** It specifies the constant name.
2. **value:** It specifies the constant value.
3. **case-insensitive:** Specifies whether a constant is case-insensitive. Default value is
4. false. It means it is case sensitive by default.

Let's see the example to define PHP constant using **define().**

*File: constant1.php*

```
<?php
   define("MESSAGE","Hello JavaTpoint PHP");
   echo MESSAGE;
?>
```

## Output:

```
Hello JavaTpoint PHP
```

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

*File: constant2.php*

```
<?php
define("MESSAGE","Hello JavaTpoint PHP",true);//not case sensitive
echo MESSAGE, "</br>";
echo message;
?>
```

**Output:**

```
Hello JavaTpoint PHP
Hello JavaTpoint PHP
```

*File: constant3.php*

```
<?php
define("MESSAGE","Hello JavaTpoint PHP",false);//case sensitive
echo MESSAGE;
echo message;
?>
```

**Output:**

```
Hello JavaTpoint PHP
Notice: Use of undefined constant message - assumed 'message'
in C:\wamp\www\vconstant3.php on line 4
message
```

# PHP constant: const keyword

PHP introduced a keyword **const** to create a constant. The const keyword defines constants at compile time. It is a language construct, not a function. The constant defined using

const keyword are **case-sensitive**.

*File: constant4.php*

```
<?php
   const MESSAGE="Hello const by JavaTpoint PHP";
   echo MESSAGE;
?>
```

### Output:

```
Hello const by JavaTpoint PHP
```

# Constant() function

There is another way to print the value of constants using constant() function instead of using the echo statement.

### Syntax :-

The syntax for the following constant function:

- **constant (name)**

*File: constant5.php*

```
<?php
    define("MSG", "JavaTpoint");
    echo MSG, "</br>";
    echo constant("MSG");
    //both are similar
?>
```

**Output:**

```
JavaTpoint
JavaTpoint
```

# Constant vs Variables

| Constant | Variables |
| --- | --- |
| Once the constant is defined, it can never be redefined. | A variable can be undefined as well as redefined easily. |
| A constant can only be defined using define() function. It cannot be defined by any simple assignment. | A variable can be defined by simple assignment (=) operator. |
| There is no need to use the dollar ($) sign before constant during the assignment. | To declare a variable, always use the dollar ($) sign before the variable. |
| Constants do not follow any variable scoping rules, and they can be defined and accessed anywhere. | Variables can be declared anywhere in the program, but they follow variable scoping rules. |
| Constants are the variables whose values can't be changed throughout the program. | The value of the variable can be changed. |

| | |
|---|---|
| By default, constants are global. | Variables can be local, global, or static. |

---

# PHP Operators

PHP Operator is a symbol i.e used to perform operations on operands. In simple words, operators are used to perform operations on variables or values. For example:

```
$ num=10+20;      //+ is the operator and 10,20 are operands
```

In the above eg , + is the binary + operator, 10 and 20 are operands and $num is variable.

PHP Operators can be categorized in following forms:

- **Arithmetic Operators**
- **Assignment Operators**
- **Bitwise Operators**
- **Comparison Operators**
- **Incrementing/Decrementing Operators**
- **Logical Operators**
- **String Operators**
- **Array Operators**
- **Type Operators**
- **Execution Operators**
- **Error Control Operators**

We can also categorize operators on behalf of operands. They can be categorized in 3 forms:

- **Unary Operators :** works on single operands such as ++, -- etc.
- **Binary Operators :** works on two operands such as binary +, -, *, / etc.

- **Ternary Operators :** works on three operands such as "?:".

## Arithmetic Operators

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

| Operator | Name | Example | Explanation |
|---|---|---|---|
| + | Addition | $a + $b | Sum of operands |
| - | Subtraction | $a - $b | Difference of operands |
| * | Multiplication | $a * $b | Product of operands |
| / | Division | $a / $b | Quotient of operands |
| % | Modulus | $a % $b | Remainder of operands |
| ** | Exponentiation | $a ** $b | $a raised to the power $b |

The exponentiation (**) operator has been introduced in PHP 5.6.

**Example :-**

```
<?php

$x= 29; // Variable 1

$y= 4; // Variable 2
```

```
        // Some arithmetic operations on
        // these two variables
         echo($x+ $y), "\n";
         echo($x- $y), "\n";
         echo($x* $y), "\n";
         echo($x/ $y), "\n";
         echo($x% $y), "\n";
    ?>
```

**OUTPUT :-**

```
33
25
116
7.25
1
```

# Assignment Operators

The assignment operators are used to assign value to different variables. The basic assignment operator is "=".

| Operator | Name | Example | Explanation |
|---|---|---|---|
| = | Assign | $a = $b | The value of right operand is assigned to the left operand. |

| | | | |
|---|---|---|---|
| += | Add then Assign | $a += $b | Addition same as $a = $a + $b |
| -= | Subtract then Assign | $a -= $b | Subtraction same as $a = $a - $b |
| *= | Multiply then Assign | $a *= $b | Multiplication same as $a = $a * $b |
| /= | Divide then Assign (quotient) | $a /= $b | Find quotient same as $a = $a / $b |
| %= | Divide then Assign (remainder) | $a %= $b | Find remainder same as $a = $a % $b |

## Bitwise Operators

The bitwise operators are used to perform bit-level operations on operands. These operators allow the evaluation and manipulation of specific bits within the integer.

| Operator | Name | Example | Explanation |
|---|---|---|---|
| & | And | $a & $b | Bits that are 1 in both $a and $b are set to 1, otherwise 0. |
| \| | Or (Inclusive or) | $a \| $b | Bits that are 1 in either $a or $b are set to 1 |

| | | | |
|---|---|---|---|
| ^ | Xor (Exclusive or) | $a ^ $b | Bits that are 1 in either $a or $b are set to 0. |
| ~ | Not | ~$a | Bits that are 1 set to 0 and bits that are 0 are set to 1 |
| << | Shift left | $a << $b | Left shift the bits of operand $a $b steps |
| >> | Shift right | $a >> $b | Right shift the bits of $a operand by $b number of places |

## Comparison Operators

Comparison operators allow comparing two values, such as number or string. Below the list of comparison operators are given:

| Operator | Name | Example | Explanation |
|---|---|---|---|
| == | Equal | $a == $b | Return TRUE if $a is equal to $b |
| === | Identical | $a === $b | Return TRUE if $a is equal to $b, and they are of same data type |
| !== | Not identical | $a !== $b | Return TRUE if $a is not equal to $b, and they are not of same data type |
| != | Not equal | $a != $b | Return TRUE if $a is not equal to $b |
| <> | Not equal | $a <> $b | Return TRUE if $a is not equal to $b |

| | | | |
|---|---|---|---|
| < | Less than | $a < $b | Return TRUE if $a is less than $b |
| > | Greater than | $a > $b | Return TRUE if $a is greater than $b |
| <= | Less than or equal to | $a <= $b | Return TRUE if $a is less than or equal $b |
| >= | Greater than or equal to | $a >= $b | Return TRUE if $a is greater than or equal $b |
| <=> | Spaceship | $a <=>$b | Return -1 if $a is less than $b Return 0 if $a is equal $b Return 1 if $a is greater than $b |

## Example :-

```
<?php

$a= 80;

$b= 50;

$c= "80"

  // Here var_dump function has
been used to

// display structured information.
We will learn

// about this function in complete
details in further articles.
```

```
var_dump($a== $c) + "\n";

var_dump($a!= $b) + "\n";

var_dump($a<> $b) + "\n";

var_dump($a=== $c) + "\n";

var_dump($a!== $c) + "\n";

var_dump($a< $b) + "\n";

var_dump($a> $b) + "\n";

var_dump($a<= $b) + "\n";

var_dump($a>= $b);

?>
```

**Output :-**

```
bool(true)
bool(true)
bool(true)
bool(false)
bool(true)
bool(false)
bool(true)
bool(false)
bool(true)
```

# Incrementing/Decrementing Operators

The increment and decrement operators are used to increase and decrease the value of a

variable.

| Operator | Name | Example | Explanation |
|---|---|---|---|
| ++ | Increment | ++$a | Increment the value of $a by one, then return $a |
| | | $a++ | Return $a, then increment the value of $a by one |
| -- | decrement | --$a | Decrement the value of $a by one, then return $a |
| | | $a-- | Return $a, then decrement the value of $a by one |

## Logical Operators

The logical operators are used to perform bit-level operations on operands. These operators Allow the evaluation and manipulation of specific bits within the integer.

| Operator | Name | Example | Explanation |
|---|---|---|---|
| and | And | $a and $b | Return TRUE if both $a and $b are true |
| Or | Or | $a or $b | Return TRUE if either $a or $b is true |
| xor | Xor | $a xor $b | Return TRUE if either $ or $b is true but not both |

| | | | |
|---|---|---|---|
| ! | Not | ! $a | Return TRUE if $a is not true |
| && | And | $a && $b | Return TRUE if either $a and $b are true |
| \|\| | Or | $a \|\| $b | Return TRUE if either $a or $b is true |

## Example :-

```
<?php

$x= 50;

$y= 30;

  if($x== 50 and$y== 30)

      echo"and Success \n";

    if($x== 50 or$y== 20)

      echo"or Success \n";

    if($x== 50 xor$y== 20)

      echo"xor Success \n";

    if($x== 50 && $y== 30)

      echo"&& Success \n";

    if($x== 50 || $y== 20)
```

```
        echo"|| Success \n";
      if (!$z)
       echo"! Success \n";
?>
```

**Output :-**

```
and Success
or Success
xor Success
&& Success
|| Success
! Success
```

## String Operators

The string operators are used to perform the operation on strings. There are two string operators in PHP, which are given below :

| Operator | Name | Example | Explanation |
|---|---|---|---|
| . | Concatenation | $a . $b | Concatenate both $a and $b |
| .= | Concatenation and Assignment | $a .= $b | First concatenate $a and $b, then assign the concatenated string to $a, e.g. $a = $a . $b |

## Array Operators

The array operators are used in case of array. Basically, these operators are used to compare the values of arrays.

| Operator | Name | Example | Explanation |
|---|---|---|---|
| + | Union | $a + $y | Union of $a and $b |
| == | Equality | $a == $b | Return TRUE if $a and $b have same key/value pair |
| != | Inequality | $a != $b | Return TRUE if $a is not equal to $b |
| === | Identity | $a === $b | Return TRUE if $a and $b have same key/value pair of same type in same order |
| !== | Non-Identity | $a !== $b | Return TRUE if $a is not identical to $b |
| <> | Inequality | $a <> $b | Return TRUE if $a is not equal to $b |

## Type Operators

The type operator **instance of** is used to determine whether an object, its parent and its derived class are the same type or not. Basically, this operator determines which certain class the object belongs to . It is used in object-oriented programming.

### Example :-

```
<?php
//class declaration
   class Developer
```

```
    {}
   class Programmer
    {}
//creating an object of type Developer
   $charu = new Developer();

//testing the type of object
    if( $charu instanceof Developer)  )
   {
       echo "Charu is a developer.";
   }
   else
   {
       echo "Charu is a programmer.";
   }
      echo "</br>";
   var_dump($charu instanceof Developer);        //It will return true.
   var_dump($charu instanceof Programmer);       //It will return false.
  ?>
```

### Output :-

```
Charu                          is                          a

bool(true) bool(false)
```

# Execution Operators

PHP has an execution operator **backticks (``)**. PHP executes the content of backticks as a shell command. Execution operator and **shell_exec()** give the same result.

| Operator | Name | Example | Explanation |
|---|---|---|---|

| | | | |
|---|---|---|---|
| `` | backticks | echo `dir`; | Execute the shell command and return the result. Here, it will show the directories available in current folder. |

***Note: Note that backticks (``) are not single-quotes.***

## Error Control Operators

PHP has one error control operator, i.e., **at (@) symbol**. Whenever it is used with an expression, any error message will be ignored that might be generated by that expression.

| Operator | Name | Example | Explanation |
|---|---|---|---|
| @ | at | @file ('non_existent_file') | Intentional file error |

## PHP Operators Precedence

Let's see the precedence of PHP operators with associativity.

| Operators | Additional Information | Associativity |
|---|---|---|
| clone new | clone and new | non-associative |
| [ | array() | left |
| ** | arithmetic | right |

| | | |
|---|---|---|
| ++ -- ~ (int) (float) (string) (array) (object) (bool) @ | increment/decrement and types | right |
| instanceof | types | non-associative |
| ! | logical (negation) | right |
| * / % | arithmetic | left |
| + - . | arithmetic and string concatenation | left |
| <<>> | bitwise (shift) | left |
| <<= >>= | comparison | non-associative |
| == != === !== <> | comparison | non-associative |
| & | bitwise AND | left |
| ^ | bitwise XOR | left |
| \| | bitwise OR | left |
| && | logical AND | left |
| \|\| | logical OR | left |
| ?: | ternary | left |

| | | |
|---|---|---|
| = += -= *= **= /= .= %= &= \|= ^= <<= >>= => | assignment | right |
| and | logical | left |
| xor | logical | left |
| or | logical | left |
| , | many uses (comma) | left |

# PHP Control Statement

All the conditional statements and loop statements are collectively termed as Control

Statements.

## Conditional Statements :-

Conditional statements are used in a code to perform an action only if a

particular condition is satisfied.

PHP provides four types of conditional statements. These are:

***If :-*** If statement is used to perform an action if a single condition is true.

***..Else :-*** If else statement is used to perform an action if a single condition is true and to perform another action if that condition is false.

***..elseif....else :-*** If elseif else statement is used to perform different actions for different conditions.

***Switch :-*** Switch statement is used to select one of the action to perform from multiple actions, relative to the true condition.

## PHP If Statement

PHP if statement allows conditional execution of code. It is executed if condition is true. If statement is used to executes the block of code exist inside the if statement only if the Specified condition is true.

**Syntax :-**

```
if(condition){
 //code to be executed
 }
```

**Example :-**

```
<?php
 $num=12;
    if($num<100){
        echo "$num is less than 100";
   }
 ?>
```

**Output :-**

```
12 is less than 100
```

## PHP If-else Statement

PHP if-else statement is executed whether condition is true or false. If-else statement is slightly different from if statement. It executes one block of code if the specified condition is **true** and another block of code if the condition is **false**.

**Syntax :-**

```
if(condition){

 //code to be executed if true
} else {
 //code to be executed if false
}
```

**Example** :-

```
<?php
$num=12;
  if($num%2==0){
     echo "$num is even number";
  } else {
     echo "$num is odd number";
  }
 ?>
```

**Output :-**

```
12 is even number
```

## PHP If-else-if Statement

The PHP if-else-if is a special statement used to combine multiple if?.else statements.

So, We can check multiple conditions using this statement.

**Syntax :-**

```
if(condition1){
//code to be executed if condition1 is true
 elseif (condition2){
//code to be executed if condition2 is true
elseif (condition3){
//code to be executed if condition3 is true
else{
//code to be executed if all given conditions are false
```

**Example :-**

```
?php
$marks=69;
  if ($marks<33){
    echo "fail";
 }
  else if ($marks>=34 && $marks<50) {
    echo "D grade";
 }
  else if ($marks>=50 && $marks<65) {
   echo "C grade";
 }
  else if ($marks>=65 && $marks<80) {
    echo "B grade";
 }
  else if ($marks>=80 && $marks<90) {
    echo "A grade";
 }
```

```
    else if ($marks>=90 && $marks<100) {
      echo "A+ grade";
   }
    else {
      echo "Invalid input";
   }
  ?>
```

**Output :-**

```
B Grade
```

# PHP Switch Statement

PHP switch statement is used to execute one statement from multiple conditions.

It works like PHP if-else-if statement.

## Syntax :-

```
switch(expression){
  case value1:
 //code to be executed
  break;
  case value2:
//code to be executed
  break;
  ......
  default:
 code to be executed if all cases are not matched;
}
```

**Example :-**

```
<?php
$num=20;
```

```php
    switch($num){
     case 10:
       echo("number is equals to 10");
     break;
     case 20:
       echo("number is equal to 20");
     break;
     case 30:
       echo("number is equal to 30");
     break;
     default:
       echo("number is not equal to 10, 20 or 30");
    }
?>
```

**Output :-**

```
number is equal to 20
```

# PHP  Loops Statement

**Loop Statements :-** Loop Statements are used to execute a block of code continuously as long as the condition of the loop is true, and stops only when the condition fails.
PHP provides three types of loop statements. These are  :

***For loop:***
For loop is used to execute a group of action only for a specified no.  of times.

***Foreach loop:***

Foreach loop is used to loop through each key/value pair in an array .

***While Loop:***
While loop is used to execute a group of action as long as the specified condition

is true.

***Do While Loop:***
Do While loop is used to execute a block of code at least once and then to

Repeat the actions as long as the specified condition is true.

# PHP For Loop

PHP for loop can be used to traverse set of code for the specified number of times.

It should be used if the number of iterations is known otherwise use while loop. This means for

loop is used when you already know how many times you want to execute a block of code.

It allows users to put all the loop related statements in one place. See in the syntax given below:

## Syntax :-

```
for(initialization; condition; increment/decrement){
  //code to be executed
}
```

## Parameters :-

**Initialization** - Initialize the loop counter value. The initial value of the for loop is done

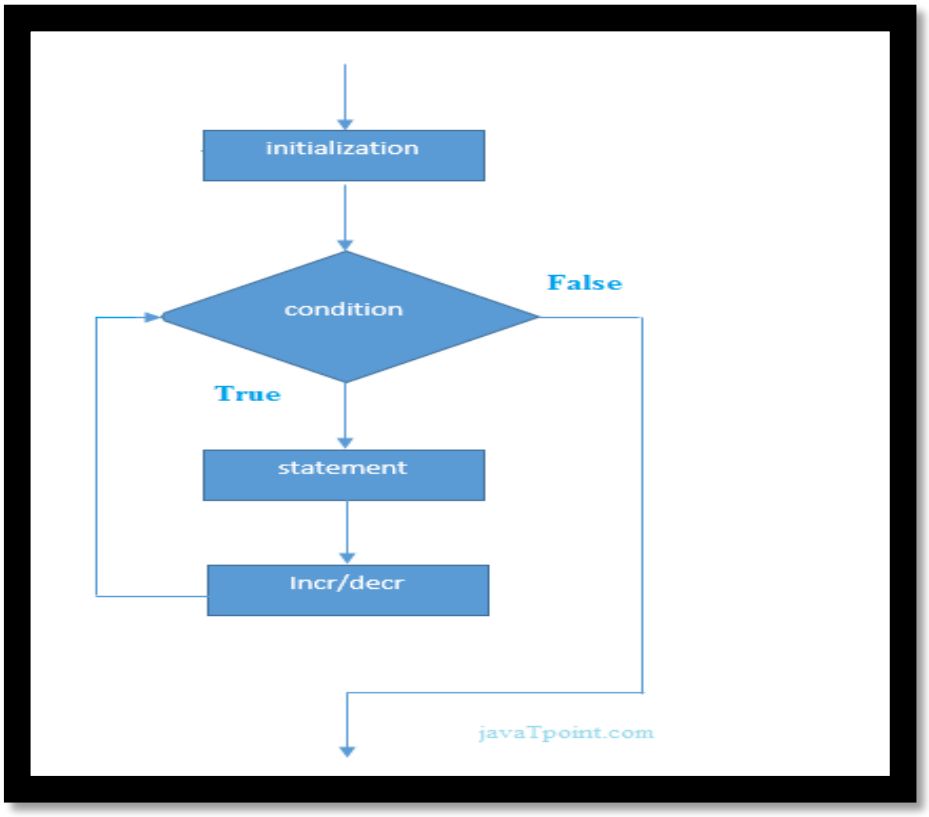
only once. This parameter is optional.

**Condition** - Evaluate each iteration value. The loop continuously executes until the conditio

is false. If TRUE, the loop execution continues, otherwise the execution of the loop ends.

**Increment/decrement** - It increments or decrements the value of the variable.

## Flowchart :-




## Example :-

```php
<?php
 for($n=1;$n<=10;$n++){
      echo "$n<br/>";
  }
 ?>
```

## Output :-

```
1
2
3
4
5
```

```
6
7
8
9
10
```

# PHP foreach loop

The foreach loop is used to traverse the array elements. It works only on array and object. It will issue an error if you try to use it with the variables of different datatype.

The foreach loop works on elements basis rather than index. It provides an easiest way to iterate the elements of an array.

In foreach loop, we don't need to increment the value.

## Syntax :-

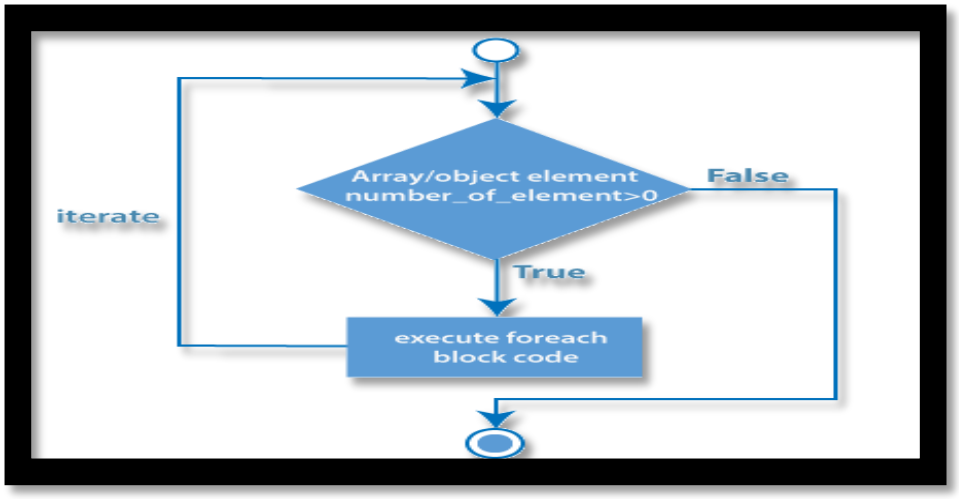
```
foreach ($array as $value) {
    //code to be executed
}
```

**There is one more syntax of foreach loop.**

## Syntax :-

```
foreach ($array as $key => $element) {
    //code to be executed
}
```

## Flowchart :-




## Example 1 : -

PHP program to print array elements using foreach loop.

## Example :-

```
<?php
//declare array
$season = array ("Summer", "Winter", "Autumn", "Rainy");
 //access array elements using foreach loop
foreach ($season as $element) {
    echo "$element";
    echo "</br>";
}
?>
```

## Output :-

```
Summer
Winter
Autumn
Rainy
```

# PHP While Loop

PHP while loop can be used to traverse set of code like for loop. The while loop executes a block of code repeatedly until the condition is FALSE. Once the condition gets FALSE, it exits from the body of loop.

It should be used if the number of iterations is not known.

The while loop is also called an **Entry control loop** because the condition is checked before entering the loop body. This means that first the condition is checked. If the condition is true, the block of code will be executed.

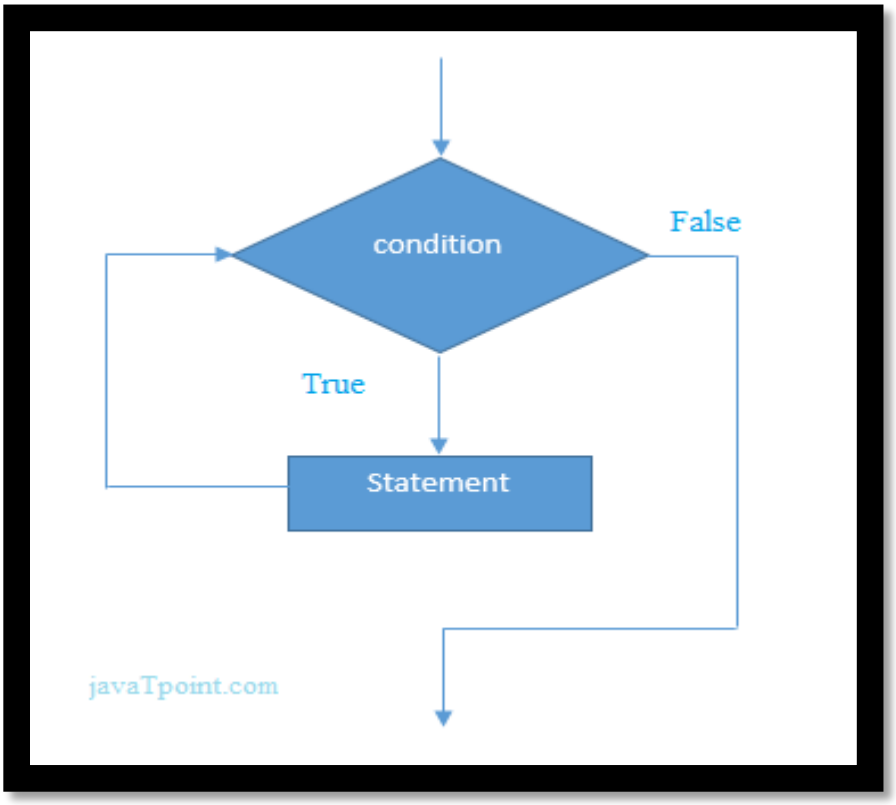
## Syntax :-

```
while(condition){
  //code to be executed
  }
```

## Alternative Syntax :-

```
while(condition):
//code to be executed
endwhile;
```

## PHP While Loop Flowchart :-




## Example :-

```
<?php
$n=1;
while($n<=10){
echo "$n<br/>";
$n++;
}
?>
```

## Output :-

```
1
2
3
4
5
6
```

```
7
8
9
10
```

# PHP do-while loop

PHP do-while loop can be used to traverse set of code like php while loop. The PHP do-while loop is guaranteed to run at least once.

The PHP do-while loop is used to execute a set of code of the program several times. I f you have to execute the loop at least once and the number of iterations is not even fixed, it is recommended to use the **do-while** loop.

It executes the code at least one time always because the condition is checked after exe-cuting the code.

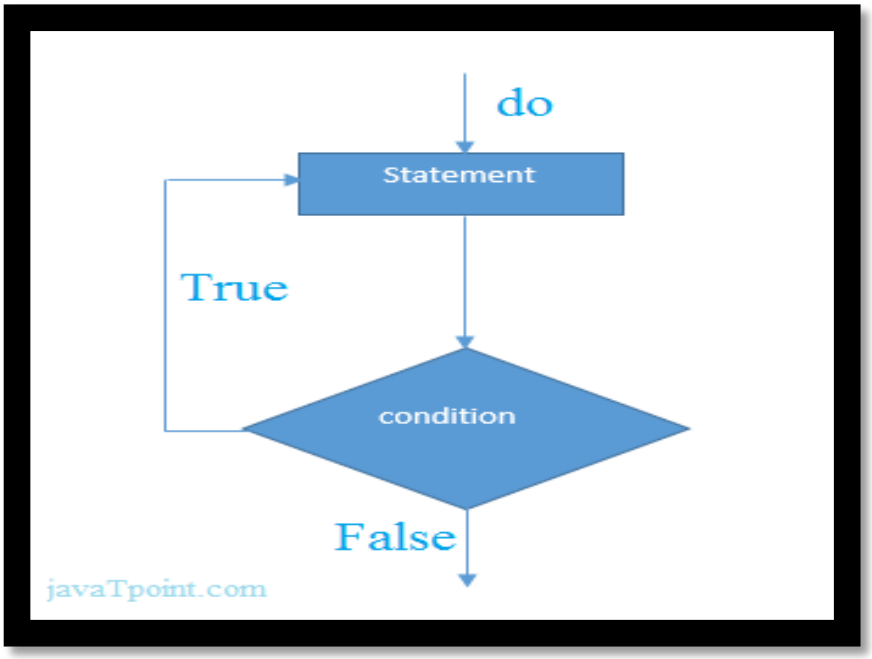
The do-while loop is very much similar to the while loop except the condition check.

The main difference between both loops is that while loop checks the condition at the beginning, whereas do-while loop checks the condition at the end of the loop.

## Syntax :-

```
do{
//code to be executed
}while(condition);
```

## Flowchart :-




## Example :-

```
<?php
$n=1;
do{
echo "$n<br/>";
$n++;
}while($n<=10);
?>
```

## Output :-

```
1
2
3
4
5
6
7
8
9
```

# Difference between while and do-while loop

| while Loop | do-while loop |
|---|---|
| The while loop is also named as **entry control loop**. | The do-while loop is also named as **exit control loop**. |
| The body of the loop does not execute if the condition is false. | The body of the loop executes at least once, even if the condition is false. |
| Condition checks first, and then block of statements executes. | Block of statements executes first and then condition checks. |
| This loop does not use a semicolon to terminate the loop. | Do-while loop use semicolon to terminate the loop. |

## Arrays in PHP

Arrays in PHP is a type of data structure that allows us to store multiple elements of similar data type under a single variable

The arrays are helpful to create a list of elements of similar types, which can be accessed using their index or key.

An array is created using an **array()** function in PHP.

There are basically three types of arrays in PHP:

- **Indexed or Numeric Arrays :** An array with a numeric index where values are stored linearly.
- **Associative Arrays :** An array with a string index where instead of linear storage, each value can be assigned a specific key.
- **Multidimensional Arrays :** An array which contains single or multiple array within it and can be accessed via multiple indices.
  - **Indexed or Numeric Arrays :-** PHP indexed array is an array which is represented by an index number by default. All elements of array are represented by an index number which starts from 0.

  PHP indexed array can store numbers, strings or any object. PHP indexed array is also known as numeric array.

  **There are two ways to define indexed array :-**

**1st way :-**
```
$season=array("summer","winter","spring","autumn");
```

**2nd way :-**
```
$season[0]="summer";
$season[1]="winter";
$season[2]="spring";
$season[3]="autumn";
```

## Example :-

*File: array1.php*
```
<?php
$season=array("summer","winter","spring","autumn");
echo "Season are: $season[0], $season[1], $season[2] and $season[3]";
?>
```

## Output :-

```
Season are: summer, winter, spring and autumn
```

# Associative Arrays

These types of arrays are similar to the indexed arrays but instead of linear storage, every value can be assigned with a user-defined key of string type.

PHP allows you to associate name/label with each array elements in PHP using => symbol. Such way, you can easily remember the element because each element is represented by label than an incremented number.

**There are two ways to define associative array :-**

**1st way :-** `$salary=array("Sonoo"=>"350000","John"=>"450000","Kartik"=>"200000");`

**2nd way :-**
```
$salary["Sonoo"]="350000";
$salary["John"]="45000";
$salary["Kartik"]="200000";
```

## Example :-

```
<?php
$salary=array("Sonoo"=>"350000","John"=>"450000","Kartik"=>"200000");
echo "Sonoo salary: ".$salary["Sonoo"]."<br/>";
echo "John salary: ".$salary["John"]."<br/>";
echo "Kartik salary: ".$salary["Kartik"]."<br/>";
?>
```

## Output :-

```
Sonoo salary: 350000
John salary: 450000
Kartik salary: 200000
```

# Multidimensional Arrays

PHP multidimensional array is also known as array of arrays. It allows you to store tabular data in an array. PHP multidimensional array can be represented in the form of matrix which is represented by row * column .

## Example :-

```
<?php
$emp = array
  (
  array(1,"sonoo",400000),
  array(2,"john",500000),
  array(3,"rahul",300000)
  );

  for ($row = 0; $row < 3; $row++) {
  for ($col = 0; $col < 3; $col++) {
        echo $emp[$row][$col]."  ";
     }
         echo "<br/>";
   }
   ?>
```

## Output :-

```
1 sonoo 400000
2 john 500000
3 rahul 300000
```

# 1) PHP array() function

PHP array() function creates and returns an array. It allows you to create indexed, associative

and multidimensional arrays.

## Syntax :- array array ([ mixed $... ] )

## Example :-

```
<?php
$season=array("summer","winter","spring","autumn");
echo "Season are: $season[0], $season[1], $season[2] and $season[3]";
?>
```

## Output :-

```
Season are: summer, winter, spring and autumn
```

# 2) PHP array_change_key_case() function

PHP array_change_key_case() function changes the case of all key of an array.

Note: It changes case of key only .

## Syntax :- array array_change_key_case ( array $array [, int $case = CASE_LOWER ] )

## Example :-

```
<?php
```

```
$salary=array("Sonoo"=>"550000","Vimal"=>"250000","Ratan"=>"200000");
print_r(array_change_key_case($salary,CASE_UPPER));
?>
```

### Output :-

```
Array ( [SONOO] => 550000 [VIMAL] => 250000 [RATAN] => 200000 )
```

### Example :-

```
<?php
$salary=array("Sonoo"=>"550000","Vimal"=>"250000","Ratan"=>"200000");
print_r(array_change_key_case($salary,CASE_LOWER));
?>
```

### Output :-

```
Array ( [sonoo] => 550000 [vimal] => 250000 [ratan] => 200000 )
```

## 3) PHP array_chunk() function

PHP array_chunk() function splits array into chunks. By using array_chunk() method, you can divide array into many parts.

**Syntax :-** `array array_chunk ( array $array , int $size [, bool $preserve_keys = false ] )`

### Example :-

```
<?php
$salary=array("Sonoo"=>"550000","Vimal"=>"250000","Ratan"=>"200000");
print_r(array_chunk($salary,2));
?>
```

### Output :-

```
Array (
[0] => Array ( [0] => 550000 [1] => 250000 )
[1] => Array ( [0] => 200000 ) )
```

## 4) PHP count() function

PHP count() function counts all elements in an array.

**Syntax :-** int count ( mixed $array_or_countable [, int $mode = COUNT_NORMAL ] )

### Example :-

```
<?php
$season=array("summer","winter","spring","autumn");
echo count($season);
?>
```

### Output :-

```
4
```

## 5) PHP sort() function

PHP sort() function sorts all the elements in an array.

**Syntax :-** bool sort ( **array** &$array [, int $sort_flags = SORT_REGULAR ] )

### Example :-

```
<?php
$season=array("summer","winter","spring","autumn");
sort($season);
```

```
foreach( $season as $s )
{
  echo "$s<br />";
}
?>
```

### Output :-

```
autumn
spring
summer
winter
```

## 6) PHP array_reverse() function

PHP array_reverse() function returns an array containing elements in reversed order.

### Syntax :- array array_reverse ( array $array [, bool $preserve_keys = false ] )

### Example :-

```
<?php
$season=array("summer","winter","spring","autumn");
$reverseseason=array_reverse($season);
foreach( $reverseseason as $s )
{
  echo "$s<br />";
}
?>
```

### Output :-

```
spring
winter
```

```
summer
```

## 7) PHP array_search() function

PHP array_search() function searches the specified value in an array. It returns key if search is successful.

**Syntax :-** mixed array_search ( mixed $needle , **array** $haystack [, bool $strict = false ] )

**Example :-**

```
<?php
$season=array("summer","winter","spring","autumn");
$key=array_search("spring",$season);
echo $key;
?>
```

**Output :-**

```
2
```

## 8) PHP array_intersect() function

PHP array_intersect() function returns the intersection of two array. In other words, it returns the matching elements of two array.

**Syntax :-** **array** array_intersect ( **array** $array1 , **array** $array2 [, **array** $... ] )

**Example :-**

```
<?php
```

```php
$name1=array("sonoo","john","vivek","smith");
$name2=array("umesh","sonoo","kartik","smith");
$name3=array_intersect($name1,$name2);
foreach( $name3 as $n )
{
  echo "$n<br />";
}
?>
```

## Output :-

```
sonoo
smith
```

# PHP | array_keys() Function

- **Read**
- **Discuss**

The array_keys() is a built-in function in PHP and is used to return either all the keys of and array or the subset of the keys.

**Syntax :-** *array* array_keys($input_array, $search_value, $strict)

**Return Value :-** The function returns an array containing either all of the keys or subset of keys the input array depending upon the parameters passed.

**Examples :-**

**Input :**
```
$input_array = ("one" => "shyam", 2 => "rishav",
                                    "three" => "gaurav")
```

**Output :-**
```
Array(
    [0] => one
```

```
            [1] => 2
            [2] => three  )
```

In the below program, we have passed a simple associative array to the function array_ keys(), to print all of its keys:

```
<?php

// PHP function to illustrate the use of array_keys()

function get_Key($array)      {

    $result = array_keys($array);

    return($result);

}

$array = array("one" => "shyam", 2 => "rishav", "three" => "gaurav");

print_r(get_Key($array));

?>
```

**Output :-**

```
Array  (
    [0] => one
    [1] => 2
    [2] => three   )
```

# PHP | array_pop() Function

- **Read**
- **Discuss**

# PHP | array_pop() Function

This inbuilt function of PHP is used to delete or pop out and return the last element from an array passed to it as parameter. It reduces the size of the array by one since last element is removed from the array.

**Syntax** :- `Array_pop($array)`

**Return Value :-** This function returns the last element of the array. If the array is empty or the input parameter is not an array, then NULL is returned.

**Note :-** This function resets the (reset()) the array pointer of the input array after use.

## Examples :-

**Input :** `$array = (1=>"ram", 2=>"krishna", 3=>"aakash");`

**Output :** `aakash`

**Input :** `$array = (24, 48, 95, 100, 120);`

**Output :** `120`

# PHP in_array() Function

- **Read**
- **Discuss**

In this article, we will see how to find the value in the array using the in_array() function in PHP, & will also understand its implementation through the examples.

The **in_array()** function is an inbuilt function in PHP that is used to check whether a given value exists in an array or not. It returns TRUE if the given value is found in the given array, and FALSE otherwise.

**Syntax** :- `bool in_array( $val, $array_name, $mode )`

**Return Value** :- The in_array() function returns a boolean value i.e, TRUE if the value $val is found in the array otherwise it returns FALSE.

We will understand the concept of the in_array() function in PHP, through the example.

**Example :-**

```
?php

$marks = array(100, 65, 70, 87);

  if (in_array("100", $marks))    {

    echo "found";      }

  else   {

    echo "not found";    }

 ?>
```

**Output :-** `found`

# PHP | array_push() Function

This inbuilt function of PHP is used to push new elements into an array. We can push one or more than one element into the array and these elements gets inserted to the

end of the array and because of the pushed elements into the array, the length of the array also gets incremented by the number of elements pushed into the array.

**Syntax** :- `array_push($array, $val1, $val2, $val3....)`

**Input :** `$array = (1=>"ram", 2=>"krishna", 3=>"aakash")`

`$val1 = "rohan", $val2 = "rajeeb", $val3 = "saniya"`

**Output :-**

```
Array  (
    [1] => ram
    [2] => krishna
    [3] => aakash
    [4] => rohan
    [5] => rajeeb
    [6] => Saniya  )
```

# PHP String

PHP string is a sequence of characters i.e., used to store and manipulate text. PHP supports only 256-character set and so that it does not offer native Unicode support.

**There are 4 ways to specify a string literal in PHP.**

1. single quoted
2. double quoted

## Single Quoted

We can create a string in PHP by enclosing the text in a single-quote. It is the easiest way to specify string in PHP.

## Double Quoted

In PHP, we can specify string through enclosing text within double quote also. But escape sequences and variables will be interpreted using double quote PHP strings.

### Example :-

```
<?php

$name = "Krishna";

echo "The name of the geek is $name \n";

echo 'The name of the geek is $name';

?>
```

**Output :-**

```
The name of the geek is Krishna
The name of the geek is $name
```

# PHP String Function Examples

## 1) PHP strtolower() function

The strtolower() function returns string in lowercase letter.

**Syntax :-** string strtolower ( string $string )

**Example :-**

```
<?php
$str="My name is KHAN";
$str=strtolower($str);
```

```
echo $str;
?>
```

**Output :-**

```
my name is khan
```

## 2) PHP strtoupper() function

The strtoupper() function returns string in uppercase letter.

**Syntax :-** string strtoupper ( string $string )

**Example :-**

```
<?php
 $str="My name is KHAN";

 $str=strtoupper($str);
 echo $str;
?>
```

**Output :-**

```
MY NAME IS KHAN
```

## 3) PHP ucfirst() function

The ucfirst() function returns string converting first character into uppercase. It doesn't change the case of other characters.

**Syntax :-** string ucfirst ( string $str )

**Example :-**

```
<?php
$str="my name is KHAN";
$str=ucfirst($str);
echo $str;
?>
```

**Output :-**

```
My name is KHAN
```

## 4) PHP lcfirst() function

The ucfirst() function returns string converting first character into uppercase. It doesn't change the case of other characters.

**Syntax :-** string ucfirst ( string $str )

**Example :-**

```
<?php

$str="my name is KHAN";
$str=ucfirst($str);
echo $str;
?>
```

**Output :-**

```
My name is KHAN
```

## 5) PHP ucwords() function

The ucwords() function returns string converting first character of each word into uppercase.

**Syntax :-** string ucwords ( string $str )

**Example :-**

```
<?php
$str="my name is Sonoo jaiswal";
$str=ucwords($str);
echo $str;
?>
```

**Output :-**

```
My Name Is Sonoo Jaiswal
```

## 6) PHP strrev() function

The strrev() function returns reversed string.

**Syntax :-** string strrev ( string $string )

**Example :-**

```
<?php
$str="my name is Sonoo jaiswal";
$str=strrev($str);
echo $str;
?>
```

**Output :-**

```
lawsiaj oonoS si eman ym
```

## 7) PHP strlen() function

The strlen() function returns length of the string.

**Syntax :-** int strlen ( string $string )

**Example :-**

```
<?php

  $str="my name is Sonoo jaiswal";
 $str=strlen($str);
echo $str;
 ?>
```

**Output :-**

```
24
```

**str_replace() function :-** This function takes three strings as arguments. The third argument is the original string and the first argument is replaced by the second one.

In other words, we can say that it replaces all occurrences of the first argument in the original string with the second argument.

**Example :-**

```
<?php

echo str_replace("Geeks", "World", "Hello GeeksforGeeks!"), "\n";
```

```
echo str_replace("for", "World", "Hello GeeksforGeeks!"), "\n";

?>
```

**Output :-**

```
Hello WorldforWorld!
   Hello GeeksWorldGeeks!
```

In the first example, we can see that all occurrences of the word “Geeks” are replaced by “World” in “Hello GeeksforGeeks!”.

1. **strpos() function :-** This function takes two string arguments and if the second string is present in the first one, it will return the starting position of the string , otherwise returns FALSE.

## Example:

```
<?php

echo strpos("Hello GeeksforGeeks!", "Geeks"), "\n";

echo strpos("Hello GeeksforGeeks!", "for"), "\n";

var_dump(strpos("Hello GeeksforGeeks!", "Peek"));

 ?>
```

**Output :-**

```
6
11

bool(false)
```

We can see in the above program, in the third example the string “Peek” is not present in the first string, hence this function returns a boolean value false indicating that string is not present.

1. **trim() function :-** This function allows us to remove whitespaces or strings from both sides of a string.

**Example :-**

```
<?php

echo trim("Hello World!", "Hed!");

?>
```

**Output :-** `llo Worl`

**6. explode() function :-** This function converts a string into an array.

**Example :-**

```
<?php

$input  = "Welcome to geeksforgeeks";

print_r(explode(" ",$input));

?>
```

**Output :-** `Array ( [0] => Welcome [1] => to [2] => geeksforgeeks )`

**strwordcount() function :-** This function counts total words in a string.

**Example :-**

```
<?php

$input  = "Welcome to GeeksforGeeks";

echo str_word_count($input);

?>
```

**Output :-** `3`

**substr() function :-** This function gives the substring of a given string from a given index.

**Example :-**

```
<?php

$input  = "Welcome to geeksforgeeks";

echo(substr($input,3));

?>
```

**Output :-** `come to geeksforgeeks`

| Function | Description | Example | Output |
|---|---|---|---|
| strtolower | Used to convert all string characters to lower case letters | echo strtolower( ‘Benjamin’); | outputs benjamin |
| strtoupper | Used to convert all string characters to upper case letters | echo strtoupper(‘george w bush’); | outputs GEORGE W BUSH |

| | | | |
|---|---|---|---|
| strlen | The string length function is used to count the number of character in a string. Spaces in between characters are also counted | echo strlen(‘united states of america’); | 24 |
| explode | Used to convert strings into an array variable | $settings = explode(‘;’, “host=localhost; db=sales; uid=root; pwd=demo”); print_r($settings); | Array ( [0] => host=localhost [1] => db=sales [2] => uid=root [3] => pwd=demo ) |
| substr | Used to return part of the string. It accepts three (3) basic parameters. The first one is the string to be shortened, the second parameter is the position of the starting point, and the third parameter is the number of characters to be returned. | $my_var = ‘This is a really long sentence that I wish to cut short’;echo substr($my_var,0, 12).’…’; | This is a re… |
| str_replace | Used to locate and replace specified string values in a given string. The function accepts three arguments. The first argument is the text to be replaced, the second | echo str_replace (‘the’, ‘that’, ‘the laptop is very expensive’); | that laptop is very expensive |

| | | | |
|---|---|---|---|
| | argument is the replacement text and the third argument is the text that is analyzed. | | |
| strpos | Used to locate the and return the position of a character(s) within a string. This function accepts two arguments | echo strpos('PHP Programing','Pro'); | 4 |
| sha1 | Used to calculate the SHA-1 hash of a string value | echo sha1('password'); | 5baa61e4c 9b93f3f0 682250b6cf8331b 7ee68fd8 |
| md5 | Used to calculate the md5 hash of a string value | echo md5('password'); | 9f961034ee 4de758 baf4de09ceeb1a75 |
| str_word_count | Used to count the number of words in a string. | echo str_word_count ('This is a really long sentence that I wish to cut short'); | 12 |
| ucfirst | Make the first character of a string value upper case | echo ucfirst('respect'); | Outputs Respect |
| lcfirst | Make the first character of a string value lower case | echo lcfirst('RESPECT'); | Outputs rESPECT |

For

# PHP Functions

PHP function is a piece of code that can be reused many times. It can take input as

argument list and return value. There are thousands of built-in functions in PHP.

## Advantage of PHP Functions

**Code Reusability** :- PHP functions are defined only once and can be invoked many times,

like in other programming languages.

**Less Code** :- It saves a lot of code because you don't need to write the logic many times.

By the use of function, you can write the logic only once and reuse it.

**Easy to understand** :- PHP functions separate the programming logic. So it is easier to

understand the flow of the application bcoz every logic is divided in the form of functions.

## PHP User-defined Functions

We can declare and call user-defined functions easily. Let's see the syntax to declare

user-defined functions.

**Syntax :-**

```
function functionname(){
    //code to be executed
}
```

**Note: Function name must be start with letter and underscore only like other labels in PHP. It can't**

**be start with numbers or special symbols.**

## PHP Functions Example

```
<?php

function sayHello(){
echo "Hello PHP Function";
}
sayHello();//calling function
?>
```

### Output :-

```
Hello PHP Function
```

## PHP Function Arguments

We can pass the information in PHP function through arguments which is separated by

Comma ( ,) .

PHP supports **Call by Value** (default), **Call by Reference**, **Default argument values** and

**Variable-length argument list**.

### Parameter passing to Functions :-

PHP allows us two ways in which an argument can be passed into a function :-

- **Pass by Value :-** On passing arguments using pass by value, the value of the argument gets changed within a function, but the original value outside the function remains unchanged. That means a duplicate of the original value is passed as an argument.
- **Pass by Reference :-** On passing arguments as pass by reference, the original Value is passed. Therefore, the original value gets altered. In pass by reference we Actually pass the address of the value, where it is stored using ampersand sign(&).

### Example :-

```
<?php
function sayHello($name){
echo "Hello $name<br/>";
}
sayHello("Sonoo");
sayHello("Vimal");
sayHello("John");
?>
```

### Output :-

```
Hello Sonoo
Hello Vimal
Hello John
```

## PHP Call By Reference

Value passed to the function doesn't modify the actual value by default (call by value).

But we can do so by passing value as a reference.

By default, value passed to the function is call by value. To pass value as a reference, you need to use ampersand (&) symbol before the argument name.

Let's see a simple example of call by reference in PHP.

### Example :-

```
<?php

function adder(&$str2)
{
   $str2 .= 'Call By Reference';
}
$str = 'Hello ';
adder($str);
echo $str;
?>
```

### Output :-

```
Hello Call By Reference
```

## PHP Function: Default Argument Value

We can specify a default argument value in function. While calling PHP function if you don't specify any argument, it will take the default argument. Let's see a simple example of using default argument value in PHP function.

### Example :-

```
<?php

function sayHello($name="Sonoo"){
echo "Hello $name<br/>";
}
sayHello("Rajesh");
sayHello();//passing no value
sayHello("John");
?>
```

### Output :-

```
Hello Rajesh
Hello Sonoo
Hello John
```

## PHP Function: Returning Value

Let's see an example of PHP function that returns value.

### Example :-

```
<?php

function cube($n){
return $n*$n*$n;
}
echo "Cube of 3 is: ".cube(3);
?>
```

**Output :-**

```
Cube of 3 is: 27
```

# PHP Parameterized Function

PHP Parameterized functions are the functions with parameters. You can pass any number of parameters inside a function. These passed parameters act as variables inside your function.

They are specified inside the parentheses, after the function name.

The output depends upon the dynamic values passed as the parameters into the function.

# PHP Parameterized Example 1

**Addition and Subtraction**

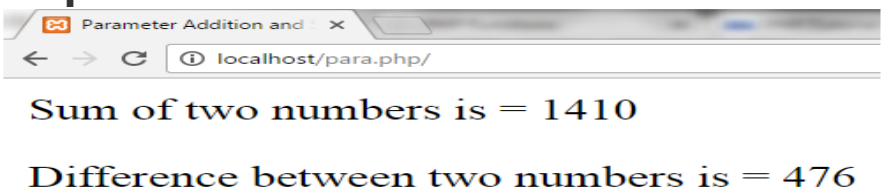
In this eg , we have passed two parameters **$x** and **$y** inside two functions **add()** and **sub()**.

```
<!DOCTYPE html>
<html>
<head>
<title>Parameter Addition and Subtraction Example</title>
</head>
<body>
<?php
	//Adding two numbers
	 function add($x, $y) {
		$sum = $x + $y;
		echo "Sum of two numbers is = $sum <br><br>";
```

```
        }
        add(467, 943);

        //Subtracting two numbers
        function sub($x, $y) {
       $diff = $x - $y;
           echo "Difference between two numbers is = $diff";
    }
        sub(943, 467);
     ?>
  </body>
  </html>
```

**Output :-**




# PHP Session

session refers to a frame of communication between two medium. A PHP session is used to store data on a server rather than the computer of the user. Session identifiers
or SID is a unique number which is used to identify every user in a session based environment. The SID is used to link the user with his information on the server like posts, emails etc.

When you work with an application, you open it, do some changes, and then you close it. This is much like a Session. The computer knows who you are. It knows when you start the application and when you end. But on the internet there is one problem: the web server

does not know who you are or what you do, because the HTTP address doesn't maintain state.

By default, session variables last until the user closes the browser.

So; Session variables hold information about one single user, and are available to all

pages in one application.

# Start a PHP Session

A session is started with the `session_start()` function.

Session variables are set with the PHP global variable: $_SESSION.

It returns existing session if session is created already. If session is not available, it creates
and returns new session.

## PHP $_SESSION

PHP $_SESSION is an associative array that contains all session variables. It is used to set

and get session variable values.

**Example :-** Store information

```
$_SESSION["user"] = "Sachin";
```

**Example :-** Get information

```
echo $_SESSION["user"];
```

# PHP Session Example

***File: session1.php***

```
<?php
session_start();
?>
<html>
<body>
<?php
$_SESSION["user"] = "Sachin";
echo "Session information are set successfully.<br/>";
?>
    <a href="session2.php">Visit next page</a>
    </body>
    </html>
```

***File: session2.php***

```
<?php
session_start();
?>
<html>
<body>
<?php
echo "User is: ".$_SESSION["user"];
?>
</body>
</html>
```

# PHP Destroying Session

PHP session_destroy() function is used to destroy all session variables completely.

***File: session3.php***

```
<?php
session_start();
session_destroy();
?>
```

# PHP Cookies

A **cookie** in PHP is a small file with a maximum size of 4KB that the web server stores
on the client computer. They are typically used to keep track of information such as a username that the site can retrieve to personalize the page when the user visits the website next time

Each time when client sends request to the server, cookie is embedded with request. Such

Way , cookie can be received at the server side.

***Note: PHP Cookie must be used before <html> tag***.

## PHP setcookie() function

PHP setcookie() function is used to set cookie with HTTP response. Once cookie is set,

you can access it by $_COOKIE superglobal variable.

**Syntax :-** `setcookie(name, value, expire, path, domain, security);`

**Parameters :** The setcookie() function requires six arguments in general which are :-

- **Name :** It is used to set the name of the cookie.
- **Value :** It is used to set the value of the cookie.
- **Expire :** It is used to set the expiry timestamp of the cookie after which the cookie can't be accessed.
- **Path :** It is used to specify the path on the server for which the cookie will be available.
- **Domain :** It is used to specify the domain for which the cookie is available.
- **Security :** It is used to indicate that the cookie should be sent only if a secure HTTPS connection exists.

### Example :-

```
setcookie("CookieName", "CookieValue");/* defining name and value only*/
setcookie("CookieName", "CookieValue", time()+1*60*60);//using expiry in 1 hour(1*60*60 seconds or 3600 seconds)
setcookie("CookieName", "CookieValue", time()+1*60*60, "/mypath/", "mydomain.com", 1);
```

## PHP $_COOKIE

PHP $_COOKIE superglobal variable is used to get cookie.

### Example :-

```
$value=$_COOKIE["CookieName"];//returns cookie value
```

## PHP Cookie Example

***File: cookie1.php***

```
<?php
setcookie("user", "Sonoo");
?>
```

```
<html>
<body>
<?php
if(!isset($_COOKIE["user"])) {
    echo "Sorry, cookie is not found!";
} else {
      echo "<br/>Cookie Value: " . $_COOKIE["user"];
}
?>
</body>
</html>
```

### Output :-

```
Sorry, cookie is not found!
```

Firstly cookie is not set. But, if you *refresh* the page, you will see cookie is set now.

### Output :-

```
Cookie Value: Sonoo
```

## PHP Delete Cookie

If you set the expiration date in past, cookie will be deleted.

***File: cookie1.php***

```
<?php
setcookie ("CookieName", "", time() - 3600);// set the expiration date to one hour ago
?>
```