**PHP – Working Principles of PHP**

* PHP is a server scripting language, and a powerful tool for making dynamic and interactive Web pages.
* PHP is a widely-used, free, and efficient alternative to competitors such as Microsoft's ASP.
* The first version of what came to be known as PHP was created in 1995 by a man named Rasmus Lerdorf.
* The name "Personal Home Page/Forms Interpreter" was later shortened to PHP/FI and eventually renamed to represent "PHP: Hypertext Preprocessor".

**Example**

<!DOCTYPE html>  
<html>  
<body>  
  
<?php  
echo "My first PHP script!";  
?>  
  
</body>  
</html>

**O/P:**

My first PHP script!

## 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

**PHP is an amazing and popular language!**

It is powerful enough to be at the core of the biggest blogging system on the web (WordPress)!  
It is deep enough to run the largest social network (Facebook)!  
It is also easy enough to be a beginner's first server side language!

### What is a PHP File?

* PHP files can contain text, HTML, CSS, JavaScript, and PHP code
* PHP code are 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 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
* With PHP you are not limited to output HTML. You can output images, PDF files, and even Flash movies. You can also output any text, such as XHTML and XML.

### Why PHP?

* PHP runs on various platforms (Windows, Linux, Unix, Mac OS X, etc.)
* PHP is compatible with almost all servers used today (Apache, IIS, etc.)
* PHP supports a wide range of databases
* PHP is free. Download it from the official PHP resource: [www.php.net](http://www.php.net/)
* PHP is easy to learn and runs efficiently on the server side
* A PHP script is executed on the server, and the plain HTML result is sent back to the browser.

### Basic PHP Syntax

A PHP script can be placed anywhere in the document.

A PHP script starts with **<?php** and ends with **?>**:

|  |
| --- |
| <?php // PHP code goes here ?> |

The default file extension for PHP files is ".php".

A PHP file normally contains HTML tags, and some PHP scripting code.

Below, we have an example of a simple PHP file, with a PHP script that uses a built-in PHP function "echo" to output the text "Hello World!" on a web page:

<!DOCTYPE html>  
<html>  
<body>  
  
<h1>My first PHP page</h1>  
  
<?php  
echo "Hello World!";  
?>  
  
</body>  
</html>

Output:

# My first PHP page

Hello World!

**Note:** PHP statements end with a semicolon (;).

### Comments in PHP

A comment in PHP code is a line that is not read/executed as 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 what you are doing
* 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

PHP supports several ways of commenting:

Example

<!DOCTYPE html>  
<html>  
<body>  
<?php  
// This is a single-line comment  
  
# This is also a single-line comment  
  
/\*  
This is a multiple-lines comment block  
that spans over multiple  
lines  
\*/  
  
// You can also use comments to leave out parts of a code line  
$x = 5 /\* + 15 \*/ + 5;  
echo $x;  
?>  
  
</body>  
</html>

**O/P:**

10

PHP Case Sensitivity

In PHP, NO keywords (e.g. if, else, while, echo, etc.), classes, functions, and user-defined functions are case-sensitive.

In the example below, all three echo statements below are legal (and equal):

Example

<!DOCTYPE html>  
<html>  
<body>  
  
<?php  
ECHO "Hello World!<br>";  
echo "Hello World!<br>";  
EcHo "Hello World!<br>";  
?>  
  
</body>  
</html>

**O/P:** Hello World!  
 Hello World!  
 Hello World!

However; all variable names are case-sensitive.

In the example below, only the first statement will display the value of the $color variable (this is because $color, $COLOR, and $coLOR are treated as three different variables):

Example

<!DOCTYPE html>  
<html>  
<body>  
  
<?php  
$color = "red";  
echo "My car is " . $color . "<br>";  
echo "My house is " . $COLOR . "<br>";  
echo "My boat is " . $coLOR . "<br>";  
?>  
  
</body>  
</html>

**O/P:**

My car is red  
My house is   
My boat is

### Variables

* Variables are "containers" for storing information.
* 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)

Remember that PHP variable names are case-sensitive!

### Creating (Declaring) PHP Variables

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

### Example

<!DOCTYPE html>  
<html>  
<body>  
  
<?php  
$txt = "Hello world!";  
$x = 5;  
$y = 10.5;  
  
echo $txt;  
echo "<br>";  
echo $x;  
echo "<br>";  
echo $y;  
?>  
  
</body>  
</html>

O/P:

Hello world!  
5  
10.5

### 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

### <!DOCTYPE html> <html> <body> <?php $txt = "Music"; echo "I love $txt!"; ?> </body> </html>

### O/P:

### I love Music

### 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

### <!DOCTYPE html> <html> <body> <?php $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>"; ?> </body> </html>

### O/P:

Variable x inside function is:

Variable x outside function is: 5

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

### Example

### <!DOCTYPE html> <html> <body> <?php 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>"; ?> </body> </html>

### O/P:

Variable x inside function is: 5

Variable x outside function is:

### You can have local variables with the same name in different functions, because local variables are only recognized by the function in which they are declared.

### PHP The global Keyword

The global keyword is used to access a global variable from within a function.

To do this, use the global keyword before the variables (inside the function):

### Example

### <!DOCTYPE html> <html> <body> <?php $x = 5; $y = 10; function myTest() {     global $x, $y;     $y = $x + $y; }  myTest();  // run function echo $y; // output the new value for variable $y ?> </body> </html>

### O/P:

### 15

PHP also stores all global variables in an array called $GLOBALS[*index*]. The index holds the name of the variable. This array is also accessible from within functions and can be used to update global variables directly.

The example above can be rewritten like this:

### Example

### <!DOCTYPE html> <html> <body> <?php $x = 5; $y = 10; function myTest() {     $GLOBALS['y'] = $GLOBALS['x'] + $GLOBALS['y']; }  myTest(); echo $y; ?> </body> </html>

### O/P:

### 15

### PHP The static Keyword

Normally, when a function is completed/executed, all of its variables are deleted. However, sometimes we want a local variable NOT to be deleted. We need it for a further job.

To do this, use the static keyword when you first declare the variable:

### Example

### <!DOCTYPE html> <html> <body> <?php function myTest() {     static $x = 0;     echo $x;     $x++; } myTest(); echo "<br>"; myTest(); echo "<br>"; myTest(); ?>  </body> </html>

### O/P:

### 0

### 1

### 2

Then, each time the function is called, that variable will still have the information it contained from the last time the function was called.

**Note:** The variable is still local to the function.

### PHP  echo and print Statements

### In PHP there are two basic ways to get output: echo and print.

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):

### Example

### <!DOCTYPE html> <html> <body> <?php 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."; ?>  </body> </html>

### O/P:

## PHP is Fun!

### Hello world! I'm about to learn PHP! This string was made with multiple parameters.

**Display Variables**

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

### Example

### <!DOCTYPE html> <html> <body> <?php $txt1 = "Learn PHP"; $txt2 = "google.com"; $x = 5; $y = 4; echo "<h2>" . $txt1 . "</h2>"; echo "Study PHP at " . $txt2 . "<br>"; echo $x + $y; ?> </body> </html>

### O/P:

## Learn PHP

### Study PHP at google.com 9

### 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

### <!DOCTYPE html> <html> <body> <?php print "<h2>PHP is Fun!</h2>"; print "Hello world!<br>"; print "I'm about to learn PHP!"; ?>  </body> </html>

### O/P:

### PHP is Fun!

### Hello world! I'm about to learn PHP!

**Display Variables**

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

### Example

### <!DOCTYPE html> <html> <body> <?php $txt1 = "Learn PHP"; $txt2 = "google.com"; $x = 5; $y = 4; print "<h2>" . $txt1 . "</h2>"; print "Study PHP at " . $txt2 . "<br>"; print $x + $y; ?> </body> </html>

### O/P:

# Learn PHP

### Study PHP at google.com 9

# 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

# 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

<?php   
$x = "Hello world!";  
$y = 'Hello world!';  
  
echo $x;  
echo "<br>";   
echo $y;  
?>

**o/p:**

Hello world!  
Hello world!

# 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 three formats: decimal (10-based), hexadecimal (16-based - prefixed with 0x) or octal (8-based - prefixed with 0)

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

### Example

### <?php  $x = 5985; var\_dump($x); ?>

**o/p:**

### int(5985)

# 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

### <?php  $x = 10.365; var\_dump($x); ?>

### o/p:

### float(10.365)

# PHP Boolean

A Boolean represents two possible states: TRUE or FALSE.

$x = true;  
$y = false;

Booleans are often used in conditional testing. You will learn more about conditional testing in a later chapter of this tutorial.

# 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

### <?php  $cars = array("Volvo","BMW","Toyota"); var\_dump($cars); ?>

### o/p:

### array(3) { [0]=> string(5) "Volvo" [1]=> string(3) "BMW" [2]=> string(6) "Toyota" }

# PHP Object

An object is a data type which stores data and information on how to process that data.

In PHP, an object must be explicitly declared.

First we must declare a class of object. For this, we use the class keyword. A class is a structure that can contain properties and methods:

### Example

### <?php class Car {     function Car() {         $this->model = "VW";     } } // create an object $herbie = new Car(); // show object properties echo $herbie->model; ?>

### o/p:

### VW

# 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

### <?php $x = "Hello world!"; $x = null; var\_dump($x); ?>

### o/p:

### NULL

# PHP Resource

The special resource type is not an actual data type. It is the storing of a reference to functions and resources external to PHP.

A common example of using the resource data type is a database call.

We will not talk about the resource type here, since it is an advanced topic.

# PHP 5 Constants

Constants are like variables except that once they are defined they cannot be changed or undefined.

# 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*)

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

The example below creates a constant with a **case-sensitive** name:

### <?php define("GREETING", "Welcome to google.com!"); echo GREETING; ?>

### o/p:

### Welcome to google.com!

### The example below creates a constant with a **case-insensitive** name:

### <?php define("GREETING", "Welcome to W3Schools.com!", true); echo greeting; ?>

### o/p:

### Welcome to W3Schools.com!

# Constants are Global

Constants are automatically global and can be used across the entire script.

The example below uses a constant inside a function, even if it is defined outside the function:

### Example

### <?php define("GREETING", "Welcome to W3Schools.com!"); function myTest() {     echo GREETING; }   myTest(); ?>

### o/p:

### Welcome to W3Schools.com!

# PHP Operators

Operators are used to perform operations on variables and values.

PHP divides the operators in the following groups:

* Arithmetic operators
* Assignment operators
* Comparison operators
* Increment/Decrement operators
* Logical operators
* String operators
* Array operators

# 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** | **SYNTAX** | **OPERATION** |
| + | Addition | $x + $y | Sum the operands |
| – | Subtraction | $x – $y | Differences the operands |
| \* | Multiplication | $x \* $y | Product of the operands |
| / | Division | $x / $y | Quotient of the operands |
| \*\* | Exponentiation | $x \*\* $y | $x raised to the power $y |
| % | Modulus | $x % $y | Remainder of the operands |

Example:

<?php

// variable 1

$x = 29;

// variable 2

$y = 4;

// 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

**Logical or Relational Operators**

These are basically used to operate with conditional statements and expressions. Conditional statements are based on conditions. Also, a condition can either be met or cannot be met so the result of a conditional statement can either be true or false. Here are the logical operators along with there syntax and operations, that PHP provides us:

|  |  |  |  |
| --- | --- | --- | --- |
| **OPERATOR** | **NAME** | **SYNTAX** | **OPERATION** |
| and | Logical AND | $x and $y | True if both the operands are true else false |
| or | Logical OR | $x or $y | True if either of the operand is true else false |
| xor | Logical XOR | $x xor $y | True if either of the operand is true and false if both are true |
| && | Logical AND | $x && $y | True if both the operands are true else false |
| || | Logical OR | $x || $y | True if either of the operand is true else false |
| ! | Logical NOT | !$x | True if $x is false |

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

**Comparison Operators**

These operators are used to compare two elements and outputs the result in boolean form. Here are the comparison operators along with there syntax and operations, that PHP provides us:

|  |  |  |  |
| --- | --- | --- | --- |
| **OPERATOR** | **NAME** | **SYNTAX** | **OPERATION** |
| == | Equal To | $x == $y | Returns True if both the operands are equal |
| != | Not Equal To | $x != $y | Returns True if both the operands are not equal |
| <> | Not Equal To | $x <> $y | Returns True if both the operands are unequal |
| === | Identical | $x === $y | Returns True if both the operands are equal and are of the same type |
| !== | Not Identical | $x == $y | Returns True if both the operands are unequal and are of different types |
| < | Less Than | $x < $y | Returns True if $x is less than $y |
| > | Greater Than | $x > $y | Returns True if $x is greater than $y |
| <= | Less Than or Equal To | $x <= $y | Returns True if $x is less than or equal to $y |
| >= | Greater Than or Equal To | $x >= $y | Returns True if $x is greater than or equal to $y |

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)

**Conditional or Ternary Operators**

These operators are used to compare two values and take either of the result simultaneously, depending on whether the outcome is TRUE or FALSE. These are also used as shorthand notation for if…else statement that we will read in the article on decision making.

**Syntax**:

$var = (condition)? value1 : value2;

Here, condition will either evaluate to true or false. If the condition evaluates to True, then value1 will be assigned to the variable $var otherwise value2 will be assigned to it.

|  |  |  |
| --- | --- | --- |
| **OPERATOR** | **NAME** | **OPERATION** |
| ?: | Ternary | If condition is true ? then $x : or else $y. This means that if condition is true then left result of the colon is accepted otherwise the result on right. |

Example:

<?php

$x = -12;

echo ($x > 0) ? 'The number is positive' : 'The number is negative';

?>

Output:

The number is negative

**Assignment Operators**

These operators are used to assign values to different variable, with or without mid-operations. Here are the assignment operators along with there syntax and operations, that PHP provides us:

|  |  |  |  |
| --- | --- | --- | --- |
| **OPERATOR** | **NAME** | **SYNTAX** | **OPERATION** |
| = | Assign | $x = $y | Operand on the left obtains the value of the operand on right |
| += | Add then Assign | $x += $y | Simple Addition same as $x = $x + $y |
| -= | Subtract then Assign | $x -= $y | Simple subtraction same as $x = $x – $y |
| \*= | Multiply then Assign | $x \*= $y | Simple product same as $x = $x \* $y |
| /= | Divide then Assign (quotient) | $x /= $y | Simple division same as $x = $x / $y |
| %= | Divide then Assign (remainder) | $x %= $y | Simple division same as $x = $x % $y |

Example:

<?php

// simple assign operator

$y = 75;

echo $y, "\n";

// add then assign operator

$y = 100;

$y += 200;

echo $y, "\n";

// subtract then assign operator

$y = 70;

$y -= 10;

echo $y, "\n";

// multiply then assign operator

$y = 30;

$y \*= 20;

echo $y, "\n";

// Divide then assign(quotient) operator

$y = 100;

$y /= 5;

echo $y, "\n";

// Divide then assign(remainder) operator

$y = 50;

$y %= 5;

echo $y;

?>

Output:

75

300

60

600

20

0

**Array Operators**

These operators are used in case of arrays. Here are the array operators along with there syntax and operations, that PHP provides us:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **OPERATOR** | | **NAME** | **SYNTAX** | **OPERATION** | |
| + | | Union | $x + $y | Union of both i.e., $x and $y | |
| == | | Equality | $x == $y | Returns true if both has same key-value pair | |
| != | | Inequality | $x != $y | Returs True if both are unequal | |
| === | | Identity | $x === $y | Returns True if both has same key-value pair in the same order and of same type | |
| !== | | Non-Identity | $x !== $y | Returns True if both are not identical to each other | |
| <> | | Inequality | $x <> $y | Returns True if both are unequal | |
| Example:  <?php    $x = array("k" => "Car", "l" => "Bike");  $y = array("a" => "Train", "b" => "Plane");    var\_dump($x + $y);  var\_dump($x == $y) + "\n";  var\_dump($x != $y) + "\n";  var\_dump($x <> $y) + "\n";  var\_dump($x === $y) + "\n";  var\_dump($x !== $y) + "\n";    ?> | | | |

Output:

array(4) {

["k"]=>

string(3) "Car"

["l"]=>

string(4) "Bike"

["a"]=>

string(5) "Train"

["b"]=>

string(5) "Plane"

}

bool(false)

bool(true)

bool(true)

bool(false)

bool(true)

**Increment/Decrement Operators**

These are called the unary operators as it work on single operands. These are used to increment or decrement values.

|  |  |  |  |
| --- | --- | --- | --- |
| **OPERATOR** | **NAME** | **SYNTAX** | **OPERATION** |
| ++ | Pre-Increment | ++$x | First increments $x by one, then return $x |
| — | Pre-Decrement | –$x | First decrements $x by one, then return $x |
| ++ | Post-Increment | $x++ | First returns $x, then increment it by one |
| — | Post-Decrement | $x– | First returns $x, then decrement it by one |

Example:

<?php

$x = 2;

echo ++$x, " First increments then prints \n";

echo $x, "\n";

$x = 2;

echo $x++, " First prints then increments \n";

echo $x, "\n";

$x = 2;

echo --$x, " First decrements then prints \n";

echo $x, "\n";

$x = 2;

echo $x--, " First prints then decrements \n";

echo $x;

?>

Output:

3 First increments then prints

3

2 First prints then increments

3

1 First decrements then prints

1

2 First prints then decrements

1

**String Operators**

These are implemented over strings.

|  |  |  |  |
| --- | --- | --- | --- |
| **OPERATOR** | **NAME** | **SYNTAX** | **OPERATION** |
| . | Concatenation | $x.$y | Concatenated $x and $y |
| .= | Concatenation and assignment | $x.=$y | First concatenates then assigns, same as $x = $x.$y |

Example:

<?php

$x = "Geeks";

$y = "for";

$z = "Geeks!!!";

echo $x . $y . $z, "\n";

$x .= $y . $z;

echo $x;

?>

Output:

GeeksforGeeks!!!

GeeksforGeeks!!!

Flow Control and Looping

Conditional statements are used to perform different actions based on different conditions.

# 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 statement - executes some code if a condition is true and another code if that condition is false
* if...elseif....else statement - executes different codes for more than two conditions
* switch 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*; }

The example below will output "Have a good day!" if the current time (HOUR) is less than 20:

### Example

### <?php $t = date("H"); if ($t < "20") {     echo "Have a good day!"; } ?>

### o/p:

### Have a good day!

# 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;* }

The example below will output "Have a good day!" if the current time is less than 20, and "Have a good night!" otherwise:

Example

### <?php $t = date("H"); if ($t < "20") {     echo "Have a good day!"; } else {     echo "Have a good night!"; } ?>

### o/p:

### Have a good day!

# 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 this condition is true;* } else {     *code to be executed if all conditions are false;* }

The example below will output "Have a good morning!" if the current time is less than 10, and "Have a good day!" if the current time is less than 20. Otherwise it will output "Have a good night!":

Example

### <?php $t = date("H"); if ($t < "10") {     echo "Have a good morning!"; } elseif ($t < "20") {     echo "Have a good day!"; } else {     echo "Have a good night!"; } ?>

### o/p:

The hour (of the server) is 00, and will give the following message:

### Have a good morning!

# PHP - The switch Statement

### The switch statement is used to perform different actions based on different conditions.

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

Syntax

switch (*n*) {  
    case *label1:*  
  *code to be executed if n=label1;*  
        break;  
    case *label2:*  
  *code to be executed if n=label2;*  
        break;  
    case *label3:*  
  *code to be executed if n=label3;*  
        break;  
    ...  
    default:  
  *code to be executed if n is different from all labels;*  
}

### This is how it works: First we have a single expression *n* (most often a variable), that is evaluated once. The value of the expression is then compared with the values for each case in the structure. If there is a match, the block of code associated with that case is executed. Use break to prevent the code from running into the next case automatically. The defaultstatement is used if no match is found.

Example

<?php  
$favcolor = "red";  
  
switch ($favcolor) {  
    case "red":  
        echo "Your favorite color is red!";  
        break;  
    case "blue":  
        echo "Your favorite color is blue!";  
        break;  
    case "green":  
        echo "Your favorite color is green!";  
        break;  
    default:  
        echo "Your favorite color is neither red, blue, nor green!";  
}  
?>

### o/p:

### Your favorite color is red!

# PHP Loops

Often when you write code, you want the same block of code to run over and over again in a row. Instead of adding several almost equal code-lines in a script, we can use loops to perform a task like this.

In PHP, we have the following looping statements:

* while - loops through a block of code as long as the specified condition is true
* do...while - loops through a block of code once, and then repeats the loop as long as the specified condition is true
* for - loops through a block of code a specified number of times
* foreach - loops through a block of code for each element in an array

# The PHP while Loop

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

### Syntax

### while (*condition is true*) { *code to be executed*; }

### The example below first sets a variable $x to 1 ($x = 1). Then, the while loop will continue to run as long as $x is less than, or equal to 5 ($x <= 5). $x will increase by 1 each time the loop runs ($x++):

Example

### <?php  $x = 1;  while($x <= 5) {     echo "The number is: $x <br>";     $x++; }  ?>

### o/p:

### The number is: 1  The number is: 2  The number is: 3  The number is: 4  The number is: 5

# The PHP do...while Loop

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

### Syntax

### do { *code to be executed;* } while (*condition is true*);

### The example below first sets a variable $x to 1 ($x = 1). Then, the do while loop will write some output, and then increment the variable $x with 1. Then the condition is checked (is $x less than, or equal to 5?), and the loop will continue to run as long as $x is less than, or equal to 5:

Example

### <?php  $x = 1;  do {     echo "The number is: $x <br>";     $x++; } while ($x <= 5); ?>

### o/p:

### The number is: 1  The number is: 2  The number is: 3  The number is: 4  The number is: 5

Notice that in a do while loop the condition is tested AFTER executing the statements within the loop. This means that the do while loop would execute its statements at least once, even if the condition is false the first time.

The example below sets the $x variable to 6, then it runs the loop, **and then the condition is checked**:

Example

### <?php  $x = 6; do {     echo "The number is: $x <br>";     $x++; } while ($x <= 5); ?>

### o/p:

### The number is: 6

# The PHP for Loop

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

### Syntax

### for (*init counter; test counter; increment counter*) {   *code to be executed;* }

Parameters:

* *init counter*: Initialize the loop counter value
* *test counter*: Evaluated for each loop iteration. If it evaluates to TRUE, the loop continues. If it evaluates to FALSE, the loop ends.
* *increment counter*: Increases the loop counter value

The example below displays the numbers from 0 to 10:

Example

<?php   
for ($x = 0; $x <= 10; $x++) {  
    echo "The number is: $x <br>";  
}   
?>

### o/p:

### The number is: 0  The number is: 1  The number is: 2  The number is: 3  The number is: 4  The number is: 5  The number is: 6  The number is: 7  The number is: 8  The number is: 9  The number is: 10

# The PHP foreach Loop

The foreach loop works only on arrays, and is used to loop through each key/value pair in an array.

### Syntax

### foreach ($*array*as$*value*) {     *code to be executed;* }

For every loop iteration, the value of the current array element is assigned to $value and the array pointer is moved by one, until it reaches the last array element.

The following example demonstrates a loop that will output the values of the given array ($colors):

Example

### <?php  $colors = array("red", "green", "blue", "yellow");  foreach ($colors as $value) {     echo "$value <br>"; } ?>

### o/p:

### red  green  blue  yellow