

Introduction to PHP

A PHP file may contain text, HTML tags and scripts. Scripts in a PHP file are executed on the server.

What is PHP?

PHP stands for **H**ypertext **P**reprocessor

PHP is a server-side scripting language, like ASP PHP scripts are executed on the server

PHP supports many databases (MySQL, Informix, Oracle, Sybase, Solid, PostgreSQL, Generic ODBC, etc.)

PHP is an open source software

(OSS) PHP is free to download and use

What is a PHP File?

PHP files may contain text, HTML tags and scripts

PHP files are returned to the browser as plain HTML

PHP files have a file extension of ".php", ".php3", or ".phtml"

What is MySQL?

MySQL is a small database server

MySQL is ideal for small and medium applications

MySQL supports standard SQL

MySQL compiles on a number of platforms

MySQL is free to download and use

PHP + MySQL

PHP combined with MySQL are cross-platform (means that you can develop in Windows and serve on a Unix platform)

Why PHP?

PHP runs on different platforms (Windows, Linux, Unix, etc.)

PHP is compatible with almost all servers used today (Apache, IIS, etc.) PHP is FREE to download from the official PHP resource: www.php.net PHP is easy to learn and runs efficiently on the server side

Where to Start?

Install an Apache server on a Windows or Linux machine

Install PHP on a Windows or Linux machine

Install MySQL on a Windows or Linux machine

PHP Syntax

You cannot view the PHP source code by selecting "View source" in the browser – you will only see the output from the PHP file, which is plain HTML. This is because the scripts are executed on the server before the result is sent back to the browser.

Basic PHP Syntax

A PHP file normally contains HTML tags, just like an HTML file, and some PHP scripting code. Below, we have an example of a simple PHP script which sends the text "Hello World" to the browser:

```
<html>
<body>
<?php echo "Hello World"; ?>
</body>
</html>
```

A PHP scripting block always starts with **<?php** and ends with **?>**. A PHP scripting block can be placed anywhere in the document.

Each code line in PHP must end with a semicolon. The semicolon is a separator and is used to distinguish one set of instructions from another.

There are two basic statements to output text with PHP: **echo** and **print**. In the example above we have used the echo statement to output the text "Hello World".

Variables in PHP

All variables in PHP start with a \$ sign symbol. Variables may contain strings, numbers, or arrays. Below, the PHP script assigns the string "Hello World" to a variable called \$txt:

```
<html>
<body>
<?php
$txt="Hello World";
echo $txt;
?>
</body>
</html>
```

To concatenate two or more variables together, use the dot (.) operator:

```
<html>
<body>
<?php
$txt1="Hello World";
$txt2="1234";
echo $txt1 . " " . $txt2 ;
?>
</body>
</html>
```

The output of the script above will be: "Hello World 1234".

Comments in PHP

In PHP, we use // to make a single-line comment or /* and */ to make a large comment block.

```
<html>
<body>
<?php
//This is a comment
/*
This is
a comment
block
*/
?>
</body>
</html>
```

PHP Operators

Operators are used to operate on values.

PHP Operators

This section lists the different operators used in PHP.

Arithmetic Operators

Operator	Description	Example	Result
+	Addition	$x=2$ $x+2$	4
-	Subtraction	$x=2$ $5-x$	3
*	Multiplication	$x=4$ $x*5$	20
/	Division	$15/5$ $5/2$	3 2.5
%	Modulus (division remainder)	$5\%2$ $10\%8$ $10\%2$	1 2 0
++	Increment	$x=5$ $x++$	
--	Decrement	$x=5$ $x--$	

Assignment Operators

Operator	Example	Is The Same As
=	x=y	x=y
+=	x+=y	x=x+y
-=	x-=y	x=x-y
=	x=y	x=x*y
/=	x/=y	x=x/y
%=	x%=y	x=x%y

Comparison Operators

Operator	Description	Example
==	is equal to	5==8 returns false
!=	is not equal	5!=8 returns true
>	is greater than	5>8 returns false
<	is less than	5<8 returns true
>=	is greater than or equal to	5>=8 returns false
<=	is less than or equal to	5<=8 returns true

Logical Operators

Operator	Description	Example
&&	and	x=6 y=3 (x < 10 && y > 1) returns true
	or	x=6 y=3 (x==5 y==5) returns false
!	not	x=6 y=3 !(x==y) returns true

Operator Examples

► Arithmetic Operators

```
<?php
$x=10;
$y=5;
$sum=$x+$y;
$sub=$x-$y;
$multiply=$x*$y;
$div=$x/$y;
$rem=$x%$y;
echo "sum=".$sum."<br/>";
echo "sub=".$sub."<br/>";
echo "Multiplication = ".$multiply."<br/>";
echo "Div = ".$div."<br/>";
echo "remainder=".$rem."<br/>";
?>
```

► Assignment Operators

```
<?php
$x = 500;
$x+= 500;
echo "sum=".$x."<br/>";
$x-= 500;
echo "subtraction = ".$x."<br/>";
$x*= 10;
echo "Multiplication = ".$x."<br/>";
$x/= 500;
echo "Quotient = ".$x."<br/>";
$x%= 2;
echo "Remainder = ".$x."<br/>";
$str = "Welcome ";
$str. = "to myclass";
echo $str."<br/>";
?>
```

► Comparison Operators

```
<?php
$x=10;
$y=10.0;
echo ($x==$y);
echo ($x===$y);

$a=10;
$b=11;
echo $a>$b;
echo $a<$b;
echo $a>=$b;
echo $a<=$b;

?>
```

Logical Operators

```
<?php
$name="abebe";
$pass="123";
if($name=="abebe" && $pass=="123")
{
header('location:https://www.myclass.com');
}
else
{
echo "Invalid name or password";
}

?>
```

PHP Conditional Statements

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

Conditional Statements

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

In PHP we have two conditional statements:

if (...else) statement - use this statement if you want to execute a set of code when a condition is true (and another if the condition is not true)

switch statement - use this statement if you want to select one of many sets of lines to execute

The If Statement

If you want to execute some code if a condition is true and another code if a condition is false, use the if....else statement.

Syntax

```
if (condition)
code to be executed if condition is true;
else
code to be executed if condition is false;
```

Example

The following example will output "Have a nice weekend!" if the current day is Friday, otherwise it will output "Have a nice day!":

```
<html>
<body>
<?php
$d=date("D");
if ($d=="Fri")
echo "Have a nice weekend!";
else
echo "Have a nice day!";
?>
</body>
</html>
```

If more than one line should be executed when a condition is true, the lines should be enclosed within curly braces:

```
<html>
<body>
<?php
$x=10;
if ($x==10)
{
echo "Hello<br />";
echo "Good morning<br />";
}
?>
</body>
</html>
```

The Switch Statement

If you want to select one of many blocks of code to be executed, use the Switch statement.

Syntax

```
switch (expression)
{
case label1:
    code to be executed if expression = label1;
    break;
case label2:
    code to be executed if expression = label2;
    break;
default:
    code to be executed
    if expression is
    different from both
    label1 and label2;
}
```

Example

This is how it works: First we have a single expression (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 default statement is used if none of the cases are true.

```
<html>
<body>
<?php
switch ($x)
{
case 1:
    echo "Number 1";
    break;
case 2:
    echo "Number 2";
    break;
case 3:
    echo "Number 3";
    break;
default:
    echo "No number between 1 and 3";
}
?>
</body>
</html>
```

PHP Looping

Looping statements in PHP are used to execute the same block of code a specified number of times.

Looping

Very often when you write code, you want the same block of code to run a number of times. You can use looping statements in your code to perform this.

In PHP we have the following looping statements:

while - loops through a block of code as long as a specified condition is true

do...while - loops through a block of code once, and then repeats the loop as long as a special 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 while Statement

The while statement will execute a block of code **if and as long** a condition is true.

Syntax

```
while (condition)
code to be executed;
```

Example

The following example demonstrates a loop that will continue to run as long as the variable *i* is less than, or equal to 5. *i* will increase by 1 each time the loop runs:

```
<html>
<body>
<?php
$i=1;
while ($i<=5)
{
echo "The number is " . $i . "<br />";
$i++;
}
?>
</body>
</html>
```

The do...while Statement

The do...while statement will execute a block of code **at least once** - it then will repeat the loop **as long as** a condition is true.

Syntax

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

Example

The following example will increment the value of *i* at least once, and it will continue incrementing the variable *i* while it has a value of less than 5:

```
<html>
<body>
<?php
$i=0;
do
{
$i++;
echo "The number is " . $i . "<br />";
}
while ($i<5);
?>
</body>
</html>
```

The for Statement

The for statement is used when you know how many times you want to execute a statement or a list of statements.

Syntax

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

Note: The for statement has three parameters. The first parameter is for initializing variables, the second parameter holds the condition, and the third parameter contains any increments required to implement the loop. If more than one variable is included in either the initialization or the increment section, then they should be separated by commas. The condition must evaluate to true or false.

Example

The following example prints the text "Hello World!" five times:

```
<html>
<body>
<?php
for ($i=1; $i<=5; $i++)
{
    echo "Hello World!<br />";
}
?>
</body>
</html>
```

The foreach Statement

Loops over the array given by the parameter. On each loop, the value of the current element is assigned to \$value and the array pointer is advanced by one - so on the next loop, you'll be looking at the next element.

Syntax

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

Example

The following example demonstrates a loop that will print the values of the given array:

```
<html>
<body>
<?php
$arr=array("one", "two", "three");
foreach ($arr as $value)
{
echo "Value: " . $value . "<br />";
}
?>
</body>
</html>
```

Arrays

The most inconvenient thing about a variable is that you can only store one value at a time. Arrays are special types that allow variables to overcome this limitation, so you can store as many values as you want in the same variable. For example, instead of having two variables “\$number1” and “\$number2”, you could have an array “\$numbers” that will hold both values. Imagine the same thing with ten numbers. What about a hundred? Because of the flexibility of the array, it can store two values or two hundred values, without having to define other variables. PHP indexes all the values within an array using a number or a string, so you will know which of the values you’re using.

Working with arrays is easy. You can process each item one after another, or you could just take one at random. Each item in an array is commonly referred to as an element. These elements can be accessed directly via their index, which can be either a number or a string. By default, PHP starts indexing elements numerically, from zero, and increments the element’s index with each new addition, so keep in mind that the index of the last elements in a numerically indexed array is always the total number of elements minus one. Indexing arrays by string can be useful in cases where you need to store both names and values.

There are two ways you can create an array: with the “array()” function or directly using the array identifier “[]”. You can use the “array()” function when you want to assign multiple elements to an array at a time. Don’t think that an array can only contain elements of a certain type (for example, only numbers). You can have numbers, strings and booleans in the same array, PHP won’t mind at all.

```
$names = array("John", 279, "Betty", TRUE);
```

This creates an array called “\$names” which holds the specified elements. You can access an array element by placing its index between square brackets, right after the array name. Not only you can retrieve a value this way, but you can also assign a value to that element.

```
print $names[2]; //outputs "Betty"
```

```
$names[3] = "Harrison"; //replaces TRUE with "Harrison"
```

Remember that PHP starts indexing from zero, therefore “\$names[0]” is “John”, and so on - the index of any element always is one less than the element’s place in the list. Another way to define an array is using the array identifier “[]” in conjunction with the array name. You can also use this to add new elements in you have already created an array – either using the “array()” function or the array identifier.

```
$names[] = "John";
```

```
$names[] = "Mary";
```

```
$names[] = "Betty";
```

```
$names = array("Harry", "Samantha", "Danny");
```

There is no need to place any numbers between the square brackets, PHP takes care of the index number, so you don’t have to figure out which is the next available slot. This doesn’t mean that you cannot add numbers, but pay attention not to skip any of them, because PHP will initialize only the elements with the index number you specify.

Arrays indexed by strings, and not numbers, can prove to be useful when you need to access elements in array by name, and not by number. For example, if you have an address book, it would be much better to have a field called “name” or “address”, instead of a numeric field called “1” or “2”. You can define an associative array pretty much the same way you define a numerically indexed array.

```
$sysinfo = array(computer_name => "My computer",
```

```
cpu_mhz => 2800,
```

```
memory_size => 512,
```

```
multimedia => TRUE);
```

Or you can use the array identifier:

```
$sysinfo[computer_name] = "My computer";
```

```
$sysinfo[cpu_mhz] = 2800;
```

```
$sysinfo[memory_size] = 512;
```

```
$sysinfo[multimedia] = TRUE;
```

It's good to know that an array element can itself be an array, so this enabled you to create sophisticated data structures called multidimensional arrays. Let's say you have an address book, and you use an array to hold the information on the people you know (John, Mary, Betty, Harry). But on the other hand, each element must be a collection of a person's details (first name, last name, telephone number). This is how you define it:

```
$address_book = array(
```

```
array(first_name => "John", last_name => "Davis", phone_number => "1234567"),
```

```
array(first_name => "Mary", last_name => "Stewart", phone_number => "1234568"),
```

```
array(first_name => "Betty", last_name => "Willis", phone_number => "1234569"),
```

```
array(first_name => "Harry", last_name => "Miller", phone_number => "1234560"));
```

You can access the data using two indices, the first one for "\$address_book" – which is a numeric index, and the second one for the person's details – which is a string index. The following text outputs the text "Mary":

```
print $address_book[1][first_name];
```

There are a lot of functions to use with arrays. You can merge, slice, shift and sort arrays, by using the functions with the same name: "array_merge()", "array_slice()", "array_shift()", and "sort()". Sorting is definitely one of the most used functions. The "sort()" function can sort both alphabetically or numerically, depending on the array elements. Read the PHP documentation to learn how to use these functions.

Array Examples

```
<?php
$arr = array(10,11,12,13,14,15);
echo $arr[0];
?>
```

OR

```
<?php
$arr=array(10,20,30,40,50);
$col=array("red","green","blue","black");
echo $arr[0];
echo $col[0];
?>
```

```
<?php
$arr[ ]=10;
$arr[ ]=20;
$arr[ ]=30;
$arr[ ]=40;
$arr[ ]=50;
echo $arr[0];
?>
```

OR

```
<?php
$arr[0]=10;
$arr[1]=20;
$arr[2]=30;
$arr[3]=40;
$arr[4]=50;
echo $arr[0];
?>
```

```
<?php
```

```
$Personage=array("Abebe"=>"30","Ali"=>"21","Lula"=>"43")
```

```
;
```

```
echo "Abebe is ".$Personage["Abebe"]."Years old";
```

```
?>
```

OR

```
<?php

$Personage[ 'Abebe' ]=30;

$Personage[ 'Ali' ]=21;

$Personage[ 'Lula' ]=43;

echo " Lula is ".$Personage[" Lula "]. "Years
old";

?>
```

PHP Forms

A very powerful feature of PHP is the way it handles HTML forms!

PHP Form Handling

The most important thing to notice when dealing with HTML forms and PHP is that any form element in an HTML page will **automatically** be available to your PHP scripts.

Look at the following example of an HTML form:

```
<html>
<body>
<form action="welcome.php" method="POST">
Enter your name: <input type="text" name="name" />
Enter your age: <input type="text" name="age" />
<input type="submit" />
</form>
</body>
</html>
```

The example HTML page above contains two input fields and a submit button. When the user fills in this form and hits the submit button, the "welcome.php" file is called.

The "welcome.php" file looks like this:

```
<html>
<body>
Welcome <?php echo $_POST["name"]; ?>.<br />
You are <?php echo $_POST["age"]; ?> years old!
</body>
</html>
```

A sample output of the above script may be:

```
Welcome John.
You are 28 years old!
```

Here is how it works: The `$_POST["name"]` and `$_POST["age"]` variables are automatically set for you by PHP. The `$_POST` contains all POST data.

Note: If the method attribute of the form is GET, then the form information will be set in `$_GET` instead of `$_POST`.

Form Examples

PHP Forms /sum of two numbers

```
<html>

<head>

<title>get_browser</title>

<?php

error_reporting(1);

$x = $_POST['num1'];

$y = $_POST['num2'];

$sum = $x + $y;

echo "Sum of two number = ".$sum;

?>

</head>
```

```

<body bgcolor="pink">
  <form method="post" >
    <table bgcolor="pink">
      <tr>
        <td>Enter number1</td>
        <td><input type="text" name="num1"/></td>
      </tr>
      <tr>
        <td>Enter number2</td>
        <td><input type="text" name="num2"/></td>
      </tr>
      <tr align="center">
        <td colspan="2" >
          <input type="submit" value="+"/></td>
        </td>
      </tr>
    </table>
  </form>
</body>
</html>

```

Displaying the result in a text box

```

<?php
if(isset($_POST['add']))
{
  $x=$_POST['num1'];
  $y=$_POST['num2'];
  $sum=$x+$y;
  echo "Result:<input type='text' value='$sum'/'>";
}
?>

```

```
<body>
<form method="post">
Enter number1 <input type="text" name="num1"/><br/><br/>
Enter number2 <input type="text" name="num2"/><br/>
<input type="submit" name="add" value="ADD"/>
</form>
</body>
```

Working with multiple buttons

```
<?php
    extract($_POST);
    if(isset($add))
    {
        $res=$fnum+$snum;
    }
    if(isset($sub))
    {
        $res=$fnum-$snum;
    }
    if(isset($mult))
    {
        $res=$fnum*$snum;
    }
?>
<html>
<head>
<title>Display the result in 3rd text box</title>
</head>
```

```
<body>

<form method="post">

  <table>

<tr> <th>Result</th>

<td><input type="text" value="<?php echo
@$res;?>"/></td> </tr>

<tr> <th>Enter first number</th>

<td><input type="text" name="fnum"/></td>

</tr>

<tr> <th>Enter second number</th>

<td><input type="text" name="snum"/></td>

</tr>

<tr> <td align="center" colspan="2">

<input type="submit" value="+" name="add"/>

<input type="submit" value="-" name="sub"/>

<input type="submit" value="*" name="mult"/>

</tr>

</table>

</form>

</body>

</html>
```