PHP

Presented By

Anuradha Kumari Singh
(Research Scholar)
Department of Computer Science

What is PHP?

PHP is a server scripting language, and a powerful tool for making dynamic and interactive Web pages.

- 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

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

What is a PHP File?

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

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
- PHP is easy to learn and runs efficiently on the server side

Set Up PHP on Your Own PC

However, if your server does not support PHP, you must:

- install a web server
- install PHP
- install a database, such as MySQL

The official PHP website (PHP.net) has installation instructions for PHP:
http://php.net/manual/en/install.php

PHP Version

To check your php version you can use the phpversion() function:

Example:

```
echo phpversion();
```

PHP Syntax

- A PHP script is executed on the server, and the plain HTML result is sent back to the browser.
- A PHP script can be placed anywhere in the document.
- A PHP script starts with <?php and ends with ?>:

```
<?php

// PHP code goes here

?>
```

A simple .php file with both HTML code and PHP code:

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

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

Comments in PHP

Syntax for comments in PHP code:

```
// This is a single-line comment


# This is also a single-line comment


/* This is a
   multi-line comment */
```

PHP Variables

Variables are "containers" for storing information.

Rules for PHP variables:

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

Output Variables

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

Example1:

```
$txt = "BHU";

echo "I love $txt!";
```

Example2:

```
$x = 5;

$y = 4;

   echo $x + $y;
```

PHP is a Loosely Typed Language

- In the example above, notice that we did not have to tell PHP which data type the variable is.
- PHP automatically associates a data type to the variable, depending on its value. Since the data types are not set in a strict sense, you can do things like adding a string to an integer without causing an error.

Variable Types

PHP has no command for declaring a variable, and the data type depends on the value of the variable.

Example

```
$x = 5;      // $x is an integer

$y = "John"; // $y is a string

echo $x;

echo $y;
```

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

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

```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>";
```

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

```
$x = 5;

$y = 10;


function myTest() {
  global $x, $y;
  $y = $x + $y;
}


myTest();
   echo $y; // outputs 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.

Example

```
$x = 5;

$y = 10;


function myTest() {

$GLOBALS['y'] = $GLOBALS['x'] +
$GLOBALS['y'];

}


myTest();

echo $y; // outputs 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.

Example

```
function myTest() {
  static $x = 0;
  echo $x;
  $x++;
}

myTest();
myTest();
myTest();
```

PHP echo and print Statements

Both **echo** and **print** are 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, while print can take one argument
- **echo** is marginally faster than **print**.

Example:

```
echo "Hello";

//same as:

  echo("Hello");
```

Example

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

echo "Hello world!<br>";

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

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

Display Variables

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

```
<!DOCTYPE html>
<html>
<body>
<?php
$txt1 = "Learn PHP";
$txt2 = "W3Schools.com";
echo "<h2>$txt1</h2>";
echo "<p>Study PHP at $txt2</p>";
?>
</body>
</html>
```

Using Single Quotes

- Strings are surrounded by quotes, but there is a difference between single and double quotes in PHP.
- When using double quotes, variables can be inserted to the string as in the example above.
- When using single quotes, variables have to be inserted using the . operator, like this:

```
<!DOCTYPE html>
<html>
<body>
<?php
$txt1 = "Learn PHP";
$txt2 = "W3Schools.com";
echo '<h2>' . $txt1 . '</h2>';
echo '<p>Study PHP at ' . $txt2 . '</p>';
?>
</body>
</html>
```

PHP Data Types:

PHP supports the following data types:

- string (text values)
- int (whole numbers)
- float (decimal numbers)
- bool (true or false)
- array (multiple values)
- object (stores data as objects)
- null (empty variable)
- resource (references external resources)

Use var_dump() to Get the DataType

To get the data type and the value of a variable, use the `var_dump()` function.

Example:

The `var_dump()` function dumps the data type and the value:

```
$x = 5;

var_dump($x); // dumps int(5)
```

```
<!DOCTYPE html>
<html>
<body>

<?php
$x = 5;
var_dump($x);
?>

</body>
</html>
```

PHP Strings

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

In PHP, strings are surrounded by either double quotes, or single quotes.

```
<?php
$x = "John";
echo "Hello $x";
?>
```

Output: Hello John

```
<?php
$x = "John";
echo ‘Hello $x’;
?>
```

Output: Hello $x

PHP - Modify Strings

PHP has a set of built-in functions that you can use to modify strings.

1. Upper Case

The PHP **strtoupper()** function returns a string in upper case.

Example:

```
$x = "Hello World!";

echo strtoupper($x);
```

2. Lower Case

The PHP **strtolower()** function returns a string in lower case.

Example:

```
$x = "Hello World!";

echo strtolower($x);
```

3. Replace String

The PHP **str_replace()** function replaces some characters with some other characters in a string.

Example:

```
$x = "Hello World!";

echo str_replace("World", "Dolly", $x);
```

4. Reverse a String

The PHP **strrev()** function reverses a string.

Example:

```
$x = "Hello World!";

echo strrev($x);
```

5. Remove Whitespace

Whitespace is the space before and/or after the actual text, and very often you want to remove this space.

The PHP **trim()** function removes any whitespace from the beginning or the end.

Example:

```
$x = " Hello World! ";

echo trim($x);
```

PHP - Concatenate Strings

String Concatenation

To concatenate, or combine, two strings you can use the . operator:

```
<?php
$x = "Hello";
$y = "World";
$z = $x . $y;
echo $z;
?>
```

Output: HelloWorld

PHP - Slicing Strings

The PHP **substr()** function is used to extract a part of a string (slice a string).

1. **Slice a String:**

Example:

```
<?php
$x = "Hello World!";
echo substr($x, 6, 5);
?>
```

Note The first character has index 0.

Output: World

2. **Slice String to the End:**

By leaving out the *length* parameter, the range will go to the end:

Example:

Start the slice at index 6 and go all the way to the end:

```
$x = "Hello World!";

echo substr($x, 6);
```

Output : World!

3. Slice String From the End

Use negative indexes will start the slice from the end of the string:

Example:

Get the 3 characters, starting from the "o" in world (index -5):

```
$x = "Hello World!";

echo substr($x, -5, 3);
```

Output: orl

Note The last character has index -1.

4. **Negative Length**

Use negative *length* to specify how many characters to omit, starting from the end of the string:

Example:

```
<?php

$x = "Hi, how are you?";

echo substr($x, 5, -3);

?>
```

Output: **ow are y**

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
- Conditional assignment operators

Arithmetic Operators

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

Assignment Operators

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

Comparison Operators

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

Increment / Decrement Operators

Operator	**Same as...**	**Description**
++$x	Pre-increment	Increments $x by one, then returns $x
$x++	Post-increment	Returns $x, then increments $x by one
--$x	Pre-decrement	Decrements $x by one, then returns $x
$x--	Post-decrement	Returns $x, then decrements $x by one

Logical Operators

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

String Operators

Operator	Name	Example	Result
.	Concatenation	$txt1 . $txt2	Concatenation of $txt1 and $txt2
.=	Concatenation assignment	$txt1 .= $txt2	Appends $txt2 to $txt1

Array Operators

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

Conditional Assignment Operators

Operator	Name	Example	Result
?:	Ternary	$x = *expr1* ? *expr2* : *expr3*	Returns the value of $x. The value of $x is *expr2* if *expr1* = TRUE. The value of $x is *expr3* if *expr1* = FALSE
??	Null coalescing	$x = *expr1* ?? *expr2*	Returns the value of $x. The value of $x is *expr1* if *expr1* exists, and is not NULL. If *expr1* does not exist, or is NULL, the value of $x is *expr2*. Introduced in PHP 7

PHP Conditional Statements

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

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

1. The if Statement

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

Syntax

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

Example

```
<!DOCTYPE html>
<html>
<body>

<?php
if (5 > 3) {
  echo "Have a good day!";
}
?>

</body>
</html>
```

2. The if...else Statement

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

Syntax

if (*condition*) {

// code to be executed if condition is true;

} else {

// code to be executed if condition is false;

}

Example

```
<!DOCTYPE html>
<html>
<body>

<?php
$t = date("H");

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

</body>
</html>
```

3. The if...elseif...else Statement

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

Syntax

if (*condition*) {

code to be executed if this condition is true;

} elseif (*condition*) {

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

} else {

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

}

Example

```
<!DOCTYPE html>
<html>
<body>

<?php
$t = date("H");
echo "<p>The hour (of the server) is " . $t;
echo ", and will give the following
message:</p>";

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

</body>
</html>
```

4. Nested If

You can have if statements inside if statements, this is called *nested* if statements.

Example

```
<!DOCTYPE html>
<html>
<body>

<?php
$a = 13;
if ($a > 10) {
  echo "Above 10";
  if ($a > 20) {
    echo " and also above 20";
  } else {
    echo " but not above 20";
  }
}
?>
</body>
</html>
```

5. switch Statement

- The PHP 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.

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

Syntax

```
switch (expression) {
  case label1:
    //code block
    break;
  case label2:
    //code block;
    break;
  case label3:
    //code block
    break;
  default:
    //code block
```

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!";
}  ?>
```

Output :

Your favorite color is red!

PHP Loops

PHP loops are used to execute the same block of code again and again, as long as a certain condition is true.

In PHP, we have the following loop types:

- 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

1. while Loop

The PHP while loop—loops through a block of code as long as the specified condition is true.

Syntax:

While (condition)

{

// statements;
}

Output:

12345

Example

```
<!DOCTYPE html>
<html>
<body>
<?php
$i = 1;
while ($i < 6) {
  echo $i;
  $i++;
}
?>
</body>
</html>
```

do while Loop

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

Note: In a do...while loop the condition is tested AFTER executing the statements within the loop. This means that the do...while loop will execute its statements at least once, even if the condition is false.

Example

```
<!DOCTYPE html>
<html>
<body>

<?php
$i = 1;

do {
  echo $i;
  $i++;
} while ($i < 6);
?>

</body>
</html>
```

for Loop

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

Syntax:

for (*expression1*, *expression2*, *expression3*) {

  // *code block*

}

This is how it works:

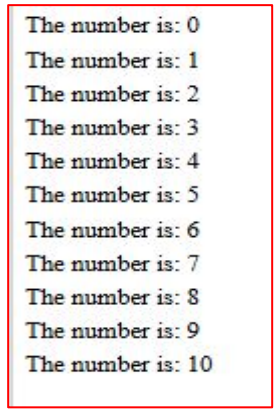
- *expression1* is evaluated once
- *expression2* is evaluated before each iteration
- *expression3* is evaluated after each iteration

Example

```
<!DOCTYPE html>
<html>
<body>

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

</body>
</html>
```



foreach Loop

foreach loop - Loops through a block of code for each element in an array or each property in an object.

Syntax:

foreach(array as value)

{

// code block

}

Example

```
<?php
$colors = array("red", "green", "blue",
"yellow");

foreach ($colors as $x) {
  echo "$x <br>";
}
?>
```

PHP User Defined Functions

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

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

1. Create a Function

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

Syntax:

```
function function_name(parameters) {
  //code block;
  }
```

Note: A function name must start with a letter or an underscore. Function names are NOT case-sensitive.

2. Call a Function

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

Syntax:

function_name(arguments);

Example

```
<!DOCTYPE html>
<html>
<body>

<?php
function familyName($fname) {
  echo "$fname Refsnes.<br>";
}

familyName("Jani");
familyName("Hege");
familyName("Stale");
familyName("Kai Jim");
familyName("Borge");
?>

</body>
</html>
```

Output:

Jani Refsnes.
Hege Refsnes.
Stale Refsnes.
Kai Jim Refsnes.
Borge Refsnes.

Variable Number of Arguments

By using the ... operator in front of the function parameter, the function accepts an unknown number of arguments. This is also called a variadic function.

The variadic function argument becomes an array.

Example

```
function sumMyNumbers(...$x) {
  $n = 0;
  $len = count($x);
  for($i = 0; $i < $len; $i++) {
    $n += $x[$i];
  }
  return $n;
}

$a = sumMyNumbers(5, 2, 6, 2, 7, 7);

echo $a;
```

Output: 29

Arrays

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

In PHP, there are three types of arrays:

- Indexed arrays - Arrays with a numeric index
- Associative arrays - Arrays with named keys
- Multidimensional arrays - Arrays containing one or more arrays

Create Array

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

Syntax:

```
$array_name =
array("values",
"values","values",.....);
```

Example:

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

Multiple Lines

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

```
$cars = [
  "Volvo",
  "BMW",
  "Toyota"
];
```

Array Keys

When creating indexed arrays the keys are given automatically, starting at 0 and increased by 1 for each item, so the array above could also be created with keys.

```
$cars = [
  0 => "Volvo",
  1 => "BMW",
  2 =>"Toyota"
   ];
```

Declare Empty Array

You can declare an empty array first, and add items to it later:

Example:

```
$cars = [];

$cars[0] = "Volvo";

$cars[1] = "BMW";

$cars[2] = "Toyota";
```

1. Indexed Arrays

In indexed arrays each item has an index number.

By default, the first item has index 0, the second item has item 1, etc.

Example:

```
<?php
$cars = array("Volvo", "BMW", "Toyota");
var_dump($cars);
?>
```

Output:

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

Access Indexed Arrays

To access an array item you can refer to the index number.

```
<?php

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

echo $cars[0];

?>
```

Output: Volvo

Change Value

To change the value of an array item, use the index number:

```
<?php

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

$cars[1] = "Ford";

var_dump($cars);

?>
```

Output:

```
array(3) {
  [0]=>
  string(5) "Volvo"
  [1]=>
  string(4) "Ford"
  [2]=>
  string(6) "Toyota"
}
```

Loop Through an Indexed Array

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

```
<?php

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

foreach ($cars as $x) {

  echo "$x <br>";

}

?>
```

Output:

Volvo
BMW
Toyota

2. Associative Arrays

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

Syntax:

```
$var = array("key1"=>"Value1", “Key2"=>"Value2", "Key3"=>Value3,.....);
```

Access Associative Arrays

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

Example:

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

echo $car["model"];
```

Change Value

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

Example:

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

$car["year"] = 2024;

var_dump($car);
```

Thank you