PHP (Hypertext preprocessor)

PHP Introduction What

is PHP?

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

PHP is an amazing and popular language!

PHP is powerful enough to be at the core of the biggest blogging system on the web!
PHP is deep enough to run large social networks!
PHP is easy enough to be a beginner's first server-side scripting language!

Why PHP?

- PHP runs on all major platforms (Windows, Linux, Unix, Mac OS X, etc.)
- PHP is compatible with all leading web servers (Nginx, Apache, Cloudflare, Microsoft IIS, etc.)
- PHP supports a wide range of databases (MySQL, PostgreSQL, MS SQL, db2, Oracle Database, MongoDB, etc.)
- 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

---

What is a PHP File?

- PHP files have the file extension .php
- 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 HTML

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

PHP Syntax

Basic 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

?>

The default file extension for PHP files is .php.

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

Below, we have an example of a simple PHP file, with a PHP script that uses the PHP echo function to output some text on a web page:

Example

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

PHP Case Sensitivity

PHP keywords (e.g. if, else, echo, etc.), classes, functions, and userdefined functions are not case-sensitive.

In the example below, both echo statements are legal:

Example

ECHO is the same as echo:

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

<body>

<?php

ECHO 'Hello World!
'; echo

'Hello World!
';

?>

</body>

</html>

PHP Comments

Comments in PHP

PHP comments are not executed as a part of the code. Their only purpose are to be read by someone who is looking at the code.

PHP comments are used for:

- Let others understand your code
- Remind yourself of what you did - Most programmers have experienced coming back to their own work a year or two later and having to re-figure out what they did
- Leave out some parts of your code - for testing code PHP supports two types of comments:
- Single-line comments
- Multi-line comments

Example

Syntax for PHP comments:

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

# This is also a single-line comment


/* This is a multi-line

comment */
```

PHP Single-line Comments

Single line comments start with // or #.

Any text after the // or # - and to the end of line, will be ignored.

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

Example

```
A comment before a code line: //

Output welcome message echo

'Welcome Home!';
```

Example

A comment at the end of a code line:

```
echo 'Welcome Home!'; // Output welcome message
```

PHP Multi-line Comments

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

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

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

Example

Multi-line comment as an explanation:

```
/* The next statement will print
a welcome message */ echo
'Welcome Home!';
```

Multi-line Comments to Ignore Code

We can use multi-line comments to prevent blocks of code from being executed:

Example

Use multi-line comment to ignore code:

```
/*
echo 'Welcome to my home!'; echo
'Mi casa su casa!';
*/ echo
'Hello!';
```

Comments in the Middle of the Code

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

Example

The + 15 part will be ignored in the calculation:

```
$x = 5 /* + 15 */ + 5; echo
$x;
```

PHP Variables

PHP 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 must start 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!

Create PHP Variables

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

Example

Create two variables, named $x and $y:

```
$x = 5;

$y = "John";
```

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

Output Variables

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

The following example will show how to output some text and the value of a variable:

Example

```
$txt = "W3Schools.com"; echo

"I love $txt!";
```

The following example will produce the same output as the example above:

Example

```
$txt = "W3Schools.com";

echo 'I love ' . $txt . '!';
```

The following example will output the sum of two variables:

Example

```
$x = 5; $y = 4;

echo $x + $y;
```

PHP Variables and Data Types

In PHP, 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 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)
- **mixed** (any value)

Use var_dump() to Get the Data Type

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

Example

The var_dump() function returns the data type and the value:

var_dump(5); var_dump("John"); var_dump(3.14);

var_dump(true); var_dump([2, 3, 56]); var_dump(NULL);

Assign Multiple Values

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

Example

Here, all three variables get the value "Fruit":

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

PHP Variable Scope

PHP variables can be declared anywhere in the PHP code.

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

PHP has three different variable scopes:

- global

- local
- static

Global Scope

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

Example

Variable with global scope:

```
$x = 5; // global scope


function myTest() {
  // using x inside this function will not work   echo
"Variable x inside function is: $x";
}
myTest();


echo "Variable x outside function is: $x";
```

Local Scope

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

Local variables are created when the function is called and are destroyed when the function finishes executing:

Example

Variable with local scope:

```
function myTest() {   $x = 5; // local scope

echo "Variable x inside function is: $x";

}

myTest();


// using x outside the function will not work echo

"Variable x outside function is: $x";
```

Static Scope

Normally, when a function finishes executing, 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 achieve this, use the static keyword when you first declare the variable.

Then, each time the function is called, that variable will have the value from the last time the function was called.

Note: The variable is still local to the function.

Example

```
function myTest() {   static $x

= 0; // static scope   echo $x;

 $x++;

}


myTest(); myTest();

myTest();
```

PHP 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;
```
