Server-side Scripting PHP

PHP stands for Personal Hypertext Processor (or PHP Hypertext Processor).

It is a ***server-side*** scripting language that can be used to generate HTML, CSS, etc... This means that before the web page is loaded on the screen, the PHP code is sent to the server for processing.

The opposite of this is called ***client-side*** scripting whereby the user's computer processes the code and is run directly in the browser. JavaScript is an example of this type of language.

Using PHP, it is possible to create custom web-pages in response to user-actions, e.g.

- a page listing the results of a search on a database (IMDB, Amazon),
- a page showing the current state of an e-commerce transaction.

A PHP file may contain only PHP code or a mixture of PHP and HTML code:

- any HTML code in the file is sent directly to the browser,
- the PHP code is executed and the resulting code is sent to the browser.

Any file containing PHP, even just a little bit of it, should have the extension **.php** e.g. (myHomePage.php). This instructs the web-server to execute the PHP code before sending the file to the client.

Writing PHP

Within the webpage, all PHP code must be surrounded by the following tags:

<?php ?>

You can open and close these tags many times within a webpage however PHP code will only be read and executed properly when it is in between those tags.

There are a few ways to comment your code with PHP. Commenting means to add a line, or group of lines, that are not read by the browser. These would contain instructions for a developer. It is good practice to comment your code.

```
<?php
    // This is a single line comment
    # This is another single line comment
    /* This is a multi-line comment.
       It must be closed off by this tag */
?>
```

All lines of PHP code must end with a semi colon ;

Variables in PHP

Variables are containers which "store" information. They are very common throughout computer programming.

Since PHP is a *loosely typed language*, we do not need to specify the variable type when declaring a variable. All that is required to create a variable is to type $ followed by your variable name.

```
<?php
        $myNewVariable = 512;
?>
```

The most common types of variables are: strings, integers, floats, arrays and Booleans.

```
<?php
        $stringVar  =  "Hello World";     // a sequence of characters. Must use " "
        $intVar     =  15;                // a whole number positive or negative
        $floatVar   =  4.25;              // a decimal point number pos or neg
        $boolVar    =  true;              // true or false
        $arrayVar   =  array("apple", "banana", "cactus");
        // an array is a collection of multiple values in a single variable
?>
```

A handy thing for debugging is **var_dump()**. When you place the variable name inside the brackets, the var_dump function will return the data type and value of the variable. For instance:

```
        var_dump($floatVar);
```

Will return **float(4.25)**.

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)

Outputting PHP to the Web

With PHP there are 2 main ways to show content on a web page. They are the **echo** and **print** statements. They are very similar however **echo** is far more common. By using either of these we can put HTML content, as well as PHP variables, on a webpage.

```
<?php
        echo “<h2>This header will appear on the page</h2>”;
        echo “This simple text will also appear”;
        echo “<button>Press Me</button>”;
?>
```

Note how it’s outputting HTML elements. It will also display the values of PHP variables on a webpage.

```
<?php
        $myName = “ Joe Bloggs”;
        $age = 38;
        echo “<p>My name is $myName</p>”;
        echo “<p>I am $age years old.</p>”;
?>
```

The above PHP code will output:

My name is Joe Bloggs
I am 38 years old.

Dealing with Strings

PHP has a number of inbuilt functions used for dealing with strings.

strlen("Hello world!");	Outputs the # of characters in string	->	**12**
str_word_count("Hello world!");	Outputs the # of words in string	->	**2**
strpos("Hello world!", "world");	Outputs the position of searched string	->	**6**

*Note for **strpos()** the string on the right “world” is being searched for in the string on the left “Hello world!”. The returned value is the number position of the first character of the searched word. In this case the “w” is the seventh character in that string however in computer science in general we count the first position as 0. Therefore world appears at 0, 1, 2, 3, 4, 5, **6**.

PHP Operators

Operators are used to perform operations on variables and values such as subtracting integers from each other, concatenating (putting together) strings together or assigning values to variables. There are a number of types of operators. For the below examples the variable **$x** is an integer equal to 2 and **$y** is an integer equal to 1.

Arithmetic Operators are used with numeric variables to perform math such as addition, subtraction…

Operator	Name	Example	Result	Show it
+	Addition	$x + $y	Sum of $x and $y	3
-	Subtraction	$x - $y	Difference of $x and $y	1
*	Multiplication	$x * $y	Product of $x and $y	2
/	Division	$x / $y	Quotient of $x and $y	2

Assignment Operators are used with numeric data to put values in variables. The most common one is "=" which states that the variable on the left side is assigned the expression on the right.

Assignment	Same as...	Description	Show it
x = y	x = y	The left operand gets set to the value of the expression on the right	x = 1
x += y	x = x + y	Addition	x = 3
x -= y	x = x - y	Subtraction	x = 1
x *= y	x = x * y	Multiplication	x = 2

String Operators are used with string variables to place them together. In the below example **$txt1** is equal to “Hello” and **$txt2** is equal to “World”.

Operator	Name	Example	Result	Show it
.	Concatenation	$txt1 . $txt2	Concatenation of $txt1 and $txt2	HelloWorld
.=	Concatenation assignment	$txt1 .= $txt2	Appends $txt2 to $txt1	WorldHello

Conditional Statements

Conditional statements are used to do different things based on the outcome of another thing. These are often called **if else statements**. For instance, if a user that has a Facebook account goes to facebook.com, they should be brought to their home page. If they don’t, they should be redirected to the sign up page. Conditional statements are very common in computer programming.

These statements rely on comparison operators to determine whether a block of code should be executed or not. The comparison operator is shown underlined in red below.

```
<?php
        $age = 19;
        if ( $age >=  18 ) {
                echo "<p>You can buy liquor.</p>";
        }
        else {
                echo “<p>You are too young to buy liquor.</p>”;
        }
?>
```

Because **$age** is greater than or equal to 18 (19 > 18), the line “You can buy liquor” will be printed on the webpage. The line “You are too young to buy liquor” will not be outputted.

These **comparison operators**, located inside the () of the if statements, are used to compare two values (number or string). Remember **$x** = 2 and **$y** = 1.

Operator	Name	Example	Result	Show it
==	Equal	$x == $y	Returns true if $x is equal to $y	False
===	Identical	$x === $y	Returns true if $x is equal to $y, and they are of the same type	False
!=	Not equal	$x != $y	Returns true if $x is not equal to $y	True
>	Greater than	$x > $y	Returns true if $x is greater than $y	True
<	Less than	$x < $y	Returns true if $x is less than $y	False
>=	Greater than or equal to	$x >= $y	Returns true if $x is greater than or equal to $y	True
<=	Less than or equal to	$x <= $y	Returns true if $x is less than or equal to $y	False

If else statements are only good for 2 outcomes. If this is true do this, otherwise do this. These statements can be extended with else if() statements which check for another condition if the first condition is not met.

```
<?php
        if ( $medal == “gold” ) {
                echo "<p>You came first.</p>";
        }
        else if ($medal == “silver”) {
                echo "<p>You came second.</p>";
        }
        else {
                echo “<p>You came third.</p>”;
        }
?>
```

*Note that the else statement comes at the end.

The PHP **Logical operators** are used to combine conditional statements to see if a range of conditions are met before executing code.

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

```
<?php
        if ( $medal == “gold” || $medal == “silver” ) {
                echo "<p>You are happy.</p>";
        }
        else {
                echo “<p>You aren’t happy at all </p>”;
        }
?>
```

Arrays

We mentioned earlier that arrays are a special type of variable. They are a single variable which can hold more than one value at a time. Think of them as a list. Using arrays we can sort through items stored in them, access specific items using an index and loop through them entirely.

```
<?php
        $fruitArray  =  array("Apple", "Banana", "Orange");
        echo "I eat " . $fruitArray [0] .", ". $fruitArray [1] . " and " . $fruitArray [2] . ".";
?>
```

This will print on the page:

I eat Apple, Banana and Orange.

Here are some useful PHP functions that deal with arrays:

count($fruitArray);
This returns the length of the array. In this case it is 3.

array_push($fruitArray,"Plum","Grape");
This pushes 2 new values into the end of an existing array.

array_splice($fruitArray, 3, 2);
This removes values from an array. The first number 3 specifies the index of the array where to begin the removal and the second number 2 specifies how many values from there to remove. In the above case "Plum" and "Grape" are removed.

There are 3 types of arrays in PHP:

- **Indexed Arrays:** Like the one above the items stored in the array can be accessed using a number. The first one is 0 while the next is 1 and so on. **$fruitArray [0]**
  These are the most common and the easiest to use when looping through big arrays.
- **Associative Arrays:** All items in the array are linked to a key which you would use to access them.

```
<?php
        $fruitPrice = array("Apple"=>"0.65", "Banana"=>"0.87", "Orange"=>"0.73");
        echo "Bananas cost " . $ fruitPrice ['Banana'] . " Canadian cent.";
?>
```

Bananas cost 0.87 Canadian cent.

- **Multidimensional Arrays:** These are arrays inside of arrays. Sort of like the movie "Inception", when there are multiple arrays inside of arrays it can get quite confusing. The most common multi array you may come across is a 2 dimensional array. It is best to visualize them as a table first.

Fruit	Price	Weight
Apple	0.65	1kg
Banana	0.87	2kg
Orange	0.73	1kg

```
$multiFruit = array
 (
        array("Apple", 0.65, “1kg”),
        array("Banana", 0.87, “2kg”),
        array("Orange",0.73, “1kg”)
);
```

In order to access a single piece of data from the array, let’s take the weight of the banana for instance, you must specify the row index followed by the column index like so:

```
echo “<p> The weight of a Banana is “ . $multiFruit[1][2] . “. </p>”
```

Essentially you are first accessing the array you want, in this case the one with the banana **$multiFruit[1][2]**, and then you are accessing a value inside that array **$multiFruit[1][2]**.

Include and Require

In PHP the include and require statements are used to import a PHP file into another PHP file. All text/code/markup and even variables of the imported file will be inserted before the server executes it. It acts in a similar way to an external style sheet for CSS.

These statements come in handy for elements that are common throughout a website. For instance a menu should probably be written once in a single PHP file and then get included into all relevant pages.

```
<?php include 'menu.php'; ?>
```

or

```
<?php require 'menu.php'; ?>
```

The difference between the two becomes apparent when the file specified cannot be found.

For **include** when the file cannot be found the rest of the script will continue to execute. For **require** the script execution will be killed after this error is discovered.