**PHP**

**(with Apache server)**

### 

**Stage 1**

**PHP BUILDING BLOCKS**

**Lecturer: Alexander Adu- Sarkodie**

**MSc. Telematics (IT & Telecom), MSc. Eng., Dip. Russ. Lang., Teach. Cert, AMIAEng (UK), MBCS (UK), MIfL (UK)**

**Blog:** [**http://www.blogger.com/profile/14800490193632788559**](http://www.blogger.com/profile/14800490193632788559)

**PHP building blocks for programmes**

One of PHP’s advantages is that you can ***embed*** the PHP code directly alongside HTML. For the code to do anything it must be passed to the **PHP engine** for **interpretation** via the **web server**.

The default delimiter syntax opens with  **<?php** and concludes with **?>**

**echo and print statements**

The echo statement is the most **basic** statement in PHP. What this does is **output** whatever you tell it to echo (or execute). Notice when we echo a **statement** or **string**, it is **contained within quotation marks.**

<?php

echo "<div>Hi there! We are happy you have chosen this course</div>";

?>

Another way to do this is to use the print function. An example of that would be:

<?php print "I like PHP" ?>

There is a lot of debate about which is better to use or if there is any difference at all. Apparently, in very large programs that are simply outputting text the **ECHO** statement will run slightly faster, but for the purposes of a beginner they are interchangeable.

The print function is slightly more **dynamic** than the echo function by **returning** a value.

The printf function inserts dynamic variables/whatever into wherever you want with special delimiters, such as %s, or %d. For example,

<?php

printf('There is a difference between %s and %s', 'good', 'evil')

?>

would return,

'There is a difference between good and evil'.

**Alternative php syntaxes**

You can also use the short hand forms below, which requires support to be enabled on the server.

1. <? …?>
2. <?= …?> Which is equivalent to <?php echo ….;?>
3. In a pure syntactically php programme, you can leave out the closing php syntax

Use :

**<?php**

**echo phpinfo();**

**?>**

to find out list of properties, functions and characteristics of your programme installation and its environment.

**Variable declaration**

A variable **serves as storage** for holding data for a programme.

A variable always **begins** with a dollar sign**, $** which is then followed by the **variable name**. Variable names can begin with either a **letter** or **an underscore** and can consist of letters, underscores, numbers or other **ASCII** characters ranging from **127** to **255**.

Variables are **case sensitive**. Once you’ve **declared** your variable you can begin to **assign values** to them. Two methodologies are available for variable assignment: by **value** and by **reference**,

By reference, you can create a **second** variable that **refers** to the **same content as another variable does**. What this means is any change to the variable referencing a particular item of variable content will be reflected among all other variables referencing that same content. You can assign variable by reference by **appending** an **ampersand (&)** to the assignment operator.

*Variable assignment:*

$color = “green”;

*Reference assignment (*The two syntaxes below are valid)

i)

<?php

$value1 = “goodbye”;

$value2 =& $value1;

$value2 = “goodbye”;

?>

ii) You can either append & after assignment operator or prepend to the old variable name.

<?php

$value1 = “goodbye”;

$value2 = &$value1;

$value2 = “goodbye”;

?>

**B. Variable scope.**

The ***location*** of the variable declaration in a program greatly **influences the realm** in which a variable can be **accessed**. PHP variables can be one of four scopes

* **Local** variables
* **Function** parameters
* **Global** variables
* **Static** variables

*i) Local variables*

A variable declared **in a function** is considered **local** and can only **be referenced in that function**. Note that when you **exit** the function in which a local variable is declared, that **variable and its corresponding values are destroyed.**

Local variables are helpful because they **eliminate** the **possibility** **of unexpected side effect which can result from globally accessible variables that are modified intentionally or not**.

Executing this:

<?php

$x = 4;

function assignx() {

$x = 0;

printf(“\$x inside function is %d <br />” ,$x);

}

assignx();

printf(“\$x outside function is %d <br />”,$x);

?>

Results in the following,

***$x inside function is 0***

***$x outside function is 4***

*ii) Function parameters*

Function parameters are declared **after** the function name and **inside parenthesis**. Note that you can **access** and **manipulate** any function parameter **in the function in which it is declared, it is destroyed once the execution ends.**

<?php

//multiply a value by 10 and return to the caller

function x10($value) {

$value2 = $value \* 10;

return $value2;

}

?>

*iii) Global variables*

A **global** variable can be accessed in **any part of the program**. However, to **modify a global variable in a function, it must be explicitly declared in the function as being global. By placing the keyword GLOBAL in front of the variable**, that should be recognized as global.

In the example below,

<?php

$somevar = 15;

function addit() {

GLOBAL $somevar;

$somevar++;

}

addit();

print “somevar is $somevar”;

?>

**The displayed value of $somevar would be 16. However, if you were to omit the line**

**GLOBAL $somevar then the value will be 1 on execution.**

Another method for declaring global variables is to use PHP’s **$GLOBALS** array as below:

<?php

$somevar = 15;

function addit() {

$GLOBALS [“somevar”]++;

}

addit();

print “somevar is ” . $GLOBALS[“somevar”];

?>

*iv) Static variables*

In contrast to function parameters, which are destroyed on function exit, a **static** **variable does not lose value and will continue to hold it for further execution if the function is called again**. Static scooping is useful for **recursive functions**. Recursive functions **call themselves repeatedly until a particular condition is met.**

You can declare a variable to be static by simply placing the keyword **STATIC** in front of the variable name.

In the example below,

<?php

function keeptrack() {

STATIC $count = 0;

$count++;

echo $count;

echo “<br />”;

}

keeptrack();

keeptrack();

keeptrack()

?>

The displayed value if **not** STATIC was used would be

***1***

***1***

***1***

However, because $count **is** STATIC, **it retains its previous value each time the function is executed**. Giving the following outcome:

***1***

***2***

***3***

*v) Variable variables*

It is possible to treat at times the **content** of a variable as **dynamic**. This is achieved by placing a **second dollar sign in front of the original variable name** and again assigning another value. The program below:

<?php

$recipe = “spaghetti”;

$$recipe = “and meatballs”;

echo $recipe ${$recipe };

?>

will evaluate to : ***spaghetti and meatballs***

**C. PHP super globals**

PHP offers a number of useful **pre-defined** **variables** that are **accessible** from **anywhere within the executing script** and provides a substantial amount of **environment specific information**.

The following code will output all pre-defined variables relevant to any giving Apache web server and the scripts execution environment.

<?php

foreach ($\_SERVER as $var => $value) {

echo “$var => $value <br />”;

}

?>

You can use this code to check the user’s IP address.

<?php

printf(“your IP address is : %s” , $\_SERVER[‘REMOTE\_ADDR’]);

?>

Where %s denotes the type of data format.Which is **string** in this case.

You can also gain information about a user’s browser and operating system using the code:

<?php

printf(“your browser is : %s” , $\_SERVER[‘HTTP\_USER\_AGENT’]);

?>