**PHP**

**(with Apache server)**

### 

**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)

**iStudio:** [**http://www.goldhawk-college.com/responsive.html**](http://www.goldhawk-college.com/responsive.html)

**PHP building blocks for programmes**

One of PHP’s advantage 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. 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.

**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)

<?php

$value1 = “goodbye”;

$value2 = &$value1;

$value2 = “goodbye”;

?>

**B. Variable scope.**

The ***location*** of the variable declaration 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) {

$value = $value \* 10;

return $value;

}

?>

*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’]);

?>

**D. Constants.**

A **constant** is a value that cannot be modified throughout the execution of the program. For example an address or a mathematical value, like “PI”. Constants are defined using the **define ()** function.

The function defines a constant by **assigning a value to the name**. In the example below, the mathematical constant PI is defined and further on manipulated. Note the name is a string.

<?php

define (“PI”,3.141592);

//define (“VAT”,15);

printf(“The value of PI is %f”,PI);

$pi2 = 2 \* PI;

printf(“<br />The value of PI double is %f”, $pi2 );

?>

Where %f is floating point data format. One that takes whole numbers as well as decimals.

This will evaluate to:

***The value of PI is 3.141592***

***The value of PI double is 6.283184***

**E. Expressions**

An expression is a **phrase** **representing a particular action in a program**. All expressions consists of at least **one operand** and **one or more operators**. Variable declarations are expressions.

*Operands*

Operands are the **inputs** for expressions. For example, $alpha, $sum, $val1 and $val2 are all operands

<?php

$alpha;

$sum = $val1 + $val2

?>

*Operators*

An operator is a **symbol that specifies a particular action**. The **precedence** and **association** of operators are significant characteristics of any programming language. Use **BODMAS** (bracket of DIVISION, MULTIPLICATION, ADDITION, SUBTRACTION) in order of precedence in a chained expressions using multiple operators.

1. Arithmetic operators: **Perform various mathematical operations**

**Example Label Outcome**

$a + $b Addition sum of $a and $b

$a - $b Subtraction Difference of $a and $b

$a \* $b Multiplication Product of $a and $b

$a / $b Division Quotient of $a and $b

$a % $b Modulo Remainder of $a divided by $b

1. Assignment operators: These **assign a data value to a variable**.

**Example Label Outcome**

$a = 5 Assignment $a equals 5

$a += 5 Addition assignment $a equals $a plus 5

$a \*= 5 Multiplication assignment $a equals $a multiplied by 5

$a /= 5 Division Assignment $a equals $a divided by 5

$a .= 5 Concatenation assignment $a equals $a concatenated with 5

1. String operators: Provide a means for **concatenating strings** together.

There are two main operators used here. The **dot operator** (**.**) and **the concatenating assignment operator** (**.=**)

**Example Label Outcome**

$a = “abc” .”def”; Concatenation $a is assigned the string abcdef

$a .= “ghijkl”; Concatenation assignment $a equals its **current** value **concatenated** with “ghijkl”

eg;

<?php

//$a contains the string value “spaghetti and meatballs”

$a = “spaghetty” **.** ”and meatballs”;

$a .= “are delicious”;

//this will evaluate to the string :”**spaghetti and meatballs are delicious**”

?>

1. Increment and decrement operators

Increment and decrement operators provide **shortened means** by which you can **add 1 to or subtract 1 from the current value of a variable and return it.**

**Example            Label             Outcome**

++$a  $a++      Increment        Increment $a by 1

--$a  $a--         Decrement       Decrement $a by 1

1. Logical operators

Logical operators make it possible to **direct the flow of a program** and are used frequently with **control structures**, such as the **if** conditional, the **while** and **for loops**. They are also commonly used **to provide details about the outcome of other operations**. Especially, those that return value.

**Example Label Outcome**

$a && $b AND True if both $a and $b are true

$a AND $b AND True if both $a and $b are true

$a || $b OR True if either $a or $b is true

$a OR $b OR True if either $a or $b is true

!$a NOT True if $a is not true

NOT $a NOT True if $a is not true

$a XOR $b Exclusive OR True only if $a or $b is true

1. Equality operators

Equality operators are used to **compare two values testing for equivalence**

**Example Label Outcome**

$a == $b Is **equal** to True if $a and $b are equivalent

$a != $b Is **not equal** to True if $a is not equal to $b

$a === $b Is **identical** to True if $a and $b are **equivalent** and the **same type.**

1. Comparison operators

Comparison operators like logical operators provide a method to direct program flow **through an examination of the comparative value of two or more variables or operands**

**Example Label Outcome**

$a < $b Less than True if $a is less than $b

$a > $b Greater than True if $a is greater than $b

$a <= $b Less than or equal to True if $a is less than or equal to $b

$a >= $b Greater than or equal to True if $a is greater than or equal to $b

($a == 12) ? 5: -1 Ternary If $a equal 12 return the value 5; otherwise return value -1

**F. String Interpolation and Escape Sequence.**

PHP offers a means for both **literal** and **figurative** interpretation. For example, the following string;

The $animal somersaulted across the lawn.\n

$animal and \n could be a literal other than PHP interpretation as a variable and new line character.

1. Escape sequence

**Sequence Description**

\n Newline character

\r Carriage return

\t Horizontal tab

\\ Backslash

\$ Dollar sign

\” Double quote

\[0-7]{1,3} Octal notation

\x[0-9A-Fa-f]{1,2} Hexadecimal notation

**G. Quotes**

i)        Double quotes

Strings enclosed in double quotes are the most commonly used in most PHP scripts. This is because they offer **more flexibility and guarantee that both variables and escapes will parse appropriately** if the right and standard syntaxes are adhered to.

The programme below

<?php

$sport = "cricket";

echo "Mike's favourite sport is $sport";

?>

will evaluate to;

**Mike's favourite sport is cricket**

Escape sequences are also parsed. **This is visible in the browser source but not in the browser window**. Use html for appropriate formatting for the browser window.

The program below;

<?php

$output = "This is one line. \n And this is another line";

echo $output ;

?>

will evaluate to in the browser

**This is one line And this is another line**

And evaluate to on two lines in source code

**This is one line**

**And this is another line**

1. Single quotes

Enclosing strings in single quotes is helpful, when the string should be interpreted **exactly** as stated. For example the program below;

<?php

print 'This string will $print exactly as it\'s \n declared';

?>

will evaluate to; **This string will $print exactly as it's \n declared**

Note it\'s is parsed or else there will be early termination. Another way is to use magic quotes. The server must be enabled for this to work.

1. Heredoc

**Heredoc** syntax offers convenient means for outputting **large amounts of text**. Rather than delimiting with single and double quotes.

Using this syntax,

* You create your own **identifiers**, but there are strict rules to having them parsed. The parsing of the string follows the same rules as parsing strings in double quotes. Variable and escape sequences are parsed but not the double quotes within the string.
* The **opening** and **closing** identifiers must be the same and are case sensitive

The opening should be preceded by **three angle brackets pointing to the left**

* No space should follow the opening identifier and should be on a **different line to the following string**
* The content of the string goes on the **immediate next line**
* The closing identifier must be on the **next line following the string and there should be no space before and after it.**

See example below:

<?php

$website = “http://www.romaterminal.it”;

echo <<<EXCERPT

<p>Rome's central train station, known as <a href=" $website ">Roma Terminal</a> was built in 1867 blurb

</p>

EXCERPT;

?>

This will evaluate to:

**Rome's central train station, known as Roma Terminal was built in 1867 blurb**

***Note: Rome terminal will be clickable to the site address:*** http://www.romaterminal.it;

**Useful debug type functions**

* print\_r : Prints human readable information about a variable.
* var\_dump **:** Dumps information about a variable
* var\_export : Outputs or returns a parsable string representation of a variable
* get\_defined\_vars : Returns an array of all defined variables
* gettype : Get the type of a variable

**Control Structures**

Control structures determine the **flow** of code within the application, defining **execution** **characteristics** such as **whether and how many times a particular code statement will execute as well as when a code block will relinquish execution**.

**H. Conditional Statement**

Conditional statements make it possible for your computer program to **respond accordingly** to a wide **variety of inputs**, using **logic** to **discern between various conditions based on input value**.

1. The **if** statement

The if statement is one of the most **common place constructs** of any **mainstream programming language**, offering convenient means for conditional code execution. The following is the syntax

if (expression) {

Statementto execute;

}

As an example suppose you want a congratulatory message displayed if the user guesses a pre-determined secret number.

<?php

$secretNumber = 045; //First set the result expected outcome

if($\_POST[‘guess’] == $secretNumber) {

echo “<p>Congratulations!</p>”;

}

?>

1. The **else** statement

The else statement is used to provide **tailored response no matter the outcome not just when it is rightly guessed.** A review of the previous example, this time offering a response in both cases is hi-lighted next.

a)

<html>

<body>

<?php

$d=date(“D”); // date(“D”) is a PHP pre-defined **date**  function

if($d==“Fri”){

echo “Have a nice weekend”;

echo “See you Monday”;

}

else {

echo “Have a nice day”;

}

?>

</body>

</html>

b)

<?php

$secretNumber = 453;

if($\_POST[‘guess’] == $secretNumber) {

echo “<p>Congratulations!</p>”;

}

else {

echo “<p>sorry try again</p>”;

}

?>

1. The **elseif** statement

Sometimes you will need a means for considering **each possible outcome**. This is accomplished with the *elseif* statement.

**Revising** the previous example: with the *elseif* statement. This time offering message, if the user’s guess is relatively close (say within ten). As seen in the quiz below.

<?php

$secretNumber = 453; //**Expected** outcome

$\_POST[‘guess’] = 461;// A **possible** outcome. **One** of **many**

//If there is a **match**

if($\_POST[‘guess’] == $secretNumber ){

echo “<p>Congratulations!</p>”;

}

//An option for which there is **no match,** but within a **range**

elseif (($\_POST[‘guess’] - $secretNumber) <10){

echo “<p>You’re getting close</p>”;

}

//**Outside** a range

else {

echo “<p>sorry</p>”;

}

?>

1. The **switch** statement

The switch statement can be thought of as a variant of the if-else combination, often when you need to compare a **variable** against a **large number of values.**

Note the presence of the **break** statement at the **conclusion** of each case block. If a break statement is **omitted** from any loop in the program, **all subsequent case blocks will execute until a break statement is located.**

Also, notice the default statement at the end.

Ex 1

<?php

switch ($category) {

case “news”;

echo “<p>what’s happening around the world</p>”;

break;

case “weather”;

echo “<p>Your weekly forecast</p>”;

break;

case “sports”;

echo “<p>Latest sports hi-lights</p>”;

break;

default;

echo “<p>Welcome to my website</p>”;

}

?>

Ex 2

<?php

$destination = "Amsterdam";

echo "Traveling to $destination<br />";

switch ($destination){

case "Las Vegas":

echo "Bring an extra $500";

break;

case "Amsterdam":

echo "Bring an open mind";

break;

case "Egypt":

echo "Bring 15 bottles of SPF 50 Sunscreen";

break;

}

?>

Ex 3

<?php

//It is always best to have a default value to use in case none of the options are found.

//Note there is no use of case in this instance.

$destination = "New York";

echo "Traveling to $destination<br />";

switch ($destination){

case "Las Vegas":

echo "Bring an extra $500";

break;

case "Amsterdam":

echo "Bring an open mind";

break;

case "Egypt":

echo "Bring 15 bottles of SPF 50 Sunscreen";

break;

default:

echo "Bring lots of underwear!";

break;

}

?>

Ex 4

<?php

//It is always best to have a default value to use in case none of the options are found.

//Note there is no use of case in this instance.

$premierClub = "FA – under construction";

switch ($premierClub){

case "Arsenal":

include('./arsenal.php');//includes Arsenal list

break;

case "Manchester U":

include('./manu.php');

break;

case "Chelsea":

include('./chelsea.php');

break;

case "Tottenham Hotspurs":

include('./spurs.php');

break;

default:

echo "No premiere club selection yet!";

break;

}

?>

1. The **while** loopstatement

The while statement specifies a **condition which must be** met before execution of its embedded code is **terminated**.

a)

<html>

<body>

<?php

$i=1;

while($i<3){

echo “The number is”. $i. “<br />”;

$i++;

}

?>

</body>

</html>

This will evaluate to:

**The number is 1**

**The number is 2**

b)

<?php

$count = 1;

while ($count <5) {

printf(“%d squared = %d <br />”,$count, pow($count,2));

$count++;

}

?>

The output is:

**1 squared = 1**

**2 squared = 4**

**3 squared = 9**

**4 squared = 16**

1. The **do while** statement

The do while looping statement is a variant of while but it **verifies** the loop conditional at the **conclusion of the block rather than at the beginning**. The difference between while and do while statements is that the code embedded within a while statement possibly **could never be executed,** whereas the code embedded within a *do,,,while* statement will **always execute at least once.**

a)

<html>

<body>

<?php

$i=1;

do{

echo “The number is” .$i. “<br />”;

$i++;

}while($i<3);

?>

</body>

</html>

This will evaluate to:

**The number is 2**

**The number is 3**

**The number is 4**

b)

<?php

$count = 11;

do {

printf (“%d squared = %d <br />”,$count ,pow($count,2));

}while($count < 10);

?>

The following is the outcome;

1. **squared = 121**
2. The **for** statement

The most basic *for* loops are based on a **counter**. You set the **beginning** value for the counter, set the **ending** value, and set how the counter is **incremented** or **decremented**

In the example below the variable $i can be used in a block of statements that is repeating. The following loop statement will display “Hello World” three times.

<?php

for($i=1;$i<=3;$i++) {

echo “$i. Hello World <br />”;

}

?>

The output from this statement is

1. **Hello World**
2. **Hello World**
3. **Hello World**

For loops are particularly useful to **loop** through an **array**. Suppose you have an array of customer names and want to display them all. You can do this easily in the loop;

a)

<?php

for($i=0;$i<3;$i++) {

echo $i. “ <br />”;

}

?>

This will evaluate to:

**0**

**1**

**2**

b)

<?php

for($i=10;$i>0;$i=$i-3) {

echo $i. “ <br />”;

}

?>

This will evaluate to:

**10**

**7**

**4**

**1**

c)

<?php

for($i=0;$i<100;$i++) {

echo “$customerName[$i] <br />”;

}

?>

Supposing you want to know the number of customers in the array list, you can use the *sizeof ()*  as part of the construct.

<?php

for($i=0;$i<sizeof($customerName);$i++) {

echo “$customerName[$i] ”;

}

?>

1. The **continue statement**

The continue statement causes the execution of the current loop iteration to **end** and **commence at the beginning of the next iteration**. Execution of the following *while* body will recommence if the $username[$x] is found to have the value  *missing.*

<?ph

$username = array (“grace”,”doris”,”gary”,”nate”,”missing”,”tom”);

for(x=0; x<count($username); x++){

if($username[$x] == “missing”) continue;

printf (“Staff member: %s <br />”, $username[$x]);

}

?>

This results in the following output

**Staff member : grace**

**Staff member : doris**

**Staff member : gary**

**Staff member : nate**

**Staff member : tom**