**PHP**

**Part 1**

**Introduction**

PHP tags are normally put somewhere within the HTML body tags, as PHP is most often used to create content displayed in the web browser.

<?PHP

?>

PHP can send data to the web browser using built-in functions; *echo* and *print*.

E.g. echo ‘hello world’;

Print “hello world”;

PHP is case *in*sensitive, so *echo* & *ECHO* will both work.

You can, and often will, use echo and print to send HTML code to the web browser.

E.g. echo ‘<p>hello, <b>world</b>!</p>’;

Like with HTML, PHP is white space insensitive, meaning you can space out your code to look and be more legible and this won’t be transferred on to the page.

**Writing comments**

|  |  |
| --- | --- |
| Single line comments | # or // |
| Multiline comments | /\* *comment here* \*/ |

One of the first comments each script should contain is an introductory block that lists: creation date, modification date, creator, creators contact information, purpose of script and so on…

E.g. #Script 1.4 – comments.PHP

#Created July 6, 2017

#Created by Ali Issaee

#This script does nothing much

It is common practice to put //end of PHP code before closing tag ?>

\*\**you can never write enough comments\*\**

**Variables**

|  |  |
| --- | --- |
| Boolean | Scalar |
| Integer |
| Floating point (decimals) |
| String |
| Arrays | Non-scalar |
| Objects (multivalued) |
| Resources |  |
| Null | No value |

Variable syntax

$name =

The syntax can contain a combination of numbers, letters and underscores. However, the first character cannot be a number.

Variables are case sensitive so $name and $Name are different!

Variables can be printed without quotes and within double quotes.

E.g. print $name

Print “$name”

Variables cannot be printed using single quotes!

Pre-defined variables – built-in variables which the script already has when script begins its execution:

$file = $\_SERVER[‘SCRIPT\_FILENAME’];

$user = $\_SERVER[‘HTTP\_USER\_AGENT’];

$server = $\_SERVER[‘SERVER\_SOFTWARE’];

**Strings**

A string is a quoted chunk of characters

E.g. $first\_name = “Ali Issaee”;

Echo $first\_name;

Concatenating strings (linking them)

Using (.) period

E.g. $city = “Hull”;

$county = “Yorkshire”;

$address = $city.’, ‘.$county;

Some string functions

|  |  |
| --- | --- |
| Strtolower() | Makes lower case |
| Strtoupper() | Makes upper case |
| Ucfirst() | Capitalise first letter |
| Ucwords | Capitalise first letter of every word |

The PHP manual ([www.php.net/manual](http://www.php.net/manual)) lists every function and feature of the language

**Numbers**

Arithmetic operators are the same as JS: +, -, /, \*, %, ++, --

Some number functions

|  |  |
| --- | --- |
| Round() | Rounds number to the nearest integer |
| Number\_format() | Turns number in to more commonly written version |

E.g. $n = 20943

$n = number\_format($n, 2); //20,943.00

E.g. 2 $quantity = 30;

$price = 119.95;

$taxrate = .05;

//calculate total

$total = $quantity\*$price;

$total = $total+($total\*$taxrate);

//format total

$total = number\_format($total, 2);

**Introducing constants**

Constants, like variables, are used to temporarily store a value. To create a constant, you use the define() function instead of the ‘=’ operator.

E.g. define(‘NAME’, ‘value’);

Constants are named using all capitals as above example shows.

A constant can only be assigned a scalar value, like a string or a number. And, unlike variables, a constants value cannot be changed.

To print a constant you need to use the concatenation operator (.).

E.g. echo ‘hello, ‘.NAME;

PHP runs with many pre-defined constants like: PHP\_VERSION, PHP\_OS etc…

E.g. define(‘TODAY’, ‘June 08, 2017’);

Echo ‘<p>Today is ‘.TODAY.’<br />This server is running version <b>’.PHP\_VERSION.’</b> of PHP on the <b>’.PHP\_OS.’</b> operating system.</p>’;

**Single vs double quotations marks**

With echo and print, or when assigning values to string, you can use either single or double quotes.

But there is a key difference between the 2 types and when you should use them.

In PHP, values within single quotations will be treated literally, whereas those in double quotations interpreted.

As valid HTML often includes a lot of double quotation marks, it is often easiest to use single quotation marks when printing HTML with PHP.

Using single quotes is theoretically faster as PHP will attempt to find variables if written in double quotation marks.

**Basic debugging steps**

* Make sure you’re always running PHP scripts through a URL.
* Know what version of PHP you are using
* Make sure display-errors is on – you can confirm this by executing phpinfo();
* Check the HTML source code
* Trust the error message displayed
* Take a break

**Part 2**

**Creating an HTML form**

An HTML form is created using the form tags and various elements for taking inputs.

e.g. <form action=”script.php” method=”post”>

</form>

In terms of PHP, the most important attribute of your form tag is ‘action’, which dictates to which page the form data will be sent. The second attribute ‘method’ has its own issues, but ‘POST’ is the value you will use more frequently.

What kinds of inputs your forms have, makes little difference to the PHP script handling it. You should, however, pay attention to the names you give your form inputs, as they will be a critical importance when it comes to your PHP code.

**Handling an HTML form**

Now that the HTML form has been created, it’s time to write a PHP script to handle it (i.e. the script does something with the data it receives).

Whatever the user types in to the input will be accessible via a PHP variable name: $\_REQUEST[‘name’]; - where ‘name’ is the name of the form element.

You can create short-hands of these, e.g. $email = $\_REQUEST[‘email’];

Handling an HTML form example:

$name = $\_REQUEST[‘name’];

$email = $\_REQUEST[‘email’];

$comments = $\_REQUEST[‘comments’];

Print “<p>Thank you, $name, for the following comments:<br /><tt>$comments</tt></p>

<p>We will email you at <i>$email</i>.</p>”;

\*\*Because the PHP script must be run through a URL, the form must also be run through a URL. Otherwise when you go to submit the form, you’ll see PHP code instead of the proper result\*\*

$\_REQUEST is a special variable type, known as a superglobal. It stores all the data sent to a PHP page through either the GET or POST method, as well as data accessible in cookies.

**Conditionals & Operators**

PHP’s three primary terms for creating conditions are IF, ELSE and ELSEIF. Every condition begins with an IF clause, similar to javascript.

Comparative operators

|  |  |
| --- | --- |
| == | Equal to |
| != | Is not equal to |
| < | Less than |
| > | Greater than |
| <= | Less than or equal to |
| >= | Greater than or equal to |

Logical operators

|  |  |
| --- | --- |
| ! | Not |
| && | And |
| AND | And |
| || | Or |
| OR | Or |
| XOR | And not |

Function – isset() = this function checks if a variable is ‘set’ (has a value other than NULL).

e.g. if(isset($\_REQUEST[‘gender’])){

$gender=$\_REQUEST[‘gender’];

}else{

$gender=NULL;

}

The above is a simple and effective way to validate a form input.

If the user checks either gender radio button, then $\_REQUEST[‘gender’] will have a value, meaning that the condition isset(‘$\_REQUEST[‘gender’]) is true. In such case, the shorthand version of this variable - $gender – is assigned the value of $\_REQUEST[‘gender’].

E.g2. if($gender==”M”){

Echo “<p><b>good day, sir!</b></p>;

}elseif($gender==”F”){

Echo “<p><b>Good day, madam</b></p>”;

}else{

Echo “<p>You forgot to enter your gender!</p>;

}

PHP has another type of conditional called the SWITCH, best used in place of a long IF-ELSEIF-ELSE conditional. The syntax of switch is the same as javascripts.

**Validating form data**

The critical concept related to handling HTML forms is that of validating form data. In terms of both error management and security, you should absolutely never trust the data being submitted by a HTML form.

Validating form data requires the use of conditionals and any number of functions, operators and expressions.

One standard function to be used is isset(), which tests if a variable has a value (including ), FALSE, or an empty string, but not NULL).

To check that a user typed something into textual elements, you can use the empty() function, which checks if a variable has an empty value i.e. 0, NULL, FALSE or an empty string.

Form validation aims:

1. Check whether a field was typed in or selected
2. Ensure submitted data is of the right type (like an email address)

To test if a submitted name is a number, use the is\_numeric() function.

E.g.

<?php

//validate name, email, comments and gender

If(!empty($\_REQUEST[‘name’])) {

$name=$\_REQUEST[‘name’];

}else{

$name=NULL;

Echo ‘<p>You forgot to enter your name<p>’;

}

//do the same for email and comments

//gender validation

If(isset($\_REQUEST[‘gender’])) {

$gender=$\_REQUEST[‘gender’];

If($gender==”M”) {

Echo ‘<p>good day, sir!</p>;

Elseif($gender==”F”) {

Echo ‘<p>good day, ms</p>;

}else{

$gender=NULL;

Echo ‘<p>Gender should be either “M” or “F”</p>’;

}

}else{

$gender=NULL;

Echo ‘<p>you forgot to select your gender!</p>’;

}

//if everything is ok, print below message

If($name && $email && $comments && $gender) {

Echo “<p> thank you, $name, for your comments:<br />$comments</p>

<p>We will contact you at $email</p>”;

}else{

Echo ‘<p>please refill the form correctly</p>’;

}

**Part 3**

**Introducing arrays**

An array can hold multiple, separate pieces of information, like a list with each value being a string, a number or another array.

PHP supports 2 types of arrays:

1. Indexed
   * 1. This uses numbers as the key (like JS)
        + 1. 0 crack
          2. 1 whisk
          3. 2 cook
2. Associative
   * 1. This uses strings as keys
        + 1. MD Maryland
          2. IL Illinois

Creating arrays

$band[] = ‘d12’;

$band[] = ‘oasis’;

OR

$band[‘fan’]=’Ali’;

OR

$band=array(‘d12’, ‘oasis’);

OR

$band = array (

‘fan’ => ‘Ali’,

Fan\_two’ => ‘Lauren’

);

How to access arrays

e.g. $band[0]; OR $band[‘fan’];

Accessing entire arrays

Use the foreach loop:

Foreach($array as $value) {

//do something

}

To access keys and values use:

Foreach($arrays as $key => $value) {

Echo “the value at $key is $value.”;

}

\*\*If the array uses strings for keys, the quotes used to surround the key will muddle the syntax, so instead of:

Echo “my name is $people[‘me’]”;

USE: echo “my name is {$people[‘me’]};

E.g.

$months = array(1=> ‘January’, ‘February’, ‘March’, ‘April’, ‘May’, etc…);

$days = range(1,31);

$years = range(1950,2017);

Echo ‘<select>’;

Foreach ($months as $key => $value) {

Echo “<option value=\”$key\”>$value</option>”;

}

Echo ‘</select>’;

Echo ‘<select>’;

Foreach($days as $value) {

Echo “<option value=\”$value\”>$value</option>”;

}

Echo ‘</select>’;

Echo ‘<select>’;

Foreach ($years as $value) {

Echo “<option value = \”$value\”>$value</option>”;

}

Echo ‘</select>’;

\*\*use the range() function to easily make an array of number\*\*

\*\*To determine the number of elements in an array, use count(): $name=count($array);\*\*

Multidimensional arrays

An array consisting of other arrays create multidimensional arrays

E.g.

$primes = array(2,3,5);

$sphenic = array(3-,42,66);

$number = array(‘primes’ =>$primes, ‘spenic’=>$sphenic);

Echo “{$numbers[‘sphenic’][0]}”;

E.g.2

$mexico = array(

‘YU’ => ‘Yucatan’,

‘BC’ => ‘Baja California’

);

$US = array(

‘MD’ => ‘Maryland’,

‘IL’ => ‘Illinois’

);

$n\_america = array(

‘Mexico’ => $mexico,

‘United States’ => $US

);

Foreach($n\_america as $country => $list) {

Echo “<h2>$country</h2>”;

Foreach($list as $a => $z) {

Echo “<li>$a - $z</li>”;

}

}

Arrays and strings

Because arrays and strings are so commonly used together, PHP has 2 functions for converting between the two:

1. Explode() – needs 2 variables
2. Implode() – needs 2 variables

E.g.

//string

$s1 = “mon-tue-wed-thurs-fri”;

//make an array

$a1 = explode(‘-‘, $s1);

//return to a string, but using a comma instead of a hyphen

$s2 = explode(‘, ‘, $a1);

Sorting arrays

|  |  |
| --- | --- |
| Sort() | Sort by value, discarding the original keys |
| Assort() | Sort by value, maintaining original keys |
| Ksort | Sort by key |
| *Rsort(), arsort(), krsort()* | *For reversing order of above* |

**For and while loops**

The while loop:

While(condition) {

//do something

}

The for loop

For(initial expression; condition; closing expression) {

//do something

}

*Basically the same as javascript*

e.g.

echo “<select>”;

for($year=1950; $year<=2017;$year++) {

echo “<option>$year</option>”;

}

Echo “</select>”;

**Part 4**

**Including multiple files**

PHP has 4 functions for incorporating external files: include(), include\_once(), require(), require\_once(). E.g. include(‘filename.php’); require(‘path/to/filename.html’);

Usually any of these functions has the end result of taking all the content of the included file and dropping it in the parent script (the one calling the function) at the juncture.

The include() and require() functions are exactly the same when working but behave differently when they fail:

Include() – script continues to run and warning is printed to web browser

Require() – an error is printed and the script is halted

It’s best not to use the \_once() functions as it requires extra work from the PHP module.

To include multiple files

\*look at p78\*

1. Design a HTML page and mark where any page specific content goes in the body
   * 1. E.g. <div id=”content”>

<h1>Content</h2>

<p>Random para</p>

</div>

1. Copy the html code from the very top, to below the opening div tag for the specific content, but do not include the closing tag.
2. Change the page’s title line to read: <title><?php echo $page\_title; ?></title>
3. Cut the above code (2) and save as ‘header.html’/
4. Copy the rest of the original code and save as ‘footer.html’
5. Create a new php file ‘index.php’
   * 1. <?php

$page\_title = “welcome to this side”;

Include(‘header.html’);

?>

<!—include content from (1) here 🡪

<?php

Include(‘footer.html’);

?>

**Handling HTML forms revisited**

There are advantages for putting the form and the PHP handle file in to one script. To have one page both display and handle a form, a conditional must check which action (display or handle) should be taken.

IF($\_SERVER[‘REQUEST\_METHOD’]==”POST”) {

//handle the form

}else{

//display the form

}

Example

<?php

$page\_title=”the cost calculator”;

Include(‘includes/header.html);

//check the form submission

IF($\_SERVER[‘REQUEST\_METHOD’]==”POST”) {

//form validation

If(isset($\_POST[‘distance’], $\_POST[‘gallon\_price’], $\_POST[‘efficiency’]) && is\_numeric($\_POST[‘distance’]) && is\_numeric($\_POST[‘gallon\_price’]) && is\_numeric($\_POST[‘efficiency’]) {

//calculate the results

$gallons = $\_POST[‘distance’]/$\_POST[‘efficiency’];

$dollars = $gallons \* $\_POST[‘gallon\_price’];

$hours = $\_POST[‘distance’]/65;

//print results

Echo ‘<h1>Total estimated cost</h1>

<p>The total cost of driving ‘ .$\_POST[’distance’] .’miles, averaging ‘ .$\_POST[‘efficiency’] .’miles per gallon, and paying an average of $’ .$\_POST[‘gallon\_price’] .’per gallon, is $’ .number\_format($dollars, 2) .’. If you drive at an average of 65 miles per hour, the trip will take approximately ‘ .number\_format($hours, 2) .’hours.</p>’;

}else{

Echo “<h1>Error</h1>”;

}

}

?>

//HTML

<h1>Calculator</h1>

<form action=”calculator.php” method=”post”>

<p>Distance (miles): <input type=”text” name=”distance” /></p>

<p>Avg. price per gallon:<span>

<input type=”radio” name=”gallon\_price” value=”3.00” />3.00

<input type=”radio” name=”gallon\_price” value=”3.50” />3.50

<input type=”radio” name=”gallon\_price” value=”4.00” />4.00

</span></p>

<p>Fuel efficiency: <select name=”efficiency”>

<option value=”10”>Terrible</option>

<option value=”50”>Great</option>

</select></p>

<p><input type=”submit” name=”submit” value=”calculate” /> </p>

</form>

<?php include(‘includes/footer.html’); ?>

**Making sticky forms**

A sticky form is simply a standard html form that remembers how you filled it out. To pre-set what’s entered in to a text input, use its ‘value’ attribute.

**Textbox:** <input type=”text” name=”city” value=”<?php echo $city; ?>” />

**Radio:** <input type=”radio” name=”gender” value=”F”<?php if($gender==’F’) {

Echo ‘checked=”checked”’; } ?> />

**Textarea:** <textarea name=”comments”>

<?php echo $comments; ?>

</textarea>

**Pulldown menu:** echo ‘<select name=”year”>;

For($y =2011; $y<=2021; $y++) {

Echo “<option value=\”$y\””;

If($year==$y) {

Echo ‘selected=”selected”’;

}

Echo “> $y</option>

}

Echo ‘</select>’;

\*\*it’s best not to refer to variables unless they exist. Use conditionals to check a variable is set before printing it

e.g. <?php if(isset($\_POST[‘name’])) {

echo “$\_POST[‘name’];

} ?>

**Part 5**

**Creating your own functions**

Very similar to javascript.

Function name() {

//do something

}

The most common reasons to create your own functions are:

1. To associate repeated code with one function to call
2. To separate out sensitive or complicated process from other code
3. To make common code bits easier to re-use

E.g. //this function creates an ad

function create\_ad() {

Echo “<p>This is an ad</p>”;

} // end of function definition

Create\_ad();

\*\*Although not required, it is conventional to place a function definition near the very top of a script or in a separate file\*\*

Creating a function that takes arguments

A function can take any number of arguments, but the order in which you list them is critical. To allow for arguments, add variables to a functions definition:

Function print\_hello($first, $last) {

//function code

}

Print\_hello(‘Ali’, ‘Issaee’);

$surname = “Foster”;

Print\_hello(‘Lauren’, $surname);

Example (using the cost calculator code above):

//create the function at the top of the script after the opening php tag

Function create\_gallon\_radio($value) [

Echo ‘<input type=”radio” name=”gallon\_price” value = “ ‘ .$value . ‘”’;

If(isset($\_POST[‘gallon\_price’]) && ($\_POST[‘gallon\_price’] == $value)) {

Echo ‘checked=”checked”’;

}

Echo “ /> $value “;

}

// replace the html radio button code with:

<?php

Create\_gallon\_radio(‘3.00’);

Create\_gallon\_radio(‘3.50’);

Create\_gallon\_radio(‘4.00’);

?>

Setting default argument values

Another variant on defining your own functions is to pre-set an arguments value. To do so, assign the argument a value in the functions definition:

Function greet($name, $msg=’Hello’) {

Echo ”$msg, $name”;

}

The end result of setting a default argument value is that that particular argument becomes optional when calling the function. If a value is passed to it, the passed value is used, otherwise the default value is used.

\*\*The required argument must always be listed first\*\*

E.g. greet($surname, $message);

OR – greet(‘Zoe’);

OR – greet(‘Sam’, ‘Good evening’);

To set default argument values example:

*Using the previous calculator.php example. Instead of having a function for just gallon price, you can have it for all radio buttons*

Function create\_radio($value, $name=”gallon\_price”) {

Echo ‘<input type=”radio” name = “’ .$name .’” value = “’ .$value .’”’;

If(isset($\_POST[$name]) && ($\_POST[$name]==$value)) {

Echo ‘checked=”checked”’;

}

Echo “ /> $value “;

}

//change the radio button code to match the function name of the above function

\*\*In PHP, square brackets are used to indicate a functions optional parameters\*\*

**Returning values from a function**

To have a function return a value, use the return statement.

E.g. function find\_sign($month, $day) {

//function code

Return $sign;

}

Example using calculator.php

Function calculate\_trip\_cost($miles, $mpg, $ppg) {

$gallons = $miles/$mpg;

$dollars = $gallons \* $ppg;

Return number\_format($dollars, 2);

}

//replace the 2 lines of code that calculate the cost with:

$cost = calculate\_trip\_cost ($\_POST[‘distance’], $\_POST[‘efficiency’], $\_POST[‘gallon\_price’]);

//adjust the echo statement to use the new variable $cost

**Test**

Part 1

1. Name the 2 bult-in functions used to print data to the web browser and give example
2. List 2 ways to write single line comments, and 1 way to write multiline comments
3. Name the 8 data types in PHP
4. Give an example of how to write a variable in PHP using correct syntax
5. Name 3 pre-defined variables
6. How do you link numerous strings/variables together? Give example
7. List 4 string functions and define what they do
8. List 2 number functions and define what they do
9. What is a constant and how do you write one?
10. List 2 pre-defined constants
11. Single vs double quotes – what are the differences?

Part 2

1. Create a basic HTML form and send to PHP file using the 2 form attributes
2. Create short-hands for each variable in the form
3. Create a basic IF-ELSEIF-ELSE condition
4. List the 6 comparative operators
5. List the 6 logical operators
6. What does the function isset() do?
7. What are the 2 aims for form validation?
8. What does the empty() function do?
9. What does the is\_numeric() function do?
10. Give an example of form validation where all criteria need to be met before correct message is show. If one isn’t filled, specify which one isn’t.

Part 3

1. PHP supports 2 types of arrays, what are they?
2. Create an array using the 2 different types
3. How do you echo an individual item from an array?
4. How do you access entire arrays? Give example
5. Create an array to cover a range of numbers i.e. 1-100.
6. Create a multidimensional array to do with countries in the UK and capital cities
7. How do you convert arrays in to strings and vice versa?
8. List 6 methods to sort arrays
9. Create a for loop which does the same as the range() function.

Part 4

1. What are the 4 functions for incorporating multiple files?
2. What is the difference between the 4 functions?
3. Use multiple files in an example to create a header, footer and ‘main content’ page.
4. Create a HTML form and create some PHP code to handle the form, both on one script. Make sure to use different inputs; text, radio, select, submit
5. Edit the above (4) document to include ‘sticky forms’ for each input

Part 5

1. Create a basic function to print some text on the page
2. Do the same as above but using arguments
3. In calculator.php, create some code, so that a user defined function creates the price per gallon radio buttons
4. What is meant to pre-set an argument value? Give example
5. Similar to *question 3*, but instead of calling functions for just gallon\_price, make the code so it can create code for all radio buttons with different names
6. What are square brackets used for in php functions?
7. Give a basic example of the return statement
8. Update the calculator.php file, so that is uses a function to determine the cost of the trip (make sure to update the echo statement)
9. Answer the review and pursue questions on p110.