Contents

Introduction

This document is a basic guide to PHP.

Conventions

```
This type of text indicates a code example / listing. You may wish to type
this code out or copy & paste.
```

Italic Text indicates code elements inline with a sentence. E.g.

The *<h1>* element

This denotes an exercise

What you need for this tutorial

- Access to the University IT Services network
- Development application, anything from Notepad® to Dreamweaver®.
- Basic knowledge of HTML
- General understanding of programming concepts: Variables, Methods and branching structures (E.g. IF statements)
- Access to the university internal PHP server (mapped to T:)

The Basics

A PHP scripting block is as follows

```
<?php
//PHP script goes here...
?>
```

Writing to The Browser - echo

To output items to the browser, you can use *echo* or *print*. The examples in this tutorial will use *echo*. The code below will ‘echo’ the string output "<p>Hello World</p>" to the requesting agent. A browser will interpret this as HTML.

```
<?php
echo "<p>Hello World</p>";
?>
```

You can use the echo as a debugging tool to show where your program gets to before it crashes, or to see what results you are getting.

Comments

Comments within PHP can either be a single line (//) or a block (/*..*/).

```
<?php

//This is a single line comment

/*
This is
a comment
block
*/

?>
```

Code Blocks

Blocks of code are denoted by braces '{' and '}'. These are used to associate several lines of code together. For example:

```
if($isChecked) {
  $value = 'Your account was saved.';
  echo $value;
}
```

If the value of the *if* statement is true then all lines within the braces will be executed (more on the *if* statement below).

Variables and Constants

All variables in PHP start with a $ but in PHP you do not have to declare what type a variable is.

```
<?php
  $myvar="Hello World!";
  $x=42;
?>
```

Variable Names

Variable names are case sensitive. So *$author* and *$Author* are different variables

For a variable name to be valid it must begin with a '$' followed by a letter or underscore and then any number of letters, underscores or numbers.

String Variables

To create a string variable you simply assign a string value, PHP will then treat it as a string.

```
<?php
$myTxt = 'Hello';
$myName = 'Fred';
?>
```

Single Quoted

The example above shows the simplest way to specify a string. Using a backlash as an escape character will only work with a backslash itself and a single quote.

```
<?php
$myTxt = 'You\’re here'; //results in You’re here
?>
```

Double Quoted

Specifying a string using double quotes allows the use of more escape sequences[1]. Variable names will also be expanded so that you will not need to concatenate variables into a string.

```
<?php
$myName = "Fred";
$myTxt = "Welcome $myName";
?>
```

Heredoc and Nowdoc

Another way to specify a string is to use 'heredoc' or 'nowdoc'.

[1] http://www.php.net/manual/en/language.types.string.php#language.types.string.syntax.double

Heredoc text behaves in the same way as a string specified using double quotes. Nowdoc behaves in the same way as a string specified using single quotes.

The code below shows an example of heredoc use.

```
<?php
$name = 'Phil';
echo <<<EOT
My name is "$name".
This should not print a capital 'A': \x41
EOT;
?>
```

The syntax is very similar for heredoc and nowdoc. They begin with '<<<' followed by an identifier. The identifier ('EOT' above) can be anything but must follow the same rules as any other PHP label (alphanumeric or underscores and beginning with either a letter or an underscore). The same identifier is then used again to close. The final identifier must begin in the first column of the line.

A heredoc identifier can optional be enclosed by double quotes. A nowdoc identifier is distinguished because it must be enclosed in single quotes as in the code example below.

```
<?phpecho <<<'EOT'
My name is Phil.
I teach at Staffordshire University.
EOT;
?>
```

Concatenation

To concatenate two string variable together you can use a period '.'

```
<?php
$myTxt = "Hello";
$myName = "Fred";
?>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
<meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
<title>Untitled Document</title>
</head>
<body>
<p>
<?php
echo $myTxt . " " . $myName;
?>
</p>
</body>
</html>
```

Useful String Functions

There are several other built in functions you can use with strings. Some of these are shown below

Code	**Definition**	**Example**
strlen(string)	Returns the length of a string	<?php echo strlen("Hello world!"); ?>
strpos(string, find)	Searches for the position of a given phrase in a string. If there is no match false is returned. If there is a match, the position is returned	<?php echo strpos("Hello world!","world"); ?>
substr(string, start, length)	Returns a sub string based on the first character being 0	<?php echo substr("Hello world!",6); ?>
trim(string)	Trims whitespace from a string	<?php $str = " Hello World! "; echo "Without trim: " . $str; echo "With trim: " . trim($str); ?>
strtolower(string)	Converts a string to lower case	<?php echo strtolower("Hello WORLD."); ?>
strtoupper(string)	Converts a string to upper case	<?php echo strtoupper("Hello WORLD."); ?>
str_replace(find,replace,string,count)	Replaces some characters with some other characters in a string	<?php echo str_replace("world","Peter","Hello world!"); ?>

Constants

PHP has a number of built in constants for commonly used values. You can also declare your own constant.

```
define(‘MILES_TO_KM’, 1.6);
$distanceInKM = 25 * MILES_TO_KM;
```

One of the built in constants is the value of PI (@3.14159) and is M_PI

An example is shown below

```
<?php
$radius = 5;
$area = M_PI * $radius * $radius;
?>
```

Operators

As seen in the previous example there are various operators available in PHP. These are shown in the table below

Arithmetic Operators

Operator	Description	Example	Result
+	Addition	$x=2 $x+2	4
-	Subtraction	$x=2 5-$x	3
*	Multiplication	$x=4 $x*5	20
/	Division	15/5 5/2	3 2.5
%	Modulus (division remainder)	5%2 10%8 10%2	1 2 0
++	Increment	$x=5 $x++ ++$x	x=6
--	Decrement	$x=5 $x— --$x	x=4

Assignment Operators

Operator	Example	Is The Same As
=	$x=$y	$x=$y

+=	$x += $y	$x = $x + $y
-=	$x -= $y	$x = $x - $y
*=	$x *= $y	$x = $x * $y
/=	$x /= $y	$x = $x / $y
.=	$x .= $y	$x = $x . $y
%=	$x %= $y	$x = $x % $y

Comparison Operators

Operator	**Description**	**Example**
= =	is equal to	5 == 8 returns false
!=	is not equal	5 != 8 returns true
<>	is not equal	5 <> 8 returns true
>	is greater than	5 > 8 returns false
<	is less than	5 < 8 returns true
>=	is greater than or equal to	5 >= 8 returns false
<=	is less than or equal to	5 <= 8 returns true

Logical Operators

Operator	**Description**	**Example**
&&	and	$x = 6 $y = 3 ($x < 10 && $y > 1) returns true
||	or	$x = 6

		$y = 3 ($x == 5 || $y == 5) returns false
!	not	$x = 6 $y = 3 !($x == $y) returns true

Simple Programming Concepts

If statement

The *if* statement executes statements depending on the outcome of a test.

```
if … then
      'first bit code
else
      'Second bit code
end if
```

An example is shown below.

```
<?php
$d=date("D");
if ($d=="Fri")
  $output = "It's Friday..door to the weekend";
else
  $output = "Not Friday yet";
?>
```

If you have more than one line of code to be executed when the *if* is triggered, you need to surround the code in {…} as shown below.

```
<?php
$d=date("D");
if ($d=="Fri") {
      $MyName = "Fred, ";
      $output =  $MyName . "Its Friday..door to the weekend";
} else {
      $MyName = "Lisa, ";
      $output = $MyName . "Not Friday yet";
}
?>
```

You can also use an 'if..elseif..else statement.

```
<?php
$d=date("D");
if ($d=="Fri")
  $output = "Have a nice weekend!";
elseif ($d=="Sun")
 $output = "Have a nice Sunday!";
else
  $output = "Have a nice day!";
?>
```

Switch statement

Switch can be used in place of *if* to help improve the clarity of the code and, in some cases, offers the potential for faster execution.

```
<?php
$d=date("D");
switch ($d) {
  case "Mon":
    $output = "Monday Monday!...the start of the week";
    break;
  case "Tue":
    $output = "Tuesday!...";
    break;
  case "Wed":
    $output = "Midweek...";
    break;
  case "Thu":
    $output = "Soon the weekend!";
    break;
  case "Fri":
    $output = "The weekend starts here!";
    break;
  default:
    $output = "Weekend!!!..Lie in!";
}
?>
```

While loop

The *While* loop executes statements while a condition holds true

```
<?php
//Matching pair of dice
$dice1 = rand(1,6);
$dice2 = rand(1,6);
$output = "<p>First dice: " . $dice1 . " and second dice: " . $dice2 .
"</p>";
$count = 1;

while($dice1 <> $dice2) {
  $dice1 = rand(1,6);
      $dice2 = rand(1,6);
$output = $output . "<p>First dice: " . $dice1 . " and second dice: " .
$dice2 . "</p>";
$count = $count + 1;
}

$output = $output . "It took you ". $count . " throws to throw a double. Hit
refresh to try again";
?>
```

Try the code above and make sure you can understand what it is doing.

Do..while

The do..while loop will always do the block of code once, and checks the condition at the end

```
<?php
$i=1;
do {
  $i++;
  echo "The number is " . $i . "<br />";
}
while ($i<=5);
?>
```

For loop

This has the syntax

```
for (init; condition; increment) {
  code to be executed;
}
```

For example

```
<?php
$output = "Hello Fred!";
for ($i=1; $i<=5; $i++) {
  $output = $output . "<p>The number is " . $i . "</p>";
}
?>
```

Arrays

Arrays provide the facility to group related data into a single logical unit. Arrays in PHP have a zero based index. This means that the first item in the array is *$myArray(0)*.

How to create arrays

Arrays can be created either by:

```
$cars=array("Saab","Volvo","BMW","Toyota");
```

OR

```
$cars[0]="Saab";
$cars[1]="Volvo";
$cars[2]="BMW";
$cars[3]="Toyota";
```

You can also create arrays using association as shown below

```
$ages['Peter'] = "32";
$ages['Fred'] = "30";
$ages['Joe'] = "34";
```

Accessing Values From An Array

Items are accessed using the name of the array, followed by the index of the required item in brackets. The first element of the array has 0 as its index if numeric.

```
<?php
$cars[0]="Saab";
$cars[1]="Volvo";
$cars[2]="BMW";
$cars[3]="Toyota";

echo $cars[0] . " and " . $cars[1] . " are Swedish cars.";

$ages['Peter'] = "32";
$ages['Quagmire'] = "30";
$ages['Joe'] = "34";

echo "Peter is " . $ages['Peter'] . " years old.";
?>
```

ForEach Loop

The *foreach* Loop helps you cycle through all the elements of an array.

```
<?php
$x=array("one","two","three");
foreach ($x as $value) {
  echo “<p>” . $value . "</p>";
}
?>
```

Array Functions

A full list of array functions can be found at http://www.php.net/manual/en/ref.array.php

Forms

Forms are used to get user data that is entered on the client side to the server so that it can be processed.

Forms are part of HTML. If you are unfamiliar with forms and form elements you should revise HTML form elements before you proceed. This section will deal with the form only insofar as it has a bearing on the use of PHP.

Form Element Attributes

The *<form>* element contains a number of attributes. Two of these attributes *action* and *method* are important in instructing the browser 'where' and 'how' the user's data is to be sent.

The *action* attribute provides the 'where' in the form of a path. This can be an absolute path or, as in the examples below, a relative path. Where a relative path is used it is relative to the original location of the web page that contains the form. In the example "a.php" may be a file within the same folder as the current web page – it may also refer to the web page itself (if the web page is called "a.php").

The *method* attribute provides the 'HTTP' method by which the data is to be sent to the server. This can either be HTTP GET

```
<form action="a.php" method="get"></form>
```

or HTTP POST.

```
<form action="a.php" method="post"></form>
```

Using GET will send the form data as part of the URL. You will see the data appended to the end of the web address using a question mark and one or more name/value pairs each separated by an ampersand ('&'). The data is therefore clearly visible and can be severely limited in the amount of data that can be sent, though this depends upon the browser and the server rather than a limitation imposed by HTTP.

Using POST will send the form data as a header. Generally this means that larger amounts of data can be sent using POST than can be sent by using GET. It is also slightly more difficult to view the data that is being sent because it would be necessary to view the request header information.

Form Data and PHP

You can access the form data using PHP, whether it is sent using GET or POST, in similar ways. If the *method* used is POST then a 'built-in' array called *$_POST* is automatically populated with the form data.

```
$name = $_POST['fname'];
```

The same is true for the 'built-in' *$_GET* array.

```
$name = $_GET['fname'];
```

Automatically populating these arrays relies on the use of the *name* attribute in the form field elements from which the data is retrieved. The value of the *name* attribute in the HTML is used as the 'key' to access the value in the array.

In the code above, 'fname' would refer to the user data taken from an <*input*> element with an attribute *name="fname"*.

This is shown more clearly in the following examples.

PHP and POST

The code below shows an expanded example of the use of *$_POST*.

"TestForm.html" (the requesting HTML page)

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
  <meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
  <title>POST Example</title>
</head>
<body>
  <form action="Welcome.php" method="post">
    <fieldset>
      <label for="fname">Name: </label>
      <input type="text" name="fname" id="fname" />
      <label for="age">Age: </label>
      <input type="text" name="age" id="age" />
      <input type="submit" name="submit" id="submit" />
    </fieldset>
  </form>
</body>
</html>
```

"Welcome.php" (the processing PHP page)

```
<?php
$name = $_POST['fname'];
$age = $_POST['age'];
$output = "<p>Welcome $name. You are $age years old.</p>";
?>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
  <meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
  <title>POST Example</title>
</head>
<body>

<?php echo $output; ?>

</body>
```

```
</html>
```

PHP and GET

The code below shows an expanded example of the use of *$_GET*. Except for the use of the HTTP GET method the code is exactly the same as the previous code.

"TestForm.html"

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
  <meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
  <title>GET Example</title>
</head>
<body>
  <form action="Welcome.php" method="get">
    <fieldset>
      <label for="fname">Name: </label>
      <input type="text" name="fname" id="fname" />
      <label for="age">Age: </label>
      <input type="text" name="age" id="age" />
      <input type="submit" name="submit" id="submit" />
    </fieldset>
  </form>
</body>
</html>
```

"Welcome.php"

```
<?php
$name = $_GET['fname'];
$age = $_GET['age'];
$output = "<p>Welcome $name. You are $age years old.</p>";
?>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
  <meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
  <title>POST Example</title>
</head>
<body>

<?php echo $output; ?>

</body>
</html>
```