# Programming the Internet—Lecture Notes

## Week 5: PHP and server-side programming

The programming technologies we have looked at so far, HTML, CSS and JavaScript, can all be tested in a browser without the need to upload them to a Web server. Additionally, because JavaScript, in the particular way we have been using it, runs in a browser rather than on a Web server, it is referred to as ‘client-side’ programming.

We are now going to turn our attention to scripts or programs that run on the Web server. The language we are going to use is PHP. It will require a slight change in the way we test our programs. Every time we make changes to our PHP program we must remember to upload it to a Web server. In this case, that space is on the Laureate Web server which we have allocated to your project group. Use your FileZilla FTP software to upload files to the web server. It might also be a Web server, such as Apache or IIS, which you have set up on your own desktop computer. XAMPP is a free software bundle that will install an Apache Web server, PHP and the MySQL database on your local machine (... APACHE FRIENDS..., 2011).

The main thing to remember is that you cannot just open PHP scripts in your browser and expect them to run. They need a server to do that. The other thing to note is that browsers store pages you have visited in their cache (an area of memory). Sometimes, clicking the refresh button is not effective in displaying the latest version of an HTML file you have uploaded to a Web server and the output from the latest PHP file. A quick way of checking is to try another browser. If it is clear that your first browser is displaying an older cached copy of a Web page you have created, clear the cache (wikiHow, n.d.).

### What Is PHP?

In the Week 2 Lecture Notes we introduced a simple example of an HTML form:

```
<form method = "post" action = "customer.php">

<input type = "submit" name = "submit" value = "Submit" />
<input type = "reset" value = "Clear" />

</form>
```

The *action* attribute has a value *“customer.php”*. It names a file that will be opened on the Web server when the user clicks the ‘Submit’ button. The extension of this file (the part after the dot) indicates that it is a script written in PHP. This handling of information input via an HTML form is one of the main reasons we have for using server-side programming. The other important function PHP performs is that of handling interactions with a database such as MySQL, and we shall be looking at that next week.

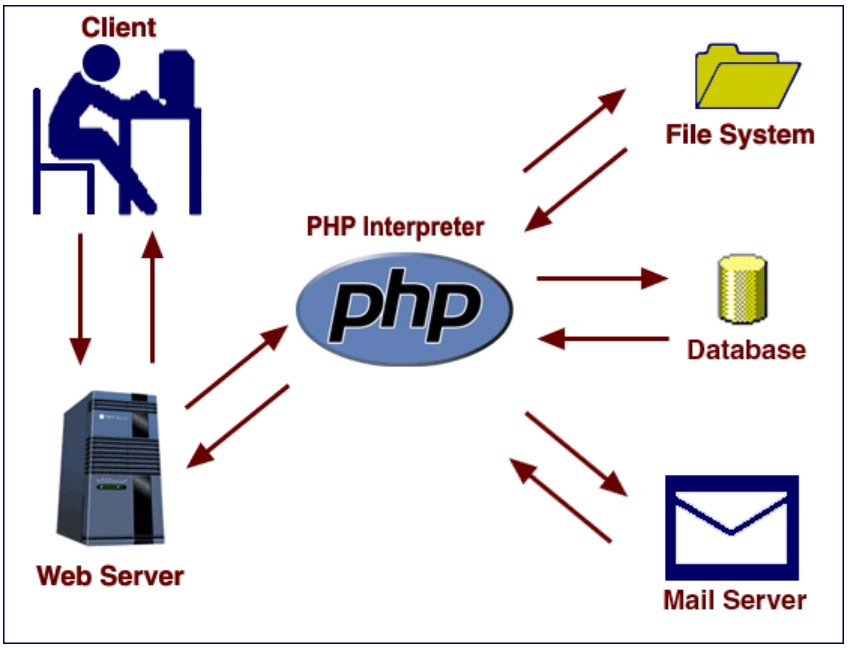


**Figure 1: How PHP Works (Webucator, 2012)**

PHP stands for PHP: Hypertext Preprocessor. This is sometimes referred to as a recursive acronym (it refers to itself). PHP is an open source language, which means that it can be freely distributed. Development is under the control of the PHP Group. In practice, people are restricted from making changes to the language by a clause in the licence which says that 'The name "PHP" must not be used to endorse or promote products derived from this software without prior written permission' (PHP Group, n.d.). This does not, however, affect our ability to write programs in the language.

There is an official PHP manual available online at the PHP Group's Web site (2013a, 2013b), and this should be of great help in learning to use new features of the language. PHP is the most popular server-side scripting technique but not the only one. Many sites use Active Server Pages (ASP), often written in VB.NET (W3 Schools, n.d.a). Additionally, another popular option is JavaServer Pages (JSP) (Oracle, n.d.).

PHP has gone through a number of different versions. Sometimes it will be worth checking which version of PHP is installed on the Web server you are using. Broadly speaking, any version of PHP after 5.1 should contain the latest functionality and will include built-in support for PHP Data Objects (PDO), which we shall use next week for database access. You can rely on the Laureate Web server to support these latest additions to PHP.

## PHP Basics

The good news about learning PHP is that much of the language is very similar to JavaScript. It too, is an interpreted language. One difference is that all variables must start with a '$' sign followed by a letter or an underscore character. Variable names may only contain numerals (0-9), letters or underscores. No spaces are allowed.

Another difference is that there is no equivalent of JavaScript's *var* declaration. PHP, like JavaScript, does not require variables to be declared before they are used, but it is good practice. In the absence of *var*, we have to implement PHP variable declarations by giving variables an initial value (we initialise them). For example:

```
$numberOne = 0;
$numberTwo = 0;
$surname   = "";
```

PHP is another loosely typed language in that the programmer does not have to declare the data type of a variable. PHP will make intelligent guesses about the kind of data that is being stored.

A file with a .php extension may contain nothing but PHP or nothing but HTML. More typically, it will contain both. The HTML will represent information that will be returned to the browser in all cases. The PHP code will normally be used to process information sent from the browser, interact with files or databases on the server and then generate customised HTML to be returned to the browser. The PHP is, therefore, used to generate information that varies from one case to another.

When a PHP script has executed on the server, it will normally generate HTML code which will be placed alongside any existing HTML in the PHP file before being returned to the browser. Thus, what the browser receives is HTML, but the address bar will show a URL containing a file with a .php extension. This will be the same file name (in our example *customer.php*) that was mentioned in the *action* attribute of the HTML form.

This sometimes causes confusion. When we are requesting an HTML file from a server by clicking on a hyperlink, the content of the file the browser receives is exactly the same as the content of the file that is sitting on the Web server. It is slightly different with PHP. We send the name of a PHP script to the Web server. We are effectively asking the server to run the PHP script. What we get back is not the code that is contained in the PHP file (because that is not visible to the client). What we actually receive is the output from the PHP script after it has executed. That output will normally consist of HTML.

### *get* and *post*

There is one more thing to explain before we look at an example of PHP in action. In an HTML form, we are likely to see a line of code like this:

```
<form method = "post" action = "customer.php">
```

The *method* attribute is here *post*. The main alternative is *get*. This specifies the method by which information will be sent from the HTML form to the Web server.

The *get* method appends or adds on the information to the end of the URL that contains the name of the PHP file. A *?* symbol is placed between the URL and the information being appended. If the method is *post*, information is returned as part of a message in the body of the submission.

*post* is more secure if confidential information is being transmitted because data does not appear as part of a URL in plain sight. The *get* method is limited to returning 100 characters and, therefore, is not suitable for transmitting a large amount of information. *get* is useful where the Web site owner wants to present the client with a link that has information appended to it (e.g. 'Click here to renew your subscription' where details about us are tagged onto the end of the URL).

Let's look at an example of a simple PHP script. Firstly, the HTML file that calls the PHP script:

```
<!DOCTYPE html>
<html>

<head>
<title> Our First PHP Example </title>
<link href="style1.css" type="text/css" rel="stylesheet" />
<meta charset="utf-8" />
</head>

<body>
<h1> Customer Registration </h1>
<p> You are a valued customer. Tell us your name </p>

<form method = "post" action = "customer.php">
<fieldset>
<legend></legend>
<label for="firstname">First name</label><br />
<input name = "firstname" id = "firstname" type = "text" size = "20"
maxlength = "30" /><br />
</fieldset>

<p>
<input type = "submit" name = "submit" value = "Submit" />
<input type = "reset" value = "Clear" />
</p>
</form>
```

```
</body>
</html>
```

**Figure 2: firstPHP.html**

The above code contains a simple HTML form that asks for the customer's first name. It will launch a PHP script on the Web server that looks like this:

```
<!DOCTYPE html>
<html>

<head>
<title> The Reply From the PHP Script </title>
<meta charset="utf-8" />
<link href="style1.css" type="text/css" rel="stylesheet" />
</head>

<?php
$firstName = $_POST["firstname"];
/* This retrieves the value typed in the HTML form text box called
"firstname" and stores it in a PHP variable */
?>

<h1> Customer Registration Part 2 </h1>
<p> Welcome, <?php echo $firstName ?>. Now tell us your surname </p>

<form method = "post" action = "customer2.php">
<fieldset>
<legend></legend>
<label for="surname">Surname</label><br />
<input name = "surname" id = "surname" type = "text" size = "20"
maxlength = "30" /><br />
</fieldset>

<p>
<input type = "submit" name = "submit" value = "Submit" />
<input type = "reset" value = "Clear" />
</p>
</form>

</body>
</html>
```

**Figure 3: customer.php**

Any piece of PHP script opens with the symbols `<?php` and ends with `?>`. Note that this notation differs from that of HTML tags, where we use pairs of angled brackets like this: `<p>`. The second thing to note is that there can be several pieces of PHP script within a file interspersed with HTML code.

In our simple example we have stored the user's first name (typed into an HTML form field) in a PHP variable called *$firstName*. We have then used the PHP command *echo* to include the text stored in that variable as part of information we return to the browser. The example serves only to demonstrate how we can include information stored in PHP variables. It also

illustrates how a new form can be presented to the user alongside the output from a PHP script.

Any HTML in the file *customer.php* will be returned exactly as it is, interspersed with content that has been generated by bits of PHP script. The PHP command, *echo*, is useful in returning such content to the browser. In our example we have said:

```
<p> Welcome, <?php echo $firstName ?>. Now tell us your surname </p>
```

An alternative way of doing this would have been:

```
<?php echo ("<p> Welcome, " . $firstName . ". Now tell us your
surname </p>"); ?>
```

In JavaScript, we joined strings together by using a '+' sign. We learned that this is called concatenation. In PHP, we use the full stop or decimal point (**.**) to do the same job. Even more neatly we could write:

```
<?php echo ("<p> Welcome, $firstName. Now tell us your surname
</p>"); ?>
```

If the string to be returned to the browser is contained within double quotes, PHP is clever enough to recognise the variable names and substitute the value that is being stored in that variable; in this case, the user's first name. Note that the dot in this line of code is a punctuation mark which will be displayed after the user's name. Within a single set of double quotes, it does not indicate concatenation. If we had used single quotes in this line of code, the variable name *($firstname*) would have been displayed in the output.

Note also that references can be made to external CSS and JavaScript files in the HTML that we return to the user when our PHP script executes. In this example it will be assumed that these files are in the same directory as the PHP script.

**PHP and JavaScript Comparisons**

Before we look at the kind of things PHP scripts can do on a server, it will be useful to complete our comparison of JavaScript and PHP. The following elements are identical:

- The mathematical symbols used by the languages
- The way functions are defined
- The syntax for including comments (but PHP will also accept '#' for single line comments)
- The *for* loop
- The *while* loop

The *if* statement is largely identical. There is one subtle difference. In JavaScript we use *else if*, which basically starts another *if* statement. We have

a series of *if* statements that are nested (one inside each other). In PHP there is an *elseif* facility which is used like this:

```
if (some condition)
      {
      some action;
      }
elseif (some other condition)
      {
      some other action;
      }
elseif (some further condition)
      {
      yet another other action;
      }
else
      {
      do something else;
      }
```

The difference in the logic employed by the use of *else if* and *elseif* is subtle and for most purposes it should not concern us.

The *for* loop is implemented in the same way in both languages (but remember that PHP variables specified in the brackets must begin with a '$' sign). The difference is that PHP also has a *foreach* loop for cycling through arrays. Here is an example:

```
foreach ($arrayName as $value)
{
echo $value;
}
```

Here, the variable *$value* stores the value being held in the current position of the array called *$arrayName*. The *foreach* loop automatically starts at the first position in the array, stores its value in *$value* and then performs the action specified between the braces (curly brackets). It will then move on to the second position in the array, store its value in *$value* and so on until all the values in the array are processed. The syntax is simpler than that of the *for* loop.

Arrays work in very much the same way in JavaScript and PHP, but there is no *var* statement for declaring arrays in PHP. Arrays are essentially declared when we start storing values in them. For example:

```
$arrayName = array("Tom" http://www.learnphp-
tutorial.com/PHPBasics.cfm "Jill", "Jack", "Jacqueline");
```

or

```
$arrayName[0] = "Tom";
$arrayName[1] = "Jill";
```

There are also associative arrays, which do not use a number to identify a position in an array but instead use a key, which is usually a name. Let us assume that we are storing a series of student marks in an array where the key is the student's name. We would assign 67 marks to Pauline Jones (or create an array) like this:

```
$marks["Pauline Jones"] = "67";
```

We would then access and display the marks like this:

```
echo "Pauline Jones gained a mark of " . $marks["Pauline Jones"];
```

The *foreach* loop can be used with an associative array as follows:

```
foreach ($marks as $key => $value) {
```

Remember that in an associative array there is a key that identifies who or what the value belongs to. In our example, we are using names like 'Pauline Jones' to do this job. The value will be a mark. Therefore, each time we go through the *foreach* loop, the current key will be stored in the variable *$key*, and the current value will be stored in *$value*. The next time we go through the loop and look at the next item in the array, the values in these variables will be overwritten. We are saying: For each element of the *$marks* associative array I want to refer to the *key* as *$key* and the *value* as *$value*. The operator '=>' represents the relationship between a *key* and *value*. You can imagine that the *key* points => to the value (tizag.com, 2008).

We could then display each name and mark like this:

```
echo $key . " gained a mark of " . $value;
```

which, in the case of the Pauline Jones example, would read:

```
Pauline Jones gained a mark of 67
```

The *while* loop is the same in JavaScript and PHP and PHP also has a *do…while* loop. The difference is that with the *while* loop, the test is carried out at the beginning of the loop, so in theory the action specified in the loop may never be performed if the condition is not met when the loop starts to execute. The *do…while* loop, on the other hand, performs the test at the end of the loop, so the action specified will always be performed at least once.

The general format will be:

```
do
{
some action to be executed;
}
while (some condition is true);
```

## Retrieving Information from an HTML Form

PHP eliminates the drama of having to unpack the data sent from an HTML form to a Web server. It chops up a long string of data and packages it for us in an associative array for both *get* and *post* methods. It then provides ways for us to access that information. We have seen how the contents of a text box can be retrieved using the *post* method. It is also possible to use the *$_POST* function as if it were a variable like this:

```
echo $_POST["surname"];
```

If, however, we are going to use the value more than once, it is good programming practice to store the value in a variable. This is so that we are not asking PHP to search for the value in an associative array more than once. That would involve an unnecessary duplication of effort.

If data is returned using the *get* method, the *$_GET* function can be used in the same way as *$_POST*.

## PHP and Radio Buttons

We could define a set of radio buttons in an HTML form like this:

```
<!DOCTYPE html>
<html>

<head>
<title> Processing Radio Buttons </title>
<link href="style1.css" type="text/css" rel="stylesheet" />
<meta charset="utf-8" />
</head>

<body>
<h1> Customer Survey </h1>
<form method = "post" action = "radioButton.php">

<fieldset>
<legend> Your company Web site provides far more information about
your services than other companies do about theirs.  </legend>
<input type = "radio" name = "SiteQuality" id = "SQ1" value = "5" />
<label for="SQ1"> Strongly Agree </label> <br />
<input type = "radio" name = "SiteQuality" id = "SQ2" value = "4" />
<label for="SQ2"> Agree </label> <br />
<input type = "radio" name = "SiteQuality" id = "SQ3" value = "3" />
<label for="SQ3"> Neutral/Undecided </label> <br />
<input type = "radio" name = "SiteQuality" id = "SQ4" value = "2" />
<label for="SQ4"> Disagree </label> <br />
<input type = "radio" name = "SiteQuality" id = "SQ5" value = "1" />
<label for="SQ5"> Strongly Disagree </label> <br />
<input type = "radio" name = "SiteQuality" id = "SQ6" value = "0"
checked = "checked" /> <label for="SQ6"> Question Not Answered
</label> <br />
</fieldset>
```

```
<p>
<input type = "submit" name = "submit" value = "Submit" />
<input type = "reset" value = "Clear" />
</p>
</form>

</body>
</html>
```

**Figure 4: processRadioButtonsPHP.html**

Remember that, at most, one radio button can be selected, so no more than one value will be returned from the form. PHP will retrieve this by using the value associated with the *name* attribute ("SiteQuality") like this:

```
$selectedButton = $_POST["SiteQuality"];
```

We can then check to see which button was selected by using the value associated with the *value* attribute ("5" or "4" or "3", etc.). This is done with code something like this:

```
<!DOCTYPE html>
<html>
<head>
<title> Demonstrating Shadows </title>
<meta charset="utf-8" />
<link href="style1.css" type="text/css" rel="stylesheet" />
</head>
<body>

<?php
$selectedButton = $_POST["SiteQuality"];
if ($selectedButton == "5")
{
echo("<h2>Your reply was: Strongly Agree</h2>");
}
elseif ($selectedButton == "4")
{
echo("<h2>Your reply was: Agree</h2>");
}
elseif ($selectedButton == "3")
{
echo("<h2>Your reply was: Neutral/Undecided</h2>");
}
elseif ($selectedButton == "2")
{
echo("<h2>Your reply was: Disagree</h2>");
}
elseif ($selectedButton == "1")
{
echo("<h2>Your reply was: Strongly Disagree</h2>");
}
else
{
echo("<h2>You did not answer the question</h2>");
}
?>
```

```
</body>
</html>
```

**Figure 5: radioButton.php**

## PHP and Checkboxes

In the Week 2 Lecture Notes we provided this example of the use of checkboxes. Here, we have amended it slightly:

```
<!DOCTYPE html>
<html>
<head>
<title> Processing Checkboxes in PHP </title>
<meta charset="utf-8" />
<link href="style1.css" type="text/css" rel="stylesheet" />
</head>
<body>
<h1> Tell Us What You Think </h1>
<p> Please answer the following questions so that we can provide a
better service for you. </p>
<form method = "post" action = "checkboxes.php">
<fieldset>
<legend> Which of these do you own? </legend>
<input type = "checkbox" name = "ItemsOwned[]" id = "IO1" value =
"laptop" /> <label for="IO1"> A laptop computer </label> <br />
<input type = "checkbox" name = "ItemsOwned[]" id = "IO2" value =
"blackberry" /> <label for="IO2"> A Blackberry </label> <br />
<input type = "checkbox" name = "ItemsOwned[]" id = "IO3" value =
"iphone" /> <label for="IO3"> An iPhone </label> <br />
<input type = "checkbox" name = "ItemsOwned[]" id = "IO4" value =
"ipad" /> <label for="IO4"> An iPad </label> <br />
<input type = "checkbox" name = "ItemsOwned[]" id = "IO5" value =
"labrador" /> <label for="IO5"> A Labrador called Donald </label> <br
/>
</fieldset>
<p>
<input type = "submit" name = "submit" value = "Submit" />
<input type = "reset" value = "Clear" />
</p>
</form>
</body>
</html>
```

**Figure 6: processCheckboxesPHP.html**

Here, the number of values returned by an HTML form is more difficult to predict because anything from 0 to 5 checkboxes could be selected. In the above example, we have given each checkbox in the set the same name, and have used empty square brackets in *ItemsOwned[ ]*. This will ensure that PHP places all the values returned to the server into an array. We shall retrieve all the values returned with code like this:

```
$checkboxArray = $_POST["ItemsOwned"];
```

Note that here we do not use empty square brackets after *ItemsOwned*. That is just a device that is employed in the HTML code. If we want to test first whether at least one checkbox has been selected, we can first say:

```
if (isset ($_POST["ItemsOwned"]))
{
$checkboxArray = $_POST["ItemsOwned"];
}
```

The length of the array holding the checkboxes will be calculated by the PHP function, *count()*, like this.

```
$lengthArray = count($checkboxArray);
```

We can then use a *for* or *foreach* loop to run through the array checking for different values. The *foreach* equivalent is included in comments. Here is the full PHP code:

```
<!DOCTYPE html>
<html>
<head>
<title> Checkboxes Processed </title>
<meta charset="utf-8" />
<link href="style1.css" type="text/css" rel="stylesheet" />
</head>
<body>

<?php
if (isset($_POST["ItemsOwned"]))
{
$checkboxArray = $_POST["ItemsOwned"];
$lengthArray = count($checkboxArray);
for ($i = 0; $i < $lengthArray; $i++)
// foreach ($checkboxArray as $key => $value)
{
if ($checkboxArray[$i] == "laptop")
// if ($value == "laptop")
{
echo("<h2>You own a laptop</h2>");
}
elseif ($checkboxArray[$i] == "blackberry")
// elseif ($value == "blackberry")
{
echo("<h2>You own a Blackberry</h2>");
}
elseif ($checkboxArray[$i] == "iphone")
// elseif ($value == "iphone")
{
echo("<h2>You own an iPhone</h2>");
}
elseif ($checkboxArray[$i] == "ipad")
// elseif ($value == "ipad")
{
echo("<h2>You own an iPad</h2>");
}
elseif ($checkboxArray[$i] == "labrador")
```

```
// elseif ($value == "labrador")
{
echo("<h2>You own a Labrador called Donald</h2>");
} // End of if $checkboxArray[$i] etc
} // End of for loop
} // End of if part of if isset
else // isset is not true
{
echo("<h2>No checkboxes were selected</h2>");
} // End of if isset etc
?>

</body>
</html>
```

**Figure 7: checkboxes.php**

### Manipulating Strings in PHP

PHP has a range of built in functions for manipulating strings. To find the length of a string (the number of characters in it) we can use *strlen()* as in:

```
$myString = "I am learning PHP.";
$numCharacters = strlen($myString);
```

In this example the value of *$numCharacters* will be 18.

To find a single character or string within a larger string we can say:

```
$position = strpos($longstring, "@");
```

If the '@' symbol is contained in the string, this function will return its position (where 0 is the first position in the string). If it is not present, the function will return the Boolean value *false*. This function can therefore be used to test whether a character or string is present as follows:

```
if ($position === false)
{
echo "The character '@' was not found in the string '$longstring'";
}
else
{
echo "The character '@' was found in the string '$longstring'";
}
```

We have to use === here because of a peculiarity in PHP. If the character was in position 0 and we asked if it was true that the character was in the string using the expression '== *false*', PHP would say no. This is because 0 is translated into the value *false* when a Boolean test (of whether something is true or false) is carried out.

Another useful function provided by PHP is *trim*. It will remove white space from the beginning and end of a string. This will include spaces, tabs, new line (line feed) characters, carriage returns and *null* characters. It is often

employed to tidy up user input because an extra space at the beginning or end of a password, for instance, would result in it being rejected. In retrieving a password from an HTML form we could write:

```
$password = trim($_POST['password']);
```

A full list of string functions can be found at PHP Group (2013c).

### Operators

PHP has a number of standard and 'C like' operators to be used when building expressions. These are summarised in the table below:

| **Operator** | **Brief explanation** |
|---|---|
| + - * / | Standard arithmetic operators |
| = += -= *= /= | Assignment operators |

| **Comparison operators** | **Brief explanation** |
|---|---|
| = = ($a = = $b) | TRUE if $a is equal to $b |
| != ($a != $b) | TRUE if $a is not equal to $b |
| < ($a < $b) | TRUE if $a is strictly less than $b |
| > ($a > $b) | TRUE if $a is strictly greater than $b |
| <= ($a <= $b) | TRUE if $a is less than or equal to $b |
| >= ($a >= $b) | TRUE if $a is greater than or equal to $b |

| **Logical operators** | **Brief explanation** |
|---|---|
| ! (!$a) | TRUE if $a is not TRUE |
| && ($a && $b) | TRUE if both $a and $b are TRUE |
| \|\| ($a \|\| $b) | TRUE if either $a or $b is TRUE |

| **String operators** | **Brief explanation** |
|---|---|
| . ( $a = $b . "hello") | Concatenates strings $b and "hello" |
| .= ($a .= "hello") | Appends "hello" to $a |

### Server-Side Validation

In studying JavaScript we saw how validation of data can be carried out on the client-side in the browser. However, we may often want to carry out much of our data validation on the Web server. Fortunately, PHP has a number of built-in functions to help us. Some of these are called filters. One check we may want to carry out is whether data is of the right type. Let us look at a test to see whether data returned from an HTML form field is an integer (whole number):

```
$validAge = filter_input(INPUT_POST, "age", FILTER_VALIDATE_INT);
```

Here, we are using the built-in PHP function, *filter_ input*. It accepts a number of parameters in brackets. The first one, *INPUT_POST*, simply indicates that our input is being transmitted by the *post* method. The second parameter tells

us that the HTML form field we are validating is called *age*. Finally, *FILTER_VALIDATE_INT* says that we are checking to see whether the returned value is an integer.

The function returns a Boolean value (true or false), which is stored in the variable *$validAge*. If the input is a valid integer, the value will be *true*. We may then use an *if* statement to specify what should be done in each case. For instance, if this is not a valid integer we may want to ask the user to input the data again.

There is also a *FILTER_VALIDATE_FLOAT* option to check whether the input is a floating point number, and a *FILTER_VALIDATE_EMAIL* option to ensure that input conforms with the basic requirements of an e-mail address (it must contain an '@' symbol, for instance).

Sometimes, we may want to define an acceptable range of values that our data can have. In our example of the user's age, we could specify that this has to be in the range 21 to 100 like this:

```
$options = array("options"=> array("min_range"=>0,
"max_range"=>100));
$validAge = filter_input(INPUT_POST, "age", FILTER_VALIDATE_INT,
$options);
```

We have added an extra parameter for the *filter_input* function. This specifies some additional options, and these are stored in the variable *$options*. We define an acceptable range of values in the first line of code. We are actually setting up an associative array, which must be called *$options*. The values *min_range* and *max_range* define the lower and upper limits of the acceptable range of values.

Full details about PHP filters can be found at PHP Group (2013d).

Here is a full working example of the use of PHP filters. First, the HTML form, and then the PHP:

```
<!DOCTYPE html>
<html>
<head>
<title> PHP Filters </title>
<meta charset="utf-8" />
<link href="style1.css" type="text/css" rel="stylesheet" />
</head>
<body>
<h1> Tell Us What You Think </h1>
<p> Please answer the following questions anonymously so that we can
provide a better service for you. </p>
<form method = "post" action = "filterPHP.php">
<fieldset>
<legend></legend>
<label for="speed">What is your average driving speed (expressed as a
real or floating point number).</label><br />
<input name = "speed" id = "speed" type = "text" size = "10"
```

```
maxlength = "10" /><br />
<label for="age">Type your age (as an integer).</label><br />
<input name = "age" id = "age" type = "text" size = "5" maxlength =
"5" /><br />
<label for="email">Type your email address.</label><br />
<input name = "email" id = "email" type = "text" size = "20"
maxlength = "30" /><br />
</fieldset>
<p>
<input type = "submit" name = "submit" value = "Submit" />
<input type = "reset" value = "Clear" />
</p>
</form>
</body>
</html>
```

**Figure 8: filterPHP.html**

```
<!DOCTYPE html>
<html>

<head>
<title> PHP Filter Results </title>
<meta charset="utf-8" />
<link href="style1.css" type="text/css" rel="stylesheet" />
</head>

<h1> Demonstrating PHP Filters </h1>

<?php
$options = array("options"=> array("min_range"=>21,
"max_range"=>100));
$validSpeed = filter_input(INPUT_POST, "speed",
FILTER_VALIDATE_FLOAT);
/* PHP will accept an integer as a valid floating point number
because it can be easily converted */
$validAge = filter_input(INPUT_POST, "age", FILTER_VALIDATE_INT,
$options);
/* PHP will not accept a floating point number as a valid integer */
$validEmail = filter_input(INPUT_POST, "email",
FILTER_VALIDATE_EMAIL);
if($validSpeed)
{
echo("<h2>Your average speed was expressed as a valid floating point
number.</h2>");
}
else
{
echo("<h2>Error: your average speed was not expressed as a valid
floating point number.</h2>");
}
if($validAge)
{
echo("<h2>Your age was expressed as a valid integer.</h2>");
}
else
{
echo("<h2>Error: your age was not expressed as a valid integer <br />
or it is outside an acceptable range.</h2>");
```

```
}
if($validEmail)
{
echo("<h2>Your email address was in a valid format.</h2>");
}
else
{
echo("<h2>Error: your email address was not in a valid
format.</h2>");
}
?>

</body>
</html>
```

**Figure 9: filterPHP.php**

## PHP Includes

We have seen that functions can help us reuse code that we should otherwise have to retype. PHP provides another way of helping us reuse code called an *include*. This is PHP's way of implementing what is generally known as Server Side Includes (SSI), meaning that the server includes or adds some code to a PHP script we have written.

The basic idea is that we store in a separate file some code (often HTML) that we reuse quite often. This could be the header or footer (beginning and end) of an HTML file, a menu, a list of variables or possibly a list of PHP assignment statements giving particular values to variables. When we use a PHP *include*, that code may be thought of as being copied and pasted into our main PHP file at the point where we use the built in *include()* function. Here is an example. Firstly, the code in our main PHP file that uses *include()*:

```
<?php
include ("header.php");
?>
<body>
<?php
echo ("Hello world.");
?>
</body>
</html>
```

Our file is a classic mixture of PHP and HTML. The call to the *include()* function names an external file *header.php*. The contents of this file will effectively be pasted into our main program at the point where the call to *include()* has been made. In this case, the contents of *header.php* will look like this:

```
<!DOCTYPE html>
<html>
<head>
<title> Our Company </title>
<link href="style1.css" type="text/css" rel="stylesheet" />
```

```
<meta charset="utf-8" />
</head>
```

Where the included file does not contain PHP, it is not necessary to use the *.php* file extension. However, using this extension in all cases is a good habit to get into. If we were to later add PHP code to the included file and had used an extension such as *.inc*, the PHP code would be visible to anyone accessing the *.inc* file directly. Using the *.php* extension ensures that, at worst, the client will see only the output from the PHP script after it has executed.

Where the file to be included contains PHP code, make sure it is enclosed in the traditional `<?php` and `?>` tags. The analogy of copying and pasting breaks down here because, without those tags, the main program will not recognise our code as PHP. Before the code is included, the extra PHP tags are automatically stripped out.

Another important thing to remember is no spaces or blank lines should precede the code in the file to be included. Make sure that the code begins in the very top left corner of the file.

**Setting Cookies**

The interaction between a user and a Web site often consists of a dialogue involving each computer requesting or supplying information several times. Typically, however, each time a party sends or receives information, the connection is ended. It is rather like a telephone conversation during which each party has to redial every time he or she says something. HTTP is a stateless protocol, which means that information is not normally stored about previous communications between the client and the server. It is thus like a phone conversation where one party (the server) has no recollection of what was said earlier in the dialogue.

This problem is described as one of maintaining ‘state’ (keeping a record of the details of the dialogue between two computers). Programmers have to use a number of techniques to store information. At its most sophisticated, this might involve storing information in a database and asking users to log in to a Web site so that all their details can be retrieved. Sometimes, however, less complex techniques are all that are required.

One option is to use cookies. Cookies are small text files that are written to the hard disk of the user’s computer by a site which he or she has visited. Users can sometimes refuse to accept cookies or disable them completely through the settings of their browser or personal firewall, and so they are not a complete solution. There may be a policy issue in deciding whether to refuse to allow visits to an organisation’s Web site from users who will not accept cookies. Cookies are reasonably secure as they can only be read by the Web servers which wrote the information in the first place.

Each site can only access the cookies that are relevant to interactions between the user and its site. When the user visits a Web site, that site's server will retrieve information about him or her from the cookie(s) saved on the user's computer. This will help the Web site provide content that is partially customised to a user's needs or possibly to provide personalised messages. Cookies usually have an expiry date, at which point they will be automatically deleted.

The European Union has required its member states to bring in legislation to ensure that cookies are being set with the user's consent. In the UK, this legislation was enacted in the form of the Privacy and Electronic Communications Regulations in May 2012 (Computer Weekly, n.d.). Web sites are using a range of approaches to draw users' attention to the fact that cookies are being set.

The following program demonstrates how we can set cookies and retrieve them using PHP. We shall see in Week 7 how the new local storage facility supported by HTML5 may soon be a viable alternative to cookies.

Here is the code for our example:

```
<!DOCTYPE html>
<html>
<head>
<title>Setting Cookies</title>
<meta charset="utf-8" />
</head>
<body>
<h2>We are going to set some cookies</h2>
<h3>Please fill in these fields so that we can demonstrate the use of
cookies. Clicking on the button will imply consent for the setting of
cookies.</h3>
<form action="cookies.php" method="post">
First Name <input type="text" name="firstname" size="20"
maxlength="30"><br />
Last Name <input type="text" name="lastname" size="30"
maxlength="40"><br />
IN Class <input type="text" name="class" size="15" maxlength="15">
<input type="submit" value="Set Those Cookies">
</form>
</body>
</html>
```

**Figure 10: cookies.html**

In the file below, it is important that cookies are set before the opening `<html>` tag.

```
<?php
if (isset($_COOKIE["login_date"])) {
/* If a cookie has already been set on your computer */
$lastlogindate = $_COOKIE["login_date"];
$lastlogintime = $_COOKIE["login_time"];
$lastclass = $_COOKIE["login_class"];
```

```
/* The details in the cookies will tell us about the last time you
used the cookies.html page */
}
setcookie ("login_date", date("D d M Y"),time()+86400*30); /* expire
in 30 days (expressed as seconds) */
setcookie ("login_time", date("g:i A"),time()+86400*30); /* expire in
30 days */
setcookie ("login_class", $_POST["class"] ,time()+86400*30); /*
expire in 30 days */
/* In all cases set cookies about your current situation */
?>
<!DOCTYPE html>
<html>
<head>
<title>Display Cookies</title>
<meta charset="utf-8" />
</head>
<body>
<h2>Welcome to the Cookies Page</h2>
<?php
$name = $_POST["firstname"] . " " . $_POST["lastname"];
$class = $_POST["class"];
/* Retrieve information from the HTML form in cookies.html */
echo("<p>Welcome to the cookies page, " . $name . ".</p>");
echo("<p>Your IN class is " . $class . ".</p>");
if (isset($lastlogindate)) {
/* If there is a cookie showing the date of your last visit */
echo("<p>You last logged in at " . $lastlogintime . " on " .
$lastlogindate . ", Web server time.</p>");
echo("<p>You also last logged in when a member of the IN class " .
$lastclass . ".</p>");
}
else {
echo("<p>You either haven't logged in before, or haven't logged in
for more than 30 days.</p>");
}
?>
</body>
</html>
```

**Figure 11: cookies.php**

## Session Variables

A more sophisticated technique, and one that is widely used in e-commerce sites, is to use session variables. They involve creating a unique id (UID) for each visitor to a Web site. These may be passed via cookies or appended to the URL. For those interested in exploring this technique, see W3 Schools (n.d.b) and PHP Group (2013e).

## References


...APACHE FRIENDS... (2011) *XAMPP* [Online]. Available from http://www.apachefriends.org/en/xampp.html (Accessed: 2 February 2013).

Computer Weekly (n.d.) *How to comply with the EU cookie law* [Online]. Available from http://www.computerweekly.com/guides/How-to-comply-with-the-EU-cookie-law (Accessed: 17 September 2012).

Oracle (n.d.) *JSP overview* [Online]. Available from http://docs.oracle.com/cd/E13222_01/wls/docs70//////jsp/intro.html (Accessed: 17 September 2012).

The PHP Group (n.d.) *The PHP license, version 3.01* [Online]. Available from http://www.php.net/license/3_01.txt (Accessed: 17 September 2012).

The PHP Group (2013a) *PHP manual* [Online]. Available from http://www.php.net/manual/en/index.php (Accessed: 6 February 2013).

The PHP Group (2013b) *PHP manual: download documentation* [Online]. Available from http://www.php.net/download-docs.php (Accessed: 6 February 2013).

The PHP Group (2013c) *String functions* [Online]. Available from http://www.php.net/manual/en/ref.strings.php (Accessed: 6 February 2013).

The PHP Group (2013d) *Filter: introduction* [Online]. Available from http://www.php.net/manual/en/intro.filter.php (Accessed: 6 February 2013).

The PHP Group (2013e) *Session handling* [Online]. Available from http://www.php.net/manual/en/book.session.php (Accessed: 6 February 2013).

tizag.com (2008) *PHP for each loop* [Online]. Available from http://www.tizag.com/phpT/foreach.php (Accessed: 26 February 2013).

W3 Schools (n.d.a) *ASP tutorial* [Online]. Available from http://www.w3schools.com/asp/default.asp (Accessed: 17 September 2012).

W3 Schools (n.d.b) *PHP sessions* [Online]. Available from http://www.w3schools.com/php/php_sessions.asp (Accessed: 17 September 2012).

Webucator (2013) *PHP tutorial: PHP basics* [Online]. Available from http://www.learnphp-tutorial.com/PHPBasics.cfm (Accessed: 6 February 2013).

wikiHow (n.d.) *How to clear your browser's cache* [Online]. Available from http://www.wikihow.com/Clear-Your-Browser's-Cache (Accessed: 8 September 2012).