

## Week 8 Lab – Introduction to Server-side Programming with PHP

### Working with Data Types and Operators

### Functions and Control Structures

PHP Control Structures: <http://www.php.net/manual/en/language.control-structures.php>  
PHP User-defined Functions: <http://www.php.net/manual/en/functions.user-defined.php>

#### Aims:

- Develop an understanding of the basic use of *variables, arrays and expressions* in PHP.
- To be able to use various control structures and develop your own functions.
- Use PHP predefined ‘superglobal’ variables to get data from a form.
- Gain some of the knowledge and skills needed to complete Assignments.

## Task 1: Using PHP variables, arrays and if statements

In this task we will

- Inspect create an **array** and initialise it with values
- Use an **if** statement to process the values
- Use PHP to generate some HTML.

- **Remember:** PHP script generates HTML on the *server*, and then **returns it to the client Browser**. So you *must* upload your PHP files to **mercury** so they are processed by the PHP engine that is attached to the HTTP server.
- All web pages delivered to the client should conform to HTML5 and should be validated.

#### Step 1: Create a PHP script

Create a new folder **lab08** under the unit folder on the mercury server in your htdocs folder.

Save today's work in this folder. Create a file **myfirst.php** with a PHP script. The script does the following:

1. Declares and initialises an **array** named **\$marks[]** and with three integer elements 85, 85 and 95.
2. Modifies the value of the second element to contain 90.
3. Computes the average score of the values from the three elements and stores the result in **\$ave**.
4. Uses an **if** statement to assign a value of "PASSED" to variable **\$status** if the average is at least 50, otherwise assigns a value of "FAILED".
5. Uses output statements to display "The average score is " along with the averaged value and " You " followed by the status.

Use any text editor on your local computer (e.g. Notepad++) and code the following:

```
<!DOCTYPE html>
<html lang="en">
<head>
  <title>Using PHP Variables, arrays and operators</title>
  <meta charset="utf-8"/>
  <!-- add other meta -->
</head>
<body>
  <h1>Creating Web Applications - Lab08</h1>
  <?php
    $marks = array (85, 85, 95);
    $marks[1] = 90;
    $ave = ($marks[0] + $marks[1] + $marks[2]);
    if($ave >= 50)
    {
      $status = "PASSED";
    }
    else
    {
      $status = "FAILED";
    }
    echo "<p>The average score is $ave. You $status.</p>";
  ?>
</body>
</html>
```

Note the syntax here.  
We have used double quotes,  
so the variables are evaluated

**Step 2: Load to the Server**

Use WinSCP or similar to copy the file `myfirst.php` to your `lab08` folder on Mercury.

*Remember that PHP only works on a server - you cannot run it from the local drive.*

**Step 3: Test**

Test in the browser, and check that the page is valid HTML5.

Do you have errors in the script? *See Task 5 that explains how to help track them down.*

*Remember that PHP only works on a server - you cannot validate your HTML from the local drive.*

## Task 2: Experimenting on arrays

In this task we will apply the approach covered in Task 1 to a similar problem.

**Step 1: Create the PHP script**

Create a file `daysarray.php` with a PHP script that declares and initialises an **array** named `$days[]` and with the days of the week Sunday, Monday, etc.

Use output statements to display “The Days of the week in English are:” along with the values in the `$days[]` array.

**Step 2: Load and test on the server**

Copy to the server, test in the browser, and check that the page is valid HTML5.

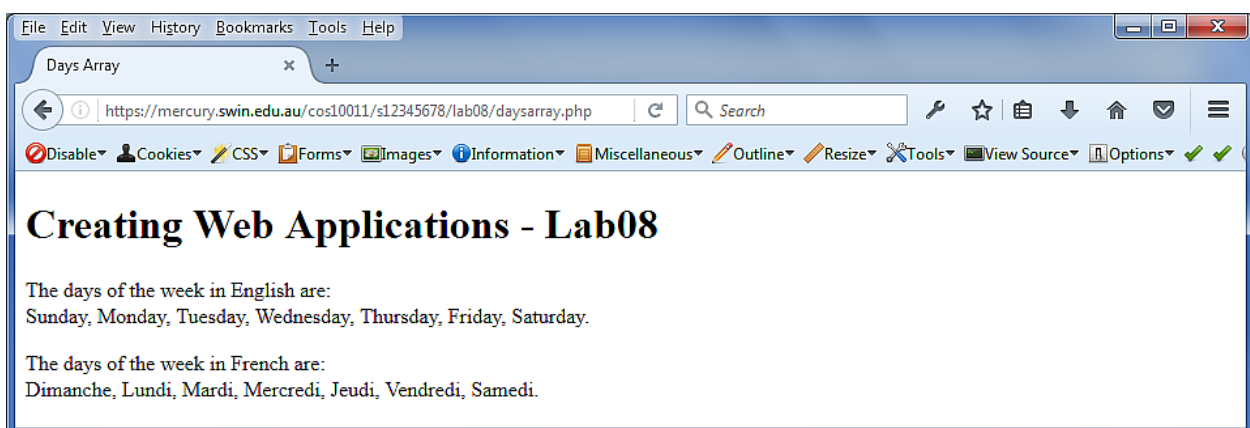
**Step 3: Change the script**

Add some more code to re-assign the values in the `$days[]` array with the days of the week in French, Sunday is *Dimanche*, Monday is *Lundi*, Tuesday is *Mardi*, Wednesday is *Mercredi*, Thursday is *Jeudi*, Friday is *Vendredi*, and Saturday is *Samedi*.

Then use output statements to display “The days of the week in French are:” along with the French values in the `$days[]` array

**Step 4: Load and test again**

Re-save the document as `daysarray.php`, to the server, test in the browser, and again check that the page is valid HTML5.



## Task 3: Functions, GET, and while statements

In this task we will apply

- User-defined functions
- In-built functions
- `if` selection statements and loops

### Step 1: Creating functions

Unzip the file `lab08.zip` into `lab08` folder.

Open the file `mathfunctions.php` in a text editor and read through the file.

The *user-defined* function `factorial` accepts a positive integer and returns its factorial value. A factorial of a non-negative integer  $n$ , denoted by  $n!$  is the product of all positive integers less than or equal to  $n$ . For example,

$$5! = 5 \times 4 \times 3 \times 2 \times 1 = 120$$

Add a function `isPositiveInteger` to `mathfunctions.php` as set out below. This user-defined function will use the *in-built* functions `is_numeric()` and `floor()` to check if a variable is an integer.

```
function isPositiveInteger ($n) { // declare the function that takes a parameter
    $result = false;
    if (is_numeric($n)) //use the inbuilt function is_numeric which returns boolean
        if ($n == floor($n))
            if ($n > 0 )
                $result = true;
    return $result; // function execution will end here if -ve or non-integer
}
```

*Aside:* Why aren't braces needed after the if condition `if (is_numeric($n))`?

While in this example braces are not used, it is good practice to always use them, as it can make the code easier to read and edit.

### Step 2: Call the functions

Create an HTML/PHP file `factorial.php` that will *include* the file `mathfunctions.php` in order to access the defined functions in the file. The code below needs to be embedded in the HTML `<body>` of `factorial.php`.

You will need to write the rest of the HTML around the `<body>`

```
<body>
<?php
    include ("mathfunctions.php"); //makes functions in the included file available
?>
<h1>Creating Web Applications - Lab 8</h1>
<?php
    $num = 5; //enter some different values here to test - remove later
    if (isPositiveInteger($num)) { // call the function we defined in the previous step
        echo "<p>", $num, "! is ", factorial ($num), "</p>"; //ditto for factorial
    } else { // number is not an integer
        echo "<p>Please enter a positive integer.</p>";
    }
    echo "<p><a href='factorial.html'>Return to the Entry Page</a></p>";
?>
</body>
```

Note: At this stage, in the code above, the value of the variable `$num = 5` is *hard-coded* for testing purposes. We will replace this with code to accept input from a form in the next task.

### Step 3: Test and validate

Copy to the server, test in the browser, and check that the delivered page is valid HTML5.

Test the output. The answer should be “5! is 120”.

Initialise `$num` to different values, and test that you get the correct output. Try:

- Different integers – positive and negative
- Non-integer numbers - e.g 3.4
- Non numbers

## Task 4: Passing parameters to a PHP file from a form

In this task we will modify `factorial.php` to receive values from an HTML form.

This will involve two steps:

1. Writing a PHP program that accepts input.
2. Writing a client HTML form that sends input to the PHP program on the server.

### Step 1: Modify the PHP to accept input

Re-name your file `factorial.php` to `factorial_with_get.php`

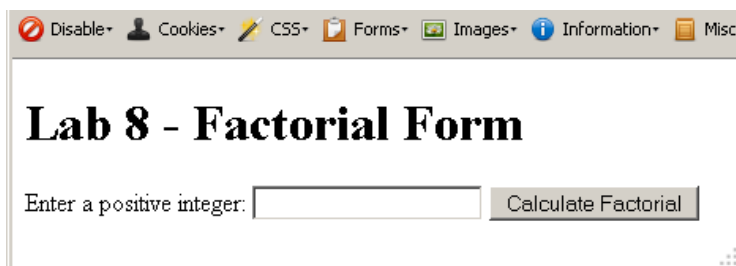
Replace the section of the PHP that has the hard-coded input `$num = 5;` with the following code.

The script will now accept input submitted from a form, that has a form control with `name="number"`.

```
<?php
if(isset($_GET["number"])) {           // check if form data exists
    $num = $_GET["number"];           // obtain the form data
    if(isPositiveInteger($num)) {     // call the function
        . . . other code
    }
}
echo "<p><a href='factorial.html'>Return to the Entry Page</a></p>";
?>
```

### Step 2: Create the form

Create a file `factorial.html` that contains a form with a single text box (as shown below) that allows a user to enter a number, and submit it to `factorial_with_get.php` using `method="get"`.



The screenshot shows a web browser window with a title bar containing icons for Disable, Cookies, CSS, Forms, Images, Information, and Misc. The main content area has a heading "Lab 8 - Factorial Form". Below the heading is a form with a label "Enter a positive integer:" followed by a text input field. To the right of the input field is a button labeled "Calculate Factorial".

*Remember:* forms send information to the server in `name-value` pairs so make sure the `name` in your form `input type="text"` matches the `number` parameter of the `$_GET` statement in `factorial_with_get.php`

Test in the browser, and check that the delivered web page is valid HTML5.

If you are having trouble getting the PHP script to work properly, see the tips on debugging in the next task.

### Optional Extended Exercise:

Repeat Task4, creating a `factorial_with_post.php` and use `$_POST` instead of `$_GET`, and modify the form to use `method="post"`

**Once you have completed the above tasks, show them to your tutor**

## Task 5: Debugging PHP

PHP does not have an internal debugging facility. And because it is running on the server, browser extensions are of little help. We will use a couple of *simple techniques* to track *syntax* and *logic* errors in PHP.

In a workplace you might use PHP IDEs like Zend Studio (<http://www.zend.com/products/studio/>) or PHP extensions like Xdebug ([xdebug.org](http://xdebug.org)) to provide advanced debugging facilities.

The program script we will debug, tests whether an input string is a perfect palindrome – that is, one that is the same when symbols are read both forward and reverse directions. The expected output on submitting is:



### Step 1: Detecting Syntax errors

As long as error reporting is turned on in the PHP set up on the server, PHP will generate errors that are sent with the HTML. In this step we will fix a PHP script that has two syntax errors (as well as some logic errors). We will fix the syntax errors first.

1. Open the file `buggysyntaxpalindrome.php` in Notepad++ or similar.  
Note that the file includes both a form and a PHP script that is actioned from the form.  
(ie. The page calls itself.)
2. Upload the file `buggysyntaxpalindrome.php` to Mercury.
3. Use Firefox to view the file output and errors generated. What line is the error on?  
What error is listed in the output? What is the source of the error?
4. Fix the error. Repeat 2 and 3 above. *Notice only one error appears at a time.* Why is that?
5. What line is the error on? What error is listed in the output? What is the source of the error?  
*Notice that errors are not necessarily on the line reported.*
6. Fix the error.

There are now some logic errors to be fixed.

### Step 2: Tracking logic errors

While PHP will report errors it detects as it interprets the syntax, it does **not** understand what you intend when are writing your code so it will **not** pick up logic errors. Logics errors can lead to faulty output or runtime exceptions.

1. The file `buggylogicpalindrome.php` has the syntax errors identified in Step 1 fixed, but has some logic errors. Upload it to Mercury.
2. Run the program and observe the output.  
Note that although no error is reported, the output is not as expected. What is/are the problem(s)?
3. If they are hard to spot on visual inspection of the code, un-comment the `print` statements in the code (on lines 22, 24, 38). Inserting `print` statements can help you trace which *path* was executed and the *state* of variables. This should help you find the source of the problems.  
(Don't forget to comment out or delete print statements before you release the code.)
4. Fix the errors (hint: lines 21, 23, 27).