Week 9 Lab – Server-side Data Validation

Note: As the demonstrations for Part 2 of the Assignment were held in Week 8, your tutor will check your lab work for both Week 8 and 9 in Week 10.

Aims:

- Create a PHP file that processes user input from a form and returns a response to the user.
- Sanitise form data at the server
- Check data for text, drop-down boxes, radio buttons and checkboxes
- Use regular expressions in PHP
- Create a PHP function
- Force the Web page to redirect if a user tries to enter the PHP processing page directly via the URL

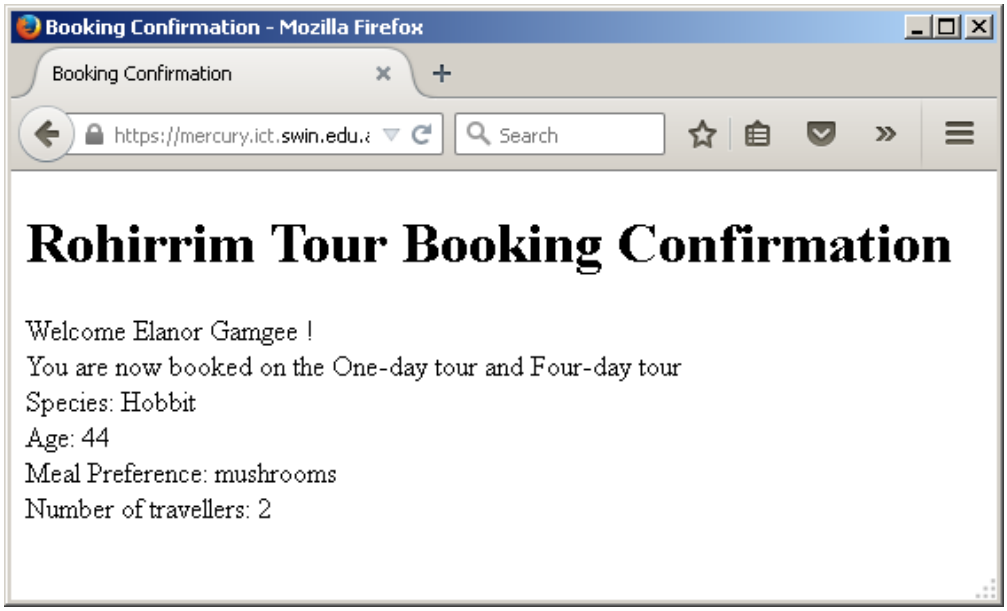
All delivered web pages should conform to HTML5 and must be validated.

NOTE: To validate that the code generated by your PHP script is valid HTML5 you must view the code received by your browser. You cannot successfully validate HTML directly from PHP source file.

Task: Checking user input data at the server

In this task we will create a script that instructs the PHP engine on the **server** to process data sent from a client webpage. The data will be *name-value* pairs sent from a form that a user has entered and submitted.
A webpage that we construct from a mixture of HTML and PHP will then send back a confirmation to the user.

The webpage will look something like this:



Step 1: Load an example HTML form.

In this step we use the same HTML form that we used in Week 7 Lab. This time however we will be validating the input data at the ***server*** using **PHP** rather than at the ***client*** using HTML5/JavaScript.

1. Create a *local* folder called **`lab09`**, download the **`lab09.zip`** file and unzip the file to the folder. This zip file contains a file called **`register.html`**.

2. We need to direct where the information collected from the form will be submitted. We will submit with method **`post`** to **`processbooking.php`**.

   Add the following **attributes** to the **`<form>`** element of **`register.html`**.

   ```
   method="post" action="processbooking.php"
   ```

3. So we can check the PHP data validation is working we will turn off HTML5 data validation by adding the `novalidate="novalidate"` attribute to the form.

4. Create a directory on Mercury in your `htdocs` folder called **`lab09`**.

5. **Upload the HTML page `register.html` to mercury and view it in your browser.**

6. ***Validate the HTML.***

Step 2: Create a PHP file to process the form

A PHP file is an HTML file with some embedded script. The PHP processor on the server will process it and generate the HTML, in the file, where the embedded scripts are located.
PHP code in an HTML file can generate *part* of the HTML or *all* of the HTML
(or even *none* of the HTML - but then there is not much point giving it a PHP extension!)

We will start by creating a standard HTML template, then add some PHP script in the next step.

1. Using Notepad++ or similar, create a standard HTML template with a doctype instruction, header and body elements. (Check out Lab 2 again if you have forgotten what should go in the HTML header).

2. Give the page the title “Booking Confirmation”

3. In the <body> add a heading element that says “Rohirrim Tour Booking Confirmation”.

4. Save it as **`processbooking.php`** and save it in the same folder you saved **`register.html`** (Step 1).

Step 3: Add some PHP to the HTML process file to extract values from the form

When a `submit` button is actioned on a form, the *name-value* pairs are sent from the client to the server through a GET or a POST HTTP request message. This request message is then processed according to the PHP code in the file specified in the `action` attribute. In Lab 8 Task 4, we sent a single *name-value* pair representing an integer number to a PHP process file via a GET `method`.

In this step, we will send all the *name-value* pairs from the form on **`register.html`** to be processed by the file **`processbooking.php`** that we created in Step 2. As we are using a POST message to send this data, we need to use the **`$_POST`** autoglobal to extract the *values* associated with each *name*.

We can use the in-built PHP function `isset()` to check if a named value in the POST message has been defined. It will not be defined if the process was not triggered by a form *submit*, for example if someone attempts to access the PHP directly by typing the URL of the PHP file.

1. In the body element, after the heading, create a PHP block
```
<?php
?>
```

2. Inside the PHP block, create an `if-else` statement as shown below. If the condition in the `if-else` statement evaluates to true, we will store the value from the form into a variable. If the value is not set we will print an error message on the Web page.

```
...
<body>
      <h1>Rohirrim Tour Booking Confirmation</h1>
<!-- Begin processing -->
<?php
    // Checks if process was triggered by a form submit, if not display an error message
    if (isset ($_POST["firstname"])) {
        $firstname = $_POST["firstname"];
        echo "<p>This is a test: You first name is $firstname</p>";
    }
    else {
        // Display error message, if process not triggered by a form submit
        echo "<p>Error: Enter data in the <a href=\"register.html\">form</a></p>";
    }
    //assign the rest of the form values to PHP variables here ...
?>
</body>
...
```

Note: comment outside PHP block so must be an HTML comment

We need to use the \ escape character to include the quote character inside a string

3. Upload **`processbooking.php`** to the mercury server.
   Check that it is saved in the same folder you saved **`register.html`**.

4. Open **`register.html`** in Firefox and make sure it echoes back the name you entered in the firstname field.

5. Immediately below the **`if - else`** statement, test if **`lastname`** is set, then create a variable **`$lastname`** and assign the appropriate $_POST autoglobal variable to it as follows:

```
if (isset ($_POST["lastname"])) $lastname = $_POST["lastname"];
```

6. In the same way, create and initialise variables for **`$age`**, **`$food`** and **`$partySize`**.
   (We will look at how to handle radio button and checkbox input in the next step.)

7. Add a statement to **echo** the variables to an HTML page so we can check they have been correctly assigned.

8. Save and upload **`processbooking.php`** again to the mercury server. ***Test your form.***

Reminder: To validate that the HTML generated by PHP is valid HTML5, we cannot directly upload the PHP to the validator. The PHP script must be processed by the PHP pre-processor on the server, create the HTML that is sent to the client. However we can view the source of the HTML page displayed in the browser, then cut and paste it to the W3C HTML5 validator https://validator.w3.org/nu/

9. ***Validate the HTML generated by your PHP script.***

Step 4: Handling input from radio buttons and checkboxes

Notice that if a value is not set in a radio button or a checkbox, then the element for that variable will not be set in the $_POST associative array. If we try to echo that we will get an error.

While a group of radio buttons that will output a single *name-value* pair, this will only happen if one of the options is selected. The server needs to handle a case where no option is selected.

1. For the species radio button we will use an **`if-else`** statement to handle the case that no selection is made. Immediately following the code you wrote in Step 3 write the following:

```
if (isset ($_POST["species"]))
        $species = $_POST["species"];
else
        $species = "Unknown species";
```

For the tour selection checkboxes the user may select multiple options. One way to handle this is to create a series of **`if`** statements that check if an option has been selected. If an option has been selected, append that option to a string that represents the selection (or you could use an array). For example:

```
$tour = "";
if (isset ($_POST["1day"]))   $tour = $tour. "One-day tour ";
if (isset ($_POST["4day"]))   $tour = $tour. "Four-day tour";
...
```

Complete the above code to display all check box options the user has selected.

2. Add some PHP code to check the user has selected a menu preference.

3. In a <p> element below the heading, add a confirmation output message for the user.
   It should read as follows (words in italics are examples of the data entered from the form):

   Welcome *Elanor Gamgee*!
   You are now booked on the *1 day* and *4 day* tours.
   Species: *Hobbit*
   Age: *40*
   Meal Preference: *Mushrooms*
   Number of travellers: *2*

4. Save and upload **`processbooking.php`** again to the mercury server.
   Test the data displayed when you submit your form.

5. ***Validate the HTML generated by your PHP script.***

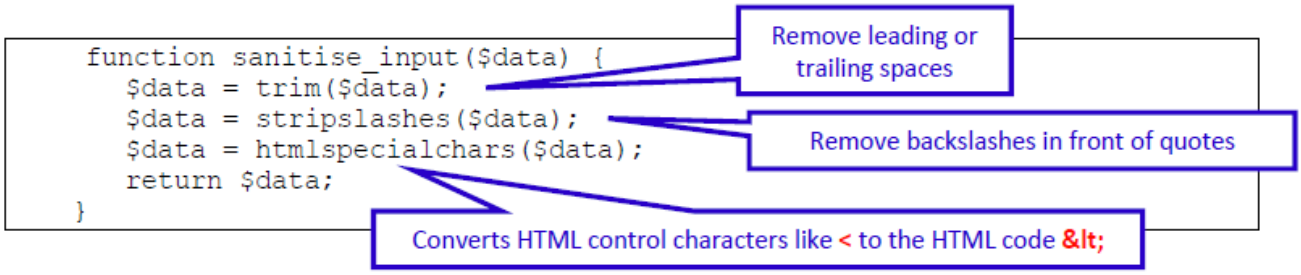
Step 5: Sanitise the input characters with a function

While we have successfully created a web page from the information we have received at the server, we do not yet have any server-side validation. If we are storing data to a database we need to make sure the integrity of the database is maintained. Lack of data validation at the server also makes the web site very vulnerable to attack using techniques such as code injection. We will look at these vulnerabilities in more detail in Week 11.

1. To illustrate how inserting HTML control characters into text can muck up the program, try filling the form by inserting a > character into the middle of the first name text, e.g. “Elan>or”.
   What happens to the output? Try with “<h1>Elanor</h1>”.

In this step we will write a function that "cleans up" such characters and removes any leading or trailing spaces from the variables. This user-defined function will call a number of in-built PHP functions.

2. You need to place *function definitions* within a PHP block. A good place to put them is at the top of your HTML file so they are easy to find. Remember, like JavaScript, the code within a function definition is not executed until it is invoked. Define the following function:



3. We now need to clean up the input data variables we have received from the form before we use them. Immediately before the output message you created in Step 4.3, invoke our function to clean-up the first name variable:

```
$firstname = sanitise_input($firstname);
```

In a similar way, invoke the function to sanitise **`$lastname`**, **`$species`**, **`$age`**, **`$food`** and **`$partySize`**.

4. Save and upload **`processbooking.php`** again to the mercury server. Test your code by entering various HTML special characters like ", < and ! in the first name and last name text boxes. Notice they are now rendered as plain text.

Step 6: Use Regular expressions to check the input data

While rendering special characters as HTML text can help prevent attacks from code injection, if there are data formats we want to enforce in our database (e.g. *age* is represented as a *number*), or business rules we want to apply (e.g. no traveller is younger than 18 years old), we need to write PHP server code to do this. We cannot rely on the client to do this safely.

Question:
If we have to do data-input validation at the server, why is it still a good idea to do it at the client as well?

In this step we will apply some of the same rules we checked using JavaScript in Lab 7

- First name is not empty and contains only alpha characters.
- Last name is not empty and contains only alpha characters or a hyphen.
- The age is an integer not less than 18 or more than 10,000.

In this step we will illustrate the use of regular expressions to check input strings. PHP has an in-built function **`preg_match`** that allows us to do this.

The general syntax is:

```
int preg_match ( string $pattern , string $variable)
```

Returns 1 if the pattern matches,
0 is no match, or FALSE is an error

1. The script below uses this PHP function to check that the first name consists of only alpha characters. The inserted code is highlighted in **blue**. Note that the pattern string **`^[a-zA-Z]*$`** is exactly the same one we used in HTML 5 and JavaScript .

```
$errMsg = "";

if  ($firstname=="") {
      $errMsg .= "<p>You must enter your first name.</p>";
}
else if  (!preg_match("/^[a-zA-Z]*$/",$firstname)) {
      $errMsg .= "<p>Only alpha letters allowed in your first name.</p>";
}
if ($errMsg != ""){
    echo "<p>$errMsg</p>";
}
else {
    echo "<p> Welcome $firstname $lastname ! <br />
        You are now booked on the $tour <br/>
        Species: $species<br/>
        Age: $age<br/>
        Meal Preference: $food<br/>
        Number of travellers: $partySize</p>";
}
```

Code from Step 4

2. Save and upload **`processbooking.php`** again to the mercury server. Test. Validate.
3. Extend the above code to check that the *last name* is not empty and contains only **alpha characters** or a **hyphen**.
4. Save and upload **`processbooking.php`** again to the mercury server. Test.
5. Write PHP code to test our rule that ages must be between 10 and 10,000.
   Hint: there is an inbuilt PHP function we can use called **`is_numeric($`*var*`)`** that returns a Boolean.
6. Save and upload **`processbooking.php`** again to the mercury server.
7. ***Test. Validate the HTML generated by your PHP script.***

Step 7: Redirection

1. In Step 3, we checked if the user had tried to enter the PHP without going through the form, by checking that the **`isset($_POST["firstname"])`** function. We displayed a message with a hyperlink back to the form. We can automatically force the user to go to the form page by using the PHP **`header()`** function to redirect them.
2. Replace the **`else`** block you wrote in Step 3 with the following:

```
else {
      // Redirect to form, if process not triggered by a form submit
      header ("location: register.html");
}
```

A PHP script cannot directly call another PHP script, so this sends a HTTP header response back to the client, and the client then sends a request for register.html

3. Save and upload **`processbooking.php`** again to the mercury server.
   Type the URL **`processbooking.php`** into your browser address text box.
   The web page displayed should be your register Web form rather than the confirmation page.

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