COMP284 Practical 1
HTML

Introduction

- This worksheet contains exercises that are intended to familiarise with HTML forms. While you work through the tasks below compare your results with those of your fellow students and ask for help and comments if required.
- You might proceed more quickly if you cut-and-paste code from that PDF file. Note that a cut-and-paste operation may introduce extra spaces into your code. It is important that those are removed and that your code exactly matches that shown in this worksheet.
- The exercises and instructions in this worksheet assume that you use the Department's Linux systems.
- To keep things simple, we will just use a text editor and a terminal. You can use whatever text editor you are most familiar or comfortable with.
- If you do not manage to get through all the exercises during this practical session, please complete them in your own time before the next practical takes place.
- Use the HTML5 Specification [4], the COMP284 Lecture Notes [1, Lecture 2], Stack Overflow [2] and W3Schools [3] as references for any information on HTML that you might need.

Exercises

1. Let us start with a 'login page'.

   a. Create a directory

   ~/public_html/

   in your Linux filestore using the command

   ```
   mkdir $HOME/public_html
   ```

   in a MobaXterm terminal connected to one one of the Linux servers.

   b. Use a text editor such as MobaXterm's built-in editor to create a file `page01A.html` in the your newly created `public_html` directory with the following content (content continues on the next page):

   ```
   <!DOCTYPE html>
   <html lang='en-GB'>
   <!-- File: page01A.html
        Creation: 2020-02-01
        Description: Forms
     -->
    <head>
     <title>Forms</title>
    </head>
    <body>
     <h1>HTML Forms</h1>
   ```

```
  <form method='post' action='https://cgi.csc.liv.ac.uk/~ullrich/COMP284
   ↪/examples/process.php'>
   <input type='hidden' name='role' value='student'>
   <label>User ID: <input type='text' name='id'></label>
   <input type='submit'>
  </form>
 </body>
</html>
```

This HTML document contains a HTML form with three forms elements, a hidden input, a text field, and a submit button.

c. Now we have to make sure that the permissions for various directories and files is set correctly, so that our web server can access them but nobody else can. In particular, your home directory and the `public_html` directory must be world-readable and world-executable, while the HTML file must only be readable by yourself. You can set the permissions correctly using the commands

```
chmod a+rx ~/
chmod a+rx ~/public_html/
chmod og-rwx ~/public_html/page01A.html
```

Instead of these Linux commands you can also use MobaXterm's permissions editor to set file and directory permissions as stated above.

d. Open a web browser, i.e. Google Chrome or Mozilla Firefox, and access the URL

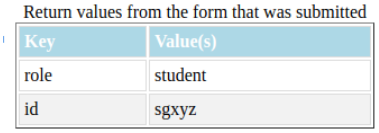
`https://student.csc.liv.ac.uk/~<user>/page01A.html`

where you need to replace `<user>` with your username. Figure 1a shows you what you should see.

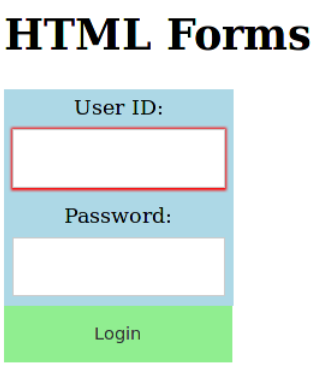
e. Enter your username into the text field then click on the 'Submit' button. You should now see the output of the PHP script `process.php` to which the inputs of the form have been submitted (Figure 1b). It should show you that there have been two inputs, one with key / name 'role', one with key / name 'id'. The first corresponds to the hidden input field, the second to the text field. The submit button itself did not produce an input to the PHP script as the corresponding HTML element has no name.

f. The quotation marks around each value in the output of `process.php` have been added to make it easier to see where a value starts and ends. To see that this makes sense for testing purposes, enter usernames that contain spaces at the start and the end.

(a) `page01A.html` (Initial)



(b) Output of `process.php`



(c) `page01A.html` (Styled)

Figure 1: Exercise 1

g. The form uses the `POST` request method. With the `POST` method, inputs become part of the body the HTTP request

```
POST /~ullrich/COMP284/examples/process.php HTTP/1.1
Host: cgi.csc.liv.ac.uk
role=student&id=sgxyz
```

Inputs become pairs *name=value*, with pairs separated by an ampersand symbol `&`. Both *name* and *value* are *URL-encoded*.

There are advantages and disadvantages to using `POST`: (i) `POST` requests do not remain in the browser history and cannot be bookmarked, (ii) `POST` requests have no restrictions on data length, and (iii) `POST` requests are suitable for the transfer of sensitive data, e.g. passwords.

h. Extend the form with an input element for a password and an appropriate label. Remember that there is a particular type for such an input element.

Test your solution. The password should not be visible but once submitted, `process.php` will show you what you have entered.

i. See what happens if you leave both inputs empty and just click on the submit button. The form will still be submitted and `process.php` will show that no value is associated with id and password.

Modify the form so that the user must enter both username and password before the form can be submitted.

j. Optional: Using CSS, we want to make this login page look nicer. Both the input element for username and password should have a width of 10em, padding of 1em, margins of 0.5em, and a 1px border. Labels should be placed centred on top of the input elements with a light blue background and an extra 4px padding on top. The submit button should be placed below the other elements and be as wide as those, it should have a light green colour and its label should be 'Login'. The end result should look similar to Figure 1c. Modify the HTML markup and provide style directives that achieve the desired look. Hint: `input[type=text]` allows you to select input elements of type text.

2. Next, let us have a quick look at check boxes and radio buttons.

a. Use a text editor to create a file named `page01B.html` in your `public_html` directory with the following content:

```
<!DOCTYPE html>
<html lang='en-GB'>
 <head>
  <title>HTML Forms</title>
 </head>
 <body>
  <h1>HTML Forms</h1>
  <form method='get' action='https://cgi.csc.liv.ac.uk/~ullrich/COMP284/
   ↪examples/process.php'>
    <p>What fruit do you like?</p>
    <fieldset>
      <label><input type='checkbox' name='fruit[]' value='a'>
             Apples</label>
    </fieldset>
    <p>Please select your preferred contact method:<p>
```

```
    <fieldset>
      <input type='radio' id='cc1' name='contact' value='email'>
      <label for='cc1'>Email</label>
    </fieldset>
   <input type='submit' value='Submit'>
  </form>
 </body>
</html>
```

b. Make sure that the permissions for the new file are set correctly.

c. Open a web browser, i.e. Google Chrome or Mozilla Firefox, and access the URL

`https://student.csc.liv.ac.uk/~<user>/page01B.html`

where you need to replace `<user>` with your username. Figure 2a shows you what you should see.

d. Select the checkbox and the radio button then submit the form. The PHP script `process.php` again shows you what then happens. See Figure 2b for the expected result. Make sure that you understand where the values come from.

e. In this example we have used the `GET` method for the HTTP requests created by submitting the form. With the `GET` method, inputs become part of the URL given in the `action` attribute. In particular, after submitting the form you should see the following URL in the brower's URL field:

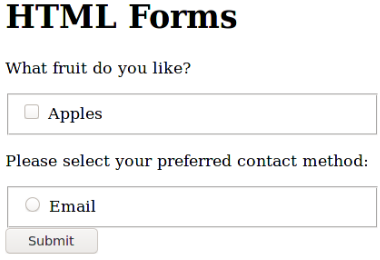
`https://cgi.csc.liv.ac.uk/~ullrich/COMP284/examples/process.php?fruit%5B%5D=a&contact=email`

You see that the inputs are appended to the `action` URL after a question mark symbol `?` as pairs *name*=*value*, with pairs separated by an ampersand symbol `&`. Both *name* and *value* are *URL-encoded*, for instance, `fruit[]` has become `fruit%5B%5D`.

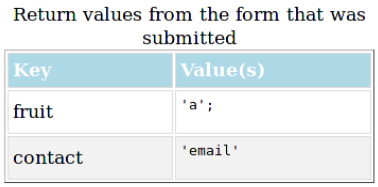
There are advantages and disadvantages to using `GET`: (i) GET requests remain in the browser history and can be bookmarked, (ii) GET requests have a length restriction, and (iii) GET requests should not be used for sensitive data, e.g., passwords.

f. One checkbox and one radio button is not particularly interesting. Add two more checkboxes for 'Oranges' and 'Peaches' with associated values 'o' and 'p', respectively. Also add two more radio buttons for 'Phone' and 'Mail' with associated values 'phone' and 'mail', respectively. The result should as shown in Figure 2c.

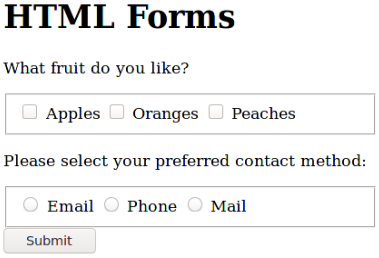
g. Test your solution. In particular, if you tick all three checkboxes and submit the form, then all three values 'a', 'o' and 'p' should be associated with 'fruit'.



(a) `page01B.html` (Initial)



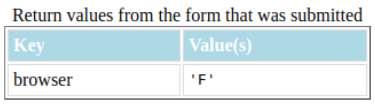
(b) Output of `process.php`



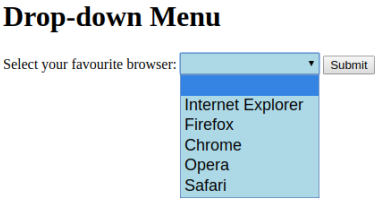
(c) `page01B.html` (Extended)

Figure 2: Exercise 2

(a) `page01C.html` (Initial)



(b) Output of `process.php`



(c) `page01C.html` (Styled)

Figure 3: Exercise 3

h. Note that the name attribute of all checkbox input elements is 'fruit[]' with '[]' indicating this is an array. Test what happens if you remove '[]' from all the name attributes and then check all the checkboxes. What does `process.php` show when you submit the form and why?

i. What happens if each checkbox input element gets a different value for its name attribute, e.g., 'fruitA', 'fruitO', and 'fruitP'?

3. Our final exercise deals with drop-down menus.

a. Use a text editor to create a new file `page01C.html` in your `public_html` directory with the following content:

```
<!DOCTYPE html>
<html lang='en-GB'>
 <head>
  <title>Drop-down Menu</title>
 </head>
 <body>
  <h1>Drop-down Menu</h1>
  <form action='https://cgi.csc.liv.ac.uk/~ullrich/COMP284/examples/
   ↪process.php'>
   <label>Select your favourite browser:</label>
   <select name='browser'>
     <option value='I'>Internet Explorer</option>
     <option value='F'>Firefox</option>
     <option value='C'>Chrome</option>
     <option value='O'>Opera</option>
     <option value='S'>Safari</option>
   </select>
   <input type='submit' value='Submit'>
  </form>
 </body>
</html>
```

Make sure that the permissions for the file are set correctly.

b. Open a web browser, i.e. Google Chrome or Mozilla Firefox, and access the URL

`https://student.csc.liv.ac.uk/~<user>/page01C.html`

where you need to replace `<user>` with your username. Figure 3a shows you what you should see.

c. Select the 'Firefox' entry in the drop-down menu and submit the form. The output produced by `process.php` in this case is shown in Figure 3b.

d. Load `page01C.html` again. By default, 'Internet Explorer' is selected in the drop-down menu. Without changing the order of options in the drop-down menu, make 'Chrome' the browser that is selected by default. Test your solution.

e. In the lectures it was mentioned that in most cases it is best if the user makes a 'conscious choice' instead of there being an option that is selected by default. To force such a choice, the first and selected option should be a 'dummy entry' with an empty value. Add the following option as the first to the `select` element and ensure that it is selected by default.

```
    <option value=""></option>
```

f. Save the modified file, reload `page01C.html` in the browser, select the empty option and submit. See what output `process.php` produces.

g. Obviously, the point of adding a 'dummy entry' was to force the user to consciously select one of the other entries. This currently is not the case as the user can leave the 'dummy entry' selected when submitting the form. Modify the drop-down menu so that the user cannot submit the form without selecting one of the other entries in the drop-down menu. Test your solution.

h. Optional: Drop-down menus are difficult to style. But we can change a few minor aspects of the way they look. We want the font size of the options to be 18px, options should have a width of 10em and both options and the selected entry should have a light blue background. Add style directives to achieve the desired look. See Figure 3c for what the result should be.

4. The last exercise is about *HTML tables*.

a. Use a text editor to create a file named `page01D.html` in your `public_html` directory with the following content:

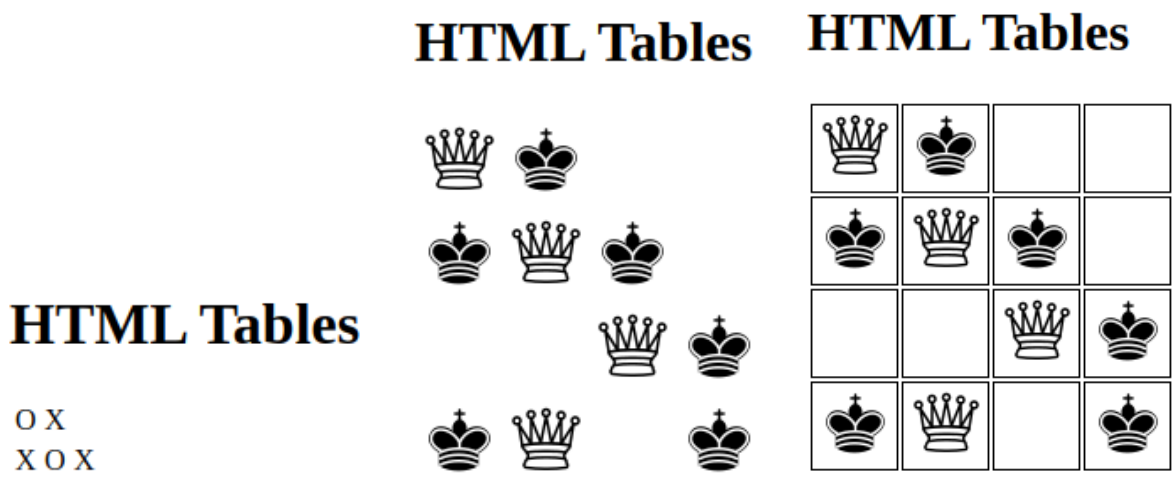
```
<!DOCTYPE html>
<html lang='en-GB'>
 <head>
  <title>HTML Tables</title>
 </head>
 <body>
  <h1>HTML Tables</h1>
  <table>
   <tbody>
    <tr><td>O</td><td>X</td><td> </td></tr>
    <tr><td>X</td><td>O</td><td>X</td></tr>
   </tbody>
  </table>
 </body>
</html>
```

b. Make sure that the permissions for the new file are set correctly.

c. In a web browser access the URL

`https://student.csc.liv.ac.uk/~<user>/page01D.html`

where you need to replace `<user>` with your username. Figure 4a shows you what you should see, namely, a table with two rows and three columns (without visible grid lines), contains the characters O, X, and space.



(a) `page01D.html` (Initial) (b) `page01D.html` (4e) (c) `page01D.html` (4f)

Figure 4: Exercise 4

d. Modify the HTML markup so that we get a table with four rows and four columns. Arbitrarily assign one of the characters O, X, and space to each table cell.

e. Modify the HTML markup so that instead of the character X the image found at

   https://upload.wikimedia.org/wikipedia/commons/thumb/f/f0/Chess_kdt45.svg/45px-Chess_kdt45.svg.png

   is shown and instead of the character O the image found at

   https://upload.wikimedia.org/wikipedia/commons/thumb/1/15/Chess_qlt45.svg/45px-Chess_qlt45.svg.png

   is shown. Note: You do not need to download the images for that.

f. Add a CSS style directive to the head element of the HTML document so that each table cell has the following style. There should be a solid black border 1px thick, the table cells should have a height and width of 3em, and the content should be horizontally and vertically centred.

References

[1] U. Hustadt. *COMP284 Web Programming: Lecture Notes*. Department of Computer Science, University of Liverpool. URL: https://canvas.liverpool.ac.uk/courses/59619/modules (accessed 24 January 2023).

[2] Stack Overflow. *Stack Overflow Site*. 24 January 2023. URL: https://stackoverflow.com/ (accessed 24 January 2023).

[3] Refsnes Data. *W3Schools Site*. 24 January 2023. URL: https://www.w3schools.com/ (accessed 24 January 2023).

[4] WHATWG, ed. *HTML Living Standard*. 24 January 2023. URL: https://html.spec.whatwg.org/multipage/ (accessed 24 January 2023).