Lab 02 – HTML Markup and Forms

Aims

- Use a validator to find errors on an HTML page
- Use a text editor to markup content with HTML5
- Use a text editor to create a valid HTML5 form
- Use HTML pattern attributes and other attributes to check the data entered by the user

Task 1: More Validation

In this task you use the HTML validator to find errors in an existing HTML file.

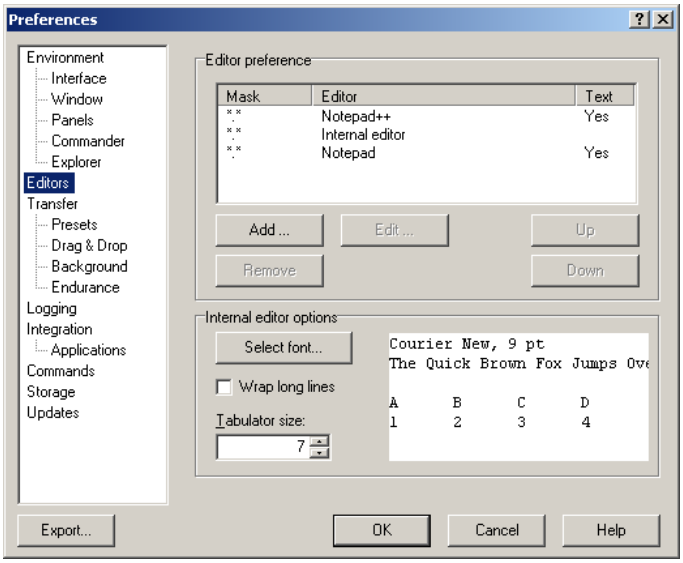
- Create a folder called **`lab02`** on your local drive.
- Create a new folder **`lab02`** under the unit folder on the mercury server ~/<unit code>/www/htdocs where <unit code> is your unit code, eg. cos10011.
- Copy the file **`lab02.zip`** from Canvas into your **`lab02`** folder and decompress it.
- Load the file **`bad_html.html`** into Firefox. Right click in the page and select 'View Source'.
- Some of the errors in the HTML should be highlighted in Red.
- Load the file into the Nu HTML checker and view the errors and warnings.
- Fix the errors and warnings and then save the file.

Note: Validators may not identify all the errors in the file, and they may not correctly identify the line where the error occurs. Errors can be cascading. For instance, forgetting to close a tag on one line means the parser does not correctly understand subsequent lines of markup.

Editing source code files directly from Mercury.

Rather than creating and saving your HTML (or any other source code) locally then uploading it to the server you can directly edit files on the server using WinSCP – just double click on the file in the server pane.

If the text comes up in the WinSCP's internal editor rather than Notepad++, set Notepad ++ (or code editor of your choice) as the default editor in WinSCP. From the menu select **`Options | Preferences`**. Click on **`Editors`** in the dialog box as shown, then **`Add`** a reference to the Notepad ++ executable. (Usually found under 'Program Files (x86)/Notepad++/notepad++.exe) Click **`Up`** to move Notepad++ to the top of the list.



Task 2: Marking up an existing text file

In this task we will use a wider range of HTML elements to markup an existing Web page

Step 1:

Use any text editor on your local computer (e.g. NotePad++), copy the text file **`outline.txt`** from **`lab02.zip`**. Save the file as `outline.html` in your working folder.

Step 2: Create the HTML structure and header

- At the start of the text file add a document declaration
- Create a `<html>` root element with a `<head>` and `<body>` children
- In the `<head>` element create meta-tags for
  - charset
  - description
  - keywords
  - author
- Give the page a title using the appropriate element,
- Because the text from `outline.txt` is not yet marked up we will place it between comment tags with the <`body` > element.

Hint: Start with all the raw 'outline' content between comment tags, then move the comment tags, as you markup the content.

Step 3: Validate the HTML

Upload your html to https://validator.w3.org/nu/ to validate it.

SUGGESTION: Do a little bit of markup then validate, markup, validate, markup, validate, …
As well as uploading the file, view the page in Firefox, then right-click to view the page source.

Step 4: Markup the content

Markup the text content using the elements discussed in Lectures, remembering to *validate* as you go.
The format to be used is shown in `outline.pdf` (in **`lab02.zip`**).

The elements to be used here are

- Heading <h#>…</h#>
- Paragraph ..
- Horizontal Rule <hr />
- List <ol>.., …
- Table <table>…</table>,
- Image
- Anchor <a … >…
- Special characters as needed
- Footer

For instance markup using heading elements such as:

```
<h1>Unit of Study Outline …</h1>
<h2>Learning Outcomes</h2>
```

When you have finish the markup, use the validator to check it is valid HTML.

Step 5: Load the file to the server and test

Using WinSCP, drag and drop the file '`outline.html`' from your local machine to the *`htdocs/lab02`* folder.

Step 6: Test and view web pages.

To view the pages through http, use any Web browser and type in the following address,

`https://mercury.swin.edu.au/<your unit code>/<username>/<folder>/<filename>`

The <*username*> is **s**< *your 7 or 10 digit Swinburne ID* > and <your unit code>. For example

`https://mercury.swin.edu.au/cos10011/s1234567/lab02/outline.html`

When the authorization request dialog pops up, use your ***SIMS username and password*** to confirm access.

Task 3: Create a HTML form

Step 1: Create the HTML file structure

Following the procedure set out in previous labs, use a suitable text editor on your local computer (e.g. NotePad++), to create an HTML file called `helpdesk.html` in a `lab02` directory in your working folder. Make sure you include the following elements to define the basic structure and <head> elements:

- At the start of the text file add a document declaration
- Create a `<html>` root element with `<head>` and `<body>` children
- In the `<head>` element create meta-tags for: charset; description; keywords; author
- Give the page a title "CWA Help Desk Appointment" using the appropriate element.

Upload your html to https://validator.w3.org/nu/ to check that is is valid.

Step 2: Add the content to the form

From the file **`lab02.zip`** copy the image file logo.png into your `lab02` folder and view the file **`example_form.html`** in Notepad++. *Using this file as an exemplar*, and referring to other resources (e.g. Lecture notes or W3Schools), create the form shown below (next page). Make sure the content is inside the <body> element you have created in Step 1.

For the `<form> … </form>` element the attributes should be **`method="post"`** and **`action="https://mercury.swin.edu.au/it000000/formtest.php"`**
The `formtest.php` script just echoes back the input data (name = value pairs) sent to the server when the submit button is clicked.

> Hint: Implement your submit button `<input type="submit" value="Book"/>` before you create the other elements in your form. Then as soon as you develop other input elements you can test them to see if they are correctly sending their name-value data to the server.

When creating your form keep in mind the following:

- <label>s within the form should link with their <input> controls using a **`for`** attribute that references the **`id`** attribute of the associated input control.
- Use form elements such as <input …/>, <select> … </select> , <text area> … </text area> as appropriate.
- Don't forget to define **`name`** attributes for **all** your inputs otherwise nothing will be sent to the server
- Remember Radio buttons that are in a group all need to share the same **`name`** attribute,
- Define the **`name`** attributes of associated checkboxes as an array.
- Use the **`checked="checked"`** attribute to make the HTML checkbox ticked by default.
- Use <fieldset> … </fieldset> and <legend>…</legend> to appropriate group input controls and labels.
- Use **`placeholder`** attributes on the *Description* `textarea`, and *Date* `input` and *Time* `input` as shown in the illustration on the next page.

`logo.png` (in zip file)
Make this logo hyperlink to
http://www.swin.edu.au
(hint: see Lecture 2 slides)

Heading 1

Paragraphs

Use the **for** attribute of the label to link it to the respective inputs

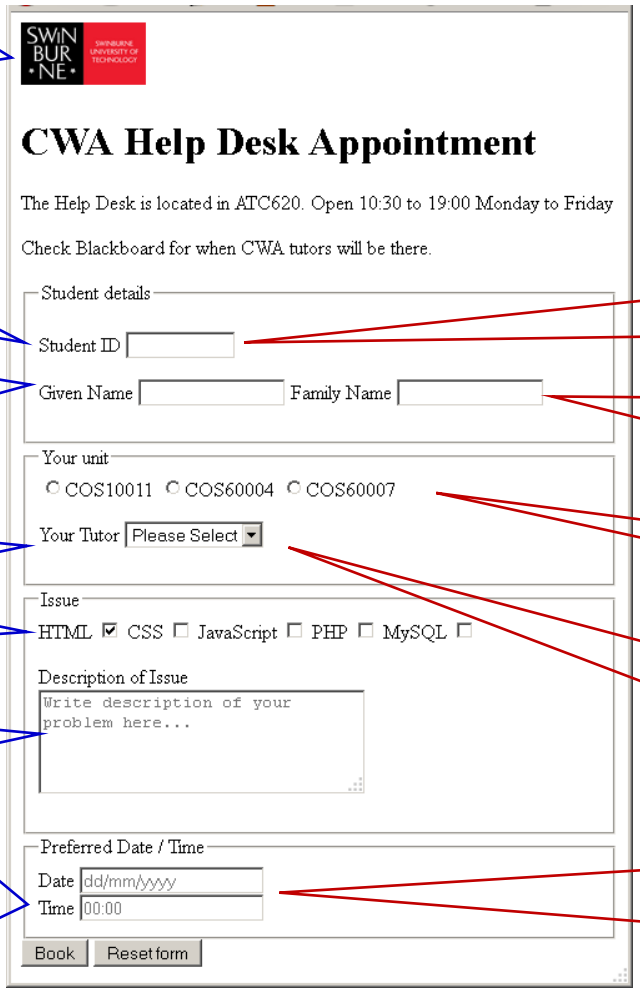
Maximum of 15 characters allowed for given and family names

Make up a list of tutor names as options for your select dropdown list.

Set this checkbox as ticked by default

Use the placeholder attribute to prompt the user

Use <input type="**date**" .../> and <input type="**time**" .../> elements here. *View the Web page in Firefox and in Chrome. What's the difference?*
Change the type to "**text**" and define the placeholder attributes as shown.



Step 3: Data Input Checking

Set the **required** attribute
Use the **pattern** attribute to ensure between 7 and 10 digits are entered.

Given and family names are required. Only alphabetical characters allowed

Set at least one of the radio buttons to required="required"

Make the select element required.
Hint: make the first option value an empty string
<option value="">Please Select</option>

Use the **pattern** attribute to ensure entered data matches the placeholder. Don't worry about using patterns to check the range.
(Hint: see Lecture 2 slides)

Regularly validate the HTML you are entering by viewing it in Firefox, then right-clicking to view the source to see if any errors are identified.

Upload your html to https://validator.w3.org/nu/ to validate it.

Step 3: Check the input data

Using the Lecture notes as a reference, add **`required`** and **`pattern`** attributes to ensure the data input formats are enforced as defined in the red callout boxes on the right of the figure above.
For example, the time input could be

```
pattern="\d{2}:\d{2}" or better still
pattern="([01]?[0-9]|2[0-3]):[0-5][0-9]"
```

where (a?b|c) is a conditional pattern if a then b else c.

Pattern symbols you might find useful

^	**Start of string**
$	**End of string**
.	**Match any character**
[a-z]	**Match the range**
\d	**Match a digit from 0 – 9**
a?	**0 or 1 instance of** a
a*	**0 or more instances of** a
a+	**1 or more instances of** a
a{3}	**exactly 3** a's
a{3, }	**3 or more** a's
a{3,6}	**between 3 and 6** a's

Step 4: Upload the html source of the form page to Mercury

Upload the file **helpdesk.html** and **logo.png** to your lab02 folder using WinSCP or similar.
As in the previous tutorial view the page in Firefox at the following address,

https://mercury.swin.edu.au/*<your unit code>*/*<username>*/lab02/<filename>

Step 5: Revalidate

Upload the html to https://validator.w3.org/nu/ to check it.

[IMPORTANT] Send your tutor the link to your web page running on the Mercury server to be marked off.

Task 4: Checking HTML is well-formed XML (Optional)

Here are a couple of ways to check this:

Method 1:

Step 1: Saving a local copy of your file as .xml

First recheck that your file `outline.html` *validates to HTML5 as is Task 4, and is loaded on the server.*
Open the page in the browser, right click the page and choose "View Page Source" in the context menu, and save the page source as a local copy (**`Save as type: HTML only`**) named `outline.xml`.

Step 2: Open the xml file in your browser

Open this local xml file in the browser. If the file is xml compliant the source xml should be displayed as a tree in the browser. (If there are errors: some browsers might display nothing, otherwise the first syntax error. Usually a 'XML Parsing Error: mismatched tag'.will be displayed.)

Step 3: Fix errors

If there any errors, fix them ***in your html file*** on the server, so it is well-formed XML.
Reload the `outline.html` file in the browser, "View Page Source", and again save a local copy as `outline.xml`, and recheck it again. The errors may be cascading so they may not necessarily be on the line indicated by the browser error message.

Note: *Later, when we use PHP, we will use this* ***same process*** *to locally check that the served pages are* ***xml*** *compliant.*

Note: This process is checking that the XML is "well-formed". When we say an XML file is "valid" we mean the file will validate against a pattern for that type of XML file – either defined by rules in a Document Type Definition (DTD) or in an XMLSchema (xsd) document. These topics are covered in more advanced Web units.

Method 2: (Optional exercise)

Step 1: Serving the HTML5 page as XHTML

As you can see, this HTML5 / XML testing process can be tedious. Another way is to change the file extension *on the server* to .xhtml. How does the browser respond now? Is the browser treating the file as html or xml?

Make a small XML syntax error in your .xhtml document (e.g. delete an end-tag from one of the elements).

How does the browser respond now?

Step 2: Validating and adding a namespace

Fix the error you made in Step 5 above, then validate the .xhtml file using "Web Developer Extension" Tools/Validate HTML. The response will show you that the page is now being served as "application/xhtml+xml" rather than "text/html". It now needs an 'xml namespace' (an 'xmlns') to be valid.

Change the line `<html lang="en">` to
`<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="en" lang="en">`

The xmlns attribute is ***required*** in XHTML, and is ***optional*** in HTML5. This code tells the browser to treat the .xml according to the rule of the "xml name space" – in this case as xhtml.

Reload and view the file in the browser – it should now be rendered as a normal webpage.
It has been served as xhtml, and we know that it is ***both*** valid **HTML5** and valid **XHTML**
and hence well-formed / compliant **XML**.
Files than can be served as HTML or XML are known as polyglot documents.