Edinburgh Napier University

SET08101/401 Web Tech

Lab 2 - HTML

Dr Simon Wells

1 Aims

At the end of the practical portion of this topic you will:

- Create HTML documents
- Add content to HTML documents using elements
- Specialise elements by setting their attributes
- Create *hypertext* documents by using links
- Have some small examples of HTML websites that you can build on and elaborate over subsequent week, partly as a learning exercise, partly to kick-start your development of a personal portfolio, and partly to stimulate your interest in the variety of types of sites we can implement for the web.

NOTICE: You should be putting your lab work into Git (Add, Commit, Push) as you develop it. This is active & deliberate practise and means that you are constantly building skills so that you no longer have to waste cognitive energy on them; you just use them. If you are looking for a good Git tool then Git-Bash is probably the best and available from here: https://git-scm.com/download/win**. If you're using a lab machine, then rather than the default that tries to auto- download, I'd choose the portable "thumbdrive" edition. You can then follow this tutorial to use git-bash with GitHub:** http://vastinfos.com/2016/09/ quick-github-and-gitbash-basics-for-beginners-tutorial/

2 Activities

You should, unless otherwise indicated, try out all of the code examples shown on lab sheets for yourself. Whilst this lab sheet can describe an outcome, this is no substitute for experiencing that outcome for yourself and allowing it to spur some experimental exploration.

2.1 HTML Documents

Last week we saw a simple HTML document that looked like this:

```
1 <!DOCTYPE html>
2 <html>
3     <head>
4         <title>Web Tech</title>
5     </head>
6     <body>
7         <h1>Hello Web Tech</h1>
8     </body>
9 </html>
```

With the exception of the **content**, "Web Tech" and "Hello Web Tech", this is probably as simple as a minimally useful HTML document can be. A slightly more generically useful bit would look like this:

```
1  <!DOCTYPE html>
2  <html>
3      <head>
4          <title>Web Tech</title>
5      </head>
6      <body>
7          <h1>Hello Web Tech</h1>
8          <p>Something about Web technologies?</p>
9      </body>
10 </html>
```

By adding the <p> tags we've added a section for a paragraph to go with our heading, <h1> tags. You could edit the content to be any text that you wanted your page to be about. Why not go ahead and do that now?

We can, and will, add more tags to our pages as develop more capable sites, but we can also simplify that some more by removing tags. However, it doesn't look very interesting in the browser if we remove all of the content. We can remove the head, body, title and heading tags and their associated content to give a minimal valid HTML document as follows:

```
1 <!DOCTYPE html>
2 <html>
3 </html>
```

We can also remove various other aspects of our HTML to varying effect. Try experimenting with this to see what effects this has. Take your example HTML and experiment with removing various elements then looking to see what the effect is in your browser. You should also see how the browser reacts when you have an empty .html file and when you have a .html file that contains content text that isn't tagged at all. Notice how the browser does it's best to provide the reader with something useful? It is worth trying all this in a couple of browsers if you have some available so that you can see what effect different HTML problems might have on the resulting rendered page. At the very least you should experiment with adding content that isn't surrounded by tags, omitting entire opening, and closing, tags, and omitting parts of a tag. For example, what happens when you forget to close a tag properly and leave off the final >? What happens if you do this on the opening tag of a pair rather than the closing tag? Why do you think there might be a difference? How do different browsers handle these circumstances? You can also try duplicating or repeating some of the tags and content in the page and viewing the results. This will all start to give you a feel for the work that the browser does in taking an HTML file and turning it into a rendered Web page. The browser is really doing a lot of handholding and fixing to try and take even the most malformed HTML and still enable the content of that HTML to be viewed usefully.

Whilst trying these exercises you should also have the browser's console window open so that you can see any messages that the browser prints.

2.2 HTML Elements

It is from this minimal set of tags in the HTML document example above that all possible valid HTML documents are built. One of the core ideas in HTML is that it is a tree structure. It starts with a root tag pair, the HTML tags, which enclose other pairs of tags, usually at this stage the head and body tags, which in turn enclose other pairs of tags, until you're done, i.e. you've described all of the text that you want the document to hold. These tag pairs are called Elements. Each element has a start tag and an end tag, for example, a start tag might look like this: <title> and an end tag might look like this: </title>. Notice that the sole difference between the two tags is a single backslash character.

Note that elements usually have content between start and end tag. If an element has no content, then it is called an empty element. Empty elements do not have an end tag, because they are not enclosing anything. An example of this is the line break element
 which stands on its own. Many people include the closing element when using empty tags, e.g.
. One this we should notice is that browsers are quite generous in what they will accept as valid HTML. This is both a blessing and a curse. A blessing because it means that lots of people can write nearly correct HTML and thus create their own websites. This means that there is a very low barrier to entry to publishing on the web which has arguably been to its advantage. However, it means that browsers have had to be a bit creative about how they render supplied HTML. This means that historically, it has been difficult to get a designer's vision to render in exactly the same way across multiple browsers because each has made its own decision about how to treat the elements that make up HTML.

2.3 HTML Attributes

Attributes are additional "settings" for an attribute, they enable you to specify how it behaves or looks. All HTML elements have attributes which provide additional information about that

element. Attributes are always specified in the start tag and always come in name/value pairs, e.g. name = "value". For every HTML element that we discover there will usually be many attributes available and you should explore the range of attributes available. As you discover new HTML elements over the remainder of the trimester, make sure to look up the element on the Mozilla Developer Reference Pages[1] to see what attributes are available.

2.4 Basic HTML Elements

Some of the most basic elements relate to core text markup. Methods for indicating what role a given piece of text plays in the wider document. Just like writing a word processor document, where we use headings, paragraphs, etc. We do the same in HTML. These basic text markup elements are generally used between the <body> tags of the document.

The heading tags are defined for six levels of heading, ranging from <h1> the most important heading, to <h6>, the least important, e.g.

```
1 <!DOCTYPE html>
2 <html>
3     <head>
4         <title>Headings</title>
5     </head>
6     <body>
7         <h1>A Very Important Heading</h1>
8     </body>
9 </html>
```

- Explore the other headings that are available and see how they look in the browser
- Make an HTML document that shows all the heading elements on one page.
- Experiment with a variety of combinations of heading elements to see how these simple tags make a huge difference to the presentation of a page.

Paragraphs are a basic building block for organising writing within a larger textual document so there is also a paragraph element using the <p> tags.

- Make an HTML document that incorporates several paragraphs of text. You can select some text from a news website, or better yet, investigate "greeking" or "lorem ipsum", a way to generate blocks of text for testing websites before the 'copy', the content of the page, is written. There are many websites that generate lorem ipsum text as a service for you to copy and paste into website designs whilst you are waiting for the content to be written. This can be very useful as you will often want to start experimenting with page layout and presentation before you have the full content of your site. Often a site grows over time, so developing a design that doesn't rely on specific content, but looks good regardless, is useful.

We probably want to have some more visual appeal that just using text in our html document. We can include images very easily using the <img> tag, e.g.

```
1 <!DOCTYPE html>
2 <html>
3     <head>
4         <title>Headings</title>
5     </head>
6     <body>
7         <img src="https://raw.githubusercontent.com/siwells/webtech/master/
              resources/vmask.jpg" />
8     </body>
9 </html>
```

We can either use a link to an image from a website on the web, e.g. https://raw.githubusercontent.com/siwells/webtech/master/resources/vmask.jpg as in the example above, or we can use a local image file. If you do use a link to an image that is someone else's site, then this is often considered bad form. You also have no control over what your link actually points to, and the image could be replaced with something else at any point. The best-case scenario is that the image link becomes broken.

[1] https://developer.mozilla.org/en-US/docs/Web/HTML/Element

Download the vmask image above or use another of your choice and put it in the same folder as your HTML file. Rather than a "fully qualified domain name" like we had above, we can now just use the local filename. e.g.

```
<!DOCTYPE html>
<html>
    <head>
        <title>Headings</title>
    </head>
    <body>
        <img src="vmask.jpg" />
    </body>
</html>
```

Note that you can place image into a subfolder, but you will have to specify the path to that folder in your HTML, e.g. if the image is in the images sub-folder then:

```
<!DOCTYPE html>
<html>
    <head>
        <title>Headings</title>
    </head>
    <body>
        <img src="images/vmask.jpg" />
    </body>
</html>
```

- Try out some different ways to organise your images using basic HTML layouts elements. You might want to investigate, for example, the table element.

There are many HTML elements, too many to list here, and too many to adequately explore in a single lab. Instead, explore the list of elements on the Mozilla Developer Reference Pages[2] and get a feel for the range of elements that can be used to mark up an HTML document.

2.5 HyperText

Arguably the most important element in HTML are the tags that support hypertext. These enable links between pages. They put the "Hyper" into HTML which would otherwise be Text Markup Language (TML) and are also responsible for putting the "web" aspect into the World Wide Web.

Links are described using the <a> tag which encloses a section of text, or some other element. The href, or hypertext reference, attribute is then used to specify a Uniform Resource Locator (URL) or "web address" which indicates where the link points to. For example:

```
<!DOCTYPE html>
<html>
    <head>
        <title>HTML Links</title>
    </head>
    <body>
        <a href="https://siwells.github.io/webtech/">Module Support Website</a>
    </body>
</html>
```

We'll create a piece of text in the body of our html document which links to our module webpage. Load it up in a browser and click the link. It should take you to our module webpage. Try adding another link to another website of your choice. What other tags might you need to use to arrange these links in a nice way. As we design our web sites we must consider how and when we want a link to work, sometimes we might want to list a collection of links, but at other times we might want to include our links inline in our paragraphs of text.

- Create an HTML document that contains a list of links to your favourite 5 websites (if you don't have 5 favourite websites then just find 5 useful links that you can use)

[2]https://developer.mozilla.org/en-US/docs/Web/HTML/Element

- Create an HTML document that contains the following paragraph of text:

  Web Tech is a module that is run in the School of Computing, Engineering and the Built Environment at Edinburgh Napier University.

  Turn the 'Web Tech", "School of Computing", and "Edinburgh Napier University" phrases into appropriate links.

You can also make many other HTML elements into links. In a sense, Hypertext is not just text but other media, for example, images. So the idea of Hypertext gives way to a more general sense of Hypermedia. You can make a clickable image by enclosing an image element in <a> tags.

- Create an HTML document that contains an image and make it into a clickable link
- Explore other HTML elements to see which can be made into hyperlinks (you might have to do some background reading at this point [and the situation might be dependent upon the browser you're using].

You can also create links between local pages, i.e. pages on the same site. For our purposes now we'll consider local to mean within the same folder hierarchy.

- Create a folder and add an index.html to it. Add some further html files to the folder with different names, e.g. 01.html, 02.html, etc. Now add text and links within each file to enable you to open the index.html and navigate to the other files. Consider how you might navigate back to the index.html.

Links provide not only the basis for Hypertext but enable us to break down our own site into separate pages and navigate between them. For this reason, they are possibly one of the most important tags available in HTML. We can do without styles, or ways differentiate a paragraph from a heading, but we can't do without hyper-links.

2.6 Challenges

In the following, don't worry too much about making your pages look nice. We will address that in other labs. For now, think about structure and content, presenting your ideas in a clear way using pure HTML. We will return to these challenges over subsequent weeks to elaborate on them with design and interactive elements as appropriate so don't ignore them. By the end of this module, you will have developed your own set of websites of various types and purposes. Think of these not just as a framework for meeting the various learning objectives of the module, but also as a way to start building your own portfolio of work. There is nothing to stop you elaborating on any of these exercises and making your own, unique demonstrators of your skills as we progress. If you are happy with what you've built then you can consider deploying some, or all[3] of your sites[4].

1. Create a set of simple and short web pages to describe yourself and your career at Napier. If that sounds boring you can choose another topic, perhaps a hobby or something else that you are interested in. Where appropriate you should include links to external resources, e.g. to various pages in the Napier website. You should be considering how to break things down into separate pages, and also what internal links you might need for navigation within each page and between your pages. For example, starting within a homepage, i.e. index.html you might have links to a personal page about you, and a course page about your degree. Your course page might be broken down by year, and each year in turn by module. At each point you should be considering the best HTML elements to apply to markup your page. Don't worry too much about how things look at this point, next week we'll look at prettifying things using CSS, what we want for now is nicely organised information that we can navigate in the browser. The important thing to consider in this exercise is how to develop and expose the "hypertext" structure of the information.
2. Find the CIA world fact book website. You might find a search engine useful for this. The factbook contains information about various countries around the world. Have a look at the kind of information available and design a simple set of pages that reproduces some of

[3] or even none :D
[4] We'll consider deployment of websites later in the module

the factbooks content. One approach might be to start with an index or "home" page that introduces things and lists the countries that you'll cover. For each country you will create a different page containing information about the country. Consider the various ways that this information can be organised and interlinked to provide different ways of navigating through the text. Try to come up with a strategy for including links in your pages to allow you to navigate amongst them. If necessary, edit your homepage to reflect this change in navigation. For example, you might have different pages that organises countries by population density, or GDP, or geographical area.

3. Create a set of pages that implement a simple "choose your own adventure" style game. The 'home' page should introduce your story and set the scene, at the bottom of the page there should be links to other pages that further the story in different ways, depending upon which link you choose. Your reader/player should be able to make selections and move through the various pages/storylines, until you reach a termination page that concludes the story. The concluding page should have a link back to the home page so that a player can start again. One way to get started is just to break an existing story down into several pages in which each page has a link to the next page. Once you have that storyline implemented, return to an earlier page and alter the link to make it a choice between options, then implement pages for the alternative options, in each case adding whichever pages are necessary to complete the storyline. In some cases, you might want to loop back to an early place in the story to allow the user to make another choice, if this works with the overall story that you're telling. In other cases, choices might lead to different conclusions or different paths might lead to the same ending via various storylines. You might want to sketch out some ideas on paper to help you keep this increasingly tangled web organised.
4. For each of the challenge sites above, create a new repository, add the files for that site and push them to GitHub. Activate Github Pages for each site/repository and bask in the satisfaction of knowing that you just added a few new Web sites to the World Wide Web.

Don't worry if you're not sure how to do all of what you want to do yet. This is why these exercises are challenges. It is also fine to "build one to throw away". Try out a test spike to get something working on screen, then, early in the process, before you've put in too much effort, decide whether things are developing the way you want and whether to continue or start again. Often in trying a test spike we learn enough about the problem and potential solutions that we are better equipped to make a second, improved, attempt.

Speak to your classmates about how they are approaching things and see if you can come up with good ideas. You can also research how others have built similar sites online to see if you can gain inspiration. Remember that when you look at websites online, you can also inspect the code that drives the website, the HTML sourcecode, and in many cases, although developers will sometimes try to hinder you, the CSS, JavaScript and other resources, are also available to inspect, so you can see how others have achieved things. If you find something that you like, then you should consider how to incorporate those aspects into your own sites where it is appropriate to do so[5].

2.7 Finally

If you want some additional practise then the W3Schools HTML exercises are a great place to start:

- HTML Exercises: https://www.w3schools.com/html/exercise.asp

[5]Note that this doesn't mean to copy and paste another's code, but to take inspiration for how to implement your own solution to a given problem.