HTML / CSS – Lab Three – HTML5, CSS3, Media Queries

Objectives

1. Understand the use of HTML5 semantic elements
2. Improve design aesthetics with CSS3
3. Understand how to add HTML shim for older browsers
4. Review HTML form elements
5. Understand how to apply media queries to create a responsive design

Use HTML5 Semantics

In the *index.html* initially (to be rolled out to the other files) identify and change the following div's to new HTML5 semantic elements.

`<div id="header">` to `<header>`
`<div id="nav">` to `<nav>`
`<div id="footer">` to `<footer>`

Update the CSS file to use HTML selectors rather than ID selectors.

Testing the Pages in Older Browsers

Test the pages in Chrome and Internet Explorer 9. In IE press F12 to open the developers tool bar. Change the Browser Mode to IE8. What effect does it have? Using IE conditional logic fix any issues that might arise.

```
<!--[if lt IE 9]>
<style type="text/css">
#logo h1{
     font-size:28px
}
</style>
<![endif]-->
```

Adding the HTML5 Shim

Older versions of IE don't understand the new HTML elements. There is a file called *html5shiv.js* in the *scripts* folder. This was downloaded from *http://code.google.com/p/html5shiv/* and should be attached to allow older browsers to at least style new HTML elements as blocks or inline as appropriate.

```
<!--[if lt IE 9]>
    <script src="scripts/html5shiv.js"></script>
<![endif]-->
```

To see what the page would look like in IE8 we can use the built-in emulator in IE11. In IE11 press F12 to open the developers panel, find the Emulation options and switch the document mode to '8'.

@font-face

Google Fonts provides a good range of free fonts that can be added to your stylesheet. In the Quick Use icon - notice the option to use `<link>` or `@import`.

Use @import to add the font style to the top of 'main.css' stylesheet ie:

```
@import
url(http://fonts.googleapis.com/css?family=Coda+Caption:800)
;
```

Then add the font to the CSS:

```
body {
     margin: 0px;
     padding: 0px;
     background: #15384C;
     font-family: 'Coda Caption', sans-serif;
     /*font-family:Helvetica, Arial, sans-serif;*/
     font-size: 15px;
     color: #545454;
}
```

CSS3 properties

Experiment with CSS3 properties such as box-shadow and border radius. For example add a rounded corner to the container and header. Try improving the design of the 'Read More' button in *presentation.html*.

For example the following will produce a button style:

```
.button-style {
     display: inline-block;
     margin-top: 20px;
     padding: 7px 20px;
     background: #CF4A41;
     border-radius: 5px;
     text-decoration: none;
     text-transform: uppercase;
     color: #FFFFFF;
}
```

Notice that when tested in IE8 the 'new' CSS3 properties are not rendered. The HTML5 Shim does not provide a fix for these. Many of the missing features can be added with various polyfills. Alternatively if a 'progressive enhancement' approach is taken, then these aesthetic improvements are not fundamental to the design, and therefore it is argued, there is no need to try and replicate them in older browsers.

IE Conditional Comments

The HTML5 shim makes use of IE conditional comments. These are HTML comments that older versions of IE interpret such that style rule can be applied to 'bug fix' issues in older version of IE.

This IE conditional commend `<!--[if lt IE 9]>` was used by the HTML5 shiv but we can also use it to add some styles that will only be applied if the browser version if it is less than IE9.

```
<!--[if lt IE 9]>
<script src="scripts/html5shiv.js"></script>
<style type="text/css">
#logo h1{
     font-size:28px
}
</style>
<![endif]-->
```

Building a Form

Edit the HTML page called *contact-us.html* and add a HTML form with the following elements:

`<form>`, `<fieldset>`, `<legend>`, `<label>`

`<input>` - types text, email, tel, number, radio, checkbox, submit

`<textarea>`, `<select>`, `<option>`

When adding form elements it is important that they have a `name` attribute. This becomes essential later when the form values are submitted to be processed in some way - for example by a PHP script. The `name` attribute and the value entered by the user are referred to as a name / value pair.

For example the following creates a simple text field with a `name` of `surname`.

```
<input type="text" name="surname">
```

If the user enters a value of *bloggs* then the name / value pair submitted is `surname=bloggs`.

Tip: See http://www.mustbebuilt.co.uk/moving-to-and-using-html5/fabulous-forms/ for a review of the core HTML form elements you should get familiar with.

Styling Elements

Create styles for the form elements like <label> ie:

```
label{
     display:inline-block;
     width:150px;
}
```

Responsive Web Design with Media Queries

In order to optimize the web site for viewing on a mobile device we should create a stylesheet to change some of the layout settings.

The 'mobile' stylesheet can be called via a 'media query'. A media query interrogates the browser to find out a whole range of different pieces of information. Screen width and specifically min-width and max-width enable us to ascertain whether a visitor has a limited screen size.

Add the following before the </*head*> in the index.html page.

```
<link rel="stylesheet" type="text/css" media="only screen
and (min-width:50px) and (max-width:600px)"
href="styles/smallScreen.css">
```

This is a standard `<link>` used to attach a stylesheet but it has the media attribute which is used to test screen dimensions. If the screen has a minimum width of 50px and a max-width of 600px then a stylesheet called *smallScreen.css* is applied to the document. The *smallScreen.css* will add to the set of rules already applied to the document. Given that *smallScreen.css* is added before the closing </head> it is last set of styling rules applied to the document. As such any rules that conflict will defer to *smallScreen.css* as it has a higher order of precedence.

Create a CSS file called `smallScreen.css` and add the following simple rule to test the page.

```
body{
     font-family:"Times New Roman", Times, serif;
}
```

This simple rule will change the default font-family but only when the screen is small enough.

Fixing the Viewport

In order for the pages to work well a mobile device we have to add a `<meta>` tag to the `<head>` of the file that will control what is known as the 'viewport'.

```
<meta name="viewport" content="width=device-width, maximum-
scale=1.0, minimum-scale=1.0, initial-scale=1">
```

This will ensure that the mobile browser does not artificially try to scale the content to fit its screen.

Testing Media Queries

There are a number of ways to test media queries:

1. Simply resize the browser window
2. In the Google Chrome Developers tools use the Device Mode option (currently on the left)
3. If possible view the pages on a mobile or tablet device.

The third option is the best but would require you to publish your site in such a way that users can find in through a mobile device.

Building a Mobile Friendly Stylesheet

To build a mobile friendly stylesheet we should consider how best to present the information in a limited screen space available. One option is to remove any widths and floats and to use percentage values to set widths.

For example add the following rules:

```
body{
     margin:0;
}
#container {
     width:auto;
     margin:0;
}
header {
     overflow: hidden;
     background: #CF4A41;
     height:auto;
}
#logo{
     float: none;
     width:auto;
     padding: 5px;
     text-transform: uppercase;
}
#page{
     width:auto;
     margin:0;
     padding:0;
}
#content{
     float:none;
     width: 100%;
     padding:0 5px;
     margin:5px;
}
.sidebar{
     float:none;
     clear:left;
     width: 100%;
     padding:0 5px;
     margin:0;
}
```

The heading <h1> is a little large so we'll resize that:

```
#logo h1{
     font-size: 1em;
}
```

The main navigation inside the <nav> is currently horizontal but would be better presented as a vertical list.

We can achieve this by overruling the existing float on the <li> in the <nav> and setting their width to auto.

To do this we'll add the following rules:

```
nav{
     clear:left;
     float: none;
     width:auto;
     padding: 0px;
     margin:0;
     background-color:#fff;
}
nav ul{
     float: none;
     margin: 0;
     padding: 5px;
     list-style: none;
     line-height: normal;
}
nav li{
     float:none;
     padding:5px 0;
     border-bottom:1px solid #ccc;
}
nav a{
     color:#000;
     padding: 0;
     text-align:left;
}
```

We'll return to the navigation in a moment but firstly lets put in a quick fix for the image.

```
#content img{
     width:80%;
     height:80%;
}
```

Tip: This is a bit of a 'dirty' fix as the image is resized in the browser so there is no saving in terms of bandwidth. Other more sophisticated ways of replacing images are available including using background-images and the emerging standard for `srcset`.

In order to improve the design we want to hide the navigation and then make it available through a button. This will require some Javascript. Hiding the navigation is straight forward just add `display:none` to the CSS rule for the `nav` above.

To create a simple show/hide 'burger' button we'll add the following HTML after `<div id="logo">`.

```
<div id="navButton">
<img src="images/whiteNavicon.png" width="20" height="20">
</div>
```

Make sure that you revisit the 'main' stylesheet to hide this `navButton` from normal 'desktop' view. To do so add a rule of:

```
#navButton{
     display:none;
}
```

The rule for the `navButton` in the *smallScreen.css* file should appear as follows:

```
#navButton{
     display:inline-block;
     float:left;
     width:60px;
}
```

To build interactively we'll need some Javascript and we're also going to use jQuery to make the Javascript a little easy on the eye.

In the *scripts* folder are a couple of JS files that contain the Javascript we need.

Add the following to the bottom of the page before the closing `</body>` tag.

```
<script src="scripts/jquery-1.11.3.js"
language="javascript"></script>
<script src="scripts/main.js"
language="javascript"></script>
```

Test the page in the browser at a point where the *smallScreen.css* stylesheet is applied.

We'll see look in more detail at Javascipt and jQuery later in the module.

Things to do

1. Attach the media query to the other files in the project.
2. Make sure the `<meta>` tag to control the viewport is added to every page.
3. Tidy up the design by adding some additional rules to *smallScreen.css*.
4. The *contact.html* page that contains the form will require some additional rules.
5. Create a second stylesheet for tablets. A suggested media query would be:

```
<link rel="stylesheet" type="text/css" media="only screen
and (min-width:601px) and (max-width:1024px)"
href="styles/medScreen.css">
```

This would need placing between the desktop/default stylesheet and the mobile stylesheet.

Sample CSS for Mobile Form Design

As with other block elements it makes sense to replace widths with auto values to maximize use of the screen space. As such a sample stylesheet for a mobile form may appear as follows:

```
/*  Form  */
 label, .label{
     display:block;
     width:auto;
      }
label.inline{
     display:inline;
     width:auto;
}
fieldset{
     padding:5px;
     margin:10px auto;
     border:none;
}
legend{
     display:block;
     border-bottom:1px solid #ccc;
     width:80%
}
input[type='text'], input[type='email'],
input[type='number'], input[type='tel'], textarea, select{
     width:80%;
 }
```

Notice the use of the HTML5 form types.