BUILDING A MOBILE FIRST WEB SITE

AIM

In this project we are going to build a simple web site that consists of 5 pages. We are going to make the pages mobile friendly as well as viewable in older browsers.

- Unzip the five pages *index.html*, *learnHTML.html*, *learnCSS.html*, *learnJS.html*, *learnPHP.html and contactUs.html*
- Open *index.html* in your editor and in Google Chrome

MOBILE FIRST STYLESHEET

A requirement of this site is that it is mobile friendly. In order to do this we will follow a technique known as 'mobile first'. This involves first designing a simple stylesheet for the site that will present the content in a mobile friendly fashion. The benefits of this are:

1. Focuses the designer on the key content.
2. The 'mobile' stylesheet will download first - other stylesheets for larger screens (laptops, desktops) will only be downloaded if required - as such this approach is kinder on mobile user's data allowance.

STYLING THE BODY AND CONTAINER

Create a file called *mobile.css* in the *css* folder. Add the following rules:

```
body{
	font-family: "Gill Sans", "Gill Sans MT", Helvetica, Arial, "sans-
serif";
	padding:0;
	margin:0;
}
.container{
	width:100%;
}
```

This sets the default font for our pages as well removing the 'native' margin and padding that belonged to the `<body>`. The class selector `.container` is used so that it can be used more than once through the file.

Add some further CSS rules to target the `<header>` removing any default margin or padding and applying a background colour and border. Also add a margin to the `.content`.

```
header{
	background-color: #488D83;
	margin: 0;
```

```
	padding: 0;
	border: 1px solid #488D83;
}
.content{
	margin:5px;
}
```

To attach this stylesheet to the *index.html* page by adding the following inside the `<head>`:

```
<link rel="stylesheet" href="css/mobile.css">
```

Test your page in Google Chrome using the developer tools to see how the page would appear on a mobile device.

STYLING THE NAVIGATION BAR

For mobile users we would like a navigation bar to appear as a list. As the HTML that forms the navigation bar is a list, this will require only a limited about of styling. Firstly, to remove the bullet points, margin and the default left-hand padding add:

```
nav ul{
	padding-left: 0;
	margin: 0;
	list-style: none;
}
```

To add a border to the bottom of each list item target the `<li>` as follows:

```
nav ul li{
	border-bottom:1px solid #ccc;
	padding:5px;
}
```

The links will still have their default link styling, so we need a more specific CSS rule to target the `<a>` elements.

```
nav ul li a{
	display:inline-block;
	width:100%;
	color: #DDEAF2;
	text-decoration: none;
}
```

This uses the display CSS property to change the `<a>` from `inline` to `inline-block` elements. HTML elements with a `display` value of `inline` cannot be assigned a `width`. Thus, by setting a `display` value of `inline-block` the element can be assigned a `width`.

ADDING A BANNER IMAGE

To make the design more interesting we would like a banner image at the top of the page. Add a class selector as follows:

```
.banner{
	background-image: url(../images/banner1-600.jpg);
	background-repeat: no-repeat;
	height: 120px;
}
```

Notice that this uses a background image as opposed to adding an image via the HTML `<img>` tag. This will mean we can change the image by using other CSS later.

VIEWPORT

In Google Chrome ensure you are in mobile view to test the page. At this point the background image appear to the left hand side and the text on the navigation bar appears too small. This is due to the 'viewport'.

Web browsers on mobile devices are designed to cope with all web pages whether they have been made mobile friendly or not. To facilitate this, the mobile web browser behaves as if it has a much larger screen that it has in reality. Mobile browsers render pages in a virtual window known as the viewport - that is wider than the physical screen dimensions. However, if a page is designed to be mobile friendly, then the browser needs to be told not to use its default viewport but to use the one set by the designer. This is achieved through the addition of a meta tag to specifically control the viewport behaviour. Add the following to the `<head>` of the HTML file.

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

This tells the mobile browser that the viewport is the same as the width of the actual device not the virtual viewport.

Now save and test your pages in the mobile emulator in Google Chrome.

REARRANGING THE <H1>

Notice that there is currently a `<h1>` in the navigation list. This is primarily there for the desktop design. We want to have a differently positioned `<h1>` for the mobile view. To hide the `<h1>` in the list add an appropriate CSS rule setting display to none.

Now insert the following into the HTML prior to the opening `<nav>` element.

```
<h1 class="mobH1">WAD</h1>
```

This `<h1>` we can style by target the class as follows:

```
.mobH1{
	padding:10px;
	font-size:1.2rem;
	color: #DDEAF2;
}
```

SHOWING / HIDING THE MENU

Currently our navigation appears by default. However, we want to hide it and add a 'burger' menu to allow the user to access it. This will take up much less screen space.

We will create the 'burger button' purely through CSS. To do so add the following HTML to the top of the `div.container` inside of the `<header>`.

```
<div class="burgerMenu">
	<div class="menuLabel">Menu</div>
	<div class="bars">
		<div class="bar1"></div>
		<div class="bar2"></div>
		<div class="bar3"></div>
	</div>
</div>
```

The HTML includes a label and then some nested `<div>` tags to create the bars of the burger button. This will need styling.

```
.bars{
	float:left;
}
.bar1, .bar2, .bar3 {
	 width: 24px;
	 height: 4px;
	 background-color: #DDEAF2;
	 margin: 6px 0;
}
.menuLabel{
	color:#DDEAF2;
	float:left;
}
```

The Javascript needed to make the navigation bar slide up and down is already included in the page. Later in this module we'll see how the Javascript did this. For now just ensure you HTML file has the following two lines before the closing `</body>` tag.

```
<script src="js/jquery-3.2.1.min.js"></script>
<script src="js/main.js"></script>
```

FLOATING THE MENU TO THE RIGHT

For this design we'll place the burger bar to the right hand side. Add a CSS rule for the `div.burgerMenu`.

```
.burgerMenu {
	display: inline-block;
	cursor: pointer;
	float:right;
	padding: 4px;
}
```

The above uses the CSS float property to move the whole contents of `div.burgerMenu` to the right-hand side.

ADJUSTING ALIGNMENT WITH POSITION:RELATIVE

The text 'Menu' isn't correctly aligned. We can fix that by using the CSS `position` property. By setting it to a value of `relative`, we can use the properties `top` and `left` to experiment with values to get the desired alignment. With `position:relative` the element is offset by any values added to any of the four properties of `top`, `bottom`, `left` and `right`.

Add the following to the rule for `.menuLabel`.

```
position: relative;
top: 8px;
left: -5px;
```

SETTING THE INITIAL STATE OF THE NAVIGATION BAR

We would like the navigation bar to be initially hidden. Add a rule setting the property of `display` to a value of `none`.

USING FLEXBOX TO CREATE TWO COLUMNS

Having the main text and image appear in one column works fine on the mobile device. However, with bigger mobile devices there is scope to have a two column design. To achieve we could have used `float` but many devices/browsers now support the new CSS Flexible Box Model - most commonly known as 'flexbox'.

Flexbox allows developers to assign a 'flex container'. The child elements of the flex container can then 'flex' to fill whatever space is available These child elements we call flex items. Flexbox is ideal for mobile first design as the flex items in a flex container can be made to flex in a variety of ways.

To introduce flexbox review the HTML element `div.imgGrid` to which we will first apply it. This `<div>` contains three child elements each of `div.grid`.

To make element `div.imgGrid` a flexbox container, simply set the display `property` to the new value of `flex`.

```
.imgGrid{
	display: flex;
}
```

Preview the page in the mobile view in Google Chrome. The contents of `.imgGrid` are now on one 'row'. To force the child elements of `.imgGrid` to wrap add the `flex-wrap` property.

```
.imgGrid{
	display: flex;
	flex-wrap: wrap;
}
```

The design now reverts to how it was prior to adding flexbox. We now have to add some properties to the child elements (the flex items) of the flex container - these have a common class value of `.grid`.

```
.grid{
	padding: 5px;
	width: 50%;
	box-sizing: border-box;
}
```

The above properties set a width of 50% to create the two columns. As classic Box Model sees the width value inside of the `margin`, `border` and `padding`, we use the `box-sizing` property that changes the way width is applied. By setting a value of `border-box`, `padding` and `border` are included in the element's total `width` and `height`.

To tidy up the images add the following rule:

```
.grid > img{
	width: 100%;
	border: 1px solid #ccc;
}
```

This CSS selector using the `>` to indicate the rule is applied when an `img` element is the immediate child of a `.grid` element.

TIDYING UP THE MOBILE

Before moving onto the desktop design add some finishing touches to the mobile design.

To reposition the 'Welcome' text over the banner image, set the `div.banner` as `position :relative`. The `<h2>` in the banner then can be positioned using `positon: absolute` to find tune the design.

```
.banner h2{
	color: #DDEAF2;
	position: absolute;
	top: 20px;
	left: 50px;
	font-weight: 100;
	font-size: 2em;
	padding: 0;
	margin: 0;
}
```

Add a `margin` value of 5px to the `.content`. Also centre the footer text and add a border to the top.

FORMATTING FOR LARGER SCREENS

The CSS added now gives us a working design for mobile devices. However, if you view the page in Google Chrome in the 'normal' desktop the current CSS gives a poor experience to desktop users.

To fix this add a second style sheet that will only be applied if the screen is greater than 601px.

As CSS rules are cascaded from one stylesheet to another the styling of the desktop view can be based on that of the 'mobile first' stylesheet. Therefore, the desktop stylesheet will amend or overwriting some of the properties already used, and in some cases add some new ones. The technique for doing this is called 'media queries'.

WORKING WITH MEDIA QUERIES

Media queries allow the designer to ask questions about the browser and device been used to view the page. If the query has a positive response then alternative stylesheets can be applied. For this project we'll ask whether the screen has a minimum width of 600px. If so, we will add another stylesheet to the page. Add the following to the `<head>` of the *index.html* immediately after the link to the mobile.css stylesheet.

```
<link rel="stylesheet" media="only screen and (min-width : 600px)"
href="css/desktop.css">
```

This is essentially the same as a normal `<link>` element but with the addition of the `media` attribute. The `media` attribute contains the 'query'.

Why 600 pixels? This is a design decision based on the current state of mobile devices. It will mean that some larger screen devices (especially in landscape mode) will have this stylesheet applied.

Test your page in the 'normal' desktop view in Google Chrome. The burger menu is no longer visible and the design is now centred based on a `max-width` of 1366px. The `div.banner` also now picks up a different background image *banner1-1366.jpg* a bigger version of the image used in the mobile design.

THE NAVIGATION BAR FOR THE DESKTOP DESIGN

The *desktop.css* has some rules but more are required to create the desktop view.

First we'll reveal the navigation that was initialled hidden in the mobile view. Add:

```
nav{
	display:block;
}
```

We can use flexbox to have the navigation items flow horizontally.

```
nav ul{
	display: flex;
}
```

This makes the `<ul>` a flex container. The `<li>` elements, as children of the `<ul>` will now 'flex' - by default from left to right to fit the space available.

This works but in the design view we would like to space these `<li>` out more evenly.

```
nav ul li{
	border: none;
	padding:5px;
	width:160px;
}
```

By adding a width of 160px, this will be the minimum width the `<li>` will have. We can refine the styling of the `<a>` by creating an appropriate rule as follows:

```
nav ul li a{
	border-left: 1px solid #DDEAF2;
	color: #DDEAF2;
	padding:5px 10px;
	transition: 0.4s;
	background-color: #488D83;
}
```

USING FLEXBOX TO CREATE FOUR COLUMNS

In the desktop view we now have plenty of width so that we can have four column of images. As the `div.imgGrid` is already a flex container all we need do is change the width of the flex items to 25% as follows:

```
.grid{
	padding: 5px;
	width: 25%;
}
```

We can also flex box to place the 'sidebar' content to the right hand side.

Firstly we need to create a flexbox container. This time we'll pick the `div.content`. By using `box-sizing` we can use percentages with confidence splitting the `.main` and `.sideBar` to 70% and 30% respectively.

```
.content{
	display: flex;
	box-sizing: border-box;
}
.main{
	width:70%;
}
.sideBar{
	background-color: #8D6E5F;
	color:#DDEAF2;
	border-radius: 4px;
	margin-left:10px;
	padding:8px;
	width:30%;
}
```

STYLING THE SEARCH BOX

A great feature of flexbox is the ability to reorder flex items. This means that you can move items up and down, placing them in a different order to those in which they would appear in the normal flow of the document. To illustrate this we'll use the flexbox order feature to move the search box so that in mobile devices it appears at the top of the screen.

Review the HTML used for adding the search box.

```
<li class="searchItem">…
```

Notice it is inside a HTML list. This list is already using flexbox in the desktop view. We'll additionally apply flexbox to the mobile view.

Open the *mobile.css* stylesheet and locate the rule for `nav ul`. Add a `display` property of `flex`.

When you save and test your page in the mobile view in Chrome you'll notice that the navigation bar is no longer presented in a column. This is because by default when a container is made a flex container, its child elements (flex items) will flex horizontally. We can change this by adding a new CSS property to control the flex direction. Amend the `nav ul` rule so it now appears as:

```
nav ul{
	padding-left: 0;
	margin: 0;
	list-style: none;
	display: flex;
	flex-direction:column;
}
```

The `flex-direction` property is now set to `column` and as such the flex items are displayed vertically. If `flex-direction` is not set the default value is `row`.

Now the navigation list is a flex container we can re-order flex-items. Create a rule for `li.searchItem` as follows:

```
.searchItem{
	order: -1;
}
```

The `order` property changes the order the flex items are displayed in. By giving the `.searchItem` and order value of **-1** we add it to the start of the list. The flex item,s are by default allocated, a number corresponding to their position in the source code. If the order set by the CSS is already used then the targeted element will assume the same position it held in the source code.

Save and test your file. Back in the desktop view the search now appears to the left hand side and the navigation is presented vertically. How could you fix this?

ROLL OUT THE DESIGN TO THE OTHER PAGES

Now we have a pretty decent mobile.css and deskTop.css stylesheets, attach them to the other files *learnHTML.html*, *learnCSS.html*, *learnJS.html*, *learnPHP.html and contactUs.html*. Don't forget to also add the meta tag for the viewpoint.

IPAD FRIENDLY MEDIA QUERIES

The page now tests well in Google Chrome mobile view. However, it is always recommended that you also test your pages on as many real devices as possible. When tested on an iPad the navigation bar was found to be a little cramped with some navigation items spreading over two lines.

Media queries can be applied in the `<link>` as we have seen, but they can also be written inside the CSS files themselves. We'll use this technique to fine tune the design for iPads.

In the *desktop.css* file add:

```
@media (min-width: 601px) and (max-width: 1122px) {
}
```

The `@media` represents the media query to apply, in this case if the browser has a minimum width of 601px and a maximum width of 1122px. This should help us target iPads as they have a screen resolution of 1024x768. Inside the curly braces of the media query rule, we can now add the rules specific to the media query ie:

```
@media (min-width: 601px) and (max-width: 1122px) {
	nav ul {
		justify-content: space-between;
	}
	nav ul li{
		width:auto;
	}
	nav ul li a{
		width:auto;
	}
	nav ul li:last-child {
		width: auto;
	}
	nav ul li form input[type=search]{
		width: 120px;
	}
}
```

BUILDING A FORM

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

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

<input> - types text, email, tel, number, radio, checkbox, search, 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 FORM ELEMENTS

Following the 'mobile first' approach create two additional stylesheets for the form on the *contactUs.html* page.

The first stylesheet will style the form for mobile and then we will produce a second stylesheet that will be applied to the desktop using media queries.

As such your styles for the form will be attached as follows:

```
<link rel="stylesheet" href="css/mobile.css" />
<link rel="stylesheet" media="only screen and (min-width:600px)"
href="css/desktop.css">
<link rel="stylesheet" href="css/mobileForm.css" />
<link rel="stylesheet" media="only screen and (min-width:600px)"
href="css/deskForm.css" />
```

With the limited space of the mobile is it common practise to place labels above form fields. As the `<label>` element is an inline element create a rule with `display:block` to force a new line.

Other design aspects to consider for the mobile stylesheet is the width of the form elements. Try to maximise this with `width:100%`.

For the desktop stylesheet try to align the labels to the left of the form elements. This could be done with `float`.

ANIMATING THE BURGER MENU

A nice touch would be animate the burger menu. We can do this using CSS tranistions. Firstly, indicate the length of the transition and apply it to all three of the bars of the burger menu. To do so add to the rule for the `.bar1` through to `.bar3`:

```
transition: 0.4s;
```

To illustrate what can be done we'll first of all fade out the middle bar of the three.

```
.animateBurger .bar2 {
    opacity: 0;
}
```

The Javascript used to animate the sliding of the navigation bar is also dynamically adding a class of `.animateBurger` to each of the three burger bars. In the above instance `.bar2` when clicked has the `.animateBurger` class added with a opacity of 0. But as the `.bar2` has a transition of 0.4s it will 'tranisiton' to this new state over 0.4 seconds.

As well as fading we can rotate using the CSS3 transform property. Add the following:

```
/* Rotate first bar */
.animateBurger .bar1 {
/*    -webkit-transform: rotate(-45deg) translate(0,0) ;*/
/*    transform: rotate(-45deg) translate(-8px, 6px) ;*/
/*    l/r t/b*/
       transform: rotate(-45deg) translate(-6px, 8px) ;
}

/* Fade out the second bar */
.animateBurger .bar2 {
    opacity: 0;
}

/* Rotate last bar */
.animateBurger .bar3 {
/*    -webkit-transform: rotate(45deg) translate(-8px, -8px) ;*/
/*    transform: rotate(45deg) translate(-8px, -4px) ;*/
      transform: rotate(45deg) translate(-6px, -8px) ;
}
```

SUPPORT FOR OLDER BROWSERS

Our design has relied on flex-box model that isn't supported by all browsers. Therefore, we should provide some fall back for users with older browsers.

There are a range of techniques for delivering fall back options. With Internet Explorer 5 to 9 supported something called 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.

For example this IE conditional commend <!--[if lt IE 9]> could be used to add some styles that will only be applied if the browser version if it is less than IE9.

```
<!--[if lte IE 9]>
<link rel="stylesheet" type="text/css" href="css/oldie.css" />
<![endif]-->
```

The *oldie.css* stylesheet is only loaded by Internet Explorer versions IE9 and below.

Fortunately these older browser are becoming more scarce and rather than detecting the browser it is more sensible to see if a feature is supported. Feature detection is something that can be done with the modernizr (https://modernizr.com). Use this site to build a JS file to detect is features are supported, then create CSS rules for the no-SUPPORT scenario.