Lab: Mobile First Design

Objectives

1. Outline the benefits of Mobile First Design.
2. Design and apply a mobile first stylesheet.
3. Demonstrate the use of media queries to a apply a desktop stylesheet.
4. Demonstrate different layouts of HTML forms.

Setting up your webspace

Download the ZIP file for this week's lab. Extract the files and folders.

To add the labs files for this week, drag the folder required into the Visual Studio Code workspace.

The files in the *mobile-first* folder will now appear on the left-hand side.

View the files

Your file structure may appear as follows:

```
f:\year1\web-dev\mobile-first\
f:\year1\web-dev\mobile-first\images
f:\year1\web-dev\mobile-first\styles
```

There will be a number of HTML files created for you: *index.html, contact-me.html, thank-you.html, qualifications.html, skill-set.html and work-experience.html*.

Create a Mobile First Stylesheet

Modern website need to be mobile friendly. In order to do this, we will follow a technique known as 'mobile first'. This firstly involves 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

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

```css
body{
  font-family: helvetica, arial, sans-serif;
  font-size: 1em;
  margin: 0;
  padding: 0;
}
```
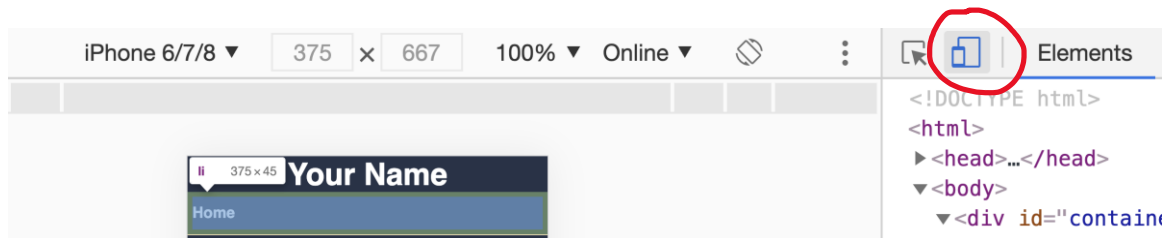
This sets the default font for our pages as well removing the 'native' margin and padding that belonged to the `<body>`.

Attach the Mobile Stylesheet

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

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

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

You should be able to tell the stylesheet has been applied by the font changing. To emphasize that the stylesheet is attached try changing the background colour of the whole document by adding the `background-color` property to the rule created for the `body`.

2

The Container and its child elements

There is a `<div>` of class container around the outside of our design. This has three child elements of:

```html
<div class="container">
  <header>
       ...
  </header>
  <main>
       ...
  </main>
  <footer>
       ...
  </footer>
</div>
```

Add rules to *mobile.css* to style `main` and `footer` as follows:

```css
.container {
  width: 100%;
}
main,
footer {
  padding: 0 15px;
}
```

This ensures the container will fill up 100% of the available space. On a mobile we want to use the space as efficiently as possible whilst ensuring that the child elements `main` and `footer` pad away from the screen edge by 15px.

Inside the `<header>` is `div#logo`. Centre this with:

```css
#logo {
  text-align: center;
}
```

Styling the Header

Style the `<header>` element with the following rules:

```css
header {
  background-color: #263248;
}
header h1 {
  margin: 0;
}
header h1 a {
  color: #fff;
  text-decoration: none;
}
```

With the mobile first approach these styles will be inherited by both mobile and desktop visitors. Therefore, we don't need to set them again in the *desktop.css*.

Styling the Navigation

Viewing the page in the mobile view in Google Chrome, the navigation bar does not appear very clear. For mobile it would be better if the navigation was in a vertical list, with more spacing and had the default link colours removed.

Start by removing the list styling and making the `<ul>` a flex container:

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

Give each `<li>` more padding a border along the bottom:

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

Next, change the link colour to white and add some padding at the top and bottom.

```css
nav ul li a {
  font-weight: bold;
  text-decoration: none;
  color: #fff;
  padding: 8px 0;
}
```

ViewPort

In Google Chrome ensure you are in mobile view to test the page. At this point it may not be displaying quite as you would expect. 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.

```html
<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.

Responsive Images

The pages are now being distorted by the images used, specifically the 900px wide image in the `section`. The image has dimension of 900px by 200px. The width of 900px is been respected by the viewport of the browser, as it tries to fit the image into the whole page. We can fix this by adding a CSS rule to resize our images.

```css
.resize-img{
  width: 100%;
  height: auto;
}
```

This rule sets images with a class selector of `.resize-img` to have a width of 100%. The 100% value refers to the width the element should assume of its parent element.

Save and test your file through the Browser emulator. It should be looking a lot better now.

To help you understand how this technique is working try adding the following rule:

```css
/* Just an experiment */
section{
  width: 50%;
}
```

When viewed all the images that are children of the `section` element now take up 50% of the screen space. Once you are happy with this idea remove the above rule.

Improving the User Experience

Now are pages is looking pretty mobile friendly we can refine the user experience. When you test the page in the mobile emulator in Chrome notice that the hyperlinks in the navigation aren't clickable on the right-hand side of the screen where it is likely the user will attempt to select the link. Imagine the way you hold your phone. Lots of users will use their thumb on the right-hand side to navigate.

We can fix this by changing the behaviours of the `<a>` tags. These are by default inline elements and as such flow left->right and cannot be assigned a width. By changing the `<a>` to have a display property of `display: inline-block` we can they assign a width of 100%. This will make the navigation links clickable across the width of the screen.

Amend the rule for the `<a>` with the additional of a `display` and `width` properties:

```css
nav ul li a {
  display: inline-block;
  width: 100%;
  font-weight: bold;
  text-decoration: none;
  color: #fff;
  padding: 8px 0;
}
```

Styling the footer

To style the footer add the following rule to the *mobile.css* file.

```css
footer {
  border-top: 1px solid #ccc;
  margin: 20px 10px 10px 0;
  font-size: 0.8em;
  text-align: center;
  font-style: italic;
}
```

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 is matched, then alternative stylesheets can be applied. For this project, we'll ask whether the screen has a minimum width of 720px. If so, we will add another stylesheet to the page. This is known as a breakpoint.

Add the following to the `<head>` of the *index.html* immediately after the link to the *mobile.css* stylesheet.

```html
<link
    rel="stylesheet"
    href="styles/desktop.css"
    media="only screen and (min-width : 720px)"
  />
```

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 720 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.

Once added save and test your page in both mobile and desktop view.

Review the Desktop Stylesheet

Review the *desktop.css* file and you will see styling such as flexbox used to create the two-column effect.

The desktop stylesheet is added after the mobile stylesheet to overrule some CSS properties and add some new ones.

For example, the `.container` rule now has the additional properties of:

```
.container {
  margin: auto;
  max-width: 1200px;
  min-width: 600px;
}
```

The `max-width` properties stops the container exceeding 1200px in width.

Notice how in Google Chrome you can see how the styling for the `nav ul` in the *mobile.css* is overruled by the styling from the *desktop.css*. `The padding-left` and `flex-direction` are struck through in the *mobile.css* as the desktop.css rules replace them.

```
media="only screen and (min-width: 720px)"
nav ul {                                              desktop.css:25
    padding-left: 0;
    flex-direction: row;
}

nav ul {                                              mobile.css:27
    list-style: ▶ none;
    padding-left: 0;
    margin: ▶ 0;
    display: flex;
    flex-direction: column;
}
```

Now we have a mobile and desktop styles roll them out to the other pages in the website. To do so ensure that on each page you:

1. Add the meta viewport tag
2. Add the two links to the stylesheets.

Styling the Form for Mobile

The form for the mobile on the *contact-me.html* page needs styling. Currently it appears ragged.

We'll style the form with the same appoach used for the rest of the styling. First of all we'll style it for mobile and then style it for the desktop view.

As such open the mobile.css stylesheet to apply some styling.

First of all add a margin to the top of the `form` using the box-model.

```css
form > div {
  margin: 15px 0;
}
```

The labels will work better if they sit on top of the `form` elements rather than to the side. The `label` is an inline element. However, we can change it to behave as a block as follows:
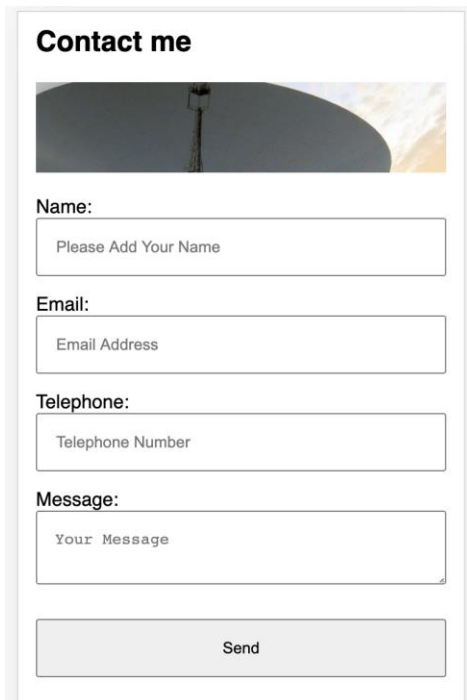
```css
label {
  display: block;
}
```

The input `text` fields, the `email` and `tel` fields we'll have stretch across the width of the mobile view. Here we'll use attribute selectors:

```
input[type="text"],
input[type="email"],
input[type="tel"],
textarea {
  box-sizing: border-box;
  width: 100%;
  padding: 15px;
}
```

The `submit` button we'll also stretch and centre its text label.

```
input[type="submit"] {
  margin: 10px auto;
  text-align: center;
  width: 100%;
  padding: 15px;
}
```

Save and view the file. It should now look as follows:
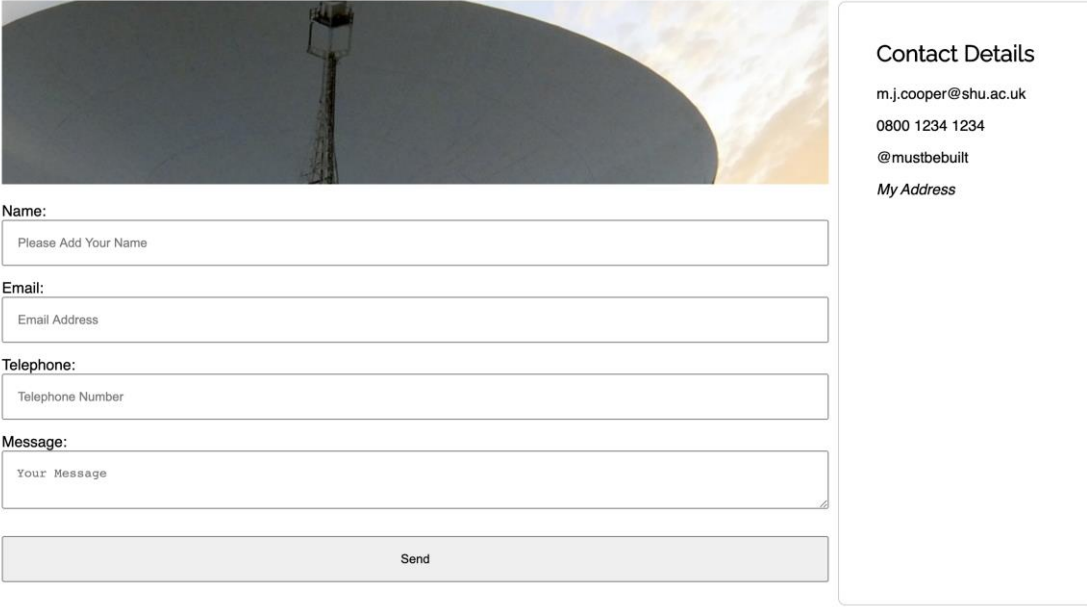
The desktop view of the form appears as follows:

The fields are now stretched a little too much. Open the *desktop.css* to amend the CSS for the form.

First reduce the width of the form and centred it with `margin:auto`.

```
form {
  width: 50%;
  margin: auto;
}
```

Add a rule to make the child `<div>` element of the form containers. As such the `label` and associated input will flex.

```
form > div {
  margin: 15px 0;
  display: flex;
}
```

The labels now appear ragged again so change them to `display` as `inline-block` such that a width can be applied:

```css
label {
  display: inline-block;
  width: 120px;
}
```

Finally have the `div.submitContainer` position the submit button to the right of the form and the submit buttons width.

```css
.submitContainer {
  justify-content: flex-end;
}
input[type="submit"] {
  width: auto;
  margin: 0;
}
```

Save and test your file.

To add a 'burger' menu to the mobile view add the following HTML under the `<h1>`.

```html
<h1><a href="#">Your Name</a></h1>
    <div class="burger">
      <div class="line"></div>
      <div class="line"></div>
      <div class="line"></div>
    </div>
```

In the mobile CSS add:

```css
.burger > .line{
  width: 35px;
  height: 5px;
  background-color: #fff;
  margin: 6px 0;
}
```

Amend the `#logo` with:

```css
#logo {
  /* text-align: center; */
  display:flex;
  justify-content: space-between;
  padding:5px;
}
```

The rule for the `<nav>` adds a CSS transition to animate the menu.

```css
nav{
  overflow: hidden;
  transition: height 0.5s;
}
```

In the desktop hide this with:

```
.burger {
  display: none;
}
```

… and add this for the nav:

```
nav {
/* for burger menu */
  overflow: auto;
}
```

Attached the Javascript file to the bottom of each page before the closing `</body>` with:

```
<script src="js/main.js"></script>
  </body>
```

Testing your Page via GitHub Pages

To test your pages on your own mobile device you can move your project to a GitHub Repo and use the GitHub Pages feature.

In HTML5 compliant browsers video can be added using the <video> element.

In many ways this is like the <img> tag in that you need to indicate the video source with the src attribute.

The video tag has the following attributes:

width	The width of the video holder
height	The height of the video holder
src	The URL to the source video file
controls	To enable playback controls
autoplay	Set the video to play automatically on page load
loop	To set the video to loop
poster	The URL to an image that is shown whilst the video is downloading and before it is played. Useful if your video fades from black or white.

As such the video element may appear as:

```
<video src="media/horses.mp4"></video>
```

But in order to be able to play it we need to add controls ie:

```
<video src="media/horses.mp4" controls></video>
```

Creating a Poster Image

By default when a video is added to a web page it will show the first frame of the video until the video is played.

Sometimes, for example when video fades in, the first frame is not an ideal one to show the user. Therefore, you can create an image that can be used as a 'Poster'. This is a still image that will be seen by users yet to play the video instead of the 1 frame of the video.

To create a poster image you need to create an image with the same aspect ratio as your video.

To add the poster image to the <video> tag use:

```
<video src="media/horses.mp4" controls poster="images/horsePoster.jpg">
</video>
```

Style the Video

To style the video, control its width and placement, nest it inside of a `<div>` element.

```html
<div class="videoContainer">
        <video src="media/horses.mp4" controls poster="images/horsePoster.jpg">
        </video>
</div>
```

In the *mobile.css* add styling that will apply as the default for both mobile and desktop:

```css
.videoContainer{
 display: flex;
 justify-content: center;
}
video{
 width:100%;
 height: auto;
 max-width:560px;
}
```

This allow the video to be up to a maximum width of 560px and uses a flex container to centre the video.