University of East Anglia
COMPUTING SCIENCES

Lab 3: CSS Media Queries and Responsive Design

Before attempting this lab, you will need to have completed lab **2: CSS Basics**.

Learning objectives

By completing this lab, you should be able to:

- Explain the concept of Media Query
- Optimise web pages for display on a mobile device using the viewport meta tag
- Apply responsive design techniques using CSS3 media queries
- Apply web icon in styling
- Configure flex layout
- Test CSS styles using various screen sizes

Background

Increasingly, many users access websites using a mobile device, rather than through a desktop computer. Mobile devices (which includes mobile phones and tablets) usually employ touch control rather than using a physical keyboard and mouse. Mobile devices also have smaller screens than desktop displays. Using HTML5 and CSS3, it is possible to display a webpage differently to mobile users than to desktop users.

The CSS Media Query gives you a way to apply CSS only when the browser and device environment matches a rule that you specify, for example "viewport is wider than 480 pixels". Media queries are a key part of responsive web design, as they allow you to create different layouts depending on the size of the browser or device, but they can also be used to detect other things about the environment your site is running on, for example whether the user is using a touchscreen rather than a mouse.

Media Query Basics

The simplest media query syntax looks like this:

```
@media media-type and (media-feature-rule) {
  /* CSS rules go here */
}
```

Figure 1. Media Query syntax

The above syntax consists of:

- A media type, which tells the browser what kind of media this code is for (e.g. print, or screen). The possible media types are:
    - all
    - print
    - screen
- A media expression, which is a rule, or test that must be passed for the contained CSS to be applied.
- A set of CSS rules that will be applied if the test passes and the media type is correct.

Media feature rules

After specifying the media type, you can then target a media feature with a rule.

Width and height

The feature we tend to detect most often in order to create responsive designs (and that has widespread browser support) is **viewport width**, and we can apply CSS if the viewport is above or below a certain width — or an exact width — using the min-width, max-width, and width media features.

These features are used to create layouts that respond to different screen sizes. For example, to change the body text colour to red if the viewport is exactly 600 pixels, you would use the following media query.

```
@media screen and (width: 600px) {
    body {
        color: red;
    }
}
```

Figure 2. Media Query example

The width (and height) media features can be used as ranges, and therefore be prefixed with min- or max- to indicate that the given value is a minimum, or a maximum. For example, to make the colour blue if the viewport is narrower than 600 pixels, use **max-width**:

```
@media screen and (max-width: 600px) {
    body {
        color: blue;
    }
}
```

Figure 3. Media Query example

In practice, using minimum or maximum values is much more useful for responsive design so you will rarely see width or height used alone.

We can apply orientation too! For example, to change the body text colour if the device is in landscape orientation, use the following media query:

```
@media (orientation: landscape) {
    body {
        color: rebeccapurple;
    }
}
```

Figure 4. Media Query example

Applying logic in media queries

To combine media features you can use and in much the same way as described above. For example, we might want to test for a min-width **and** orientation. The body text will only be blue if the viewport is at least 600 pixels wide **and** the device is in landscape mode.

```
@media screen and (min-width: 600px) and (orientation: landscape) {
    body {
        color: blue;
    }
}
```

Figure 5. Media Query example

You can negate an entire media query by using the **not** operator. This reverses the meaning of the entire media query. The example below, the text will only be blue if the orientation is portrait.

```
@media not all and (orientation: landscape) {
    body {
        color: blue;
    }
}
```

Figure 6. Media Query example

Responsive Design Approach

There are two approaches we can take to create responsive design. The first approach is to start with your desktop or widest view and then add breakpoints to

move things around as the viewport becomes smaller. Hang on, what is viewport? What are breakpoints?

The **viewport** refers to the entire area where the content is displayed and can be viewed on the browser. This means what we see on a browser window is the viewport, the user's visible area of a web page. The viewport varies with the device and will be smaller on a mobile phone than on a computer screen.

In the early days of responsive design, many Web designers would attempt to target very specific screen sizes. Lists of the sizes of the screens of popular phones and tablets were published in order that designs could be created to neatly match those viewports.

However, there are now far too many devices, with a huge variety of sizes, to make that feasible. This means that instead of targeting specific sizes for all designs, a better approach is to change the design at the size where the content starts to break in some way. For example, it could be the line lengths become far too long, or a boxed-out sidebar gets squashed and hard to read. That's the point at which we would want to use a media query to change the design to a better one for the space available. This approach means that it doesn't matter what the exact dimensions are of the device being used, every range is catered for. The points at which a media query is introduced are known as **breakpoints**.

Now back to the approach we can take to create responsive design, we can also start with the smallest view first and add layout as the viewport becomes larger. This approach is described as mobile first responsive design and is quite often the best approach to follow.

The view for the very smallest devices is quite often a simple single column of content, much as it appears in normal flow. This means that you probably don't need to create a lot of layout for small devices — the key is to markup your HTML well and you will have a readable layout by default.

Using the viewport meta tag to control layout on mobile browsers

The browser's viewport is the area of the window in which web content can be seen. This is often not the same size as the rendered page, in which case the browser provides scrollbars for the user to scroll around and access all the content.

Some mobile devices and other narrow screens render pages in a virtual window or viewport, which is usually wider than the screen, and then shrink the rendered result down so it can all be seen at once. Users can then pan and zoom to see different areas of the page. For example, if a mobile screen has a width of 640px, pages might be rendered with a virtual viewport of 980px, and then it will be shrunk down to fit into the 640px space.

This is done because not all pages are optimized for mobile and break (or at least look bad) when rendered at a small viewport width. This virtual viewport is a way to make non-mobile-optimized sites in general look better on narrow screen devices.

However, this mechanism is not good for pages that are optimised for narrow screens using media queries — if the virtual viewport is 980px for example, media queries that kick in at 640px or 480px or less will never be used, limiting the effectiveness of such responsive design techniques. The viewport meta tag mitigates this problem of virtual viewport on narrow screen devices.

A typical mobile-optimized site contains something like the following:

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

Figure 7. Viewport configuration and syntax

More details on viewport and its usage see MDN

Task: configure viewport on `index.html` and `classes.html`

1. Create a new directory/folder in your student directory (U: drive) called `css-media-grid`. Copy all the files in your `yogacss` directory and paste them in this newly created `css-media-grid` directory.

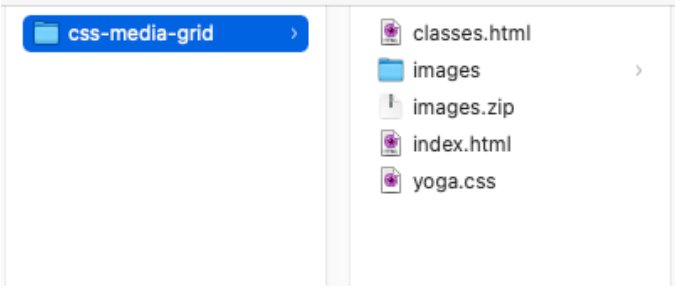

Figure 8. new directory for this week lab

2. On Visual Studio Editor open this folder "`css-media-grid`". Click the `index.html` file so that it appears on the right side of the panel. Add the viewport configuration shown in Figure 7 in this file. [**Hint**: the configuration should be placed inside the `head` tag]
3. Repeat the same for `classes.html`
4. Now test both your webpages using Chrome.

Next, we will style our page to look different when displayed on a smaller screen. We will do this using Media Query. Figure 9 below shows the look of the home page when on a desktop screen. As we can see, the main navigation bar is placed at the centre of the page, below the header – Figure 10.

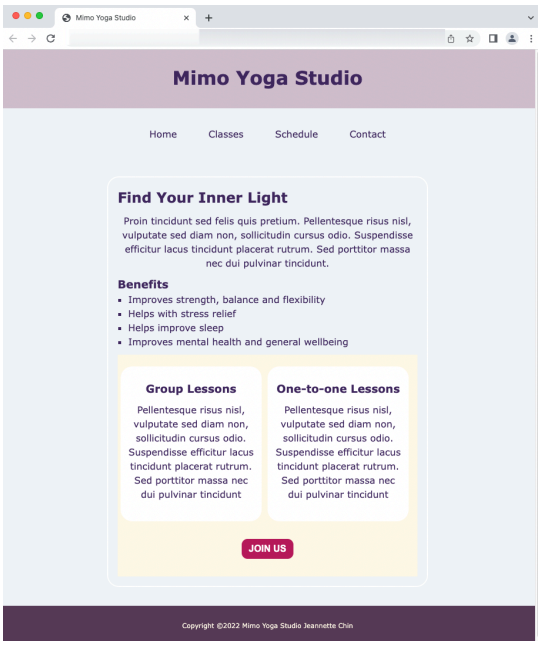

Figure 9. Homepage on desktop screen

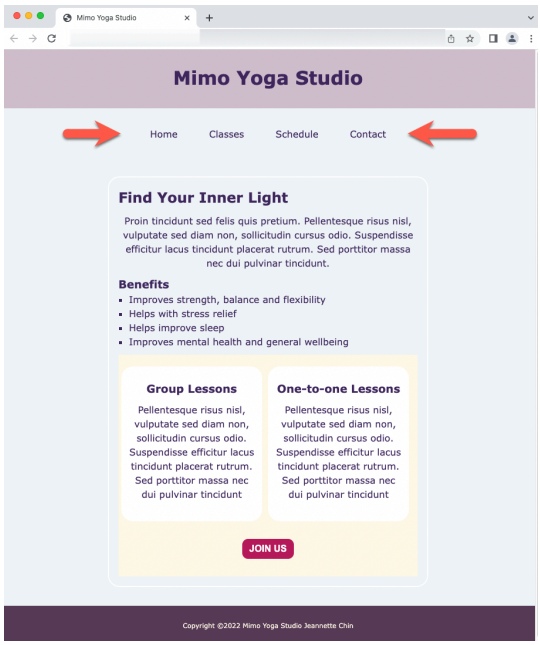

Figure 10. The main navigation bar on homepage

This navigation is alright, easy to use, but it takes up some space particular on the small screens. Wouldn't it be nice if we could hide this navigation bar somehow so that the page looks nice and neat? One way of achieving this is using toggle menu, such that when the screen width is reduced, navigation items will automatically move from the top navigation to a drop-down list and stay as a toggle menu. Our user will click or tap on the menu to expand the navigation links. We will use hamburger menu system for this job. Hamburger menu was created by Norm Cox in the 1980s [source]. It has been suggested that for phone users, the click rates for hamburger menus on the top left are low, so we will be placing our hamburger menu on the top right for demonstration purposes. Figure 11 below shows the version of our webpage on small screen device. As you can see, the main horizontal navigation has become a dropdown list, and rendered just below the hamburger menu icon.

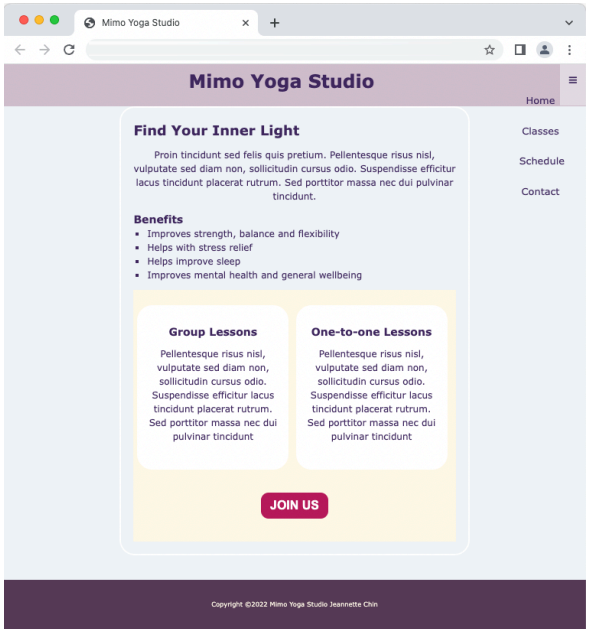

Figure 11. The navigation links live in a toggle menu on small screen

The toggle action is done by using *JavaScript*. I will cover this part later in the course, but for this lab, we are going to style this hamburger menu (Figure 11) using CSS.

First, we need a hamburger icon to represent our toggle menu. Font awesome (https://fontawesome.com) provides a lot of nice icons we can use. Before we can use the icons, we will sign up and get the link. Go to https://fontawesome.com/start and sign up using your UEA email address (Figure 12)

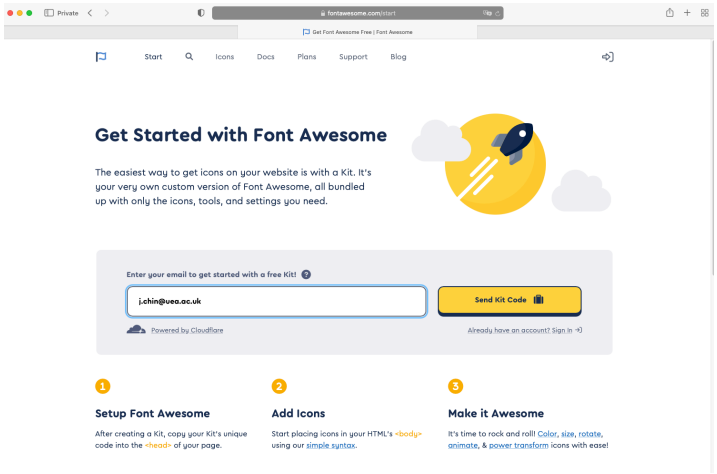

Figure 12. Sign up for fontawesome

Once you have entered your UEA email address, click the “Send Kit Code” icon. You will receive an email from fontawesome asking you to confirm your email (Figure 13)

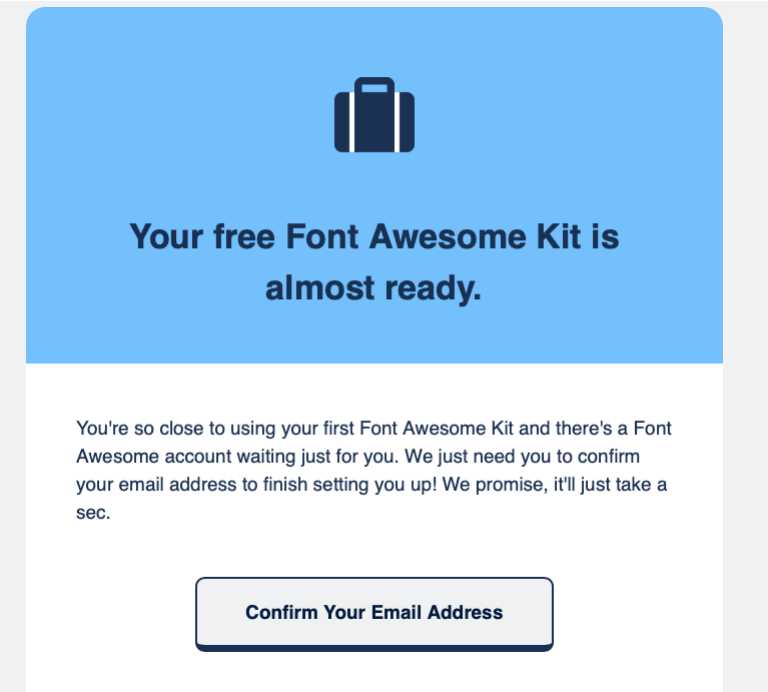

Figure 13. Confirm your email message by fontawesome

Once you receive this message, click the button “Confirm Your Email Address”. When this is done, you should see a page similar to below:

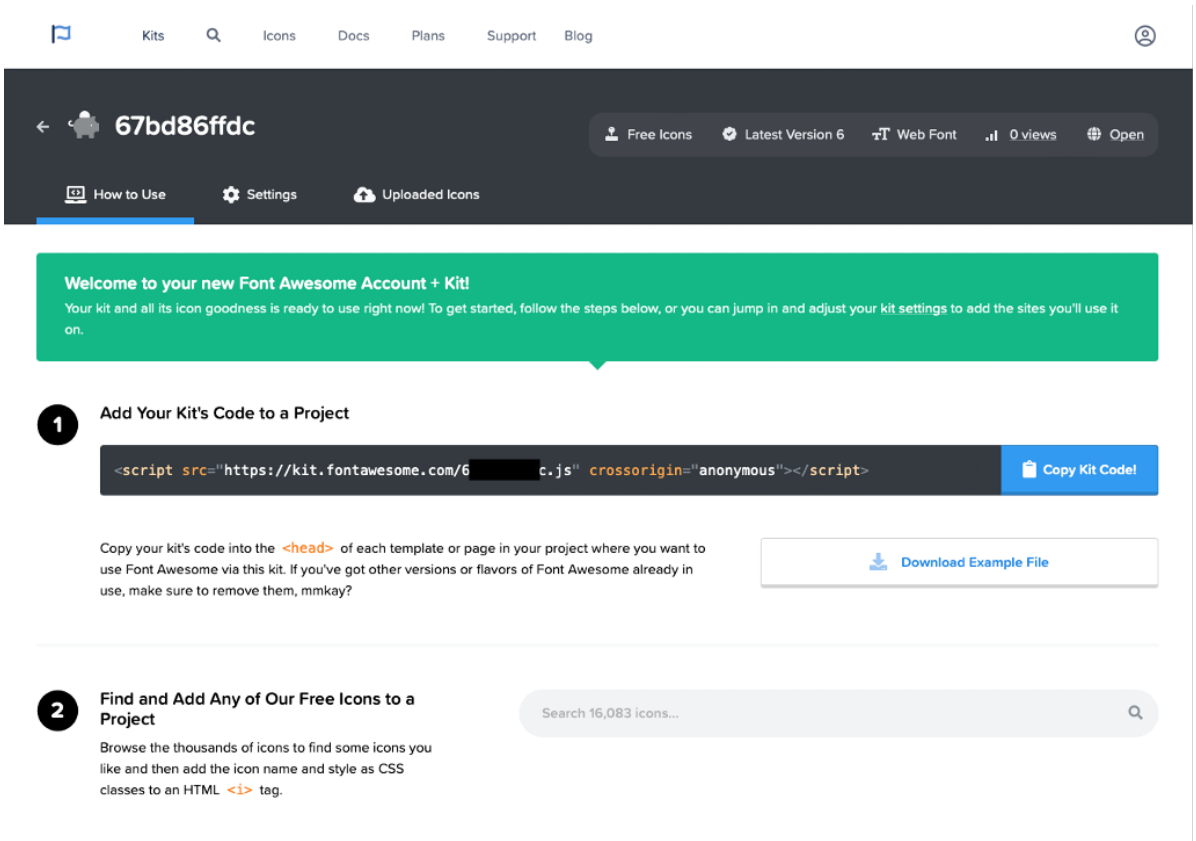

Figure 14. fontawesome user page after email validation

Now click the “Copy Kit Code” button to copy the code and paste this code inside the `head` block of `index.html` and `classes.html`. Figure 15 below shows the code in my account. **NOTE:** The code is a `script` tag linking to fontawesome JavaScript. Yours will be different from mine.

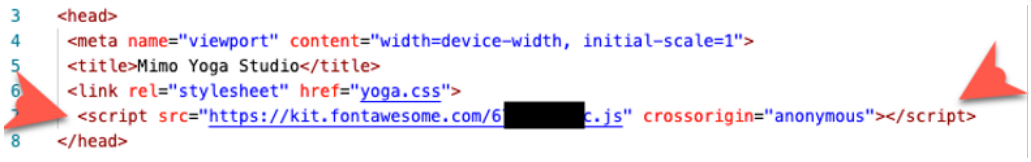

Figure 15. Index.html with fontawesome script placed inside the head element

Next, we will add this hamburger menu icon to our page. Where should this icon go? As mentioned earlier, the hamburger menu is our toggle menu, we want to place this toggle menu on the right top of the page, as shown in Figure 16.

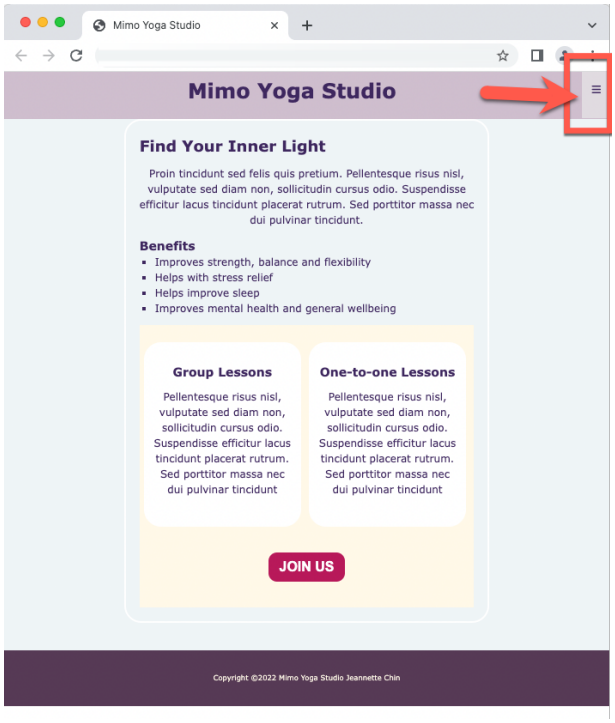

Figure 16. Hamburger menu icon placed on the right top of index.html

Notice that the location of the hamburger icon is inside the `header`, the same container where the title of the page - “Mimo Yoga Studio” is displayed, as shown in Figure 17.

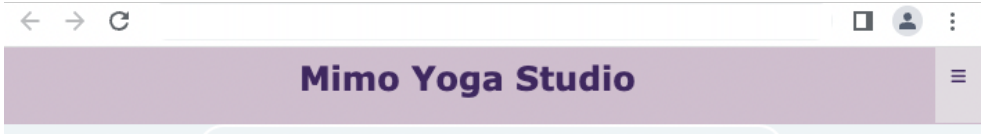

Figure 17. The output of header container

This means we should place our hamburger inside the `header` element in our HTML files (Figure 18)

```
11  <header>
12      <h1>Mimo Yoga Studio</h1>
13  </header>
```

Figure 18. The header element in index.html

The **hamburger** icon we are going to use is shown in Figure 19. URL reference: https://fontawesome.com/icons/bars?s=solid

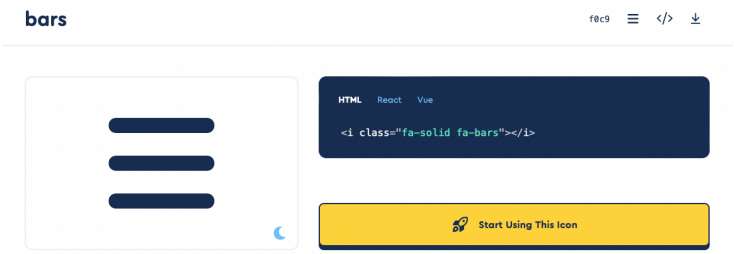

Figure 19. Hamburger menu icon on fontawesome

The code is relatively straightforward. All we need to do is copied the code there and paste in file. We will use and `anchor` tag to wrap around this code so that it will become the toggle menu when we apply JavaScript.

Tasks:

1. Copy the icon code shown in Figure 19 and paste the code inside the `header` element on your `index.html`.
2. Wrap the icon code inside an anchor (`<a>`) element. That is, the parent of this icon is the `<a>` element. Result see Figure 21 below.

Save and open the file using Chrome. Your output should look similar to Figure 20. Notice that the hamburger icon is on a separate line. Why?

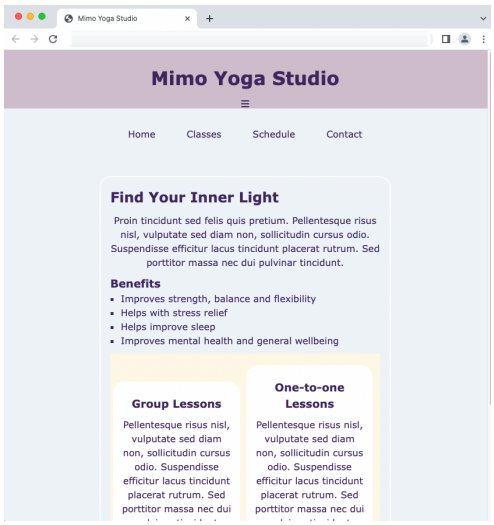

Figure 20. The output on index.html

We need to fix the location of the hamburger icon. Before we do this, let us examine our code – Figure 21.

```
<header>
    <h1>Mimo Yoga Studio</h1>
    <a >
        <i class="fa-solid fa-bars"></i>
    </a>
</header>
```

Figure 21. The code in index.html

The reason why the hamburger icon is displayed on a separate line is because the `h1` element is a block element (MDN). This means the content occupies the entire horizontal space of its parent element, the container, and vertical space equal to the height of its contents, hence creating a "block".

To understand this, I configured a border for `h1` element, the output is shown in Figure 22 below.

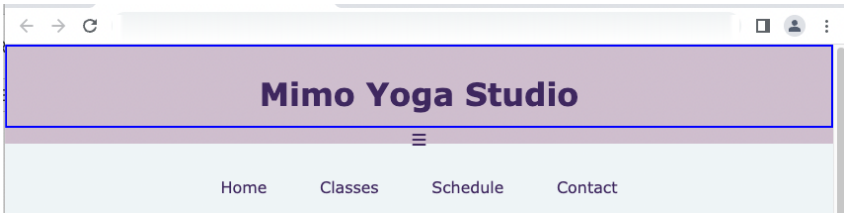

Figure 22. The h1 element has a border.

As you can see, the title “Mimo Yoga Studio” occupies the entire horizontal space. The browser renders the hamburger icon on a separate line. We will need to fix this because we want our hamburger icon to be displayed on the same line as the title. Fortunately, we can change the default display behaviour of `h1` element. However, we only want this behaviour for our small screen users. To achieve this we will configure this behaviour using a media query.

Tasks:

1. Configure media query:
    a. At the bottom of `yoga.css`, write a media query - for screen only with a max-width of 768px, as shown in Figure 23.

```
138   @media only screen and (max-width: 768px)
```

Figure 23. media query code

    b. Configure the `display` property of `h1` element to `inline-block`, as shown in Figure 24. Notice I am using a direct reference because I only want this behaviour applies to the `h1` – the child element of `header` element, not any other `h1` element on the same page.

```
130  @media only screen and (max-width: 768px) {
131
132    header h1 {
133      display:inline-block;
134    }
135
136  }
```

Figure 24. configure h1 display property

c. Save the file and refresh the page. Reduce the size your browser window to smaller than 768px to see the media query effect. Figure 25 below shows the output.

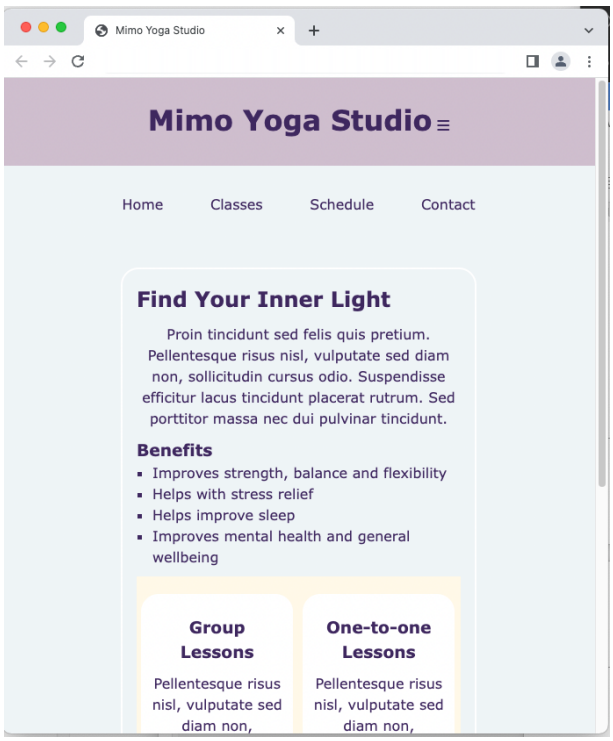

Figure 25. The output of window size smaller than 768px.

Next, we will fix the position of our hamburger icon. Examine our code in `index.html` we will see that our hamburger icon is wrapped inside an anchor `<a>` element – Figure 26.

```
11  <header>
12      <h1>Mimo Yoga Studio</h1>
13      <a >
14          <i class="fa-solid fa-bars"></i>
15      </a>
16  </header>
```

Figure 26. Fragment code on index.html

The anchor element is an inline element (MDN). This means the content only occupies the space bounded by the tags defining the element, that is, it only takes up the space as much as it needs. Unlike a block element, an inline element does

not break the flow of the content, it carries on the same line. This is the reason why our hamburger icon displays next to the title “Mimo Yoga Studio”, as shown in Figure 25. For *block elements*, we can configure the width and height. However, it is not possible to do so for *inline elements*. We need a strategy if we want our hamburger icon to be displayed on the right top of the page, same line as the title “Mimo Yoga Studio”. We can change this default behaviour by changing the value of `display` property of `<a>` element.

d. In `yoga.css`, inside the media query, create an **ID** named `hamburger`
   i. Configure the value of the `display` property to `block`
   ii. Configure a rule that move the element to the `right` by using the `float` property (MDN)
   iii. Save the file

e. Now add this CSS **ID** `hamburger` to the `<a>` element (the parent element of your icon) on your `index.html`

f. Refresh the webpage. Your output should look similar to Figure 27.

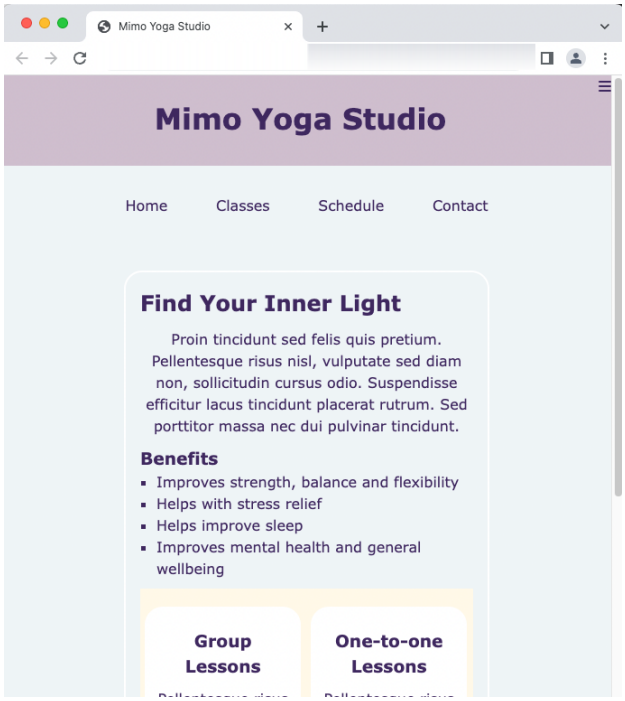

Figure 27. The output of index.html

g. The hamburger icon is now displayed on the top right corner of the page. We will need to tweak a little so that it stays on the same line as the title “Mimo Yoga Studio”. Notice that the height of the header is quite big on a small screen. We will reduce this height first. In your media query, create a rule for `header` element such that the `height` is set to 55px.

h. Configure a rule - `line height` as 50% for the `h1` element, the one which is the direct child element of this `header` element [Hint: this rule is an addition rule shown in Figure 24 above]

i. Now back to your **id** `hamburger`, configure `padding 12px`, and `height` of 55%

j. Save the `yoga.css`. Refresh the browser. Your output should be similar to Figure 28 below.

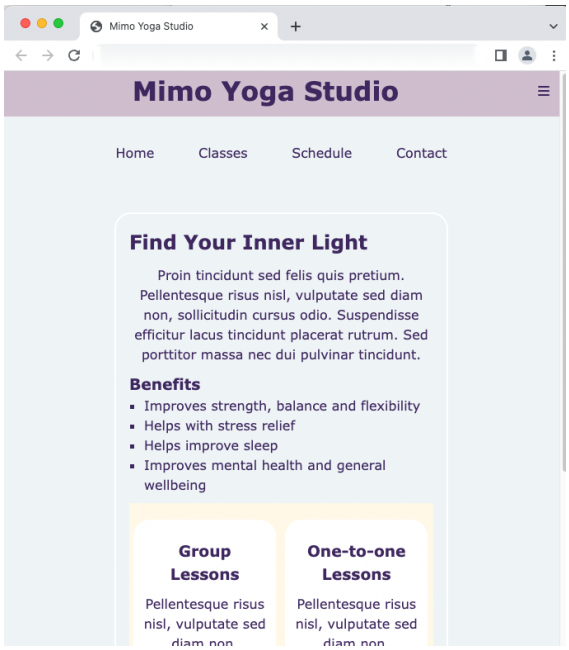

Figure 28. The output of index.html

Now that we have fixed the hamburger menu icon (obviously this can be improved), our next task is to change the horizontal navigation links so that they are rendered as a dropdown list, below the hamburger icon.

k. Back to your media query, create a **class** named `nav`, and a declaration which set the value of `display` property to `block`

l. Save the `yoga.css` and refresh the page. The navigation links are now being rendered vertically as shown in Figure 29 below.

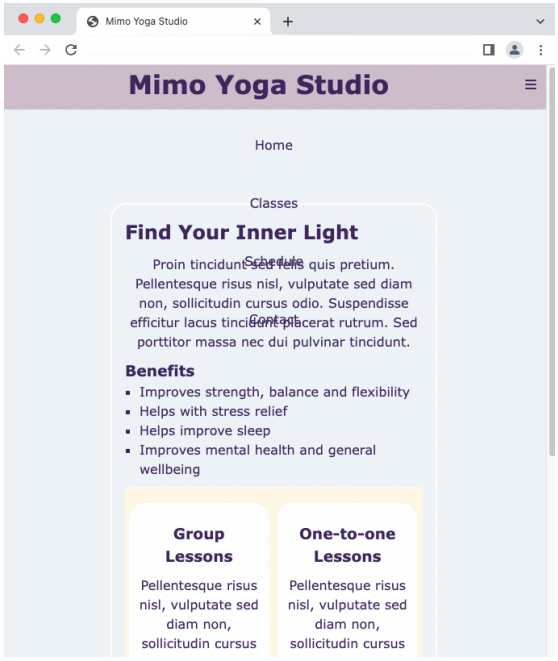

Figure 29. The output of index,html

m. Still on the `nav` class, configure `padding` to 10px, `width` as 70%, and `font-size` as small.

n. Save the file and refresh the page. The gaps between the navigation links have now reduced, as shown in Figure 30 below

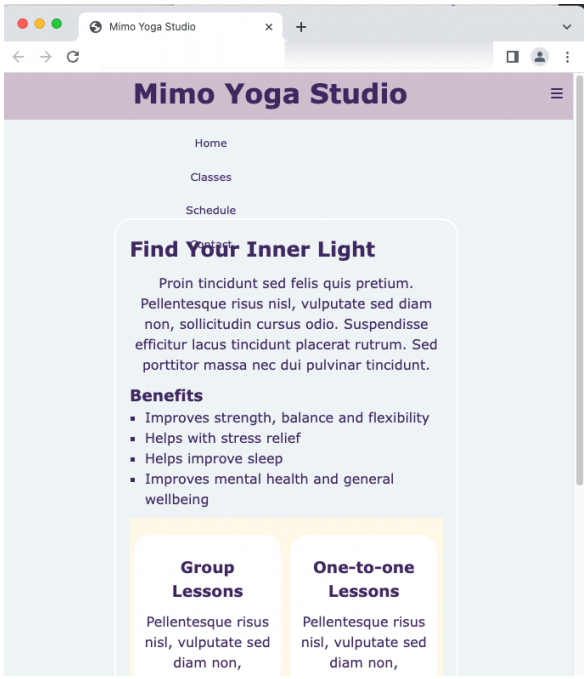

Figure 30. The output of index.html

Next, we will want to move the entire navigation bar to the right side of the page, overlap and sit on top of the header element but below the hamburger icon, as shown in Figure 11 above. To achieve this look, we will configure the position of our

main navigation as absolute and place this element inside a container whose position is relative.

First, let sort out our HTML structure. Examine the `index.html`, notice that our main navigation `<nav>` element is a child of `<body>` element, as shown in Figure 31.

```
<body>
<header>
    <h1>Mimo Yoga Studio</h1>
    <a id="hamburger">
        <i class="fa-solid fa-bars"></i>
    </a>
</header>
<nav id="mainNav">
    <ul>
        <li class="nav"><a href="index.html">Home</a></li>
        <li class="nav"><a href="classes.html">Classes</a></li>
        <li class="nav"><a href="schedule.html">Schedule</a></li>
        <li class="nav"><a href="contact.html">Contact</a></li>
    </ul>
</nav>
```

Figure 31. Fragment code on index.html

We do not want to mess around the `<body>` element. The easiest way to do this is to create a wrapper container and set its position to relative. So,

o. In your `index.html`, create a `div` element with and `id` called *`wrapper`* and place this `div` element below the `body` element, as shown in Figure 32 below, line 10. Don't forget to close this `div` element properly!

```
 9  <body>
10  <div id="wrapper"> <!-- to style for smaller screen -->
11  <header>
12      <h1>Mimo Yoga Studio</h1>
13      <a id="hamburger">
14          <i class="fa-solid fa-bars"></i>
15      </a>
16  </header>
17  <nav id="mainNav">
18      <ul>
19          <li class="nav"><a href="index.html">Home</a></li>
20          <li class="nav"><a href="classes.html">Classes</a></li>
21          <li class="nav"><a href="schedule.html">Schedule</a></li>
22          <li class="nav"><a href="contact.html">Contact</a></li>
23      </ul>
24  </nav>
```

Figure 32. Fragment of code on index.html

p. Now back to your media query create an **ID** called *`wrapper`*, and configure the `position` as `relative`.

q. Next, create another **ID** called *`mainNav`*, and configure the `position` as `absolute`. Now apply this **ID** `mainNav` to the `nav` element in your `index.html`, shown in Figure 32, line 17.

r. Save the files and refresh the page. The navigation links should be rendered on the left side of the page, below the header. Notice that the gap between the navigation links and the header is now gone, as shown in Figure 33.

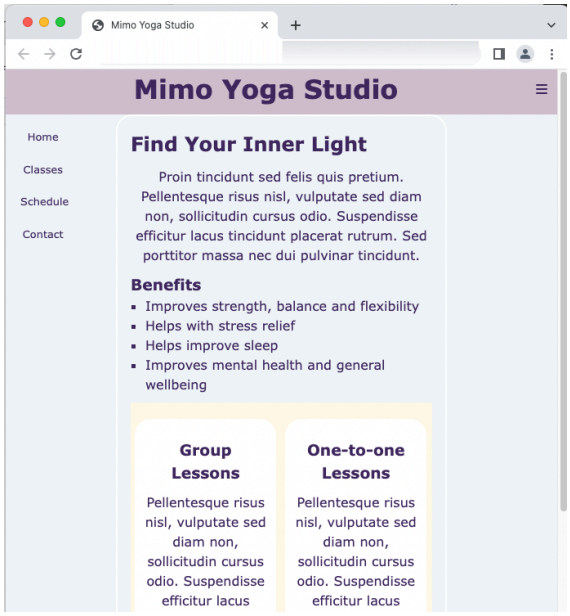

Figure 33. The output of index.html

Next, we will move the navigation element to the right side of the page. We will use z-index (MDN) property for this job.

s. Back to your media query, look for this ID `mainNav`, configure `z-index` to 1, `right` 10px, `top` 20px.

t. Save the file and refresh the page. We can see that the navigation links are now moved to the right side of the page, sitting on top of the `header` element but below the hamburger icon, as shown in Figure 34.

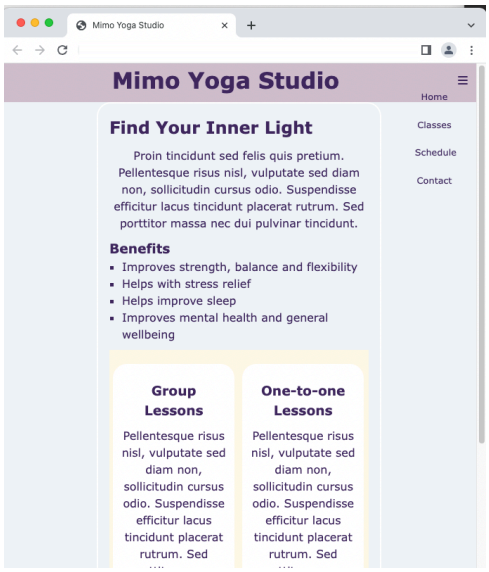

Figure 34. The output of index.html

u. To improve accessibility, we will change the background of the hamburger icon so that the menu stands out a little. Configure the background colour of this hamburger icon to `#e1dbe1`. Your output should look similar to Figure 35 below.

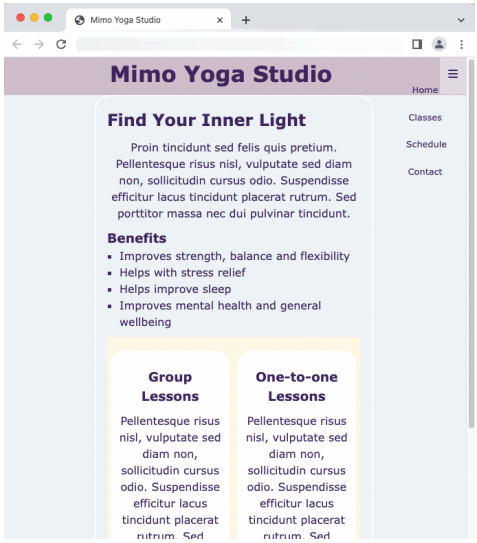

Figure 35. The output of index.html

v. If you resize (make it bigger) your browser window, you will notice that the hamburger icon is there sitting just below the title. Your task now is to fix this by setting the `display` property to `none`.

The final output is shown in Figure 36 and Figure 37 below.

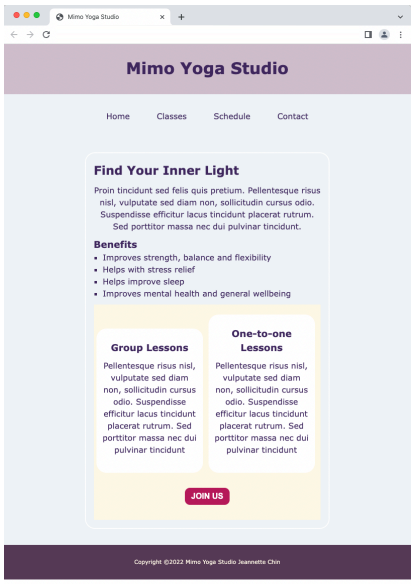

Figure 36. The output of screen bigger than 768px

Figure 37. The output of screen smaller than 768px

w. Change the necessary configurations for `classes.html` such that the navigations behave the same way as `index.html`, as shown in Figure 38 and Figure 39 below.

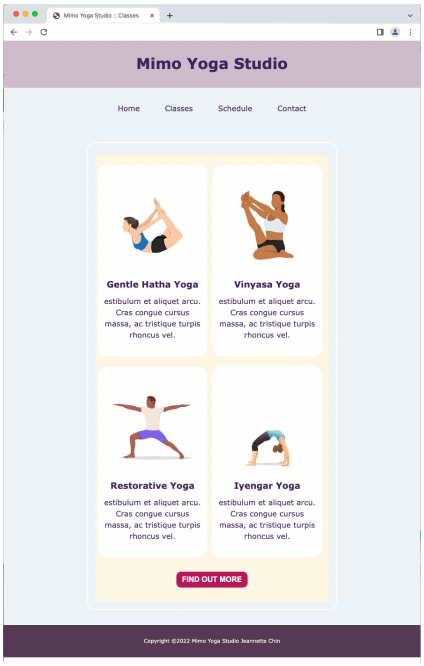

Figure 38. The output of classes.html for screens bigger than 768px

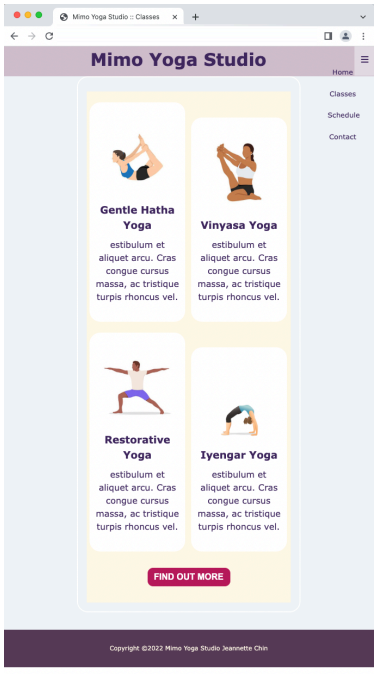

Figure 39. The output of classes.html for screens smaller than 768px

Notice that the four articles on `classes.html` (Figure 39) look rather odd on screen size smaller than 768px. First, they have different heights, and two of them display in a row makes it look rather small and difficult to read. It will be much better if we have one article displays on each row, one after another, and fits in nicely with the container, as shown in Figure 40 below.

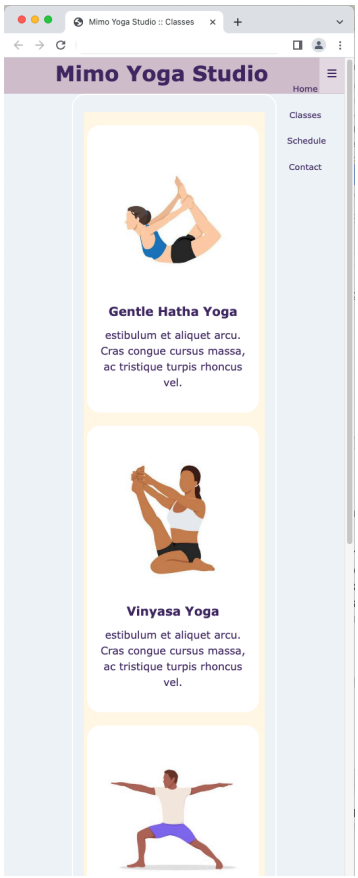

Figure 40. The output of classes.html on screen size less than 768px

We will fix this by using flex box.

a. In your yoga.css - configure the `display` property for `section` to `flex`, and the `flex-wrap` (MDN) property to `wrap`
b. In your media query:
    a. create a rule for `section` and change the `flex-direction` property to `column`.
    b. Create a rule for **class** `service` and set the `width` to 95%
c. Save the files and refresh the page.

Your output should look similar to Figure 40.

Phew! Well done! You have now successfully applied CSS style rules using media queries.

Now that you have completed this lab, show your code and output to the lab tutor. They will sign the lab sign-off sheet if the lab has been satisfactorily completed.