CMP-4011A: Web-Based Programming

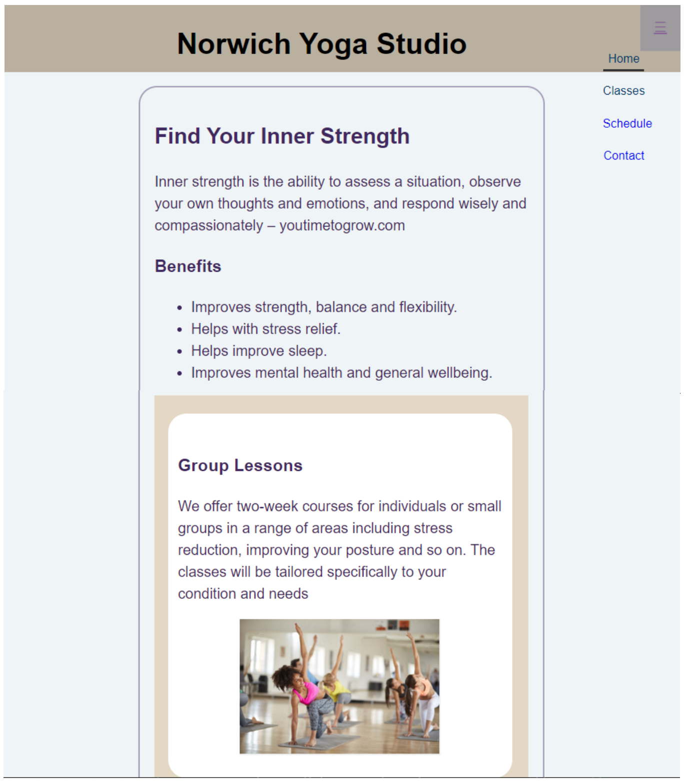
Lab Session (Week 3) – Building Responsive Web Apps

Learning Objectives

- Understand responsive strategies such as media queries, flexible media, etc.
- Optimise web pages for display on a mobile device using the viewport meta tag.
- Apply media queries to target different screen sizes.
- Configure flexible layouts.
- Test responsive designs across various screen sizes.

Background

The objective of this practical session is to apply responsive design strategies to web pages to provide good user experiences regardless of the browser, device, or screen size used to render your website. The selected task involves turning the layout of pages created for the Norwich Yoga Studio (index.html and classes.html created in week 1 and 2) into a responsive design using media queries to target tablet and phones.



Norwich Yoga Studio

Find Your Inner Strength

Inner strength is the ability to assess a situation, observe your own thoughts and emotions, and respond wisely and compassionately – youtimetogrow.com

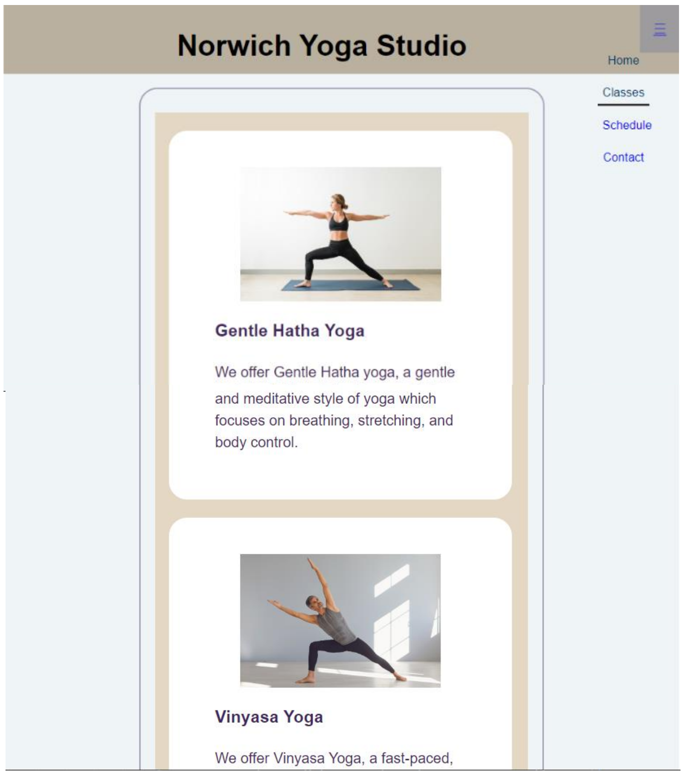
Benefits

- Improves strength, balance and flexibility.
- Helps with stress relief.
- Helps improve sleep.
- Improves mental health and general wellbeing.

Group Lessons

We offer two-week courses for individuals or small groups in a range of areas including stress reduction, improving your posture and so on. The classes will be tailored specifically to your condition and needs

Figure 1.



Norwich Yoga Studio

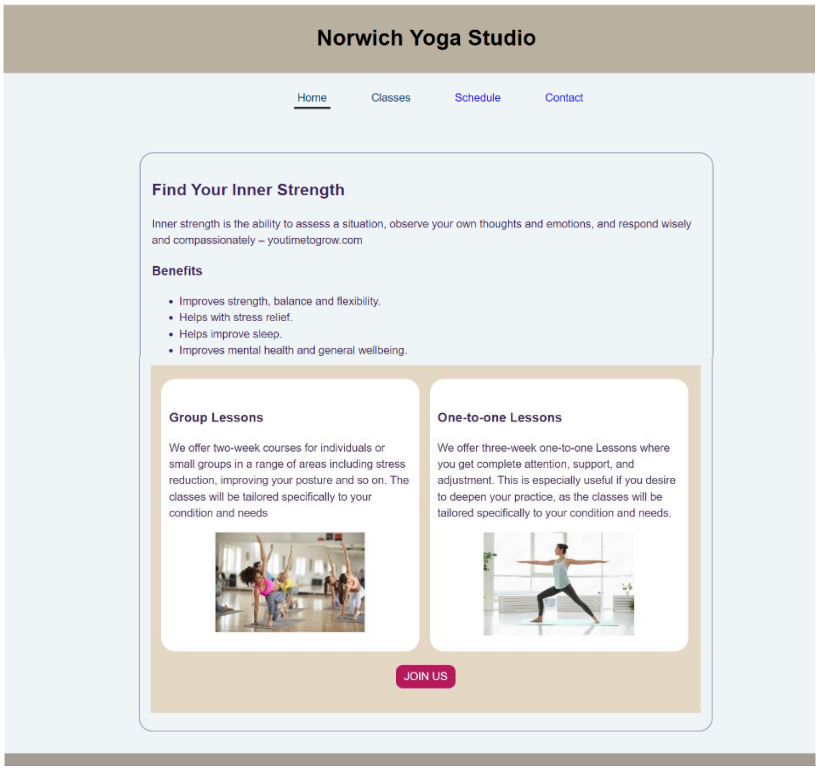
Gentle Hatha Yoga

We offer Gentle Hatha yoga, a gentle and meditative style of yoga which focuses on breathing, stretching, and body control.

Vinyasa Yoga

We offer Vinyasa Yoga, a fast-paced,

Figure 2.



Norwich Yoga Studio

Find Your Inner Strength

Inner strength is the ability to assess a situation, observe your own thoughts and emotions, and respond wisely and compassionately – youtimetogrow.com

Benefits

- Improves strength, balance and flexibility.
- Helps with stress relief.
- Helps improve sleep.
- Improves mental health and general wellbeing.

Group Lessons

We offer two-week courses for individuals or small groups in a range of areas including stress reduction, improving your posture and so on. The classes will be tailored specifically to your condition and needs

One-to-one Lessons

We offer three-week one-to-one Lessons where you get complete attention, support, and adjustment. This is especially useful if you desire to deepen your practice, as the classes will be tailored specifically to your condition and needs.

Figure 3.

Tools Required:

- Text editor (Visual Studio Code – follow a separate instruction on how to start VS code editor on the lab machines)
- Web Brower (Google Chrome)

The practical is quite open-ended and requires your active involvement in discovering and learning various responsive design techniques. Consider the style rules specified as a starting point for you.

The main resources that will assist you are:

Lecture Notes

https://web.dev/responsive-web-design-basics/ - Google's Web Fundamentals

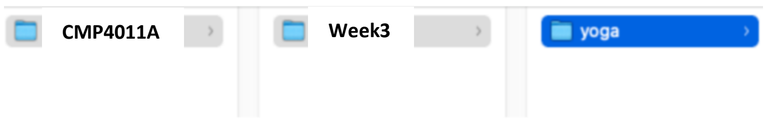
https://www.w3schools.com/css/css_rwd_intro.asp- W3Schools Responsive Web Design

Task 1: - Set up your development environment and folder structure – [10 minutes]

1.1 Open Visual Studio Code editor.

1.2 Create a new directory/folder in your Document directory (U: drive) called week3 and place it in the CMP4011A folder created in week 1.

1.3 For this exercise, copy yoga folder from week 2 and place it in the new week3 folder. This new yoga directory will be the project directory. The folder structure should look like this:



Task 2: - Set up index.html and classes.html for responsive web design – [20 minutes]

2.1. On your code editor, open index.html file and set the viewport, i.e. override default behaviour of mobile browsers by making the screen's width to match the actual display of the phone.

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

2.2. Repeat step 2.1 for classes.html (recall that the meta tag should be specified within the head element).

2.3. **Create a hamburger icon for navigation (to the top right corner) for smaller screens** (Figures 1 and 2). Figure 3 shows index.html on a desktop screen, with the horizontal navigation bar that spans the top of the page. On smaller screens, the nav bar will take too much of the top of the screen (too much of space). **Add a hamburger menu to toggle the menu.** There are several ways to add icons to your web pages, you can load an icon library such as Font Awesome into your project and place icons by using the prefix fa and the icon's name as shown below. Add the following reference to Font Awesome within the <head> section of index.html and classes.html. [Note that the toggle behaviour will be added later in week 4 using JavaScript].

```
<link rel="stylesheet"
      href="https://cdnjs.cloudflare.com/ajax/libs/font-
      awesome/4.7.0/css/font-awesome.min.css">
```

2.4. Place the hamburger icon inside the header element, and wrap the icon inside an anchor <a> element as shown below.

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

Save the file and refresh the page (notice that the hamburger icon is rendered on a separate line. Why?

Task 3: - Position and style the hamburger icon for small screen sizes – [30 minutes]

3.1 At the bottom of yoga.css, create custom style rules for screen only with a max-width of 768px using media queries (@media directive).

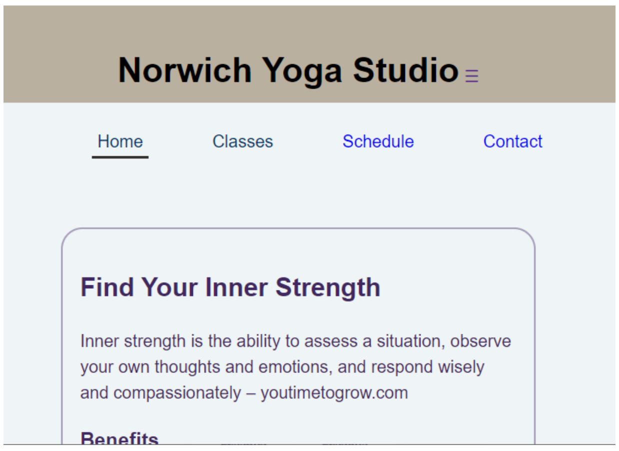
```
@media only screen and (max-width:786px){

}
```

3.2 Display the h1 element within the header inline-block to allow the hamburger to stay on the same line for small screen sizes. The hamburger icon is currently rendered on a separate line because the h1 element is a block level element. As such, its width occupies the entire horizontal space (add a border to verify this). The display property changes this.

```
@media only screen and (max-width:786px){
       header h1{
         display: inline-block;
}
```

Save and refresh the page. Reduce the size of your browser window to a maximum width of 768px to see the effect of the media query as shown below.



Find Your Inner Strength

Inner strength is the ability to assess a situation, observe your own thoughts and emotions, and respond wisely and compassionately – youtimetogrow.com

Benefits

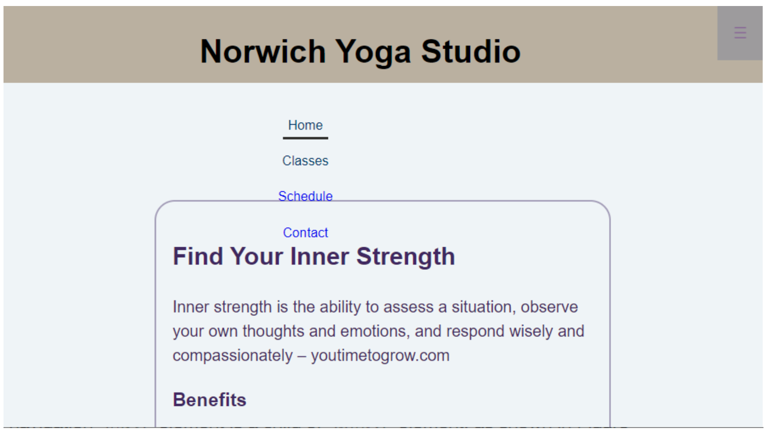
3.3 Next, position the hamburger icon to the top **right** corner of the header element using the **float** property, display as a block level element, add a background colour of #7f869a, add some spacing around it using padding – 2% and some opacity – 0.5 to make it transparent. [Hint: Add an **id attribute** to the <a> element of the hamburger icon in index.html to enable you to reference it for styling]. Note that the hamburger icon is an inline element, as such, to add spaces to the top and bottom of it will require changing it to a block level element using the display property. **Save and refresh the page to see the effect**.

3.4 Tighten up spaces around the header element by shrinking the height of the header element to 18vh, reducing the line height of the h1 element to 50%. **Save and refresh the page to see the effect**. (You should further improve the presentation of the hamburger icon).

```
@media only screen and (max-width:786px){
...
header{
        height: 18vh;
    }
    header h1{
        line-height: 50%;
}
```

Task 4: - Change the horizontal navigation to a drop-down menu for small screens – [30 minutes]

4.1 Within the @media rule, render the list items used for navigation (hint: reference using a contextual selector – `nav ul li`) as block level elements so that they are rendered vertically on the page. Tighten up spaces around the list items by reducing the width to 70%, paddings to 10px and use a small font size (`font-size:small`). **Save and refresh the page**.



Find Your Inner Strength

Inner strength is the ability to assess a situation, observe your own thoughts and emotions, and respond wisely and compassionately – youtimetogrow.com

Benefits

4.2 Move the navigation to the right side of the page so that it sits just below the hamburger icon as shown in Figure 1. To achieve this, we need to create a positioning context for the navigation bar. Create a container element in index.html to contain the header element and the nav element.

```
<div id="wrapper">
    <header>
        <h1>Norwich Yoga Studio</h1>
              <a><i class="fa fa-bars"></i></a>
    </header>
    <nav>
        <ul>
          <li class="active"><a href="index.html">Home</a></li
          <li><a href="classes.html">Classes</a></li>
          ...
        </ul>
    </nav>
<div>
```

4.3 Within the @media directive of the stylesheet, set the position of the above container to relative, and set the position of the nav element to absolute with right and top values of 10px and 20px respectively. The absolute position applied to the nav element is relative to the four edges of the container element. Here is what the CSS looks like:

```
#wrapper{
        position: relative;
}
   nav{
        position: absolute;
        right:10px;
        top: 20px;
  }
```

Save and refresh the page. The navigation element should appear on the right side of the page just below the hamburger icon as illustrated in Figure 1.

4.4 Notice that on bigger screen size, the hamburger icon is rendered at the bottom of the header element, fix this by setting the display value to none outside of the media query. The final output for the navigation area of index.html should look like Figures 1 and 3 for small and large screens respectively.

4.5 Notice that for small screen sizes, the section element, the articles, JOIN US button, footer, etc are now out of place. Fix the styling for these elements by adding the *.services* properties to the @media directive and update its values for small screen. Configure the articles for a single column by removing the float property, remove the height property and value, and specify a width of 92%. Here is what the CSS should looks like:

```
.services{
        width: 92%;
        background-color: white;
        padding: 24px 11px;
        margin: auto auto 20px 15px;
        border-radius: 20px;
        box-sizing: border-box;
 }
```

4.6 Add some white spaces at the top of the main element. **Save and refresh the page.** The final output should look like Figures 1 for small screen sizes.

4.7 Validate your CSS rules using the CSS validator at https://jigsaw.w3.org/css-validator/.

Task 5: - Check and test classes.html for responsive design – [30 minutes]

5.1 Check classes.html for any misplaced content across small and big screen sizes, and fix any issues found.

5.2 Your final output should look like Figure 2.

5.2 Consider tightening up white spaces, adjusting font sizes, and hiding content on small screens.

Well done! You have now successfully made your web pages responsive using media queries.

Show your code and output to the lab tutor. They will sign the lab sign-off sheet if the lab has been satisfactorily completed.

You should upload the completed work to your OneDrive to avoid losing your work.

In any time that you have remaining, you should **meet with your coursework assessment group members** to begin to plan your web application development.

Temi

Oct. 2025

Appendix: Further References

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 */
}
```

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;
    }
}
```

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;
    }
}
```

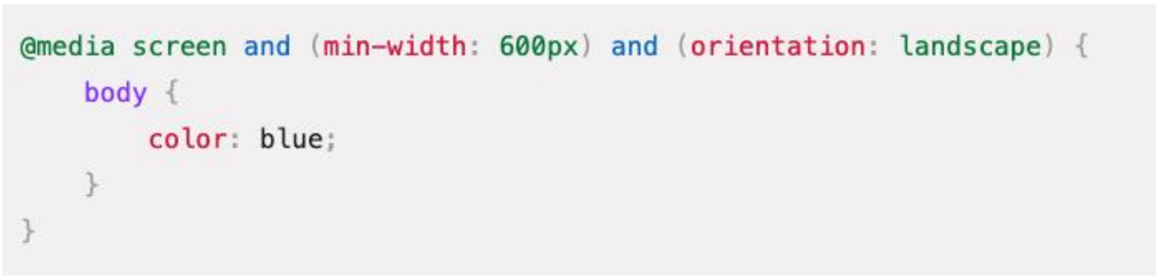
In practice, using minimum or maximum values is much more useful for responsive design so you will rarely see width or height used alone.

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;
    }
}
```

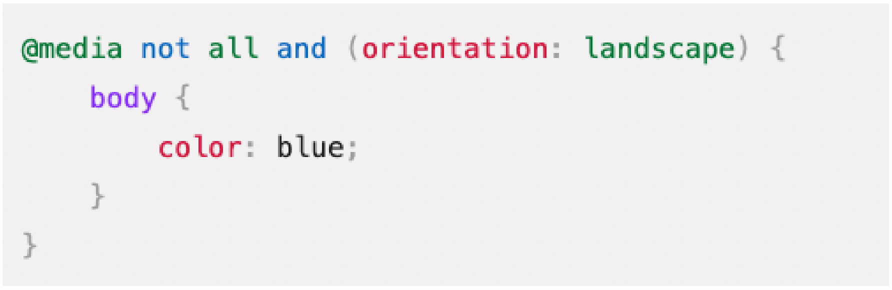
Applying logic in media queries

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;
    }
}
```

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;
    }
}
```

<u>Responsive Design Approach</u>

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 layouts 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:

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

Temi

Oct. 2025