GGR472 Week 2: Web programming basics

Class exercise i: Building a basic webpage and working with Bootstrap

The objective of the exercise is to gain practical experience working in Visual Studio Code and building a basic webpage.

You will create an HTML file with different tags and start to explore functionality using JavaScript. You will have the opportunity to incorporate Bootstrap into your webpage and start to develop an understanding of the style and responsive layout capabilities of the built-in classes within the library.

The exercises are ungraded but their primary purpose is to introduce you to basic web programming knowledge and skills which may be applied in your labs and final web map project. You may therefore use these instructions and your output code as resources when working on future submissions.

PART 1: Building a basic webpage in Visual Studio Code

Step 1: Set up a working folder in Visual Studio Code

All instructions will be provided for the recommended text editor for the course: Visual Studio Code. However, if you have familiarity with alternative software such as Sublime, the exercises may also be completed in a different text editor.

Open Visual Studio Code and select **File > Open Folder**

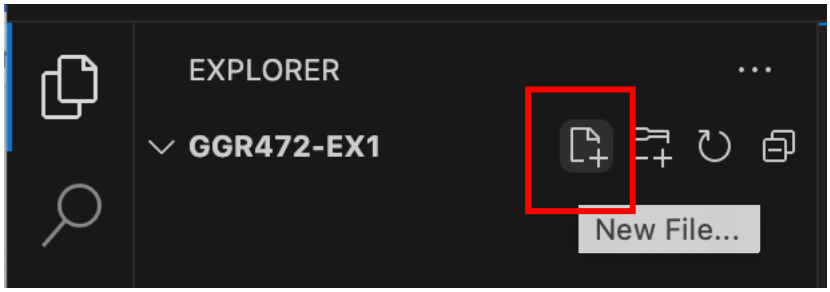
Navigate to a sensible location on your computer and **create a new folder** called *ggr472-ex1.* If you are working on your own computer, you may have a GitHub folder in Documents where you may like to store your coding projects.

> *Note that when you create a GitHub repository in Exercise ii, the folder name cannot have any spaces in it*

Select and open the *ggr472-ex1* folder you have created. **Click "Yes, I trust the authors"** if you receive a message

Step 2: Create a basic HTML page

In the Explorer tree, **create a new html file** called index.html



> *Note that the filename index.html is the conventional name used for the homepage of a website. When you arrive at a website, you are pointing to a directory of page file (e.g., www.websitename.com) and so the web server looks for a file called index by default. This differs to pointing to a specific page (e.g., www.websitename.com/about.html). Using naming conventions also helps other users who may be working with your code.*

In the index.html file, **type an ! and then hit the TAB or ENTER key** to use the Emmet Abbreviation to populate the file with boilerplate code (i.e. the essential html tags).

> *Take time to familiarise yourself with the required tags. Remember that the <head> holds information about the webpage structure and <body> holds information about content. Note that the meta viewport tag enables the browser to scale the page's dimensions so that it is responsive and optimized for viewing on a mobile device.*

Save your file

Edit the following elements of your HTML file:

1. Update the text inside the <title> tag which currently reads Document

   This text will be displayed in the browser tab

2. Add a Heading into the <body> using an <h1> tag
   - In Visual Studio Code, if you type "h1" then hit TAB or ENTER, the tag syntax will populate automatically
   - Add a heading for your webpage by adding text inside the h1 tag

3. Add a comment above the heading using the following syntax:
   - <!-- This is a comment -->
   - For a shortcut, hold either Command (Mac) OR Ctrl (Windows) AND ? key

   Comments are to help make the code more readable and can be used to explain sections of code to other users. Any text inside a comment is ignored when the code is compiled.

4. Add a paragraph in the <body> under the heading using a <p> tag
   - Use a break tag
 to create a new line of text in your paragraph

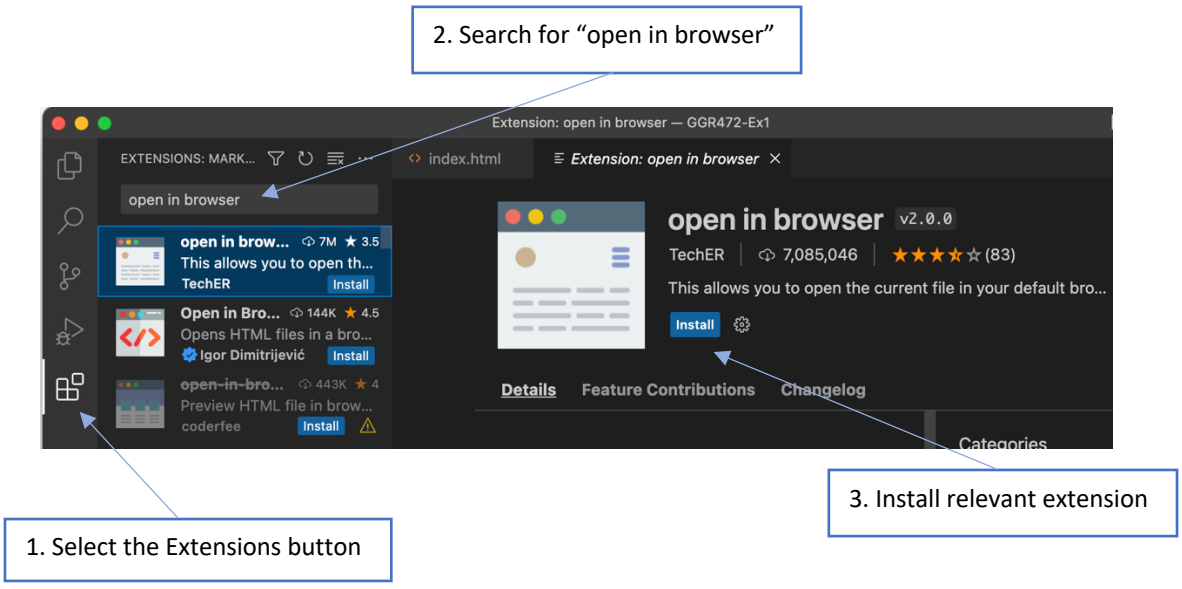
   *
 is an example of a self-closing tag as it does not contain any content inside it. You therefore only need to type
 and not </br>*

Save the file and then open it in your browser. You can do this using the following options:

1. Double-click on the file from the file location in Navigator/Windows File Explorer

2. In Visual Studio Code, right click on the filename in the Explorer tree and select "Open in Default Browser". You may need to install an extension to do this, as shown below

   I *recommend using Google Chrome as your browser. You may therefore need to set the default browser to Chrome or select the "Open in Other Browser" option*



Step 3: Add additional HTML tags and attributes

In the previous step, we explored some basic HTML tags. There are many additional tags which are documented at https://www.w3schools.com/tags/

In this step, tags, which divide an HTML page into sections, as well as <button>s will be introduced. We will also practise incorporating attributes into tags which provide additional information for the browser. Attributes are specified inside the starting tag and are typically denoted as name="value" and may point to a source or style.

Place your heading and paragraph **inside a tag**

Inside your paragraph, **add an anchor <a> tag with an href attribute**. Set the href to the URL of an external website that you want your hyperlink to direct to. E.g.:

```
<a href="https://www.utoronto.ca/">Click here</a>
```

Save your file and test your link by refreshing your webpage in the browser

Notice that the same tab is updated with the external webpage. To open a new tab from the link, **add a target attribute**:

```
<a target="_blank" href="https://www.utoronto.ca/">Click
                        here</a>
```

Create a new tag underneath the previous one and **add a <button> tag** inside it

```
<button>Click me!</button>
```

Step 4: Practise adding basic JavaScript functionality

Test the button. Notice that it acts like a button but nothing happens. To add an action, we will add an event listener to the button element which will trigger an action in the browser. In this instance, a mouse click on a button will start a JavaScript function.

First, **update your button tag** to include an ID attribute

```
<button id="my-button">Click me!</button>
```

In the Explorer tree in Visual Studio Code, **add a new file** and name it *script.js.* The new file should be stored in the same directory as *index.html*

Open the new file and add the following code:

```
const btn = document.getElementById("my-button");

btn.addEventListener("click", function(){
    alert("You clicked me!");
})
```

The <button> element has an event called 'click' that fires when a user clicks the button. Objects that can fire events have an addEventListener() method which in this example takes two parameters: the name of the event ('click') and a function to call when the event happens. In this instance, we call an alert method which opens an alert window

Use // to add comment to your code explaining what each section does

In your HTML file above the closing </body> tag, **add a link to the new JavaScript file** using the <script> tag and source attribute:

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

> *console.log() is a JavaScript method that adds a message to the web console. It is useful for testing and debugging purposes. For example, if you have written a function which updates value x, you can test x is updated as expected with the following syntax:* `console.log(x);`

Save and view your updated file in the browser

To test the button works as expected, right-click anywhere on the page and select **"Inspect".** This will open the Chrome Developer tools, including the console. Take time to explore your options under the Elements tab by selecting different tags and viewing the Styles, Layout, and Event Listeners tabs.

At the top of the Developer tools window, select the Console tab and then click your button. The .log() message should appear in the console.

PART 2: Updating the style of your webpage

The class attribute is often used to point to a class name in a style sheet. For example, if the tag with your heading was updated to `<div class="heading">`, the heading class could now be called within a CSS file (style.css) and updated with specified styling as follows:

```
.heading {
    background-color: aliceblue;
    color: blueviolet;
}
```

Note that the following code would have to be placed within the <head> of the HTML file to link to the CSS style sheet:

```
<link rel="stylesheet" href="style.css">
```

Classes may also be pre-defined in existing libraries or frameworks. In the following section, we will use built-in CSS classes in the Bootstrap framework to update the layout and components of your webpage.

Bootstrap is a JavaScript based library with predefined CSS which enables responsive and stylized elements.

Step 1: Integrate Bootstrap into your webpage

To link the Bootstrap 5 library to our webpage, first navigate to:

> https://getbootstrap.com/docs/5.3/getting-started/download/
>
> *There are different ways to work with Bootstrap. Instead of downloading and saving CSS and JS files to our working folder, we will use the BootstrapCDN (content delivery network) to link directly to them*

Scroll down to **CDN via jsDelivr** and copy the first block of code:

Place the <link> tag in the <head> for the CSS, **and the <script> tag** for the JavaScript bundle below your current script tag for you *script.js* file and above the closing </body> tag.

Save your file and refresh your browser. Notice the difference in font.

Step 2: Update the layout of the webpage using Bootstrap classes

Below the opening <body> tag, **add a new tag** with a Bootstrap class called container

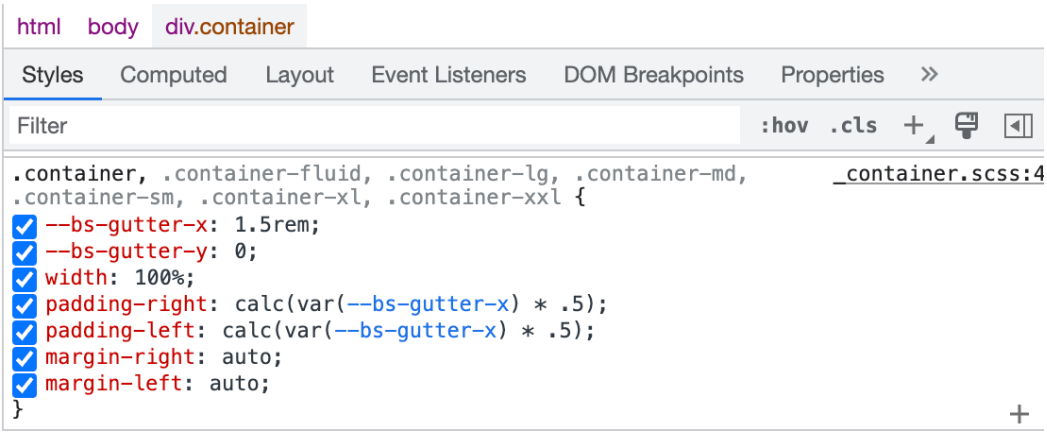
> *. followed by a name signifies a class*
>
> *For a short cut, type .container then ENTER or TAB*

Move both tags created in Part 1, Step 3 inside the container div

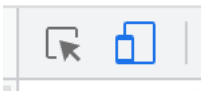
Save the file and refresh the page in the browser.

> *Notice how the layout updated, adding spacing to the lefthand side of the text and button*

Inspect the container class in the Developer Tools and notice the range of Bootstrap classes and pre-loaded style information in the Styles tab. Try toggling the padding and margin styles off and on



In the Developer Tools, switch to "mobile view" and see how the layout responds



Bootstrap is particularly responsive to changing screen sizes because it relies on a grid system. Each screen is divided up into 12 columns and components within the webpage may be specified to cover a number of columns. For example, a button set to 12 columns will cover a full screen while a button set to six columns will cover half a screen.

As an example, `<div class="col-xs-4">Col width 4</div>` sets the minimum width of a column to 4 for extra small screens (phone size) and above.

Update the ** containing the button to `<div class=row>`. Inside the row , **add three tags with the column class:

```
<div class="col">Col 1</div>
<div class="col">Col 2</div>
<div class="col">Col 3</div>
```

Check the updated layout. Note that as we have not specified the width of the columns, they are automatically distributed equally across the page.

In the Developer Tools, select the Layout tab and notice that there are now options inside the Flexbox window. Try toggling on div.row and notice how gridlines for the row and columns are shown. Switch to mobile view and reduce the size of the screen to see how the columns become stacked to optimize spacing of elements.

Step 3: Update webpage components using Bootstrap classes

Bootstrap also has several built-in component classes that allow you to add styled elements to your webpage. For example, **navigate to:**

https://getbootstrap.com/docs/5.3/components/navbar/

Copy the first block of code and paste it below your <body> tag and above the container

Refresh your page. It should instantly look closer to how you might expect a webpage to look. For now, we do not have time to activate all the components in the navbar but we will practise adding functionality by updating our button.

In the Bootstrap documents, explore the available custom styles and corresponding code for different components. You may also like to explore the Bootstrap documentation further when building webpages for your labs.

Remove or comment out the existing button using <!—COMMENTED CODE HERE -->

Add code for a Bootstrap primary button type inside the first column with an id attribute called *my-btn*. The id attribute creates a unique ID for the HTML element and cannot be repeated inside other tags:

```
<div class="col">
     <button type="button" class="btn btn-primary"
     id = "my-button">My Button</button>
</div>
```

Because the ID for the Bootstrap button is the same as the one used previously, the event listener in script.js can be reused.

Check the functionality in the browser