# CPSC 2030 – Web Development II

**Lab5: [65 marks] Flexbox and JavaScript DOM Manipulation**

|  |
| --- |
| **Objectives** |
| * Handle button click events * Fill page with random images * Configure menu and layout with flexbox |
| **Preparation** |
| 1. Read Chapter 9 of the textbook 2. Create a new folder named **Lab5** under your CPSC2030\Labs folder. |
| **What to do** |

**Setup:**

* Download Lab5Files.zip from D2L. The folder Lab5Files contains two starter files: **index.html** and **style.css**. These files implement the basics of the webpage. You shouldn't modify these files at all except where specifically directed to.
* Create a new CSS file called **mystyles.css**. Link this to the **index.html** file. Please write all your CSS styles in this file! Use a mobile-first approach as you style the page.
* Create a new JavaScript file called **script.js** and link this file to the **index.html** file using a <script> tag. Either place the script tag under the page content, or place it in the head of the document with a defer attribute to have it run only after the page content has loaded.

**Task 1 [20 marks]: Info Button with JavaScript**

Notice the grey **"i"** button in the top-right corner of the page. When we click on that, a full-page modal should appear with some additional info about the site. Near the bottom of **index.html** you will find the **<aside>** element that will be used for this modal; you'll notice that this aside contains a **<button>** element (with Font Awesome "x" icon) that is meant to close the modal after it's been opened.

Your job is to implement this through JavaScript in your **script.js** and through CSS in your **mystyles.css**. Use the following strategy:

* Get the "i" and "x" buttons (for opening and closing the modal, respectively) using **document.getElementById** and store them in variables.
* Create an event handler function called **handleInfoToggle**. The function should get the <aside> element and **toggle** a class on that element called **visible**. (so clicking on the "i" or "x" buttons either adds or removes the **visible** class)
* Add the above event handler function to the two buttons using **addEventListener**. The function should be activated on **click**.
* To test your JavaScript to make sure this is working, examine the **<aside>** element with the browser **inspect tool** to see if the class is being added and removed when the buttons are clicked.
* Using the **visible** class and some other selectors, write some styles inside your **mystyles.css** that make the <aside> element visible and style it as in the video. A few notes:
  + You can make the element visible on the page by setting **display:block**. (It was set to **display:none** by default.)
  + Use **position:fixed** to place the <aside> at a fixed position with respect to the browser window. Set the values of **top, bottom, left and right** to **0.** (This will stretch the element across the entire viewport.)
  + Set a limit on the width of the **.aside-wrapper** element and center it horizontally. (You can use the usual wrapper centering techniques for this. Don't worry about centering it vertically yet. It's fine if it's stuck to the top of the viewport for now.)
  + Position the "x" button in the top right corner. Use your knowledge of **position:absolute** to make this happen.
  + To center the **.aside-wrapper** element vertically, use CSS grid and the **align-content** property. We did not introduce at this property in this course, so figure out how it works! Some help: <https://css-tricks.com/snippets/css/complete-guide-grid/>

**Task 2 [20 marks]: Filling the Page with Random Images**

Our next task is to write JavaScript in your **script.js** file to fill the page with images! Let's create a new function to do this:  
  
***function populateImages(subject) {***

***// your job is to fill in this part!***

***}***

***populateImages("nature"); // calling the above function***

The ***subject*** parameter is meant to take a **string** that corresponds to the subject of the images we would like to generate. As you can see, we're initially calling the function with "nature" as the parameter, although we'll eventually allow the user to choose a new subject by clicking on one of the nav buttons.

So, your job is to implement the **populateImages** function. The function should generate a bunch of random images and place them in the DOM. Do it like the following:

* Use **document.getElementById** to get the <div> element with id "**photos"**. This is the container where we will put the images.
* Use the **innerHTML** property to clear out whatever is inside the <div>. (Initially it's empty, but later it will have images in it, and we'll have to clear them out every time we generate a new set of images.)
* Create a **for loop** that loops between **0** and **13** (including 0 but not 13.)
* Using **innerHTML** once again, add a new <img> tag to the "photos" div in each iteration of the loop. Note that you can add to whatever is already in the **innerHTML** by using the **+=** operator.
  + For the value of the **src** attribute of each image tag, we're going to use the **Unsplash Source** API that randomly fetches images based on search terms. Check it out here: <https://source.unsplash.com/>
  + The relevant API endpoint is this:   
      
    ***https://source.unsplash.com/1600x900/?nature***  
    Try pasting the above URL into your browser to see what happens. As you can see, we can choose the size of the image (1600x900 in this case) and the subject of the image (nature in this case) by formatting the URL.
  + So, your job is to format the **src** URL in the following ways:
    - Set the width of the image in the search URL to **440** (not 1600)
    - Set the height of the image in the search URL to a random integer between **500** and **700** (It is important to change the width or height each time you search the site in the loop. If you don't, you will always get the same image in the loop). Some tips:
      * You can use **Math.random()** to generate a random decimal number between **0** and **1**. Figure out the math necessary to translate this to a random number between **500 and 700**. You can use **Math.ceil()** to round a decimal number up to the nearest integer.
    - The search term for the image (the part after the **?** in the above URL) should be the **subject** parameter that has been passed into the function when it is called.
    - Remember that you can create a new string by concatenating several strings together using **plus signs** +. Example:   
        
      ***console.log("Here's a random number for you: " + (Math.random()\*10) + ". I hope you like it!"); // outputs "Here's a random number for you: 9.346110309236629. I hope you like it!"***
    - The **alt** attribute of each image should be set to the **subject** parameter that has been passed into the function.
    - You don't need to set the width and height attributes in the img tag.

Once you have this working, you should see 13 images appear on the page! A new set of images should appear every time you reload.

**Task 3 [10 marks]: Subject Buttons Implementation**

We'd like to call the **populateImages** function that we wrote earlier every time the user clicks on a nav button. So we should attach event listener/handlers to each of the nav buttons. To do this:

* Create an event handler function called **subjectButtonClickHandler** that does the following:
  + Calls **populateImages,** passing in the text of the button that was clicked as a parameter. You can get the text of the clicked button like this: **event.currentTarget.textContent**
* Get all the nav buttons and store them in a variable by using **document.querySelectorAll**. This function is used to get a list of HTML elements based on a CSS selector.
* Iterate over the nav buttons using a **for** loop. For each button, use **addEventListener** to add the **subjectButtonClickHandler** as an event handler function on **click** events:

Once you have this working, clicking on any of the nav buttons should clear out the old images and reload a new set of images with the chosen subject!

**Task 4 [15 marks]: Navigation Menu**

Your next task is to style the navigation menu using the **flexbox** layout system. As you may know, **flexbox** is a good choice for styling one-dimensional layouts, or layouts that gain a second dimension from elements wrapping to a new row or column.   
  
Inside your mystyles.css, create a new media query with the min-width of 900px. Inside this media query, create a flex layout for the navigation menu items with the following properties:

* + The items should be horizontally arranged with a bit of space between them;
  + The items should be ordered right to left instead of the default left to right order that we would get if using display:inline-block. So, this means that the last item in the HTML (art) should be furthest to the left, and the first item (science) should be furthest to the right.
  + Each item should grow to fill the space so that all the items together fill the full width of the wrapper.
  + When the viewport is made smaller, the items should automatically shrink to the appropriate size so that they all still fit in one row without wrapping.
  + Display the second subtitle vertically on the right.

|  |
| --- |
| **What to hand in** |
| * Zip your Lab5 folder and upload it to BrightSpace. |
| **When to hand in** |
| * By 11:59pm, Monday, October 9, 2022 |