ENSF-381: Full Stack Web Development Laboratory

Ahmad Abdellatif

Department of Electrical & Software Engineering

University of Calgary

Lab 2

Objectives

Welcome to the ENSF381 course lab. This lab introduces the basics of creating and understanding CSS. The main objective is to learn how CSS properties affect the layout and appearance of a webpage, and to practice applying styles in a systematic way. Follow the instructions carefully.

Groups

Lab instructions must be followed in groups of two students. Submissions from groups with fewer than two students will not be accepted.

Submission

- Use only the HTML tags and CSS properties that were covered in class or listed in the lab instruction document.
- The groups must submit the completed Answer Sheet and the complete source code, ensuring it can be executed without any modifications. If requested by the instructor, you may need to submit additional documentation (e.g., word documents and images).

Deadline

Lab exercises must be submitted by **11:55 PM on the same day as the lab session**.

Exercise 1: Styling a Webpage using CSS

Objective: In this exercise, we will learn to use CSS classes and selectors effectively to create reusable and modular styles, promoting efficiency and maintainability in web design. We will style a given HTML page to match a predefined visual design of the UofC history webpage: https://ucalgary.ca/about/our-history

1. Paste the HTML code from the supplementary material (exercise1.html) into your editor (e.g., VS Code). This gives you the webpage's structure, but it will not include styling beyond basic HTML defaults. You will add the styles yourself, following the instructions.

2. **Create a CSS file** and save the new file named styles.css in the same directory as the HTML file you copied in step 1.

3. Add a <link> tag in the <head> section of your HTML file to connect it to your CSS file. This allows the browser to apply your CSS styles to the HTML.

4. Remove the default margins and paddings applied by browsers to <body> element. Note the impact of this step on the space between the banner image and the page body.

5. Create a class for the second table (e.g., table2) and set it to add vertical spacing between rows using a spacing value of 20px. Apply the created style to Table 2 in the provided HTML file. This adds a visual separation between rows in the table that contains images and text, making the content easier to read.

6. Now, we need to apply background colors to the content boxes describing historical events. Thus, create three classes, one for each box. Change the text-align for each box to center and assign the following background colors:

   - Box 1: Dark Red
   - Box 2: Deep Pink
   - Box 3: Brown

7. After updating each box's background color, style the text by creating a class called box_text_color and applying it to all text inside the boxes. Use this class to:
    - Change the text color to white.
    - Add padding of 10px around the text.
    - Remove margins.
8. Note that the style above does not affect the content within the <a> tag. This is because in most browsers, <a> tags do not inherit the color of their parents. Change this behavior by setting the color attribute of all <a> tags that occur within box_text_color containers to inherit.
9. In the last lab, all images (except the banner image) were given the same width and height. This makes the webpage harder to maintain. For example, changing the image size would require updating three different places. In this step, you will standardize the image sizes in sections like "Calgary's Urban Evolution."

    Create a class named td_image_cell and apply it to all image containers **in the second table**. In this class, set the container width to **570px**. Then, using an appropriate CSS selector set the dimensions for all images within the td_image_cell containers to **100% wide** and **100% tall**, displayed as **block** elements.

10. Next, style the text describing the history and milestones on the webpage to match the expected output (expected_output/exercise1_output.pdf in the supplementary material). For **all paragraphs**, set the font to **Arial**, the font size to **16px**, and the text color to **dark grey** (#3d3d3d).

11. Open your HTML file in a browser to verify the applied styles. Focus on the appearance of tables, boxes, paragraphs, and images. In case you page does not match the output, revisit the related class and check for typos or incorrect values.

Submission:

- Fill out the Answer_sheet.docx, including:
    - The names and UCIDs of all group members.
    - A screenshot showing the output of Exercise 1.

- Create a new GitHub repository and upload the code for Exercise 1 to the new repository.

Exercise 2: Flexbox

Objective: In this exercise, we will explore Flexbox and apply its properties such as flex-direction, flex-wrap, order, align-items, justify-content, and align-self, enabling them to design responsive and visually appealing web interfaces.

1. Create a new html file in your code editor.
2. Create a flex container with five items, arranged in the **exact order** listed in the `exercise2/info.txt` file within the supplementary material. For each item, include the following:

- **An image**: one of the images within the exercise2 folder from the supplementary material (e.g., schulich.png).
- **A level three heading**: Provide a title for each item (e.g., "Schulich School of Engineering").
- **A description**: A brief description for each item (e.g., "UCalgary's hub for innovative engineering education.")

3. Create the following styles and apply them to the container and items you created in Step 2. Place these styles inside a `<style>` tag in the `<head>` of your HTML document.

```
.container {
  border: 2px solid black;
  padding: 10px;
}
.item {
  padding: 20px;
  margin: 5px;
  background-color: lightblue;
  border: 1px solid darkblue;
  text-align: center;
```

```
}
.item img {
    width: 150px;
    height: 150px;
    display: block;
    margin: 0 auto 10px;
}
```

4. Save and open the file in your browser. Make a note of the current layout.
5. Now edit the style applied to the container and item classes to match the following:

```
.container {
  display: flex;
  border: 2px solid black;
  padding: 10px;
}

.item {
  flex: 1;
  padding: 20px;
  margin: 5px;
  background-color: lightblue;
  border: 1px solid darkblue;
  text-align: center;
}

.item img {
  width: 150px;
  height: 150px;
  display: block;
  margin: 0 auto 10px;
}
```

6. Save the file and open it in your browser. The items should now appear in a single horizontal row because flex containers use a row layout by default. Take a screenshot that clearly shows all five items and insert it into AnswerSheet.docx.

7. Add the following style to the .container class:

   flex-direction: column;

8. Observe the arrangement of the items. Take a screenshot and add it to the AnswerSheet.docx. Then remove the style added in step 7.

9. Experiment with the following:

   - flex-direction: row-reverse;
   - flex-direction: column-reverse

   For each change, take a screenshot, insert it into the AnswerSheet.docx, and remove the just-added style afterward.

10. Add the following styling to the .container class: flex-wrap: wrap;

11. Reduce the width of the browser window and observe the container's default behaviour as the width reduces.

12. What does the layout look like when you have reduced the width of the browser window to the minimum? Take a screenshot and add it to the AnswerSheet.docx.

13. Remove the style added in step 10**.**

14. Experiment with:

    flex-wrap: wrap-reverse;

    Open the document in your browser again and observe the changes. Take a screenshot and insert it into AnswerSheet.docx. Then remove the styles you added.

15. Add the following styles to specific items:

```
.item:nth-child(1) {
  order: 3;
}
.item:nth-child(2) {
  order: 1;
  flex: 4;
```

```
}
.item:nth-child(3) {
 order: 2;
}
```

16. In your browser, observe how the items' visual order and relative size changes. Take a screenshot and add it to AnswerSheet.docx. Your final screenshot should match the expected output (expected_output/exercise3_output.png)

Submission:

- Add the screenshots you took during exercise 2 to the same AnswerSheet.docx you edited for exercise 1.
- Upload the code for Exercise 2 to the GitHub repository you created for this lab session.

Exercise 3: Navigation Bar

Objective: In this exercise, we will use Flexbox to design a responsive navigation bar that efficiently arranges menu items. In addition, we will implement a dropdown menu to enhance user interaction, ensuring smooth and intuitive navigation. This will help in understanding Flexbox properties for layout structuring and dynamic UI components.

1. Copy the HTML code and CSS (exercise3.html and style3.css) provided in the supplementary material into your editor (e.g., VS Code). This code gives you the structure of the webpage.
2. Modify exercise3.html and style3.css to create a Flexbox-based navbar containing the following items (see exercise3_output.png in the expected_output folder for the expected output):
   - Schulich School of Engineering (https://schulich.ucalgary.ca/).
   - Electrical Engineering (https://schulich.ucalgary.ca/electrical-software).
   - Biomedical Engineering (https://schulich.ucalgary.ca/biomedical/home).
   - Chemical Engineering (https://schulich.ucalgary.ca/chemical-petroleum/home).
   - Geomatics Engineering (https://schulich.ucalgary.ca/geomatics).

3. The background color of the navbar should change to #333 when the user hovers the mouse over it **(check the output in the supplementary material).**
4. Add a dropdown menu beneath the first navbar item ("Schulich School of Engineering"). The dropdown should appear only when you hover over the first item (see the supplementary output). Include the following links in the dropdown:

- Home
- About
- Research

NOTE: All styles should be added to the style3.css file.

Hints:

1. Use Flexbox to lay out the navbar horizontally.
2. Add a hover effect to the links so their background color changes to #bdbbbb when hovered.

Submission:

1. In the Answer_sheet.docx you edited for Exercise 1, include a screenshot showing the output of Exercise 3.
2. Upload the code for Exercise 3 to the same GitHub repository you created for Exercise 1.
3. Compress the code for Exercise 1, Exercise 2, and Exercise 3 into a single file and upload the compressed file to D2L. Also, upload the completed Answer_sheet.docx.
4. Include a link to your GitHub repository in the D2L submission box.