Tutorial and Assignment: CSS Frameworks

The purpose of this is assignment is to get practice using a few popular CSS frameworks to design a user interface, namely **Tailwind CSS** and **DaisyUI**. These are just a few CSS frameworks available; there are many others!

CSS Frameworks

At this point in the course, you've probably already noticed that writing CSS is hard! Not only is it difficult to implement the graphic designs and functionalities you want, but the software design aspects of front-end development—such as maintainability, extensibility, etc.—are non-trivial.

CSS frameworks are meant to simplify this and take some of the burden away by providing pre-written CSS primitives you can put together to build the interfaces you want. You may be able to do it without ever touching CSS code! The tradeoff is less control over the design and implementation and more coupling between the structure, content, and style.

Tailwind CSS

Tailwind CSS is currently a very popular framework; it presents an alternative to the paradigm used by older frameworks such as Bootstrap that provided pre-styled components for common UI elements. Instead, Tailwind CSS provides so-called ***utility classes****:* classes you can add to HTML elements to apply small, single-purpose styles.

Utility classes are like small building blocks that one combines to develop a design. They abstract CSS by combining complex properties, or sets of properties, with pre-chosen values, into a single class. Here's an example:

...

The two classes above, **grid** and **grid-cols-3**, set up the as a three-column grid. One doesn’t even have to understand how the CSS in the background works; one only needs to look up which utility classes are necessary to achieve the desired result.

DaisyUI

DaisyUI, on the other hand, does follow the more traditional approach of providing a set of pre-built components, but the components are themselves built with Tailwind CSS, and this framework is meant to be used alongside Tailwind CSS. It is one of many frameworks that provide Tailwind-based components. Using this framework often involves copying and pasting some HTML code for a given component into your markup and adjusting its classes as needed to configure it.

Tutorial: Tailwind CSS

Installation and Setup

Before you start creating a UI design with Tailwind, let's first go over some of the basics of how it works, starting with installation. For learning and experimentation, the easiest way to install is to simply link to Tailwind's CDN. Create a new HTML file with a basic template, and paste the following code into the **<head>:**

<script src="https://cdn.jsdelivr.net/npm/@tailwindcss/browser@4"></script>
<style type="text/tailwindcss"></style>

This downloads the Tailwind JavaScript code and CSS styles from a specialized server. (When you're getting ready to deploy to production, you'll want to use one of the other build methods on the above page.) More information can be found here: https://tailwindcss.com/docs/installation/play-cdn

Now in the **<body>** of the document, paste the following code to use for testing:

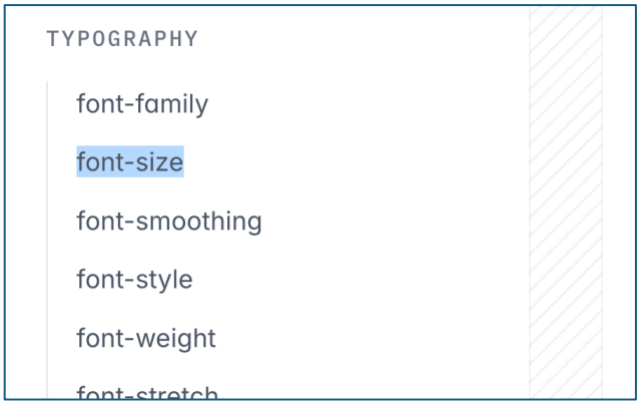
```
<div>
    <h1>Lorem Ispum Dolor Sit</h1>

    <p>Lorem ipsum dolor sit, amet consectetur adipisicing elit. Ipsa debitis
    aspernatur fuga laborum, sequi quas aut quidem nulla, minima animi deleniti
    tempora modi cupiditate soluta nemo reiciendis delectus? Corporis, illo?</p>

    <p>Lorem, ipsum dolor sit amet consectetur adipisicing elit. Repudiandae
    assumenda, at perferendis nihil itaque, earum culpa explicabo, quis porro
    commodi in numquam nemo ipsam neque mollitia maiores molestiae soluta
    doloribus.</p>
</div>
```

Documentation and Basic Styling

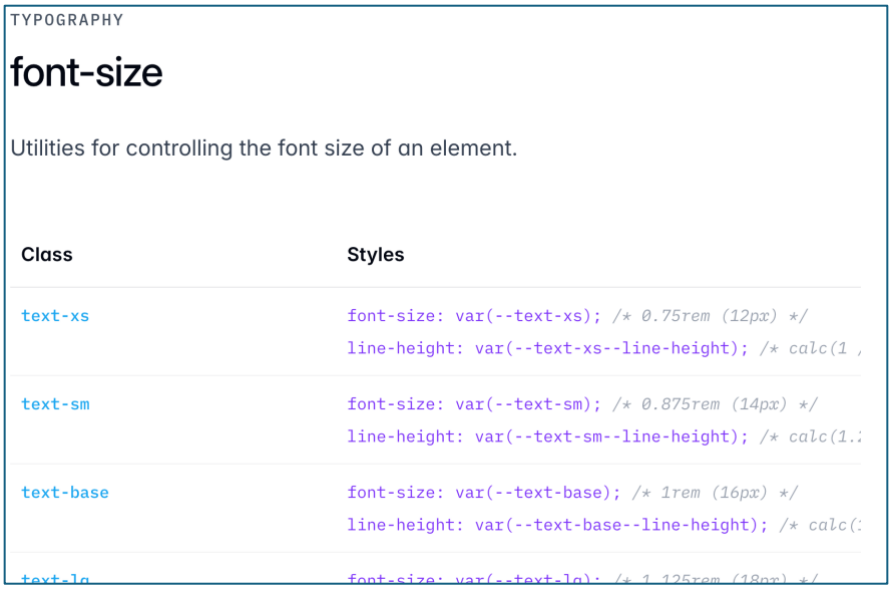
When you view this page in the browser, you'll see that most browser-default styles have been stripped away. Let's add a few styles using Tailwind CSS's utility classes.

You'll find all the documentation you need on Tailwind's official documentation site: https://tailwindcss.com/docs/installation/using-vite. The menu bar on the left has links to different sections of the documentation. Interestingly, the sections are named by the CSS property that is being replaced.

Let's start by increasing the size of the **<h1>** heading. Press **CTRL + F** on the documentation site and search for **"font-size"**, which is, of course, the name of the CSS property we would use if we were to do this with CSS. You should find the link the menu section "TYPOGRAPHY" highlighted:



When you click this link, you'll be taken to a page with a list of **utility classes** for setting the size of the font. Go ahead and pick one that corresponds to a larger size.

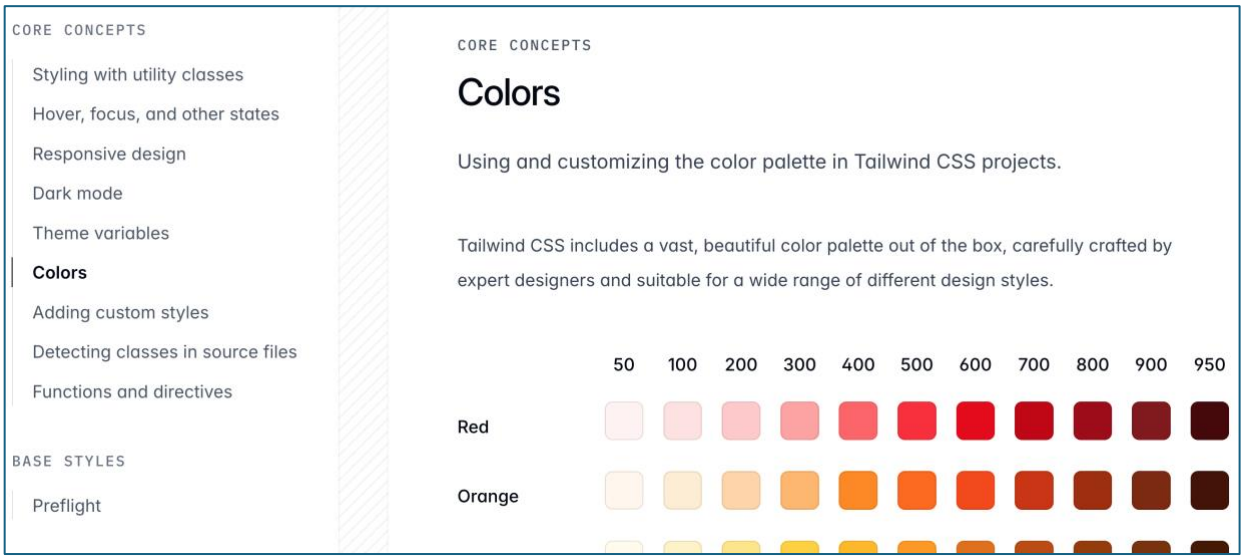


When you've chosen one (I chose **text-6xl**) paste it into the **class** property of the **<h1>** tag:

```
<div>
    <h1 class="text-6xl">Lorem Ispum Dolor Sit</h1>

    <p>Lorem ipsum dolor sit, amet consectetur adipisi
```

When you refresh the page, you should now see that the heading has a respectable size.

Let's try changing the color now. In the "CORE CONCEPTS" area of the navigation menu on the documentation page, find the **Colors** page:



As you can see, there's a grid of colors to choose with examples of how you can use these colors. Colors are applied by classes that have the following format: **target-name-darkness,** where the **target** is the thing to apply the color to (**bg-** for background, **text-** for text, etc.), the **name** is the name of the color (**red, orange,** etc.) and the **darkness** is a number between 50 and 950 representing how dark the color is. To practice this, let's set the **text color** of the **<h1>** heading to **Amber 600** and the **background color** of the ** ** to **Blue 50:**

```
<div class="bg-blue-50">
    <h1 class="text-6xl text-amber-600">Lorem Ispum Dolor Sit</h1>

    <p>Lorem ipsum dolor sit, amet consectetur adipisicing elit. Ipsa
    aspernatur fuga laborum, sequi quas aut quidem nulla, minima anim
```

Layout and Box Model

Now that we have font size and colors figured out, let's adjust the box model properties. In particular, it would be nice to have some space between the elements. On the documentation site, CTRL + F to search "margin":

Once again, you'll see a long list of utility classes that can add margin in different ways. You'll also see that the size of the margin can be controlled with a <number> parameter, which takes an integer as a value, like this:

m-8

Larger numbers correspond to larger values, but more precisely, a difference of **1** corresponds to **0.25rem**. We can also add margins on different directions, like the **bottom**. Try this:

```
<h1 class="text-6xl text-amber-600 mb-8">L

<p class="mb-8">Lorem ipsum dolor sit, ame
debitis aspernatur fuga laborum, sequi qua
deleniti tempora modi cupiditate soluta ne
p>

<p class="mb-8">Lorem, ipsum dolor sit ame
Repudiandae assumenda, at perferendis nihi
porro commodi in numquam nemo ipsam neque
doloribus.</p>
```

As you can see, **mb** stands for "**margin-bottom"**, and the **8** indicates a spacing of **8 * 0.25rem = 2rem.**

Currently, the container of the content (the ** **) spans the entire width of the screen, which is usually not what we want. To automatically set the container width based on the current media query breakpoint, use the class **container**, like this:

```
<div class="bg-blue-50 container">
    <h1 class="text-6xl text-amber-600 mb-8">Lorem Is

    <p class="mb-8">Lorem ipsum dolor sit, amet conse
```

To center the container, we would normally set **margin: auto**, which we can do with a Tailwind class **m-auto,** like this:

```
<div class="bg-blue-50 container m-auto">
    <h1 class="text-6xl text-amber-600 mb-8">Lo

    <p class="mb-8">Lorem ipsum dolor sit, amet
```

Responsive Design

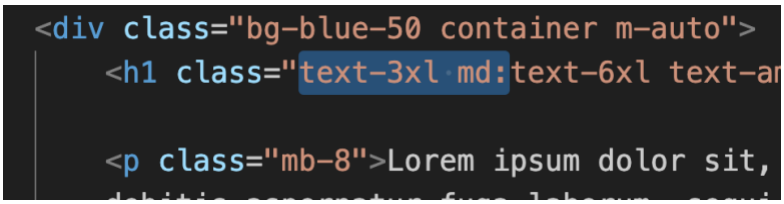
Obviously responsive design is something any CSS framework must be able to handle. Tailwind's approach to this is to explained in the documentation at **CORE CONCEPTS > Responsive Design**:

Basically, one can target any utility class to a particular viewport size by prepending one of the following to the class:

sm:
md:
lg:
xl:
2xl:

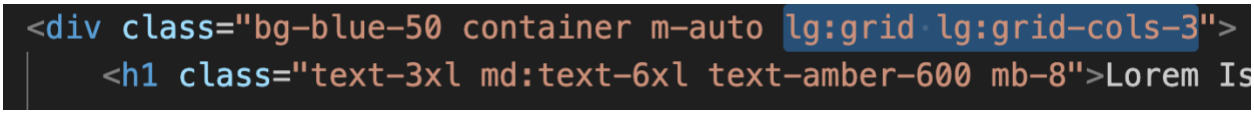
Tailwind uses a **mobile-first** approach, which means that mobile styles don't require any of the above prefixes, and each prefix is understood to affect all sizes above it.

Try this out by applying two different heading sizes depending on the screen size:



As you can see, the text size for mobile and small screens is **3xl,** while the text size for medium and larger screens is **6xl.**

We can also use this to control the layout. For large screens, let's use a grid with three columns:



Assignment: Styling a Page with Tailwind CSS and DaisyUI

Now that you've got the hang of the basics, it's time to actually create a web design with Tailwind CSS! We'll also add a few pre-built, pre-styled components using DaisyUI.

In the starter files, you'll find screenshots and a video of how the page should approximately look when it's done. Your version doesn't have to perfectly match this, and you'll make some creative decisions along the way.

You'll also find a starter HTML file called **index.html**. Feel free to fill in your own favorite TV shows, although you should use the same HTML structure that I used. Once styling begins, you should not edit the HTML structure of the page at all except to add classes.

Task 1: Utility Classes

For now, your design should follow that depicted in the screenshots within the **only-tailwind** folder.

As you work through this assignment, you'll have to consult frequently with Tailwind's documentation: https://tailwindcss.com/docs/installation/using-vite

All styling should be applied through utility classes. DO NOT use CSS (until directed to in Task 2).

Here is a guide to help you through the process:

- For now, the font family should be set using one of the classes from here: https://tailwindcss.com/docs/font-family. Later, we'll set font families using a slightly more sophisticated approach.
- The layout should be fully responsive to every screen width.
    - Note the grid layout for various sizes; try to replicate this.
    - The navigation menu should switch from a vertical to horizontal layout at a breakpoint that you think is appropriate.
        - The navigation should be **sticky** for larger screens.
- Choose colors you like from Tailwind's palette.
    - Don't use the same color scheme I used.
    - The background color of the navigation should be **partially transparent**.
- Choose appropriate font sizes for all text elements, using your own judgement.
- Set appropriate box model and border properties; you can choose the values.
- The image should have text wrapping.
- Use utility classes to set any other remaining styles you think are needed.

Task 2: Tailwind CSS Directives

It turns out that not only does Tailwind provide a library of utility classes, but also other helpful tools to simplify more burdensome tasks and plug certain holes in the utility class system.

- You may have noticed that applying utility classes directly to HTML elements results in a lot of repetition that can be labor intensive and lead to errors. Figure out how to use Tailwind's **@apply** directive within the **<style>** tag to mitigate this problem for **<h2>** headings: https://tailwindcss.com/docs/functions-and-directives#apply-directive
    - This will mess up the syntax highlighting in VS Code, but you can fix it by installing the **Tailwind CSS IntelliSense** extension from **Tailwind Labs.**
- We designers are creative people, and only having a limited set of style options to choose from just doesn't sit right. Luckily, we can create our own style **theme** by overriding Tailwind's default variables, or even creating new variables that are automatically mapped to new, customized utility classes. In the **<style>** tag, use the **@theme** directive—explained here: https://tailwindcss.com/docs/functions-and-directives#theme-directive —to do the following:
    - Override one of the colors you used in your design, giving it a slightly different value. (I overrode the **Indigo 800** color.)
    - Create two new variables for font families. I chose **Nunito** and **Outfit** as my families, but you should choose different ones from **Google Fonts**. The names of the variables should be the following:
        - **--font-paragraph**: as the name suggests, this should represent a font family for paragraph text and other non-heading text on the page. Tailwind will automatically generate a new utility class **font-paragraph**; add this class to your HTML in the necessary places.
        - **--font-headings:** same thing, but for headings. Once again, choose a font you like and apply this to your HTML markup.

Task 3: DaisyUI

As mentioned earlier, DaisyUI provides pre-built and pre-styled HTML components that you can configure as use as you like; these components are designed to be used alongside Tailwind CSS. For this part of the assignment, use the screenshots in **with-daisyui** folder as your guide.

Figure out how to do the basic CDN installation here: https://daisyui.com/docs/cdn/

As with Tailwind CSS, DaisyUI has extensive documentation with a list of available components in the menu on the left. Use this documentation as you work through this task.

Implement the following components in your UI:

- An image **carousel:**
    - Use the **Lorem Picsum** service to randomly generate **five** photos. Image URLs have this format: https://picsum.photos/680/280 where the numbers represent the **width** and **height.** The randomness is seeded by these numbers, so choose slightly different widths for each image that are all around 700.
    - The images should have a colored border/background, using your customized Tailwind CSS color.
    - The carousel motion should stop in the **center** of each image, except the two images at the start and end.
    - Buttons at the bottom should be available to navigate, in addition to the usual swipe/scroll gestures.
- **Star ratings** for each TV show:
    - The ratings should be editable but have default values on load.
    - The design with smaller stars should be used.
    - Your customized Tailwind color should be used for the stars.

Grading

- **Task 1:** [3 marks]
    - 3/3: all design features implemented with utility classes; unique colors and font families chosen;
    - 2/3: some features not implemented; utility classes not used for all styles;
    - 1/3: significant errors and ommissions, but some progress;
    - 0/3: little progress towards the solution.
- **Task 2:**
    - [1 mark] @apply directive for h2 heading
    - [1 mark] @theme for an alternate color and two additional Google font families; font families applied to the page with new utility classes
- **Task 3**
    - [1 mark] carousel with config as specified
    - [1 mark] star ratings with specified config

Total: 7 marks

As usual, up to -25% could be deducted for improper hand in, poor organization and formatting, etc. Ask me if in doubt.

Hand In

Rename your project folder to **a7-firstname-lastname,** delete unnecessary files (like these instructions), zip the folder, and hand in to Brightspace.