Advanced CSS

Couch to Coder | BRIGHT NETWORK

Learning Objectives

- Have a better grasp on CSS fundamentals
- Be able to use CSS selectors
- Understand how the box model works
- Have a basic understanding of flexbox
- Assign IDs and classes

Intro

In our first advanced session, we will look into enhancing our website with CSS. Even though we covered the basics, there is much more to know.

This session will focus on giving you tools more than just following instructions. The aim is that you should be able to play around with the toys given to you to customise your website.

CSS3 is the currently used version of CSS, which stands for Cascading Style Sheets. CSS is the code which browsers use to alter the cosmetics and the layout of content rendered on a web page.

Like HTML, CSS is used in almost every web page you'll have ever visited. In this lesson you will learn the commonly used ways to select HTML elements to style in CSS, as well as some of the commonly used CSS properties used when changing the cosmetics of a web page.

We will use the same front end page we used for our JS front end/full stack session, but the principles will apply to any project.

Logo

Let's start with our logo. We can use custom fonts in 2 different ways - either we add the font to our project folder, or if the font is free and available online, we can add the font's link to our head in the `index.html` - the easiest resource for this is Google Fonts.

In our head tag, let's add the following:

```
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <link rel="stylesheet" href="style.css">
  <link rel="preconnect" href="https://fonts.googleapis.com"> <!-- NEW -->
  <link rel="preconnect" href="https://fonts.gstatic.com" crossorigin> <!-- NEW --
>
  <link href="https://fonts.googleapis.com/css2?family=Pacifico&display=swap"
rel="stylesheet"> <!-- NEW -->
  <script src="app.js" defer></script>
  <title>InstantGramen</title>
</head>
```

Now we can apply this font using the font's name - pacifico!

In our `style.css`:

```
h1 {
  font-family: "Pacifico";
}
```

We're treading dangerous waters here with all the similarities to a well known image sharing site!

Next, we should look into our footer - as of now, the links are looking a bit weird, since they are still, well, links. My recommendation is to change the styling of these anchor tags, to remove decorations from them, and make them more like clickable buttons.

The first step is to remove the text decoration added to links by default. But if we do that, all the links will change in the entire page, which is not what I want - most anchor tags should remain the same.

Enter the descendent selector. As we discussed, the HTML tags are kind of built like a tree, with each node on the tree having possibly multiple children, and each child having a single parent. We can tell that we would like the CSS to apply ONLY to a specific tag's children (or even more distant descendents).

```
header a {
  text-decoration: none;
  color: #F8F8F8;
}
```

This will apply to every `<a>` tag inside the `<header>` tag, but anything outside of that remains the same!

Lovely - the next step is to make the clickable area larger. For this, we need to talk about the box model.

The box model

The CSS Box Model is a fundamental concept that defines how elements are structured and spaced on a webpage. Every element is treated as a rectangular box, which consists of four main components: the content, padding, border, and margin. The **content** is the innermost part, where the actual text or images reside. Surrounding the content is the **padding**, which adds space between the content and the border. The **border** outlines the element, and the **margin** creates space between the element and other elements on the page.

Understanding the box model is crucial for precise layout control in CSS. By adjusting the padding, border, and margin, you can control the spacing and positioning of elements. For example, increasing the padding will create more space inside the element, while increasing the margin will push the element further away from others. The `box-sizing` property can also be used to control how the total width and height of an element are calculated, ensuring consistency across different designs.

It's important to note that the behavior of the box model can vary depending on whether an element is **inline** or **block**. Block elements (e.g., `<div>`, `<p>`, `<h1>`) naturally take up the full width of their container and start on a new line, allowing their box model properties (margin, padding, border) to affect their spacing and layout fully. Inline elements (e.g., `<span>`, `<a>`, `<strong>`), on the other hand, only take up as much width as their content and do not start on a new line. While inline elements can have padding and borders, their vertical margins do not affect the layout, and they cannot have a fixed width or height.

So to increase the available area to click on for an `<a>` tag, let's add a 12px padding to it:

```css
header a {
  text-decoration: none;
  color: #F8F8F8;
  padding: 12px;
}
```

You can add values individually to the box elements, in order: top - right - bottom - left. If you only use one value, every side gets the same.

THere are also pseudo-classes in css selectors, like the `:hover`. This enables us to target elements in specific states, like hovered over, selected, or clicked.

```css
header a:hover{
  text-decoration: underline;
}
```

Fantastic!

For our next step, we need to think about the layout of our code. You probably heard the jokes about the hardest thing in CSS being centering stuff - luckily, that's not really a concern anymore, since we have `flexbox`!

Flexbox is a bit of a beast, could definitely use its own lesson, but we will keep it short. If you're interested, remember - studybuddy, or have a read here: Flexbox

But in short, this is how it works: It enables us to control tags *inside another tag*, and we can either center them, or move them around in one dimension - meaning either vertically or horizontally.

So if we want to put the logo and the links on the same line, we can do so like this:

```
header {
/* same as before */
  display: flex;
}
```

Well, this certainly looks like... something...

How can we fix this?

We can add a couple of other modifiers to make sure everything is on the same line, and spaced nicely:

```
header {
/* same as before */
  display: flex;
  justify-content: space-around;
}
```

Better!

To center elements vertically, we can also use flexbox. Simply make the `<a>` tags also a display: flex; type, and then we can use the `align-items` property:

```
header a {
  text-decoration: none;
  color: #F8F8F8;
  padding: 12px;
  display: flex;
  align-items: center;
}
```

Awesome!

> (OPTIONAL) Note: You can use `div` tags to group elements together you'd like to move like one. For example, you can group all the `a` tags together to move them in a group closer to each other, while still keeping them next to the logo. You need to change the html for this like so:

```
  <header>
    <h1>InstantGrameN</h1>
    <div>
      <a href="">Recipes</a>
      <a href="">About us</a>
      <a href="">Contact</a>
    </div>
  </header>
```

> And then in your css file:

```
header div {
  display: flex;
}
```

There is plenty more customization when it comes to flexbox - take a look if you have the time!

Font size can also be changed. We could use the following types: `vw`, `vh`, `px`, and a couple more.

`vw` and `vh` are useful (viewport width and viewport height respectively) because their size depends on the viewport, i.e. the size of the screen, therefore it could look good on both a laptop screen and a mobile without changing much!

```
header a {
  text-decoration: none;
  color: #F8F8F8;
  padding: 12px;
  display: flex;
  align-items: center;
  font-size: 1vh;
}
```

Let's apply what we've learned so far to the new recipe form - and while we're at it, let's look at class attributes!

The most important thing is to try and change the layout of the form by making changes in the HTML. Since we want each label and input field to be laid out nicely, we should wrap these in a div tag to act as a container - without it we can't apply flexbox well!

Add the following divs to the form (feel free to copy/paste everything if you'd like!)

```
<form>
    <p>Add new recipe:</p>
    <div class="form-row">
      <label for="name">Name: </label>
      <input type="text" name="name" id="name">
    </div>
    <br>
    <div class="form-row">
      <label for="cuisine">Cuisine: </label>
      <input type="text" name="cuisine" id="cuisine">
    </div>
    <br>
    <div class="form-row">
      <label for="time">Time: </label>
      <input type="text" name="time" id="time">
    </div>
    <br>
    <div class="form-row">
      <label for="ingredients">Ingredients (put new ingredients on new line):
</label>
      <textarea name="ingredients" id="ingredients"></textarea>
    </div>
    <br>
    <div class="form-row">
      <label for="steps">Steps: (each step on a new line): </label>
      <textarea name="steps" id="steps"></textarea>
    </div>
    <br>
    <input type="submit" value="Add new recipe">
</form>
```

We are adding classes to make sure only the divs with the `form-row` class will be turned into flexboxes.

Classes can be targeted by using the `.` notation:

```
.form-row {
  display: flex;
  justify-content: space-around;
}
```

Well, this looks atrocious. Let's change some things around - it would be nice if the form only took up a third of the screen at most!

```
form {
  text-align: center;
  width: 33%;
  margin: 48px;
}
```

This has the downside of looking weird at first on mobile devices, but with some media queries, we can make this work:

```
@media screen and (max-width: 600px) {
  form {
    width: 80%
  }
}
```

Media queries are used to apply responsive design - since most of the traffic online is coming from mobile devices, it's worth investing time in learning how to do it well!

Further reading

There is an overwhelming amount of content when it comes to CSS - here are some resources if you'd like to read about them more:

- Specificity:
  - MDN Specificity
  - A fun introduction (for the comic lovers): Batificity
- Flexbox:
  - Guide on the above link on CSS tricks
  - An extremely fun game: FlexBoxFroggy
- Grid
  - A different layout system, an alternative to flexbox: CSS Tricks - GRID
  - Another game: Grid Garden
- Animation
  - Learn basic animations using CSS: CSS Tricks - Animation
- CodePen
  - A great resource for inspiration - look around for HTML/CSS/JavaScript snippets to learn from the best
  - A personal favourite of mine: Girl with the CSS earrings
    - Note: This is NOT an image. It's entirely done with HTML and CSS!