{ Syntax and Selectors. }

Objectives:

By the end of this chapter, you should be able to:

- Define what CSS is and demonstrate three different ways to include it on your page
- Use selectors to target HTML
- Explain how specificity plays a large factor when using selectors

Getting started with CSS

When we first introduced HTML, we said that it was primarily responsible for holding the content of the page, while the styling was handled by CSS. So what is CSS, anyway, and how can we use it to style our web pages?

CSS stands for **C**ascading **S**tyle **S**heets, and it's a way to describe style rules that we'd like to set for our HTML elements. Using CSS we can alter things like colors, fonts, margins, text alignment, and much more! Here's some further reading on the background and history of CSS, if you're interested.

There are three ways to include CSS in our web pages:

1. Inline styling. This is done by adding a `style` attribute to an HTML tag. We will see shortly that this can have some unintended consequences.
2. `<style></style>`. This is done by using a `style` tag and placing CSS inside of it.
3. External stylesheets. **This is the most common way and almost always the best practice**

Since we will be using external stylesheets, we need to figure out how to link our CSS to our HTML. The answer is with a `link` tag! If we have a file called `style.css` in the same folder as our HTML file, we can link them with the following tag, which you should place inside of the `head` element:

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

Syntax, Selectors

Each CSS rule consists of a selector (how we find the element/elements) to target and a pair of curly braces, a.k.a. `{}`. Inside of `{}` we specify properties and values; we separate the two with a `:`, and after the value we add a `;`. To add multiple selectors we separate each one by a `,`. This looks something like this:

```
body {
    background-color: green;
}
```

In this rule, we are targeting the `<body></body>` element and applying a background color to it. While this works, it might be a bit too general (maybe we only want a background color of green on certain parts of the page, not the whole body). In order to be more specific, we can use attributes like `class` and `id` to target our elements. We'll talk more about these attributes in the next chapter; for now, you can think of them as a tool that makes it easier to find the element you're looking for on the page. The biggest difference is that `id`s must be unique: two elements should never have the same `id`. But you can have multiple elements share the same `class`.

Let's take a look at a sample HTML file. We will call this file `index.html`. Notice the `<link/>` tag we are using to link to an external stylesheet,which we will call `style.css`:

```
<!DOCTYPE html>
<html>
<head>
    <title>Learn CSS</title>
    <link rel="stylesheet" href="style.css">
</head>
<body class="main_styles" id="container">
    <header class="main_heading">
        <h1>Welcome to our website</h1>
        <h4>We're so happy to have you</h4>
    </header>
    <section class="main">
        <ul> Here are some reasons you should keep reading....
            <li>You're learning things!</li>
            <li>CSS is fun!</li>
            <li>Well.....not always, but it is essential to know!</li>
        </ul>
    </section>
    <footer class="main_footer">
        <p>
            Thanks for checking out this page!
            <a href="#">Here is a link that goes nowhere! It's for showing
you stuff with CSS......we promise</a>
        </p>
    </footer>
</body>
</html>
```

We can see from this HTML that we have added some `class` and `id` attributes. Right now they do not alter anything about the page, but we can use them as selectors with CSS. To target a class with CSS, add a `.` before the name of the class. For an id, add a `#` before the name of the id.

This is what some CSS might look like for our page (imagine our filename is `style.css` and is in the same directory as our `index.html`:

```
#container {
    font-size: 16px;
}
.main_heading {
    background-color: grey;
}
.main {
    background-color: blue;
}
.main_footer {
    background-color: green;
}
.main_header,
.main_footer {
  color: red;
}
```

After typing this code into your `style.css` file, refresh the page. If you've set things up properly, you should see some big changes in the way your page looks!

Other useful selectors

Descendant Selector

The descendant selector matches all elements that are descendants of a specified element. The selector below will find all of the `p` tags inside of a `footer` tag (read the selectors from right to left).

```
footer p {
    background:red;
}
```

Adjacent Selector

The adjacent selector finds all elements that are directly adjacent to some other specified element. The selector below will find all the `h4` tags that are directly adjacent to an `h1` tag.

```
h1+h4 {
    background:blue;
}
```

Direct child selector

This is commonly confused with the descendant selector. The direct child selector matches all elements that are *direct children* - not just ancestors - of a specified element. The selector below will find all `li`s which are children of a `ul`.

```
ul > li {
    background:blue;
}
```

Attribute Selector

The attribute selector finds elements based on the value of some attribute. The selector below will find all `a` tags whose `href` attribute is set to "#".

```
a[href="#"] {
    background:blue;
}
```

First / Last Child Selector

The `first-child` and `last-child` selectors find all elements which are the first (or last, respectively) children of their parents. The selector below will find all the li's which are the first children of their parents.

```
li:first-child {
    background:blue;
}
```

n-th child selector

This is a sort of generalization of the selectors above. `nth-child` will find all elements which are the nth children of their parents, for some specified value of n. The selector below will find all the li's which are the second children of their parents.

```
li:nth-child(2) {
    background: teal;
}
```

These aren't the only selectors, but they are some of the more commonly used ones. For even more examples, check out MDN.

One important thing to know about CSS is the **C**, or **cascading** nature of CSS. This means that your code is read top to bottom, so if you have the same level of specificity, the rule closest to the bottom wins. For example, consider the following stylesheet:

```
p {
    font-size: 16px;
}

p {
    font-size: 100px;
}
```

These two rules conflict, but because they have the same degree of specificity, cascading will kick in, and the latter rule will overwrite the former. The text on this page will be huge!

Not sure what we mean by *specificity*? Perfect. Read all about it in the next chapter!