**Intro to CSS**

The basic structure of every web page, HTML, is very plain on its own. The beautiful websites that you see across the internet are styled with a variety of tools, including CSS.

*CSS*, or Cascading Style Sheets, is a language that web developers use to *style* the HTML content on a web page. If you're interested in modifying colors, font types, font sizes, shadows, images, element positioning, and more, CSS is the tool for the job!

In this lesson, you'll learn how to select which HTML elements you wish to style and set up your CSS file structure.

**Inline Styles**

Although CSS is a different language than HTML, it's possible to write CSS code directly within HTML code using *inline styles*.

To style an HTML element, you can add the style attribute directly to the opening tag. After you add the attribute, you can set it equal to the CSS style(s) you'd like applied to that element.

<p style="color: red;">I'm learning to code!</p>

The code in the example above demonstrates how to use inline styling. The paragraph element has a style attribute within its opening tag. Next, the style attribute is set equal to color: red;, which will set the color of the paragraph text to red within the browser.

You might be wondering about the syntax of the following snippet of code: color: red;. At the moment, the details of the syntax are not important; you'll learn more about CSS syntax in other exercises. For now, it's important to know that inline styles are a quick way of directly styling an HTML element.

If you'd like to add *more* than one style with inline styles, simply keep adding to the styleattribute. Make sure to end the styles with a semicolon (;).

<p style="color: red; font-size: 20px;">I'm learning to code!</p>

# The <style> Tag

Inline styles are a fast way of styling HTML, but they also have limitations. If you wanted to style, for example, multiple <h1> elements, you would have to add inline styling to each element manually. In addition, you would also have to maintain the HTML code when additional <h1>elements are added.

Fortunately, HTML allows you to write CSS code in its own dedicated section with the <style> element. CSS can be written between opening and closing <style> tags. To use the <style> element, it must be placed inside of the <head> element.

<head> <style> </style> </head>

After adding a <style> tag in the head section, you can begin writing CSS code.

<head> <style> p { color: red; font-size: 20px; } </style> </head>

The CSS code in the example above changes the color of all paragraph text to red and also changes the size of the text to 20 pixels. Note how the syntax of the CSS code matches (for the most part) the syntax you used for inline styling. The main difference is that you can specify which elements to apply the styling to.

Again, the details of the CSS syntax in the example above aren't important at the moment. You will learn more about the details of CSS syntax in later lessons.

# The .css file

Developers avoid mixing code by storing HTML and CSS code in separate files (HTML files contain only HTML code, and CSS files contain only CSS code).

You can create a CSS file by using the **.css** file name extension, like so: **style.css**

With a CSS file, you can write all the CSS code needed to style a page without sacrificing the readability and maintainability of your HTML file.

# Linking the CSS File

Perfect! We successfully separated structure (HTML) from styling (CSS), but the web page still looks bland. Why?

When HTML and CSS code are in separate files, the files must be linked. Otherwise, the HTML file won't be able to locate the CSS code, and the styling will not be applied.

You can use the <link> element to link HTML and CSS files together. The <link> element must be placed within the head of the HTML file. It is a self-closing tag and requires the following three attributes:

1. href — like the anchor element, the value of this attribute must be the address, or path, to the CSS file.
2. type — this attribute describes the type of document that you are linking to (in this case, a CSS file). The value of this attribute should be set to text/css.
3. rel — this attribute describes the relationship between the HTML file and the CSS file. Because you are linking to a stylesheet, the value should be set to stylesheet.

When linking an HTML file and a CSS file together, the <link> element will look like the following:

<link href="https://www.codecademy.com/stylesheets/style.css" type="text/css" rel="stylesheet">

Note that in the example above the path to the stylesheet is a URL:

https://www.codecademy.com/stylesheets/style.css

Specifying the path to the stylesheet using a URL is one way of linking a stylesheet.

If the CSS file is stored in the same [directory](https://en.wikipedia.org/wiki/Directory_(computing)) as your HTML file, then you can specify a [relative path](https://en.wikipedia.org/wiki/Path_(computing)#Absolute_and_relative_paths) instead of a URL, like so:

<link href="./style.css" type="text/css" rel="stylesheet">

Using a relative path is very common way of linking a stylesheet.

# Tag Name

CSS can select HTML elements by using an element's tag name. A tag name is the word (or character) between HTML angle brackets.

For example, in HTML, the tag for a paragraph element is <p>. The CSS syntax for selecting <p> elements is:

p { }

In the example above, all paragraph elements will be selected using a CSS selector. The selector in the example above is p. Note that the CSS selector matches the HTML tag for that element, but without the angle brackets.

In addition, two curly braces follow immediately after the selector (an opening and closing brace, respectively). Any CSS properties will go inside of the curly braces to style the selected elements.

# Class Name

CSS is not limited to selecting elements by tag name. HTML elements can have more than just a tag name; they can also have attributes. One common attribute is the class attribute. It's also possible to select an element by its classattribute.

For example, consider the following HTML:

<p class="brand">Sole Shoe Company</p>

The paragraph element in the example above has a class attribute within the <p> tag. The class attribute is set to "brand". To select this element using CSS, we could use the following CSS selector:

.brand { }

To select an HTML element by its class using CSS, a period (.) must be prepended to the class's name. In the example above case, the class is brand, so the CSS selector for it is .brand.

# Multiple Classes

We can use CSS to select an HTML element's class attribute by name.

So far, we've selected elements using only one class name per element. If every HTML element had a single class, all the style information for each element would require a new class.

Luckily, it's possible to add more than one class name to an HTML element's class attribute.

For instance, perhaps there's a heading element that needs to be green and bold. You could write two CSS rules like so:

.green { color: green; } .bold { font-weight: bold; }

Then, you could include both of these classes on one HTML element like this:

<h1 class="green bold"> ... </h1>

We can add multiple classes to an HTML element's class attribute by separating them with a space. This enables us to mix and match CSS classes to create many unique styles without writing a custom class for every style combination needed.

# ID Name

If an HTML element needs to be styled uniquely (no matter what classes are applied to the element), we can add an ID to the element. To add an ID to an element, the element needs an id attribute:

<h1 id="large-title"> ... </h1>

Then, CSS can select HTML elements by their id attribute. To select an id element, CSS prepends the id name with a hashtag (#). For instance, if we wanted to select the HTML element in the example above, it would look like this:

#large-title { }

The id name is large-title, therefore the CSS selector for it is #large-title.

# Classes and IDs

CSS can select HTML elements by their tag, class, and ID. CSS classes and IDs have different purposes, which can affect which one you use to style HTML elements.

CSS classes are meant to be reused over many elements. By writing CSS classes, you can style elements in a variety of ways by mixing classes on HTML elements.

For instance, imagine a page with two headlines. One headline needs to be bold and blue, and the other needs to be bold and green. Instead of writing separate CSS rules for each headline that repeat each other's code, it's better to write a .bold CSS rule, a .green CSS rule, and a .blue CSS rule. Then you can give one headline the bold green classes, and the other the bold blue classes.

While classes are meant to be used many times, an ID is meant to style only one element. As we'll learn in the next exercise, IDs override the styles of tags and classes. Since IDs override class and tag styles, they should be used sparingly and only on elements that need to always appear the same.

# Specificity

Specificity is the order by which the browser decides which CSS styles will be displayed. A best practice in CSS is to style elements while using the lowest degree of specificity, so that if an element needs a new style, it is easy to override.

IDs are the most specific selector in CSS, followed by classes, and finally, tags. For example, consider the following HTML and CSS:

<h1 class="headline">Breaking News</h1>

h1 { color: red; } .headline { color: firebrick; }

In the example code above, the color of the heading would be set to firebrick, as the class selector is more specific than the tag selector. If an ID attribute (and selector) were added to the code above, the styles within the ID selector's body would override all other styles for the heading. The only way to override an ID is to add another ID with additional styling.

Over time, as files grow with code, many elements may have IDs, which can make CSS difficult to edit, since a new, more specific style must be created to change the style of an element.

To make styles easy to edit, it's best to style with a tag selector, if possible. If not, add a class selector. If that is not specific enough, then consider using an ID selector.

# Chaining Selectors

When writing CSS rules, it's possible to require an HTML element to have two or more CSS selectors at the same time.

This is done by combining multiple selectors, which we will refer to as chaining. For instance, if there was a .special class for h1 elements, the CSS would look like:

h1.special { }

The code above would select only the h1elements that have a class of special. If a pelement also had a class of special, the rule in the example would not style the paragraph.

# Nested Elements

In addition to chaining selectors to select elements, CSS also supports selecting elements that are nested within other HTML elements. For instance, consider the following HTML:

<ul class='main-list'> <li> ... </li> <li> ... </li> <li> ... </li> </ul>

The nested <li> elements are selected with the following CSS:

.main-list li { }

In the example above, .main-list selects the .main-list element (the unordered list element). The nested <li> are selected by adding li to the selector, separated by a space, resulting in .main-list li as the final selector (note the space in the selector).

Selecting elements in this way can make our selectors even more specific by making sure they appear in the context we expect.

# Chaining and Specificity

In the last exercise, instead of selecting all h5elements, you selected only the h5 elements nested inside the .description elements. This CSS selector was more specific than writing only h5. Adding more than one tag, class, or ID to a CSS selector increases the specificity of the CSS selector.

For instance, consider the following CSS:

p { color: blue; } .main p { color: red; }

Both of these CSS rules define what a pelement should look like. Since .main p has a class and a p tag as its selector, only the pelements inside the .main element will appear red. This occurs despite there being another more general rule that states p elements should be blue.

# Important

There is one thing that is even more specific than IDs: !important. !important can be applied to specific attributes instead of full rules. It will override any style no matter how specific it is. As a result, it should almost never be used. Once !important is used, it is very hard to override.

The syntax of !important in CSS looks like this:

p { color: blue !important; } .main p { color: red; }

Since !important is used on the p selector’s color attribute, all p elements will appear blue, even though there is a more specific .main p selector that sets the color attribute to red.

The !important flag is only useful when an element appears the same way 100% of the time. Since it's almost impossible to guarantee that this will be true throughout a project and over time, it's best to avoid !importantaltogether. If you ever see !important used (or are ever tempted to use it yourself) we strongly recommend reorganizing your CSS. Making your CSS more flexible will typically fix the immediate problem and make your code more maintainable in the long run.

# Multiple Selectors

In order to make CSS more concise, it's possible to add CSS styles to multiple CSS selectors all at once. This prevents writing repetitive code.

For instance, the following code has repetitive style attributes:

h1 { font-family: Georgia; } .menu { font-family: Georgia; }

Instead of writing font-family: Georgia twice for two selectors, we can separate the selectors by a comma to apply the same style to both, like this:

h1, .menu { font-family: Georgia; }

By separating the CSS selectors with a comma, both the h1 and the .menu elements will receive the font-family: Georgia styling.

# Review CSS Selectors

Throughout this lesson, you learned how to select HTML elements with CSS and apply styles to them. Let's review what you learned:

* CSS can change the look of HTML elements. In order to do this, CSS must select HTML elements, then apply styles to them.
* CSS can select HTML elements by tag, class, or ID.
* Multiple CSS classes can be applied to one HTML element.
* Classes can be reusable, while IDs can only be used once.
* IDs are more specific than classes, and classes are more specific than tags. That means IDs will override any styles from a class, and classes will override any styles from a tag selector.
* Multiple selectors can be chained together to select an element. This raises the specificity, but can be necessary.
* Nested elements can be selected by separating selectors with a space.
* The !important flag will override any style, however it should almost never be used, as it is extremely difficult to override.
* Multiple unrelated selectors can receive the same styles by separating the selector names with commas.

Great work this lesson. With this knowledge, you'll be able to use CSS to change the look and feel of websites to make them look great.