December 8, 2021

Styling Websites with CSS

Understanding Inline and Embedded CSS

We're going to go ahead and dive right into our project, so go ahead and open our project files in the text editor of our choice, or Brackets, if you're following along with me. And if you look over here in the file pane, I have the 02 folder open, and we have a variety of HTML files for our website project. We also have an images folder with some images files for the site, and I have the index.html file, or the home page, open in the browser window. So you can open it up and the browser of your choice, if you're using a different text editor. Or if you're following along in Brackets, in the top right, you can click the lightning bolt icon, and it'll open it using what's called Live Preview, which will show changes live without having to reload the page. And what I want to take a look at first is different ways of adding CSS. There are actually three different methods for adding CSS to a web page or a website. You can add them as embedded styles, you can add them inline to an element, or you can add them as an external stylesheet as well. If you add them as embedded styles, you do so through adding a style element to the head of an HTML file, or you can add them inline by applying them to a specific element using a style attribute, or you can create an external CSS stylesheet and attach it to each HTML page you want those styles to be pulled into. So let's take a look at our project and see a specific example of each way or method of adding styles. If you come down to line 11, you can see we have a style, embedded stylesheet in the head of our index.html file. And i inside of this style element, there is a set of CSS styles. Now, if this syntax looks unfamiliar to you, that's completely fine. We'll be learning exactly how to write CSS from scratch on our own later on in this course. Now I'm going to scroll down to line 87, and on line 87 we have a div element, and inside of it we have a style attribute. And here we have some more CSS styles applied within this style attribute. And what this does is, it only applies it to the element with that attribute. So let's say we wanted to style a set of headings in the same way, then we would have to add this style attribute to each heading element where we wanted the styles to be applied. Our third method of adding CSS, if you go over here and our file pane, you can see there's a styles.css file, which is an external stylesheet. And you can see the format looks pretty similar to that embedded style element, except for the difference is it's stored in a file of its own. In summary, inline styles, are styles that are added using a style attribute to a single HTML element. When you add styles in this way, you're mixing presentation was structure, which isn't considered a best practice. It also makes other best practices like scalability, maintainability, and reusability hard to achieve. Updates have to be made to every single element affected, and it's really easy for changes to get out of sync. Embedded styles are styles added to a style element in the head of a single HTML file. Similarly to inline styles, if you want to apply styles to a whole website, you end up with a lot of repetitive code because it has to be repeated across each HTML file. When you make updates, you have to update the same style across multiple files, which is not only time consuming, but it's really easy to miss a file or end up with small differences over time, which makes it difficult to maintain consistent styles. External stylesheets, on the other hand, are when you store your CSS in its own file. CSS gets applied by using a link element to link the stylesheet to a specific HTML file. Unlike inline and embedded stylesheets, you can make all of your CSS changes in a single location. And for this reason, it makes it easy to scale your CSS, and it's also easy to maintain it, since you can make changes in one single location. And all of your CSS is reusable as well, because you can use that link to apply the same stylesheet to multiple files, which is what we're going to learn how to do in our next clip, when we learn how to create and link a stylesheet to an HTML file.

Creating and Linking a Stylesheet

Let's take a look at how we can create our very own CSS stylesheet from scratch. And we can do things like move our embedded styles over to our brand‑new stylesheet. To start, we'll want to create a blank file, but first I'm going to create a folder to store all of our CSS files, or in this case, we'll just have one. But when you're working on a website, it's best practice to store all your similar file types together so you can keep things nice and organized. So over in our file pane, I'm going to right‑click and choose to create a new folder, and we can call it something like CSS, or I'm going to name our folder styles. Inside of this new styles folder, I'm going to right‑click to create our new file, and I'll create a blank file. And you can name it whatever you want, but one common convention is to name it something like styles.css or main.css to signify that's where all the main or bulk styles of a website are being stored. So I'm going to name this one main, and then we'll give it the extension .css, which is what's going to tell the browser that this is a CSS file. So I'm going to press Enter, and we have our brand‑new blank CSS file. I'm going to switch back over to our index.html file, and I'm going to go ahead and cut and paste all of our styles inside of this embedded style element, so starting on line 12, going to line 72. And then I just hit Ctrl or Command+X to cut those out, and I'm going to delete our empty style element, since it no longer has anything inside. And then I'm going to hit Command+S to save our page, since we made a change. Switch back over to our main.css file and then use Command or Ctrl+B just to paste all of our CSS styles inside of this file. Now if we go look at our home page, I'm going to refresh this in the browser window, since we made that change, you can see our site now looks much different because we moved a bulk of those styles over to that main.css file, except for those styles which are still displaying as inline the element. So to get those back, what we want to do is, we want to link the CSS file to our HTML page. That way, the browser knows to where to look for our CSS file. And we want to do that inside of the head section of the file. So down here on line 10 you can see there's already a link element, which we'll use to link the file. In this case, it's linking to a favicon, or little icon that shows to the left of the title name in the browser window. So I'm going to press Enter, and then I am going to open our link tag, which does not need a closing tag. And then inside of our link tag, we need three different attributes. The first one we need to add is the rel attribute, which stands for relationship. And our relationship is that it's a stylesheet, so go ahead and type that out. And we also need to tell it what type of stylesheet it is, so we use the type attribute for that. And it's a text file. You can see it's already suggesting the type in brackets. That's exactly what we need. It's a text file, and it's also a css file. The last attribute we need to do is to tell it what link to look for, so just like with an anchor tag, we use the href attribute, and we'll tell it where our stylesheet is located. So since we created that styles folder, we want to tell it to look in the styles folder, so I'm going to tell it to look in that styles folder. And then our file is named main.css, and I'm going to hit Ctrl or Command S to save. And if we refresh this, you can see our styles are pulled back in to the page. In summary, to link a stylesheet to an HTML file, we need to use that link element with three different attributes. The first is the rel, attribute, which explains the relationship, which in our case is a stylesheet. We also need the type, which in our case is a text file containing CSS, and we also need to use the href attribute to describe the location of our CSS file.

Writing Rules

Now that we have our style sheet created and linked to our index file, we're ready to write our first CSS style. So I'm going to scroll down past our existing styles and press Enter twice to leave a space between our existing styles just to give some organization there, be able to scan it a little bit more easily, so not everything is scrunched together. And the first thing we want to do is write what's called a selector. And just like its name sounds, a selector is telling the browser what we want to select. There are multiple types of selectors, which we'll learn throughout this course, but what we're going to use is called a type selector. And a type selector is selecting an HTML element by its name. So, for example, if we wanted to select all paragraphs, we could use the tag for a paragraph, which is P. So I'm going to add a P, and then we enclose the rest of our style in a set of curly brackets. So I'm going to go ahead and add those, and press Enter, and it'll add the closing curly bracket on a new line. This is a common syntax format for what's called a CSS rule, and we'll add a property inside of our CSS rule. Now a property describes what about our selector or element is going to be changed. So let's say we want to change the color of a paragraph. There's a property for that, and it is called the color property. And if you look over in our live preview, you can see that all of our paragraph elements are being highlighted above and below, so we can see what's going to be affected when we add this style. So I'm going to go ahead and type out that color property, and then we add a colon and a value, and the value will tell the browser what we want that color to change to. So let's say we want to change it to red just so we can see how this is going to affect our page. I can use a key word there just to type out the color name red, and then a semicolon, which ends our property and our value. And over on the right in the browser window, you can see that now all of our paragraph elements are now colored red. So all together, our selector, property, and value are known as what's called a CSS rule, which consists of the selector, which is telling the browser what you want to style or select, and inside of our selector, we have a property, which describes what is going to be changed, followed by a value describing what you want to change that property to. Together, our property and value are known as a declaration because you're declaring what you want the property and value to be. And this is what CSS consists of is a set of CSS rules defining how you want to change the default presentation of HTML. In summary, we can add CSS in three different ways. We can add it inline using a style attribute, we can add it using a style element in the head of an HTML file referred to as an embedded style sheet, or we can add it in the form of an external style sheet using the link attribute to link an HTML file to a CSS style sheet. External style sheets are generally considered best practice. This is because they have the benefit of maintainability and scalability due to the fact that you can make all of your changes in a single location. It also cuts down on the amount of repetitive code because unlike an inline style attribute where you have to add it to every single HTML element to be affected, or an embedded style sheet that has to be added to every single HTML file, you can make all of your changes and store all of your CSS rules in a single location. CSS style sheets are made up of rules. A rule consists of a property describing what's going to be changed, and a value, or what you are changing about the property. Collectively, a property in a value are referred to as a declaration.

Styling Text

Module Overview and Formatting CSS

In this module, learn how to style text. Throughout this module, learn how to change the style of text, such as making it bold, changing its size, and altering its style, such as italicizing it. We'll learn how to align text to the left, right, or center. We'll change the amount of white space between lines and letters, referred to as leading and tracking. And we'll also learn how to use both web‑safe and custom fonts and as well as exactly what that means. Before we start writing our first textiles, I first want to take a look at our main.css file. As you may have noticed in our last module, it's kind of inconsistent in its formatting. It looks a little bit messy, which makes it hard to scan and read. But there is a tool we can install to a text editor that helps us format it and keep things clean and easy while also saving time so you don't have to do it manually, and that's what's called a beautifier. So in Brackets, to install beautifier, you can come over here to the brick icon on the right, and then we can just search for a beautify tool. And then I'm going to choose this second option. This one's PHP specific, so I'm going to choose one that has JavaScript, HTML, and CSS files. So that's good for our purposes. And if you're using a different text editor, the process should be pretty much the same. You'll search through the extensions and just find a beautifier that'll help you format your code. Now that we have this installed, we can come up to Edit, and at the very bottom we have two new options. We can now beautify, and we can beautify on save. So I'm going to select Beautify. And if you pay attention to our file, as a click this, you can see it's automatically formatted all of our CSS rules. So it saves a lot of time and keeps things neat and organized. I'm going to go back up to Edit, and I'm going to select Beautify on Save, and what this will do is every time we save our file, it'll automatically clean up and organize our CSS.

Changing the Size, Weight, and Style of Text

Let's learn how we can style text in different ways starting with changing the size of text. So I have our main.css file open and I've scrolled to the bottom and I'm just going to press Enter twice so we have some space between our other CSS rules. And the first thing we want to do is choose our selector or by selecting the element name of whatever element we're going to p styling. So let's say we want to change the size of all of our paragraph text. Well, since that's in a P element, we could use that for our selector, so I'm going to type in P and then add our opening and closing curly braces and you can see over on the right that all of the paragraph elements are being highlighted in the browser window, which is cool because we can see exactly what we're about to change. And the property name for changing text size is the font‑size property and that has a hyphen so it's font‑size, then we want to add our colon, followed by our value. For our value, we'll choose a number for the size and then a unit, and the most common unit for the font‑size property is the pixel, which stands for 1 pixel on a screen, and the default font size for a paragraph in the browser is 16 pixels. So let's say we want to bump it up just a little bit to 18 pixels. I'm going to go ahead and add that in pixels as px for short. Then as I add our semicolon, you can see that the size bumped up just a little bit for our paragraph font size. So let's try again with another element. Let's say we want to change this heading down here for the weekly promotions, increase the size of it just a little bit to make it stand out more. Let's take a look and see what type of element that is wrapped in. So I'm going to scroll down to the bottom and you can see it's in a h3 element so I want to use that for our selector. I'm going to switch back over to our main.css file, press Enter twice again to start on a new line, and then we'll want to use that h3 as our element selector, and it's currently 19 pixels or approximately 19 pixels big, so let's say we want to bump it up maybe 4 sizes, 4 pixels to 22 pixels. So I'm going to type out that font‑size and then our value, so 22 pixels. And if you look over there, it's bumped up our size a couple sizes, and now it stands out a little bit more, since it is a promotion, you want it to stand out as featured content, and it looks pretty good. In addition to changing the font size property, we could also change the weight of text. So let's say you're handed off a design that had all of the heading styled in a normal weight, instead of a bold weight. This might be a use case for this. So let's say, let's try it out on our h3 property. I'm going to press Enter to come onto a new line and that property is the font‑weight property, So it's font‑weight, and for the value, we can use keywords or predefined words in CSS. So normal would be the normal weight or I could use bold as well if you were to make this bold, but it's already bold. So I'm going to type out normal is our value. And if you take a look, you can see that our h3 heading is now a normal weight. I'm going to go ahead and delete that because I want to keep it consistent with the style of the other headings. And what I do want to change is I want to come up here in this sidebar and let's style these links to be bold just like all of the links in our navigation so to match those styles. All of our links in the sidebar are in a UL or unaltered list element, and each one is an individual list item. So we could use the li selector to target these elements. So I'm going to use li and then our opening and closing curly braces, which just a note here, you want to be careful when you're selecting every element because then every li element affected by the style sheet will then be changed, so just a note there. That's the font‑weight property, so I'm going to type out font‑weight once again, and then instead of normal, this time we're going to use our other keyword, which is bold, so B‑O‑L‑D, and our semicolon. And then if you look over here in the browser window, you can see that all of those links are now bold. Another way we can alter the appearance of text is by changing the style of text. So I want to make this caption in the image italicized, instead of being a normal style of text. And that is a figcaption element, so right under here, I'm going to type out that element name, figcaption, and use the font style property to change the style of that text so font‑style and you can see brackets is pulling up our options here. We have three different keywords we can use. We can use italic, normal, or we could also use oblique. So I'm going to type out italic and you can see that it's italicized our text. Now if I were to use the oblique value, you can see it looks pretty similar. It depends on what font styles you have in the font that's being used itself. So if it's not there, the browser will try to synthesize its own version, it'll just try to slate the text for both oblique and italic if there is not a version available so you want to be careful there and make sure it looks good to you. I'm going to change this back to italic and hit Cmd or Ctrl+S to save and format our CSS. So in summary, we can use the font‑size property, which allows us to change the size of text. The most common unit that can use with a value is the pixel unit, or px for short. We can also use the font‑weight property to change the weight of text. We can change it to be normal weight if it's bold, or we can change it to be bold if it's a normal weight. We can also use the font‑style property, which allows us to take a normal text and make it italicized or oblique, or we can take italicized or oblique text and make it to be styled normally as well.

Altering Line-height and Letter-spacing

Another way we can alter the appearance of text is by changing the amount of whitespace both in and around it. So let's take a look at how we can do just exactly that. What I want to take a look at is taking our sidebar navigation and increasing the amount of space between each list item. That way, it makes it a little bit easier to scan. We have a list item selector already on line 71, so I'm going to press Enter and come onto a new line so we can add another property. And the property for changing the amount of leading or whitespace between lines of text is called the line‑height property. So I'm going to go ahead and type that out, it's just line‑height, and we can specify amount of pixels here, so let's take a look at that. I want to increase it to, let's say, 30 pixels of space and add our semicolon. And if you take a look at our list item, it's added that exactly, 30 amount of pixels of space. But one thing we can run into this is if the font size changes, so I'm going to press Enter and come onto a new line so we can take a look at that. If we were to increase the size of our text to 40 pixels, then you can see our lines of text are now jamming together. Because a pixel is a fixed unit, it doesn't flex or change, then the line‑height property won't respond to changes in text. Let's say you have a different font size on the mobile version of your site, or if a user changes the default text size in the browser, then they can run into these types of issues. Fortunately, one thing we can do with the line‑height property is to use a unitless value, which is pretty cool, so let's go ahead and take a look. I'm going to delete that 30 pixels in there. And a unitless value for the line‑height property is calculated in relation to the font size of the element itself. So if our line items are 16 pixels, then it'll add a 16‑pixel line‑height. So I'm going to change us from 30 pixels to 1, and then if I change it from 1 to 2, it'll double it and add 32 pixels. And you can see it's done exactly that, or, excuse me, 80 pixels since our font size is now 80. You can see it now has 80 pixels of line height between each line item, which is pretty cool about using a unitless value. That way it's going to respond to those changes, and you don't run into those issues. I'm going to go ahead and delete that font size that we added. And I'm going to change our 2 to our final value, which a good value, a starting value, you always want to check it with your individual font to make sure it looks good, but is around 1.5 for text readability on the web. So I'm going to change that to 1.5. And if I look over there, that looks pretty good to me. In addition to changing the amount of leading, we can also change the amount of tracking or the space between individual characters as well. So I want to add a little bit of tracking to our headings to make them feel a little bit airier, adding that space in. So below our figcaption style I'm going to press Enter twice, and I want to target all of our headings with this style. And there is a way to do this with a selector. You can create what's called a group selector or add your selectors separated by commas. So I'm going to type and start with our h1 element. And then if I wanted to target our h2 elements as well, I could just add a comma and then add on to that selector, and then our h3 all the way down to our h6. So that way, if you want to add a set of styles to a group of elements, you certainly have the ability to do so. And for our heading elements, we can use the letter‑spacing property to change that amount of tracking. So I'm going to type that out, just letter‑spacing, and then we can use a pixel value. So just so we can see this, I'm going to add a larger value. Let's say we have 5 pixels, And if you look over on the right, you can see there have been added 5 pixels of space between each character in all of our headings. So I'm going to bump this down a little bit, so it's not so drastic, maybe to be around 1 pixel or maybe 2 pixels, which looks pretty good. So to change the amount of whitespace for text, we have two different properties we can use. We can use the line‑height property, which changes the amount of whitespace, or the leading between lines of text, or we can use the letter‑spacing property to change the tracking, or the amount of whitespace between characters.

Aligning and Transforming Text

Sometimes a design or a situation calls for changing the alignment of text, and you can do that using the text‑align property. So let's say we want to change the alignment of the heading in our sidebar. That's in an h4 element so I'm going to press Enter twice. and we'll use h4 as our selector, and then we can use that text‑align property. And then we have a set of keywords we can use, and one of those is center, which will do just that. It'll center the text inside of the width of that heading element. So however large that is, it'll just put it exactly in the center. So you can see it's doing that in our sidebar, and then also down below for our other h4, which just as a side note, we'll be learning how to affect just single elements using special types of selectors later on in the course. In addition to center aligning, we have the option to align it left if it happens to be centered, or right aligned, or we could also change it to right; it will align it to the right of that element. And another thing we can do, I'm going to come up here in the p element is we can justify text. So if I say text‑align and use justify as the value, you can see, it's going to spread it or justify it equally between the lines or the width of the element, which you want to be careful using justify on the web since text and containers tend to be fluid, they stretch to take up the width of their container element. There aren't very many situations where it's very fixed and perfect, such as you find on print. So just be careful that you don't end up with a lot of awkward gaps, which make it hard to read when you're using justify. I'm going to go ahead and delete that, and then I'm going to change our text‑align value down here to be center, and another way we can change text is by transforming it. So let's say we wanted to capitalize the letters in our heading so they're always a consistent case, just like they are in this h1. We can use what's known as the text‑transform property to alter its case. So in all of our group of heading elements, I'm going to add that property, so text‑transform. And we can use the value capitalize, you can see it popping up there as the first option, which will capitalize the first letter of every word automatically. So if I add that, you can see that now, each letter in every single heading in our group selector is now being affected. If you wanted to have it always to be uppercase, you can use that as a value, or you could also change it to lowercase if you always wanted the text to be lowercase as well. So I'm just going to take this back to capitalize and hit Command or Ctrl+S to save our changes.

Using Fonts for the Web

We've learned how to change the style of font in different ways, but let's now learn how to change the font itself. We've switched gears here from working on our home page to the promotions.html file, and I have that open in the browser window. And what I want to look at first is the default browser styling for tech. So I'm going to come in and use an HTML comment to comment out our style sheet, and then refresh, and now we can see those browser default styles. So I'm going to zoom in so we can see this a little bit better, and the browser uses serif as the default go‑to text. If you are unfamiliar serif fonts, they have little bits of stroke or a line at the ends of the characters, and which font actually gets used depends on your system, but it's generally Times or Times New Roman. I'm going to go ahead and scroll up and switch over to our main.css file, but of course, we don't want to forget to comment back in our style sheet. Then I'm going to scroll to the very bottom and also make sure we refresh so that style sheet is applied once again. And I want to take a look at how we can change the font that our headings are using. So I'm going to come into our headings group selector, and press Enter to come on to a new line, and the property we can use to change the font that's being used is called the font‑family property, and that's font hyphen family, and we have multiple options of values to use here. One of those options is to use a keyword or a generic grouping of text. So let's say, if I were to apply a serif font, then it's going to use whatever the default serif font is. In this case, it's using Times New Roman, or if I was to use sans serif, then it's going to use Helvetica or Arial or whatever the default sans serif font is for the system, or we can use a more decorative font. like we could use cursive as an option. We could also use fantasy as well, but generally the most common types of fonts and the most versatile types of fonts are going to be serifs and sans serifs generally, because they have good readability for text on the web. But we aren't just limited to keywords. We can use specific fonts here as well. So let's say I want to use a font like Proxima Nova. I'm going to delete out this fantasy keyword, and we want to include that in a string or by wrapping it in quotation marks. That way, the browser knows that this is all one word, and it knows exactly where the name of the font family begins and ends. So I'm going to add those quotation marks, and type out the font name. And if you look over in the browser, you can see it's still using a serif font; it's not using Proxima Nova, and the reason for this is because I don't have Proxima Nova installed on my computer. So for this reason, it's a good idea to use what's known as a web‑safe font or a font that comes installed on most operating systems. Examples of web‑safe fonts include serifs like Times New Roman and Georgia are both web‑safe fonts. In the sans serif category, we have fonts like Arial or Verdana, Comic Sans, and Trebuchet, and we also have in the monospaced category fonts like Courier New. And the reason these are considered web‑safe is because they're installed on over 90% of users' computers, at least for those who use Windows and OS X. So it's good to be familiar with what is considered a web‑safe font, or what's considered safe to use. You can also go to different websites and get an idea of what web‑safe fonts are as well. One of those websites is cssfontstack.com, and one great thing about websites like this is you can come in and see exactly what percentage of support there is for a specific font. So if we take a look at Arial, you can see it's installed on 99.84% of Windows machines and 98.74% of Mac machines. So you know there's a pretty good chance that Arial is going to be what's displayed to the user as you use it as your font. I'm going to scroll down a little bit more, and what we're going to be using for our headings is the font Verdana, which you can see has 99.84% support for Windows and 99.1% on Mac. So I'm going to take that Proxima Nova font and I'm going to replace it with Verdana. And if you look over in the browser window, we can see Verdana being applied to our headings. Now what if Verdana wasn't installed, but we wanted the option to serve that to the user if they had Verdana installed, but we wanted to have a fallback option as well? We can create what's called a font stack to do just that, to provide a set of fallback fonts in case a user doesn't have them installed. That way, you can still have some control over what font is being used while having those backup options as well. And the way to create a font stack is to create a comma‑separated list of values. So let's say I want Arial to be of my next option in case Verdana is installed. Then I can add a comma, and then type out the name, and what the browser is going to do is it's going to start at the left side of your list and go down until it finds a font that's installed on the user's computer. I'm going to add as a last option one of those generic keywords. So I'm going to say if there isn't specifically Verdana or Arial, at least use a sans serif to get that feel. So I'm going to go ahead and type that out. And if you're thinking that this seems very limiting in terms of the fonts you can use, then you are correct, and we do have an option to use externally hosted fonts, which we'll learn about in our next clip.

Using Externally Hosted Fonts

The web would get a little bit boring if we were limited to web‑safe fonts. While they certainly have a very important purpose, luckily, we have the ability to use other fonts as well. And the way to do this is through hosting fonts, either with a website's files or using an external service like Typekit or Google Fonts who host the fonts for you. We're going to be using two fonts from Google's Font service, so you can go to fonts.google.com. And the first one you want to look for is called Cookie, and you can see it's a nice script font, which we'll be using for our headings. And you want to come over and choose Select this style. And the other font we're going to be using is called Lato, so if we do a search, you can see it here. It's a nice sans serif font that we'll be using for our body. And there's a bunch of different weights available, and while we could add them all, you want to be careful about the amount of font files you're adding to your site because it can significantly increase the load time. So we're going to be using the regular weight and then also the regular italics weight as well. And in the top‑right corner, you can view your selected font families, and we'll want to copy this link element. So I'm going to come and hit Ctrl+C or Command+C to copy that. And we want to add that link to the head of every HTML file where we want these fonts to be pulled in. So we want these to be pulled in on our whole site, so you would add it to the top of every HTML file. So I'm going to come in and add it above our stylesheet and hit Ctrl or Command+V to paste that in. That way, we're importing our fonts before our CSS stylesheet. And you can see in this URL fragment that we have Cookie and both Lato being pulled in. So if I hit Command or Ctrl+S to save that, we can now use those fonts. With our font files available, we can now use them in our CSS as a value, so I'm going to start by adding that Cookie font to our heading. So I'm going to build off of our existing font stack. And since I want it to be the first option that gets pulled in, I'm going to add it at the far left or as the first item in our font stack list. And that's called Cookie. I'm going to add it in quotation marks, since that's a custom font. And we know our font is getting pulled in correctly, because over on the right you can see we now have that script font. One thing you want to be careful about with custom fonts, I'm going to zoom in so you can see this, is when you apply it to something like a heading, the browser default styles make headings bold, and the Cookie font has only one weight available, which is a regular weight. And since the browser expects this to be bold text and it doesn't find it in the font files, then it's going to create its own version of the font that's bold. This is called browser synthesis. So it's a little bit hard to tell with this font, but you can see it's a little bulkier than it should be, so I'm going to press Enter and come on to a new line. And one of the ways we can fix this is, we can use that normal value for the font‑weight property. So I'm going to come in here and type out that font‑weight property and set it to normal. And now you can see we don't have that rather ungraceful browser‑synthesized bold font. One thing that that made obvious, though, is it may be our letter spacing is a bit too much, since our individual glyphs, or characters, aren't connecting like they should for a script, so you might want to come and adjust our letter spacing a little bit. I'm going to bump that down to 1, and now that looks pretty good. Now that we have are heading styles, I want to change our body text to use that Lato font, which pairs nicely with Cookie. So we want it to apply to any type of normal website text, not only paragraphs, but we want it to apply to list items and other types of regular text. And one way we can do that is, we can apply it to a parent element because the font‑family property is inherited by descendant elements from their parent element. So if we used one of the biggest parent elements like the body, for example, which wraps all of a website's content, it would be a good way to apply that font to all of our text. So I'm going to scroll up here to our body element. We have one already, a body selector, on line 1, and there's already a font stack started. So I'm going to come in and add our Lato font, and then make sure we add a comma after. And now we have that applied to our site, which I'm going to zoom out a little bit so you can see all of our site. And it's applied to our footer, our list items, and the rest of our text. You'll notice it didn't apply to our headings because, although our headings are a descendant element of the body, they have that Cookie font applied directly to that element itself. In this module, we learned how to style text such as changing its size, its weight, and its style. We also learned how to align it left, center, or right aligned. We learned how to capitalize, make text uppercase or lowercase with the text‑transform property. We learned how to change its line height or it's leading, the space between lines of text, or we it learned how to add white space between characters using the letter spacing or adding tracking. We learned how to apply fonts through an external service like Google Fonts, or how to use web‑safe fonts or fonts that come installed on the majority of operating systems.

Sizing Elements

Module Overview

In this module, we'll learn how to change an element's width and height using both relative and absolute units and talk about the difference between the two. We'll add color and borders to elements and work with images, and we'll also talk about margins and padding and how they affect elements by discussing the CSS box model. Throughout this module, we'll be working on the promotions page from module two, and we'll learn how to style it in a new way.

Changing the Color and Size of Elements

We're going to start styling our promotional items, starting with adding a background color to these elements. So we want to choose our selector first, and looking at each promotion, each one is inside of a div element, but since we have multiple div elements on the site, we probably don't want to style them all to have that background color. So one thing we can do to target one element or a set of elements is use what's called a class selector. And a class selector selects a set of elements. Using a class attribute, you can assign it a value, and every single element with that class value will be affected using the selector. So let's take a look at that class selector. What we want to do is we want to add that markup to the HTML first. That way we have a hook for our CSS. And I'm going to start with our top div element, and we'll add that class attribute. And then we want to give our class a name, something that's semantic and it's descriptive, and it will tell us what we're styling. That way, when we refer to it at a later date, we know what this class is for. So something like promotion makes sense to me. So I'm going to come in and name our class promotion, and then I'm just going to copy and paste this, and I'll add it to our other promotion as well. And let's switch back over to our CSS, and I'm going to scroll to the bottom, and the way we can use a class selector is by using a period in front of our selector. So we add that period, and then the class name, which is promotion, and then we want to add our property. So I want to change the background color, and there's a CSS property for that, and it's called background‑color, just like it sounds. That's background hyphen color. And we could use a keyword here, but I want to use what's called a hex value, which is a six‑digit code, which stands for an RGB value. So the first two values represent the red value, the second two represent the green value, and then the final two represent the blue value. And a hex value always starts with a hashtag. So we add that hashtag, and then the value, and I have mine copied to the clipboard. It's 23cea6 to get a blue/green that matches the Bethany's logo. And then we want to add our semicolon, and now I want to change the color of the text as well to match the look and feel of the logos. So I'm going to press Enter and come on to a new line. We can use that color property, and even though we're not applying it directly to the heading or the paragraph, since we're using a class here, then the child elements are going to inherit those properties from that class. So I'm going to type out white. We can use our keyword here, and that looks pretty good. And now I want to change the width of the element as well. It's currently taking up the full width of the section, this main content section, and I want to shrink it down a little bit to make those little promo boxes. And we have a CSS property called the width property, which allows us to do just that. So I'm going to add in that property, width, and we can specify a pixel value as that value. So let's say we want to change this width to about 450 pixels. As I add that in, you can see over on the right that it's changed the width of our container to be 450 pixels. Now, if we didn't have that background color in there, it'd be a lot harder to see, because that background color highlights the render box or the boundary box of the actual content. So this is looking pretty good so far. I also went to center our text in that element. So we can use our handy text‑align property that we learned about in our last module, with a value of center. And you can see it's going to center our text. Another thing I want to add to this is an image to give some visual interest to these promotional boxes, since we're trying to entice people into buying these products, into taking advantage of these promotions. So we want to do that in our HTML. And I'm going to come back over to our promotions.html file, and we're going to add an image above each heading. So starting above our Buy One Get One heading, we're going to add an image of a fruit pie, specifically an apple pie. So in our source attribute, we're going to look in the images folder, and there's a sub folder called products. And then it's that first image pulling up there, applepie.jpeg, and then I'm going to add an alt attribute so apple pie. And if you look over on the right, you can see that this image is huge and it's overflowing our content box. So in our next clip, we're going to learn about how to use relative units to solve problems like this.

Relative and Absolute Units

We saw when we added our pixel width to our promotion class, it caused a little bit of a problem with the layout, in that our image is larger than that fixed width, and it's overflowing its container. And this is one of the drawbacks of using a fixed unit like a pixel. You have to be careful that you have enough room left for the elements inside and that they're not overflowing the width of their parent container. But there are other options. One of those is to use a relative unit. Relative units are flexible, and they're calculated relative to the size of their parent element. So I'm going to come down onto a new line, and we're going to target that image and change its width using a relative unit. And that image is in that promotion class applied to that parent div. And then we can use a descendant selector using that space to target images inside of that promotion class. And then we're going to give it a width, and instead of using pixels, were going to use percentages. And how percentages are calculated is based on the parent width, so let's say we added in 50% here. It's going to take that 450px and take 50% of that. So our resulting width will be 225px. So if we change this to 100px, then it's going to be exactly 450px, which is pretty cool because, let's say later we wanted to come and change the width of our promotion class to maybe 650px, then our image is automatically going to be updated and calculated based on that increased width, so we don't have to recalculate and edit the width of the image itself. In addition to percentages, we also have ems as relative units, which are relative to the font size of the parent element., and then we also have rems, which are calculated based on the root element, like the body or HTML. I'm going to bump down our promotion width to be 450px again, and let's try out using an em instead of a percentage and see how this changes. So to start, I'm going to add a font size to our parent element, since that em is going to be calculated based on that. So I'm going to add a font size of, let's say, 50px. And then if I edit the width of our image, let's say I make it 1em, it's going to be 1 of the parent font size. So if we add 1, it's going to be 1 times 50, which is 50px. And if you look over in the browser window, you can see we now have a small image. If I bump up this to 2, it'll be 2 times that 50px, so our image is now 50px. If we bump it up to 3, it'll be 150px, and so on and so forth. If we were to change this to rems, it's going to be the font size of the body or the HTML element. And the default is 16px, and we haven't changed that default, so if we were to change this to 1rem, then our image is going to be 16px wide. If we were to bump it up to 2 it would be 36px, and that size would change depending on if we change that root font size. But for our situation, a percentage of the parent container is easy to calculate, and it makes sense for us. So I'm going to change this to be, I think that 50% of the promotional element looks good. And then I'm also going to remove the font size as well.

Adding Borders and Outlines

Our promotions box is looking pretty good with its background color, but let's add a border to add a little bit more color to this element and make it stand out a bit more. And we can do that using the border property. So I'm going to press Enter inside of our promotion class, and use that, it's just border. And we have three different values we can supply, one's a width, one's, a border style, and the other is a color value. So I'm going to start with a width. Let's say we want to make it 10 pixels. And then our second value will be the border style, and we have multiple options here. We can make it solid, dotted, dashed, double, groove, ridge, inset, and outset. So let's take a look at that first, we could make it solid, and you can see our border is already being applied even though we didn't supply a color yet, and that's because we have the option to leave this off, and it's going to pull the text color from the element itself. So since our color is white, I'll click out of this so you can see, we're getting a white border. If our text color for this element was red, then you can see the border is going to be red as well. So I'm going to change that back to white, and then the color I want to use for our border is this purple color in the navigation. So you'll want to start with that hashtag, and then our purple value is going to be A693C2. And now you can see we have that purple border being applied. If we wanted to apply a border to just one side, we could do that as well. We could specify one of the four values or four sides of an element, top, right, bottom, or left, and it'll apply to only that side. So we'd want to add a hyphen, so let's just say I want this on the left side only. Then I could say border‑left, and you can see we get our purple border only on the left side, which I want it to go on all four sides, so I'm going to go ahead and take that out. And let's take a look at the different styles of borders we can use. So we have that solid border, if we wanted it to be dotted, then we're going to get 10 pixel with dots around our element. We could change it to a ridge, and then we get this ridge outset style border. So it can be kind of fun to play around with the different styles and see what you can create, which I just want a solid border, so I'm going to change that back to solid. And you might have noticed that our borders on the bottom and top are touching, and that's because the width of a border gets added to the outside of the render box of an element. So it adds to the width and height on all four sides or wherever the border is being applied. So you want to be careful and make sure you don't end up with any overlap in this type of situation. Similar to borders, we can add outlines as well using the outline property. So I'm going to come on to a new line. And just like the border property, we have three values, we can specify a width, a style, and a color. So let's say we wanted to add a 10 pixel outline, and we want it to be solid, and let's just say we wanted it to be red. If I click out of here, you can see we have some overlap, which is to be expected because they're very similar elements, the main difference is that a border adds to the content box of the element, and the outline is just tacked on outside. So if I were to increase this to 30 pixels, you can see that the blue background or the render box of the element itself doesn't change at all. So I'm going to bump that back down to 10 compared to if we were to change the border value to 50 pixels. You can see it is added to the content box, and the overall size of the element increases with the border change, which we don't want that big old border, so I'm going to bump it down to 10, and I'm also going to remove our outline as well. And overall, this looks pretty good, except for we could make some adjustments to the white space, we have these borders touching, we have some text touching and jetting up on the side of the element, so that's what we're going to learn how to change in our next clip.

Adding Margins and Padding

We've been neglecting our cheesecake promotion a little bit, so I want to take a moment to add an image and to style it so it has its own unique style. So I'm going to scroll down in our HTML file and start by adding that image, which it's in the images folder and then in the products subfolder, and it's called cheesecake.jpg. Then we're going to add an alt attribute. So, it's a picture of a cheesecake, so we'll type that in. Then we want to add our right bracket, and I can see our cheesecake image is pulling in. I also want to style the colors a little bit differently as well, so I'm going to add another class to this element, which elements can have multiple classes. If we go into the class attribute, we can add an additional class just by using a space. So we're going to color this purple, so I'm going to call it promotion‑purple. And then hit Command or Ctrl+S to save, and now we can use that as a hook to style just this second promotion. So I'm going to scroll down to our CSS style sheet, and right below our promotion image selector I'm going to type in that promotion‑purple, and we're just going to flip the colors from the other promotion. So its background color is going to be that purple, and its border is going to be that teal blue. So I'm going to copy our border color since that's going to be our background color, copy and paste that value in there, and now our background is now purple. And we want to add that border, which we already have a border pulling in through our promotion class, which is being applied as well, but we have a property to just change the border color itself. So I can add that, it's border‑color, and then we want to copy and paste the background color of our other promotion, that teal. And overall, this looks pretty good. One thing we do want to add is some whitespace since, as we mentioned before, the text is jutting up against the left side, the image is touching the top, and the elements are touching in between as well. So with a little bit of whitespace, we can start to make this look a lot cleaner and a lot nicer. And one way we can do that is using the padding property. And what the padding property does is it allows us to add whitespace inside of an element, which will allow us to add some whitespace here by the text and above the image as well. So I'm going to come into our promotion class, and we have multiple options for using the padding property. We can add it on all four sides, we can add it to the top, we can add it to the right, the bottom, or the left side individually. So let's say we just wanted to add it to the left side. We can use padding‑left and then use a pixel value. So let's say we want to add 20 pixels of space on the left, then you can see our text is no longer touching on that left side because it's added 20 pixels of whitespace inside of our element's content box. But what I want to do is I want to add an equal amount of padding on all four sides, so we could use all four properties individually. Let's say I wanted to add it on the right. We could add it on the right. But there is a simpler way to do this when we want all four values to be the same, which is to use a shorthand property. So I'm going to go ahead and delete this, and that shorthand is just padding, and it's going to apply it on all four sides. So I want that 20 pixels of padding, so I'm going to add that as the value. And if you look over in the browser, we have it added to the left, we have it on the top, the right, and the bottom. So our image is no longer jutting up right against the top of the element. We have a nice even amount of whitespace all around the inside as well. In addition to setting all four values at the same time, we also can use a padding shorthand to affect the padding in different ways. We can supply four values, and it'll affect, respectively, the top, the right, the bottom, and the left. If we added two values to the padding shorthand, the first value would affect the top and the bottom, and the second would be applied both to the left and right. Or we can apply one value like we just did, and it will get applied to all four sides equally at the same time. So if we had a situation where we wanted 20 pixels on the top and bottom, and let's say we wanted 100 pixels on the left and right, I could add a space and add that second value, and our first value is going to be applied to the top and the bottom, and then our second will be applied to the left and the right. If we didn't want to separately type out a padding‑left, padding‑right, a top, and a bottom value, we could also use the padding shorthand with four values. Let's say we wanted 200 pixels on the bottom, which is a lot, and we wanted 50 pixels on the right, just four different values so we can see this, then we have the ability to set those all at the same time. So I'm going to go ahead and delete those four values because we just want those 20 pixels around evenly. And now all the whitespace looks good inside of our element. There is another property we could use to add space outside of an element as well. And that element is known as the margin property, and it allows us to add whitespace outside of the content box of an element. Similarly to the padding property, we can set all four values at the same time using the shorthand. So if we have four values, it'll respectively apply to the top, right, bottom, and left. We can use two bottom, two values to apply to the top and bottom and the left and right, or we can set all four values with one value at the same time. And in our situation, since we want to apply a margin to the bottom of our element and to the top of this other one, we're going to use those specific values to add that whitespace in between. So I'm going to add a margin bottom here, margin‑bottom. Let's say we want to add 50 pixels of space. Then you can see over here our elements have now pushed away from each other. One thing to watch out for with vertical margins is something called vertical margin collapse. So if we had a margin‑top just on our purple element, I'm going to go ahead and add that, let's say we have a margin‑top of 50 pixels, what you think might happen is now that we have a margin of 100 pixels between the two elements. But if I add this in, you can see that nothing has happened. And that's because when there are margins that are touching each other on the bottom and on the top, the largest gets applied instead of both. Or, in our case, since they're equal, only one is getting applied. So if I were to increase this to 100, instead of adding those margins together and having 150 pixels of whitespace in between, then we have 100 instead. It'll just take the largest margin and apply that one. So kind of a weird quirk in CSS. I'm going to go ahead and delete this one since we just want that 50 pixels of space. And in our next clip, we're going to talk about the box model.

Understanding the Box Model

So we started adding margins and padding to our content, which affects our content's render box, and with our background color, like in our situation with our promotions box, we can see visually what effect that it's having, we can see when it adds it inside the box or when it adds it outside of the content box as well. But we don't always have a background color on element, so sometimes this change can be hard to see, or it can be hard to visualize what's a margin and what's a padding. So let's apply it to you an element that doesn't have a background color. I'm going to come into our CSS, and let's say we added a 60‑pixel amount of padding on all sides to our footer. You can see we have 60 pixels now of white space around that, but if I were to change this to a margin, if you look over here visually, it looks identical, but what's happening is not exactly the same, although visually it looks like it has the same effect. And this is where something like what's called the box model comes in handy, especially when a layout breaks, it could help us understand exactly what's happening to an element when we make these types of changes. So the box model is just that, it's a model that helps us identify exactly how the width and height of an element is calculated. So let's say we have an element, it could be either, have its width and height calculated by the browser depending on the width of its contents, or it could be set specifically in the CSS using the width and height property. When you add padding to an element, then that padding gets added to the render box, we've been using that word a lot, which means that default width and height is either is determined by the browser or as you by the author. so when we add padding, it gets added to that width and height. When we add borders, it gets added outside of that padding, and it also effects the width and height of the element. Margins get added as white space outside of the content box or the render box of an element, and they don't affect an element's width and height. So let's say we add an element with a width of 400 and a height of 200. If we were to add 40 pixels of padding on all 4 sides, the new width of the element would be 480 by 280. If we were to add a 20‑pixel border on all 4 sides, then the new width and height of the element, or the total width and height of the element would be 520 by 320. If we were to add 100‑pixel margin on all 4 sides, then the width and height of the element itself would be 520 by 320, since the margins don't get added to the width of the element itself. So one common reason a website layout breaks is because there's not enough width, or height, or room for an element in a specific place, and often the culprit is forgetting to add the padding and the amount of borders to the total width and height of the element to figure out how much space it needs. And it could be hard to calculate that in your head, but one great thing you can do is the browser will show you the actual box model of an element. So if we use the inspectory, you can come down and right‑click Inspect to bring up the developer tools, and it'll be different every browser, but in Chrome you can come over in the Styles panel, and you'll have this handy‑dandy graphic of the box model and action, so we can use the Inspector tool, and we can select one of our promotion boxes, and so we can see that the content box of the element itself is 450 by 250.438,, we have 20 pixels of padding on all 4 sides, and the borders and the margins, and we can quickly calculate it to figure out how much room you might need for something, or if something's breaking, if there might be something that's part of this that could be at fault. So with that, that brings us to the end of our module, in which we learned how to change the color and size of elements. We learned how to use relative units like percentage, em, and rem, and about absolute units like using pixels as well. We learned how to add borders and outlines to elements, and also how to use margins and padding to add white space with outside and on the inside of an element's content box, and we also learned about the box model as well, and how to use it to calculate an element's actual width and height.

Using Selectors

Module Overview

In this module, we'll be building on the selectors we've learned throughout this course such as class and descendant selectors and learning additional types and the ways it can be used such as IDs and how they differ from classes, combinators, which depend on HTML hierarchy, pseudo classes that use element states and locations, and pseudo elements, which allow you to style part of an element as if it were an element itself.

IDs and Classes

We've learned how to use class selectors, which allow us to apply styles to a single element or a set of elements with a matching class. But sometimes there will be a situation when we only want to style a single element. Using an ID selector, we have the ability to assign an element a unique identifier or an ID and style only that specific element. So let's say we want to style the heading in our sidebar right under our logo. Well, we can do this using an ID, since this is just a single element that we want to style, and it's the only place that this element appears on the page. It's the only sidebar with the only heading. So to start, we want to come over to our HTML file, promotions.html, and I'm going to scroll down to that element, which is an h4. And similarly to applying a class to an element, we use the id attribute, so that's just id. And then we'll assign it a value, which we'll use as our selector in our CSS. So we want to name it something semantic or descriptive, so we can name it something like sidebar‑heading. And now we have a hook in our CSS, so I'm going to come over in our CSS and scroll to the very bottom. And what I want to do is, I want to increase the size of the heading. Since as we switched fonts, it looked fine before when we were using original font, but as we switched to that Cookie font, you can see that even though the font size hasn't changed, it looks quite small because there are variations in size across different fonts. So we can fix this. So I'm going to use that id. And the way we target an ID or create an ID selector is using a hashtag. So we can add that hashtag, or pound sign, and then we'll want to use the ID name. So since this is called sidebar‑heading, we'll just type that, sidebar‑heading. And it's currently 16px, so we want to bump it up a couple sizes to make this easier to read. And so I'm going to change the font size to be about 24px. And if you look over on the right, the font size of our heading has now changed, and it's a lot easier to read. So IDs can be great for targeting a single element in this way. But one downside of IDs is they can only be used one time for HTML page or HTML document. If an element only appears one time for a document, then that's totally fine and an ID can be an appropriate selector to use. But in general, if you can use a class, it's easier to maintain your CSS because class styles are reusable, since you can apply them to multiple elements.

Combinators

There will come a time when you need a special type of selector, not based on a class you assign or an id you assign to specific elements, but based on content patterns or the structural relationship of elements. One example of this is our promotion image selector, which is targeting any image inside the promotions class. Sometimes we need a selector that's more specific than this. Maybe we have multiple images inside that promotion class, and we want to target certain ones based on their structure, and leave out other images out of our selector. There's a special category of CSS selectors called combinators that allow you to do just that. They allow you to create styles based on HTML patterns and the relationships between elements. Our .promotion image selector is an example of this called a descendant combinator, in which it targets any descendant of the specified parent. So in this case, we're targeting any image inside of the promotion class. There are additional types of combinators like this selector that we can use as well. So let's say our website is growing and changing, and we have new content patterns starting to emerge. For example, our sidebar has now been expanded to include subcategories within each category of pie. So we've expanded our sidebar navigation. If you look at the HTML structure for our sidebar, we have one ordered list and an embedded unordered list inside of that navigation. If we were to want to style this sub list differently from the overall list, let's say we wanted to style these in a different color or a smaller font size to represent the hierarchy of content, then if we were to use something like a general descendant selector, then all of our styles would be affecting both of these lists. So I'm going to switch over to our main.css file and using a special combinator, let's say we wanted to target the outside list only, we could certainly do that. With combinators you use a special character to signify a specific relationship. So one of these is the child combinator, which targets only the direct child of an element, and it uses a caret sign. So let's take a look. First, we want to specify the parent element that we want to target the children of. In this case, it's our sidebar, which has an id of leftmenu so we can use that as our parent. So leftmenu, and then to signify that we only want to style the children of that element, we can use a caret, and then we want to name a child. In this case, since we just want to target the outer list, we can specify that we want to target an unordered list, which is a direct child of that leftmenu parent. And our links are inside of a link item so we can target our list items. And here's where we leave out that embedded list out of our style, because it's a descendant of the list item and not a direct child. So if we add another caret, we can build a more complicated selector this way, showing that exact specific HTML structure. If we target the links inside of those list items, I'm going to just change the color to red so we can see this. You can see, we're only targeting that outside list. Now if we wanted to target the embedded list instead, then we just want to think about the HTML structure of what we're trying to target. So if you look at our HTML again, our unordered list is a direct child of a list item, so we can use that to target just that list. So it's a direct child of the list item. So immediately following our list item, we can say we want a target an unordered list, and you can see it's changing the bullets to be read already, but we want to target the links themselves so those are in a list item element, and it's those anchor tags. So you can see how useful, if you're familiar with the structure or if you have a very specific situation, how useful using combinators can be. In addition to descendant combinators and child combinators, we have a couple additional types as well. We can use an adjacent sibling selector, which will target any sibling immediately following the specified element. So in this case, it would be targeting any h2 directly following an element with a sidebar class. There's also a general sibling selector. What this does is it matches all siblings following the specified element type. In this situation, it would find all elements matching the sidebar class and target any h2 that's a sibling of the sidebar element. Combinators are very specific to a certain situation, so they're best used sparingly, but they are extremely useful if you need styles based on HTML relationships. They can also be very useful when you don't have control over markup. For example, in a content management system, it can allow you to apply styles when you might not otherwise be able to.

Pseudo Classes

In addition to styling elements based on their HTML structure or relationship to other elements, sometimes you want to style an element based on its element state as well. Using a special type of selector called a pseudo‑class, you're allowed to do just that, in addition to styling an element based on its position. Certain types of elements have interactivity built in, such as links, and because they have this interactivity built in, they have specific states of being that we can actually target and style using pseudo‑classes. So let's take a look at how we can use those pseudo‑classes to style some links. I want to apply some specific styles to our sidebar links, so I'm going to press Enter and come on to a new line, and we have that leftmenu class that we can use to target just that set of links. And then we want to target any link inside of that leftmenu, so we can just use that anchor element. And we can start by supplying a base color to the element itself, so I'm going to add that color value. And then I'm going to copy that blue‑green teal color from the logo, which we have right up here, and add that as our color. And you can see it's a nice color, but maybe it's a bit hard to read, so I'm going to use the color picker in brackets, which you can access by hitting Command or Ctrl+E, and I'm just going to darken this color a little bit to see, where it's a little bit more readable, and I think this looks pretty good at 199175 as the hex code. And this is a good start, but if we click over here, we don't have a hover state applied to our link. So when we hover over a link, that's actually a pseudo‑class that we can target. So I'm going to come down onto a new line. And then to use a pseudo‑class, we can use a colon and then the name of the class. So that's the hover state or hover class, so you just type that in. And then we can assign it a color value or any other special styles you want to add to the link when it's in that hover state. So I want to change the color, so I'm going to add that color property, and then I'm going to make it a little bit darker than the default link, so I'm going to paste that color in as a starting place and then just use that handy‑dandy color picker to pick a darker color. That looks pretty good to me. So that's 116552. We of course want to test it to make sure it looks good. And now, as you hover over these links, you can see we have a color change, which we didn't have before. So this is the exact type of thing that pseudo‑classes allow us to do. They allow us to target a specific element state that we wouldn't otherwise be able to style. Other examples of pseudo‑classes include focus, so if someone uses keyboard navigation, you can see the browser has some default focus styles that adds a border to the element, so you can see where you're navigating. Or, for example, when you fill out a form, when you highlight an input field, or when you select an input field, then that's receiving focus as well. So using a pseudo‑class, we would be able to a style that state and add some special styles. Other types of pseudo‑classes you can use include location‑based pseudo‑classes. For example, we can select the first type of an element or the last type. So let's go ahead and take a look. I'm going to go over to our pieoverview page, and on this page we have a pretty large table, and this is a good example of how you could use something that's location based to improve the readability of a table. For example, if we wanted to alternate colors of each row to make this information easier to scan, then rather than adding classes to each individual element, then we could use a pseudo‑class instead. And every time we have a table on our site, we aren't required to add that class to each and every element; we already have that style in place to automatically style our table in that way. So let's go ahead and take a look. I'm going to come over to our CSS file, and right below our hover state, we can target any table row on our site, so that's the tr element, and then we can use a pseudo‑class to target a specific position in that table. So one example of a pseudo‑class is the nth‑of‑type class, so I'm going to use that semicolon, and that's nth‑of‑type. And then in parentheses we can specify the number or the position of what we want to target. So I'm just going to target the first row for now so we can take a look at this, and we'll just add a background color of red. And you can see that the first row only of our table is having that background color applied, which is pretty cool. If I were to change this to 2, it would highlight the second row. If we changed it to 4, it would highlight the fourth row, and so on and so forth. In addition to specifying a specific number, we could also specify keywords as well, such as odd and even, which is perfect for this type of scenario if we want to alternate different color rows. So I'm going to start with styling the odd row, which you can see is now highlighted as red. And instead of that red, we're going to do something that matches a little bit closer to our color scheme, and that is a hex value of b4ddd3, which is just a lighter teal. And we can style those even rows as well, so that's the nth‑of‑type, and then instead of odd, we can use that even keyword. And then this time we're going to add a different background color. It's going to be a light purple, and that is e0d7ef. Now I'm going to hit Ctrl+S to save. And now if we go through and look at the other tables throughout our pages, we can see that they're already styled automatically without having to go in and add those additional classes, so that's pretty cool. In addition to selecting an element based on its number position, we can also select an element for being the first of its type or the last of its type. We can specify it based on being a specific number child of an element or also the first or the last child.

Course Conclusion

There are many different types of selectors from type selectors for general elements to author selected groups of elements with classes and IDs to combinators that allow you to style elements based on HTML relationships and pseudo classes for styling based on interactive state and element position. By learning the choices you have available to you and the situations where each can apply, you can learn to write CSS that is maintainable, scalable, and reusable. So with that, our course has come to a close in which we learned how to add styles in different ways such as embedded, inline, or externally. We learned how to create a stylesheet and link it to an HTML page. We wrote style rules from scratch, learned how to style text and add external fonts. We also used relative and absolute units like the pixel, percentages, and ems and rems. We changed an element's width and height and also learned about how margins, paddings, and borders affect elements using the box model. We also learned about the different types of selectors available to us in CSS.