Introduction

Hi this is Scott Allen and this module is an introduction to Cascading Style Sheets. In the mid 1990s when the web first entered the mainstream, websites and web browsers were relatively simple. Web pages consisted mostly of text and hyperlinks. Colors were generally solid and images were rare. Most design was done using simple tabular layouts. Fast forward to today and we've been working with cascading style sheets for over a decade. CSS is the technology that allows us to create beautiful, maintainable, and flexible website designs and in this module we'll learn the basic principles behind CSS. We're going to look at how to add styles to a web page and talk about fundamental principles like selectors, rules, and units of measurement. This is the first module in a series of modules that will give you everything you need to know about styles for moderate websites.

Why CSS?

CSS is a language you can use to control the formatting, presentation, and overall look of a web page. So thinks like colors, layouts, and fonts. To get an idea of how CSS works and why CSS exists, let's look at some HTML markup that is not using CSS. The markup you see here is displaying a short quote from a fantasy book I enjoyed recently. The HTML contains not only the content, which is things like the quote, the author's name, and the name of the book, but also presentation instructions, stuff like the background color for the page and a font size. This approach can work, as you can see in the output, but mixing content and presentation makes things difficult. Every time I create a new page I have to remember to set the background color if I want any consistency across a website. And if want to add additional quotes to the page I have to remember to set the font size again. Imagine in 6 months if I wanted to redesign a website with a fresh new look or even just make simple changes, like using a different font size for all the quotes or a different background color, I'd have to go through all of the HTML and change these color and font settings everywhere. CSS will allow me to remove all the presentation information from the HTML and leave behind only the content. That makes the HTML easier to maintain, because it is only focused on content, but it also makes it easier to change and experiment with different designs. All of the presentation instructions will be in an external style sheet. It's a best practice to use CSS to manage all the presentation aspects of a website, so let's look at a demo to see how it works.

Adding Some Styles

Here on the file system I have an htm file that contains the markup was saw in this slide earlier. Only this markup doesn't contain any presentation information so the page renders in a rather plain fashion, just black text on a white background. What I'd like to do is add some color to this page and one way to do that would be to go into the head tag of this page and add a style definition inside of a style tag. This style type is text/css and it has inside of it a style rule. We'll be talking about rules later in this module, but this rule tells the browser, go out and find the body element and set it's background color to blanched almond. If we save that file and refresh the browser we can see that change has taken effect. Now this approach to adding styles to a web page is effective somewhat in separating the presentation rules from your content because now at least all the presentation rules are in one place. And what we'll see as we go through this module is that you can use these styles rules to influence different elements on a page, so we could also change the paragraph tag and so forth that's in this page, but it does miss one of the advantages of using a true external cascading style sheet file and that is being able to reuse these style rules throughout a website. So I'm not going to take this approach, I'm going to remove the style and instead I'm going to open up a .css file, this is the common extension for a cascading style sheet. We're just going to open it in Notepad++, which is a text editor, because CSS files are just text files. Inside the style sheet I'm going to put the same style that we had earlier, which is going to tell the browser to set the background color of the body to a specific color. With the way things are set up right now, this isn't going to work simply because the page that is loaded doesn't know anything about the style sheet. What I also need to do is come into the HTML page and provide a link that will point the browser to pick up the style sheet, that's done using a link tag. The row attribute specifies the relationship between this document and the link, we're telling the browser that this is a style sheet for the current document. The place to go get it is in the same relative place, just look for something called stylesheet.css, but of course you can specify an entire http address here and the MIME type of this document is going to be text/css. So with that link in place and a style to find inside of our style sheet, what we can now do is have all of our presentation information defined in a separate file, a CSS file, and we also can include that CSS file in as many web pages as we want. We can also put our styles across many different CSS files and have multiple links in here to pull in different style sheets from different locations. Since this web page is relatively simple, we're going to stick with one style sheet because we can define multiple style rules inside of this style sheet. For instance, we could also add a rule that says for all paragraph tags, the font size for these paragraph tags should be larger than normal. Let's save that and watch the text carefully, as I refresh it becomes a little bit larger. So these are two quick and simple demonstrations of what you can do inside an external style sheet with style rules and styles and how you can influence the presentation of your content just using the style rules. And now that we know a little bit about how CSS works, let's dive into some specifics.

CSS Levels

The CSS features we are going to look at are all backed by specifications maintained by the World Wide Web Consortium or the W3C, you can read the specifications and the latest CSS news at the W3.org website. It's important to understand a little bit about the specifications because they do evolve and one of the unfortunate characteristics of CSS is that you'll find varying support among the different browsers. We'll talk about some of those scenarios throughout the course. CSS was first introduced in 1996 with the CSS 1 specification, but it took 3 to 4 years for browsers to provide reasonable implementations of that first spec. These days most modern browsers provide pretty good support for CSS 2.1, which started gaining traction around 2004. There is also some support for some of the nearly complete features of CSS 3, that's still a work in progress, but pieces of the specification are relatively stable, and some of the latest browsers, like IE 9, have already implemented those pieces. Most of this course we'll be using features defined in CSS 2.1, but we can take a peek at some of the CSS 3 features too. If you're wondering if a particular CSS feature is supported in any given browser, you can visit quirksmode.org. This site provides a table allowing you to cross reference specific CSS features against different versions of IE, Chrome, Safari, Opera, and Firefox. This table will tell you if a given specification or piece of a specification is implemented by any particular browser.

Rules

The CSS specifications that we just talked about will tell us how to format a style rule. Style rules follow a very simple format. First you have a selector and then you have a set of name value pairs. The selector will tell the browser where to apply a rule. We'll talk in more detail about selectors in just a bit, but then the name value pairs that come, they have to be enclosed inside of curly braces and the names and the values that you can use here, like background-color and #cccc99, those are things that are defined in the specifications. The name and the value are separated by a colon and then each name value pair ends with a semicolon. That's because you can have multiple pairs on a given rule. This example is setting not only the background color for the body of the page, but also the padding. We'll talk about padding in the third module. For now I'll just tell you that that particular setting will push the content of a web page further away from the edge of the containing window by 10 pixels, that's the 10px. The background color, by the way, is specified in hexadecimal rgb format. Let's take a look at how we could ever know this. First, if you go to the W3.org website you can find something known as a full property table for the CSS 2.1 specification. This table will tell you, generally speaking, the available properties that you can set in a style rule. So there you can see background color, background image, background position, just to give you an idea let's scroll down here to the bottom, there's padding, page break settings, all of these are available. If we look specifically at background color we can see that the initial value should be transparent and that the value that we can set a background color to can be transparent, inherit, or a color value, let me click on that and then this will show me that there are named colors for the web, like maroon, red, orange, yellow, olive, and this diagram also lists the hexadecimal codes for those colors, so when you set a color you can use a standard name, you can use a hexadecimal code, and you can also use what's known as the functional notation because it looks like you are calling a function named rgb and passing in individual values for the amount of red, the amount of green, and the amount of blue to use. Many people will use color pickers to find out what values they want to use. See here's one that I found by searching the web, it's on HTML/color/codes.info. Inside of here I can select sort of what base color I want and then fine tune it a bit. This will give me both the hexadecimal value that I can use in a style sheet and also the constituent red, green, blue values, the decimal values that I could use in the rgb notation.

Simple Selectors

A selector is what we've been using to tell the browser to apply a background color to the body of a web page. Selectors in CSS are very flexible and very powerful so you'll want to spend some time getting to know them. Again the body selector we've been using is known as a simple type selector, it looks for instances of an element type in the browsers document and it applies the styles inside to that element, so the body selector finds body elements and although you should only find one body element in a page, that selector would apply and find all of them and you can do that with any type, input, divs, paragraphs, links, the style will apply to all of the elements of that type that it finds. You can also use an @ sign to create an ID selector. This selector will find an element with the given ID, in this case it would find the element with the ID of menu. A third selector is the class selector. The class selector allows you to identify elements with a matching class attribute. In this last example we want to italicize the text in any element that has a class=bookTitle attribute. The easiest way to understand selectors is to try them out with some HTML, a style sheet, and a browser. I've changed the sample HTML around just a little bit to include some additional elements, so now we have a couple heading tags, a div tag, a paragraph tag, and also a paragraph inside of a div and we're still applying a very basic style sheet that says find the body and set the background color to #cccc99. But what if we change the style sheet and said, I just want to set the background color on h1 elements? Let me save that and refresh the browser and you can see that we set the background color just on that h1 tag. Now you can stack up multiple selectors for a given rule just by separating them with a comma. Now I'm saying I want to select h1 element and h2 elements, which means both of those get that background color. Notice there's a little bit of white space between h1 and h2, we're going to come back and talk about that in the third module. Now through the style sheet I want to tell the browser to go out and find all the div elements and set the background color to red, green, blue, so these should be all red, and you'll notice that catches both of the div elements. So it captures the div here that's in the middle of the page and it's also coloring the background of this div, which means the paragraph inside of that div has a red background. Something we'll be talking about in the next module is how style rules are inherited and how they cascade and we'll see a quick demonstration of that here if I save the paragraph tags, it should have a background color of and let's make this red, green, blue, which means if I save this we should now seen two green paragraphs. And now what has happened is that this paragraph here has turned green because even though it's inside of a red div, it's the paragraph here that's actually taking up space and making content here and it gets the background green because of this last rule, therefore it renders as green on the page. You can think of that paragraph as having overridden the background given to it by the div, but we'll talk about that more in the next module. Now you will certainly run into scenarios where you do not want to set properties on all the elements of a given type, you want to set properties on specific elements, so I don't want all the paragraph tags to have a green background, only specific paragraph tags. There's a couple ways to do that. For one, I could give paragraphs a specific ID or any element a specific ID. For example, I could say that this is the content paragraph. Now that that ID property is there I can come into the style sheet and say that instead of all paragraphs being green, I just want #content to be green, anything with that ID. And there should only be one element with a given ID, but if we make that change and save everything and refresh, you can see that just that content paragraph is green, the second paragraph reverted to having a red color because it's inside of that red div. Another extremely useful technique is to specify styles on elements that have a specific class attribute. So for instance, I could say that this paragraph has a class of quotation and now inside of the style sheet I can say that anything that has a class of quotation, and I do that by using the dot, so a pound is by ID, a dot is by class, and let's give that a background color of #0000ff. So that should be blue after I save this and refresh and that paragraph that's a quotation is now blue. The neat thing about a class is that you can define it and use it anywhere, but IDs only one element can have any given ID, but classes can be anywhere. So I can say that this div has a class of quotation and save the HTML and now that div is also blue. And again, you might be wondering why did that div come out to be blue when we have a style that says all divs should be red? Again, that's a topic for the next module when we talk about how specifically styles are applied in those situations where more than one style rule could apply.

Other Selectors

Although the simple selectors we've looked at are powerful, the selector syntax allows us to use more complicated selectors and more complex patterns to identify elements. One example is the descendant selector. The selector we are looking at will apply to paragraph elements that are descendants of, or that is anywhere inside of a div element. So the target her is the p tag, not the div tag, and the style will only apply to paragraphs inside of a div, anywhere inside. The syntax is just two simple selectors separated by white space. There is no comma be div and p, otherwise we'd be building a rule to apply to both div and p elements. Another example is the child selector. This selector here is looking for paragraphs that are direct child elements of a div, so not just anywhere inside of a div, but directly beneath a div, its' the greater than symbol that makes this a child selector. You can also select elements based on any attribute value. So we've selected elements by ID with a pound sign and elements by class with a dot, but you can also say, like in this example, find me all the images that have an alt attribute of spacer and then set their padding to 0 pixels. Finally, CSS has the concept of a pseudo class and pseudo classes. These are classes that are not explicitly set on an element, but are inherent based on some characteristic of the element. For example, every anchor tag will have a pseudo class of visited if its href points to a URL that the user has already visited. That's just something that the browser will implicitly do. The anchor tag will have a pseudo class of link if the user has not visited the URL. So what we're doing where is trying to say, take the visited URLs and make their color sort of a light gray that will blend them out and make them fade into the background so that the unvisited links will come to the foreground and jump out with some more color. And by the way, yes, the /\* green text that you see in these examples, that is a CSS comment so you can provide comments inside of your CSS file. We're going to make use of some of these more advanced selectors as we move through the rest of the course. You can also visit the CSS specifications on the w3.org website and from there you can see the list of all the possible selectors that are available as well as examples of how to use them.

Specifying Property Values

One thing that CSS is not short on is the syntax available to specify property values. This is probably because there are so many things that you have to specify, like colors of a background, the size of a font, the width of a table. Some of these property values are set using dedicated keywords, like thin, thick, and larger. You cannot use these keywords just anywhere, only on specific properties where CSS allows them. For example, you can use thin to specify the width of a border, but it wouldn't work when setting a font. When specifying sizes, CSS offers a wide array of measurements. You can use inches, centimeters, millimeters, and points. Point measurements are generally associated with fonts as you've probably learned when using a word processor and one point is 1/72 of an inch. You can also specify measurements using pixels. There are also relative measurements like percentages and ems. Ems are quite popular these days because a relative measurement allows you to scale well and allows your design to adapt to different devices. Color codes we have seen and there are keywords, hexadecimal numbers, and the rgb style that you can use for colors. Fonts have a special case syntax, we'll dig into this syntax more in a later module. And finally, there are a couple examples of functional notation with CSS, not only with rgb colors, but also when you have to specify a URL like for a background image. Let's see an example of some of these in a demo. I'm going to switch over to using Visual Studio to edit my style sheet because Visual Studio knows what can go into a CSS file, it understands the syntax, and it can help me out a little bit if I don't understand what keywords are available. So for instance, if I want to increase the size of the font inside of a paragraph, then I know that I want to change the font size, but I may not know that I can choose from extra large or extra small or extra extra large. So let me just try extra large and refresh the browser and I can see that the text grows a little bit. And because this is a font I can also specify font sizes using point, so I could say that font size should be 28 points, that's going to be a significantly larger increase and I can also specify ems, So an em of 1.0 is the current size of the font for this page, so this should reset it to what it looked like originally, but I could also say that this text should be 1.5 times bigger than what the default size is, so let me refresh this and we've grown about 50%. I might also want to change the border that is around these paragraph tags and again that's something that we'll be talking about a little bit in a later module, but for now what I can do is I can say that the border style is going to be a solid line around this paragraph tag and I can say that the border color is going to be black, later we'll learn a shortcut for some of this, and then the border width. This can be a keyword like thick. So let me try to save this and refresh. There you can see a thick black border, a thick black solid border that's around the paragraph. And border width is something that I could specify using, let's say, pixels. So I want this to be 12 pixels. I could also specify millimeters or centimeters, cm. Let's try 12 millimeters, but this is a little bit too much so let's change it back to 2 millimeters, that seems like a much more reasonable border. I can also set the width of this paragraph tag. One way to do that is using percentages. So I can say that the width of this paragraph tag should be 50%. And the answer might be, well 50% of what? And it's 50% essentially of the parent, that means in this case it's 50% of the width of the window itself. But what if I go in here and I say that all divs should have a width of 50%? Now if I refresh the page, you'll notice that this paragraph is half the size of the paragraph above it. And that's because this paragraph is inside of a div whose width is 50% of the page and its width is 50% of that 50%, so therefore it's about a quarter of the page that it has available to expand its width. If this is not the behavior that I want, I can go in and I could say all paragraph tags, do not use a relative unit of measurement, instead just set your cells to 200 pixels, that way it doesn't matter how big the parent is. I also wanted to show you an example of setting a background image, that is done using the URL function in CSS. Inside of here I could specify the http URL to an image anywhere on the internet. I can also use relative URLs. In this case I have a bg.jpg image in the same directory as the CSS style sheet and that's where the browser will load these images from relative to the location of where the style sheet is unless you have a fully qualified URL inside of there. But by putting this here, I should be able to save and see that background image. Now the text here is a little bit too dark, it's not easy to see on that dark background. There's a couple different ways I could fix that, I could come in to the body and I could say that the default foreground for everything here should be white, but I also wanted to introduce you to the star selector, which is a way of saying everything matches and everything should have a color of white, that would also work. If all of these settings and measurements are a little bit overwhelming, that's when you can lean on the tool a little bit more. So for instance, Visual Studio, when you're building a style you can actually come into the menu, click build style, and inside of here, all of these different properties are categorized. So here are background properties, border properties, position properties, font properties, and there's drop down lists and aids to help you pick things like do I want pixels or ems or points or centimeters or percentages?

Summary

In this introduction we've learned all the basic concepts of CSS. And if there's one important point from the beginning of this module, it was that CSS is about separating content from presentation. Leave only the content in your HTML markup and keep the colors, sizing, and font selection in your CSS files. The separation will make your HTML easier to maintain and also allow your design to evolve easier over time. We've seen how CSS rules are made out of selectors. Selectors find the elements that you want to affect and then the set of property value pairs that contain the settings you want to put on specific elements. In the next module we're going to take a more detailed look at how these style rules are applied to elements by the browser.

Cascading and Inheritance in CSS

Introduction

Hi this is Scott Allen and in this module we'll look at Cascading and Inheritance with CSS. The C in CSS stands for cascading, so this is a very important topic to learn, and this is the module where we will find out how the web browser uses cascading and inheritance rules to set the properties of HTML elements. When working with CSS it's important to understand how these rules will affect the presentation of your content. As an example, let's look at the following three style rules. You might remember some of these selectors from the introductory module that started this course. The first rule says that divs inside of the body tag should have a background color of red. The second rule says all divs should have a black background, and the last rule, you might remember that being the wildcard selector, says that all elements should have a green background. If you put all of these rules in a stylesheet and include the stylesheet in a page that has a div element, will the div has a red, black, or green background? In this module I'll explain how the browser will interpret those seemingly conflicting instructions and show you the tools and techniques you can use to understand the cascade and inheritance.

The Cascade

To understand the cascade is to understand what styles a web browser will apply to HTML elements because styles can come from multiple sources. Let's take the example of a web page called default.html. One set of styles can be applied by the author, that is you, the person who is creating the web page. As we saw in the first module, you can have styles defined inline in the HTML and you can also have styles defined in a .css file that you include into the page with a link tag. You can have multiple link tags inside of that page to include multiple CSS files and there's nothing to prevent you from using a combination of link tags and embedded styles. I'll also show you an example of how a CSS file can import rules using the @import rule. So just the styles that you author can come from a variety of sources, but there is also two sets of styles included implicitly into every web page. One set of implicit rules comes from the users stylesheet. Most modern web browsers will allow a user to create their own stylesheet that the browser will apply to every web page that they view. User stylesheets primarily exist to improve the accessibility of the web. For example, I can define my own user stylesheet to increase the font size of every page I visit and that would make the pages easier to read for me. Or I can define colors that make it easier for me to see the content of a website. I'll show you where you can apply a user stylesheet in Internet Explorer and Chrome. Another set of implicit rules comes from the browser itself. We call this set of rules the default stylesheet, you will also see these styles referred to as the user agent styles because in the CSS specifications a web browser is called a user agent. These default rules tell the browser the default colors and fonts to use when there are no other style rules in effect. The CSS cascade assigns a weight to each style rule and when several rules could apply to a single element, the rule with the greatest weight will take precedence. The user styles have more weight or more importance than the default styles. So the user style can effectively override the default styles defined by the browser. Author styles have more importance than user styles. So the styles that you define in your stylesheets effectively override anything provided by the browser or by the user, although there is an exception I can show you in a demonstration, which is one of the easiest ways to understand the cascade.

Using the Cascade

Let's look at a simple web page. It's basically the same web page that we were looking at in the introductory module, it has a couple headings, a couple divs, a couple paragraphs inside. It also links to a stylesheet called styles.css, but currently styles.css is empty so we're not applying any author styles to this document. And right now the page renders like this and you might look at that and say, well who determines that the background is white and that hyperlink should be underlined and that Heading 1 should be this much bigger than Header 2? Those are all settings that are in place because of the user agent stylesheet, that's the default display that Internet Explorer gives you. If you ever wanted to see what the settings are in a user agent stylesheet, you can generally find those on the internet somewhere. For instance for Internet Explorer there's a website called IECSS.COM and this lists all of the Internet Explorer user agent stylesheets for IE 6 through IE 9. One of the entries you can see here is that an anchor tag, by default, will have the text-decoration of underline. And that is why an anchor tag, aka a link or an href, will have an underline when it displays. Now as I mentioned in the slides, you can also have a user stylesheet that is in effect. In Internet Explorer you can see this by going to Tools, Internet options, clicking on Accessibility, and they'll be a checkbox here, Format documents using my stylesheet and I currently have that set to a user.css file that is in one of my temp directories and that file just so happens to be opened inside of Visual Studio, but we can see it's currently empty. Let's try to do something interesting with it. Let me say that for all the text in the body of this document the font size should be 150%, so I'm going to save user.css and then sometimes to get this to take effect you have to reopen Internet Explorer or come into a new tab. So let me come into a new tab in Internet Explorer and open that up and you can see the text on this tag is now 50% larger than the text in the original tab before I edited that style in user.css. So user.css is at work here. But now let's come back to the page and remember the page is including one of my stylesheets, styles.css, and let me come in and edit this and say that the body here, the font size should really be 80% and now let me refresh and you can see that that is actually 80% of the original font size, so effectively what has happened is my style rule has overridden the style rule that was in user.css because author styles are more important than the styles that are defined in the user stylesheet. So the browser is using my rule, it's overriding what's in that user stylesheet. And I did want to point out that one condition where a style rule in a user stylesheet can have more importance than a style rule defined in a author stylesheet, and let me show you what that is. Inside of the user stylesheet I'm going to put !important here and that means when the browser sees a body style rule coming from the user stylesheet and a body style rule coming from an author stylesheet it's actually going to give more importance to this user style rule because of that !important, so if I save that user stylesheet and I come out and let's open up one more tab to make sure that we pick up the changes from that user stylesheet. You can see that we're back to a large font size. So this style rule has the most importance out of those rules that are defined between the user agent stylesheet, the author stylesheet, and the user stylesheet.

Ordering Rules

For the next demonstration I have cleared out the user stylesheet and I have cleared out my styles.css author stylesheet and the page is back to displaying with its default styles because now that we've learned how the browser ranks style rules with a specific weight or importance depending on the source of that rule, let's also take a look at how the browser treats conflicting rules that are coming from the same source, that is the author. So what I'm going to do inside of styles.css is say that for paragraphs I want to have a background color and let's just use a background color of green, so I can save that and refresh and you can see that the two paragraphs here are now green and let me copy that style rule and paste that here inside of styles.css and just change the color and let's try gray. So now we have a rule that says paragraphs should be green, a rule that says paragraphs should be gray, and the winner is gray because when the browser sees these conflicting rules that come from the same source it just uses the last rule that it sees that would apply to that element. So it's using the last rule here that applies to a paragraph that says things should be gray. If I were to take that rule and place it up here at the top of the stylesheet and refresh you can see we're back to green because it's choosing this rule, it's the last rule. So the order of rules is significant and this also counts when you're using multiple stylesheets, so let me take this gray style rule and cut it out of styles.css and put it in styles2.css and save that. And now we can come back to page.html, I'm going to copy this existing link tag, paste it in here and include not just styles.css, but styles2.css and refresh and you can see that we're back to gray because styles.css said paragraph should be green, but styles2 came in last and said paragraphs should be gray, therefore my paragraphs are gray. This ordering behavior, where effectively the last style rule wins, is also in play when you have imports. So let me take the link to styles2.css out of page.html, which will leave us with green paragraphs and let's come into styles.css and what I'm going to do is add an @import rule that says please go out and fetch styles2.css and effectively place it into the stylesheet, so anything that is in styles2.css will come into styles.css and that's being linked into our page. Let me temporarily delete the style that's here, so effectively all we're including is styles that are in styles2.css. I'm going to save everything and refresh and you can see that we have gray divs as specified by styles2.css, but then if I come back in here and add the style rule inside of styles.css that says paragraphs should have a green background and refresh again, we're going to go back to green. So effectively, again, this rule was the last one that says paragraphs should have this background color, it's green, that came in after the rule that was imported that said paragraphs should have a gray background.

Developer Tools and CSS

As your web pages grow more complex and your cascading stylesheet files grow bigger and bigger, it can sometimes be hard to see which rule is winning and which rule is losing and being overridden by some other rule and that's when developer tools can help you. So Internet Explorer has some built in developer tools, if you just press F12 in Internet Explorer 9, this will bring up the developer tools window and one of the first views that you'll get is an html view where here on the left hand side you'll essentially see the DOM, the Document Object Model, and that is all the elements that have been created in the browser, so there's the html element, the html element has a body element, and this is a tree, the DOM is a tree, the body has a few children, one of those children is the paragraph element that we've specified and here on the right hand side of the developer tools, under the Style tab, you can see that there were two styles targeting that paragraph, trying to set the background color. The first one was from styles2.css, it tried to set the background color to gray, because that has the strike through font on it we know that that rule was overwritten by some later rule and we can see that that later rule was a rule that came from styles.css that says the background color should be green. So these developer tools can be quite invaluable when you're trying to track down why a stylesheet might not be setting something that you expected it to set. Other browsers also have this, let me show you in Chrome if we come out and view this in Chrome, that Chrome also has a set of developer tools. You can access that from the Tools menu, Developer tools, and it's a very similar experience. Over here on the left hand side you see the DOM, the tree of elements. If we find our paragraph and select that, you can see here are the CSS rules that matched that element so the selector said, find a paragraph tag. This one was from styles2.css, since it has that strike through we know that that style didn't apply because some other rule overrode it. And that other rule came from styles.css and it set the background color to green. What's interesting about Chrome is it will also show you the user agent stylesheet styles and the user styles. I don't have any user styles to fine in Chrome, but of course it does provide a default stylesheet, the user agent stylesheet, but in this case it wasn't setting any background colors on a paragraph tag, therefore nothing there was overwritten. So again most every browser has these developer tools that you can either install as an extension or they come out of the box. Firefox, Internet Explorer, Chrome, they all have them, and they can be invaluable when you're trying to track down what style rules actually apply to this element and why does it appear this way.

CSS Reset

A CSS reset is something that you will probably encounter in web design so I wanted to take a moment to let you know what a reset is when you see it. First, let me say that you do not need to use a CSS reset, but many designers will still want to use them, because a CSS reset is a stylesheet with rules that effectively override the default styles given by a web browser for all HTML elements. We've talked about the default user agent stylesheets, those are the stylesheets that a web browser defines to provide the default look for a web page. A CSS reset stylesheet will set the default margins and font sizes so that you always start from a known place when you write your own styles. Because every web browser will have slightly different defaults. Using a CSS reset means it doesn't matter which browser the user is using, you know where you're starting point is. There's a number of open-source and free CSS reset stylesheets available, so let me show you how to use one and what the effect is. First let me demonstrate what a CSS reset style sheet is designed to do. What we're looking at here is our simple web page as it appears inside of the Google Chrome web browser and behind that Internet Explorer 9 and although the content appears to start at exactly the same position, by the time you get to the bottom of the page where that refresh link is, the refresh link is actually just a few pixels off between the two browsers. Now to many web developers and designers those few pixels do not make a difference at all, we're not going to care about them. But for some designs you want to have very precise control over exactly where your content appears and those few pixels can make a difference, even inside of the same web browser between different versions content can appear differently. So the icon here beside the address bar in Internet Explorer 9 is the capability view button. When I click that, we switch from the IE 9 rendering back to the Internet Explorer version 7 rendering and when I click that you can see the content moves around every so subtly because Internet Explorer used a slightly different default stylesheet than Internet Explorer 9 does. If you are extremely particular about your web design, this may be a cause for concern and that is when you can go out and look for a CSS reset stylesheet and there's many available. This is one that comes with the Yahoo user interface library, that's a collection of JavaScript files and CSS files. One of the CSS files is a CSS reset stylesheet that you can download and just include inside of your web pages using a link tag. Another popular CSS reset style sheet comes from Eric Meyer. It's on his website, meyerweb.com, in fact the source code to the stylesheet is right here on this web page at meyerweb.com/eric/tools/css/reset. It just so happens that I have that downloaded and put into a CSS reset.css file on my local drive and I can include that into page.htm, so let me paste this in here and save this and now let's come out and refresh our browsers and you can see that the reset stylesheet removed all the margins, removed all the font sizes, and now when we design our own CSS to style this page we'll really be starting from scratch. It is a lot more work, but some people do want this level of control, and now you'll know what a CSS reset is if you come across one.

Specificity

Now we know that style rules are weighted and the browser considers our styles, the author styles, to be more important than the default styles provided by the browser's default stylesheet. We also learned that the order in which a rule appears is important. I showed you how the last rule wins, however, what we've learned so far still does not explain the sample I showed you at the beginning of this module. Given the three styles that we see on the screen, why would the div appear with a red background? You'd think that the background would be green or black because those rules appear later than the rule that says divs inside the body should have a red background. To answer this question we have to understand specificity. It turns out that the browser gives each style rule a specificity rating, which is a way to quantify the importance of a rule. The higher the specificity number, the more important the rule is. You can think of the specificity number as consisting of three parts, A, B, and C. C is the count of type selectors in the rule, B is the count of class and attribute selectors that are in a rule, and A is the count of ID selectors. This all means that a rule using a wildcard selector would have a rating of zero. A rule using a selector of LI to select list items would have a rating of 1, that 1 is in the C position, and a rule using UL LI to select list items in an unordered list, that would be more important because it would have a rank of 2, there are 2 type selectors in that rule. The LI.red rule is selecting list items with a class attribute of red and then the last selector here, the one with the highest ranking, is the one selecting an element with the ID of content. That puts a 1 in the A position, that means the rating is 100. Remember, rules that have a higher number here will win out over rules that target the same element but have a lower number. Now before I give you a demonstration based on this concept, let's see why our div is red. It's because the wildcard selector in this code on the screen has no specificity rating, it will try to set everything to have a green background, but it's such a low priority rule it probably will not have much effect. It's literally not a very specific rule. The rule selecting divs and setting the background color to black is more specific and has a higher rating, however, the rule setting the div elements appearing underneath the body tag is even more specific and has a higher specificity rating and thus it wins and sets the divs background to red. Of course that div would have to be an immediate child of the body element and not just any descendant because that's what that first selector rule is looking for, but let's look at some examples in a demo.

Specificity in Action

To get a better idea about specificity, let's go into our simple web page and set some styles inside of styles.css. First I'll go inside of here, and it's currently empty, and say that all paragraphs should have a background color of green and if I save that, you can see that the background color on both of the paragraphs that are inside of the simple web page, they both turn green. However, now let's target this paragraph, which is a child element to this div and if I write a rule that says please set the background color of paragraphs that are inside of a div to something different, let's pick gray, then that rule will set this paragraph that is inside of a div to have a background color of gray because this rule that has two type selectors in the rule is more specific than the rule that just says set all paragraph backgrounds to green. And I could even make a more specific rule that targets this paragraph by ID. So I could say, when you see something that has an ID of content, then please set the background color to fuchsia. And it doesn't matter the order that these rules appear in, they are all different rules. This one is selecting paragraphs inside of a div, this one is just selecting paragraphs, and this rule is more important than that rule that is just selecting paragraphs. This rule that is selecting by ID is more important still. And if you think about it, this is the way that you want CSS to work. When you make a more specific rule you're probably very specific elements that are on your page, you want that rule to work against those specific elements, but when you're making a general rule, like this one that is setting the background color to green, that's not as specific, that's generally trying to set the background color to green, but there's always going to exceptions if that paragraph has an ID of content or if that paragraph is inside of a div. Now there's one more catch to specificity that I should probably point out and that is a rule that is embedded in a style attribute is the most specific. So if I come in here and I set the background color to black, that style rule is actually going to win out over any style rules that are inside of styles.css, so indeed that paragraph turns black. So let me take this inline style out once so that we can go back to seeing the actual text that's inside of here, and let me point out something else about these rules. When we talk about a rule winning, we're not saying that this rule completely replaces all of the settings that were in the lesser rules. In other words, if I come into the rule that is selecting all paragraph tags, and say that the font size for all paragraphs should be 2em, which would make the font quite a bit larger, you can see that applies to both paragraphs, even the paragraph that is inside of a div because there is no other rule, more important rule that is trying to manipulate the font size, and indeed if we come into the Developer Tools, if I hit F12, we can go to that particular paragraph tag that is inside of a div and you can see, here are the property values that were actually set on that paragraph and where they came from. We set the font size using a rule that was selecting all paragraphs. We also tried to set the background color, but that particular property setter was overwritten by a more important rule, the one that was trying to select paragraphs inside of a div and set the background color to gray and that particular rule was also overwritten by a more important rule that was setting the background color to fuchsia for anything with an ID of content. The styles that appear in these developer tools for Internet Explorer will be ordered according to their specificity and that is why the #content appears at the bottom, but you can see that multiple different rules can still have an impact on our elements even when they're not the most important rule that could apply to that element. This rule was the only one that was trying to set the font size, therefore that particular property set made it through.

Inheritance

There is one more important aspect of CSS to understand if you're trying to come to grips with why a particular element appears in a certain fashion and this aspect is inheritance. Some property values of an element will be inherited from the elements parent. I've already talked about such and such an element being a child or being inside of some other element, and this parent child relationship goes back to the DOM, the Document Object Model, which if we visualize a web page, would look very much like a tree. On our simple web page we have a paragraph that is a child element of the body tag and we also have a paragraph that is a child to a div tag. The div tag is the paragraph's parent because the P tag is nested inside of the div tag. There are certain properties that we could set on that parent div or on the parent body that will flow through and be inherited by all of the children underneath that element. One example is the font size property and let me show you in a demo why this makes sense.

Inheritance in Action

To see inheritance at work let's go into our web page and modify the styles for our web page to say that divs should have a font size of, and let's make it very small so make this noticeable, 8 pixels. I'll save that and refresh the page and you'll notice that the text inside of this div is now 8 pixels and the text that is inside of a paragraph that is inside of a div is also 8 pixels and that's because the paragraph inherited this font size from its parent div. We can verify that by going into the Developer Tools in Internet Explorer or in Chrome. In Chrome I'll hit F12, come down to this paragraph, and you can see inherited from div is a font size of 8 pixels. Now of course you can override inherited values, in this case I can go and I can explicitly say that paragraphs should have a font size of 22 pixels, this will make this very noticeable. And you'll notice that the text inside of a div is 8 pixels, but the text inside of our paragraphs, even the ones that are inside of a div, is now 22 pixels and in fact you can see where font size of 8 pixels that is inherited from div, that now displays in a strike through font, which means it was overridden by some other rule, in this case the matched CSS rule that says paragraphs have a font size of 22 pixels. If you think about it, then again this inheritance for font size makes sense. Imagine that inside of this div we had some additional markup, like a word that was inside of a strong tag and a word that's inside of an emphasis. I'll add that to both the div and also here at the end of this paragraph and save this and refresh, and you'll notice that those individual elements, emphasis and strong, inherited the font size settings of the parent container that they were inside of. And this is a good thing because this is probably the way that you want the text to display, just because you're emphasizing a word doesn't mean that you want it to jump back and use some different font size, you want it to fit with the rest of the text that is inside of a div or inside of a paragraph. And I want to be able to do that without writing individual style rules that say okay the emphasis inside of a paragraph should be 22 pixels, but the emphasis inside of a div should be 8 pixels, everything is just inherited and is just going to work for me. So let's look here and see that emphasis inherits a font size of 22 pixels from being inside of a paragraph. What will make this slightly confusing is that not every property value is inherited. Font size is an example of a property value that can be inherited from a parent, but a border is an example of something that cannot be or is not inherited from a parent by default. And over time you'll learn what property values are inheritable and what property values are not. And typically, again, these settings make sense. So for example, let's save the stylesheet and see that I now have a border around my 22 pixel font inside of a paragraph, but notice that the emphasis and the strong words did not inherit that border. What we don't want to do is draw a border inside of the border to have a border around this emphasis that's inside of the border that's part of the paragraph. That particular value is not inherited. What you'll find as you work more and more with CSS is that font related properties are typically inheritable, so font size, font family, also listed related styles are inheritable, so things like list style type and list style, those will inherit. When it comes to margins and padding and border, those values typically do not get inherited by child elements.

Summary

In this module we learned about three important and core concepts to working with cascading stylesheets. We saw how the cascade works to assign an importance to style rules coming from different sources, like the default browser style rules, the user style rules, and the author rules. We also learned how specificity assigns a weight to individual rules depending on how specific those rules are and the rule with the highest specificity will always win when setting a particular property value. Finally, we looked at inheritance and how some property values, like font size, will flow down the document tree, that is those values will be inherited by child elements and the children of those child elements all the way down unless they are overridden. All of the rules about how CSS works internally will typically make sense with how you want to work as a designer. Specific rules should override general rules and property values should be inherited when they make sense and allow you to write fewer rules and less code.

CSS and the Box Model

Introduction

Hi this is Scott Allen and in this module I want to demonstrate the box model of CSS and HTML. The box model is important to understand because every HTML element on your web page will produce a box. You cannot always see the size and dimensions of these boxes, but divs, paragraphs, anchors, headers, lists, and the rest of the elements are all producing boxes and quite often you'll have boxes inside of boxes. Understanding how to size and arrange these boxes is a large step towards understanding web design with CSS.

The Big Three

A div, like every visual HTML element, will produce a box and there are three key box properties you can influence with CSS. First there is the border around the box. Most borders are not visible by default, but you can control the width, style, and color of this border. Secondly is the padding. Padding adds space between the border of a box and the content of a box. In this case, the content is the word div and we have quite a bit of padding that is pushing the content inwards away from the border. Lastly, there is margin. Margins are the space between a box and any adjacent elements. While padding pushes content inward, margins push boxes away from other boxes. Before I show you how margins work in a demonstration, I want to point out that boxes have four sides. I know that's not a ground breaking discovery, you probably knew that before you started watching this video, but in the context of CSS it's important because you can set the border, margin, and padding settings for a box differently for each of the four sides of a box or you can set them all to the same values. Let me show you how this works.

Border, Margin, Padding

As a demonstration for this module, I want to show you some of the things you can do with CSS to make a collection of list items appear dramatically different so it's not just a collection of list items, but appears a bulleted list as they do now. Because right now we have a style sheet included, but the styles are empty and before we get to anything fancy, I want to start off and give us an understanding of border, margin, and padding. Let's first go in and add some background color to our unordered lists so that we can precisely see the box that is being drawn by that unordered list. I'll set this to a light gray color and refresh the browser and there we can see the box. And let me go ahead and zoom in so that we can see some of the changes we're about to make. And so that we can compare the box represented by this UL element and the box represented by one of the list item elements, let me go in and use some CSS to go in and say find me the list item inside of a UL, but find me a very specific list item. What I'm going to do is look for the nth child, the 3rd nth child. We haven't covered this syntax in this module, but this is literally going out and saying count the number of children inside of a UL element and find me not the first one, not the second one, but the third one, and what we'll do here is we'll say set the background color to a slightly different color. The last two hexadecimal digits represent the amount of blue so it's going to be slightly less blue and we get sort of a goldish brown color there. And now that we can clearly see the dimensions of the box drawn by the list item, let's go in and set the padding to that element to be 10 pixels. Remember padding pushes content inwards, away from the outer borders. So when I refresh the browser, you'll see that Blueprint now appears at least 10 pixels in from the left border, 10 pixels down from the top border, and 10 pixels up from the bottom border. It does extend all the way to the right, we'll talk about that in a bit, but this is the effect that padding would have. Now you'll notice that the box doesn't extend fully to the right, it doesn't reach the window of the browser. We will have some white space here on the right and the top and the left. Where's that white space coming from? It comes from some default padding and margin rules that are inherited from the user agent stylesheet. We talked about that in the last module and it's one of the reasons that some people will use a CSS reset or define a style rule that says the default padding for all of my elements should be 0 pixels and the default margin for all of my elements should be 0 pixels and now when I refresh you can see that our box drawn by the unordered list is now flush against all the borders of the window. Or at least the left, the top, and the right borders. The box is only drawn to a height that will fully encapsulate all of the content that's inside. Now let's add a margin of 10 pixels. This margin is going to push that list item away from the adjacent list items, the one above it, the one below it, and it's also going to push it away from the left side of the window. In fact you can see that without that margin we're not even seeing the little dot that typically appears in a bulleted list. Let me increase the margin just a little bit more to make sure that dot fully appears and I'll show you the proper CSS approach to removing that, that is the set the list style type and from here you can actually pick what sort of icon you want to appear. What I'm going to pick is none so that when we refresh the browser that dot doesn't even appear. There are no bullets associated with this list anymore. Many times you'll see this abbreviated to just list-style: none, that will have the same effect. And so that we can more clearly see this effect, let's remove this nth child selector and just apply the background color, padding, and margin to all of the list items and let's also draw a border around each of them. So let me say that the border-style is going to be a solid line. Notice you can also pick from a double line, a dotted line, a ridged line. Solid is one of the more popular selections though. I'm going to set the border color, the color of that line to be black. And I'm also going to set the border width in pixels so that we'll have a 2 pixel border around each of these. And now it should be very obvious that the space inside of that border is the padding setting, the space outside of that between the elements is the margin, and we're already starting to see that we've transformed this list into something that doesn't quite look like a bulleted list anymore.

Top, Right, Bottom, Left

As I mentioned earlier, every box has four sides and the border margin and padding syntax in CSS allows you to set properties for all four sides equally or independently. The first line in this rule on the screen is only setting the padding on the left side of all LI elements. The second line in the rule is setting the margin on all four sides of the LI element and it does so by specifying the margin at the top and then working clockwise around the box, so we have top, right, bottom, left. This is a shortcut syntax you can use instead of setting one value with margin-top and one value with margin-right, and so forth. The border specification also has a syntax you can use to set the appearance of a border using a single line. It's the width, style, color. So the third line in this rule will give us a solid black border with a width of 3 pixels. And notice this is only the border bottom, so you can set the border appearance on the four sides independently also. The side of the box that you target with all of these rules will always be one of top, right, bottom, or left. Let's use this syntax on our demo page. In the demo page let's make a few changes and the first thing I'm going to do is take the background color that's currently assigned to the UL element and assign it as the background color to the body so that we have a nice consistent light gray background throughout the page and that's easier on the eyes. And now let me remove the margin and padding so that we can start to work with them. One of the great things about margin and padding is how you can use them to precisely position some of your elements just by adding a few pixels here and there. Without padding, without margin everything just goes to the top of the window, it's flush against the sides of the window. I do want to add a little bit of space back in there, but not as much as we had, so let's just have a margin of 5 pixels on all four sides and that pushes the elements a little bit away from the sides and a little bit away from each other. And with the padding I want to do something a little bit different. I want some more padding on the left side of the box to push that content away from the border, but I'm not so concerned about the other sides. From the top let's just say 3 pixels, the right and bottom I don't care about so I'll just say 0 and 0, but from the left let's use a 10 pixel padding on the left to push those elements further away from the left hand side of the box. And now let's also try some things with order. So I'm going to use Visual Studio to type into these bottom three lines and make them all to apply only to the order bottom, which gives us just that thin black line along the bottom, but I know now that I can shortcut the style and I can just say border-bottom is going to be a 3 pixel solid black line, so you can specify that as a keyword or rgb values as we talked about earlier in this course, and when I refresh I have that solid black line at the bottom of the elements. And I want to do something a little bit different for the left border, so what we can say is border-left is going to be a 10 pixel solid, it looks a little more maroon. And now what we have is a bit of an interesting effect where the border is actually going to call attention to these four list items.

Vertical Margins Collapse

One of the subtle aspects of the demo we've been looking at so far is the space between list items. In the last modification that we made to the stylesheet we specified a margin of 5 pixels around all four sides of a list item. This ultimately meant that the top list item was at least 5 pixels away from the top of the page, but notice that a list item in the middle is also just 5 pixels away from the top list item. If the top item has a bottom margin of 5 pixels and that middle item has a top margin of 5 pixels, why isn't the space between the list items adding up to a full 10 pixels, 5 from top, 5 from the bottom? The answer is that in HTML when top and bottom margins meet they will overlap or collapse. Technically they will overlap until one of the margins touches the border of the other element, meaning you'll always have a margin that is the maximum amount of the bottom and top margins that are meeting. In this case both the bottom and the top between a list item is 5 pixels so we have a 5 pixel margin. This behavior only applies to vertical margins. When we stack elements next to each other horizontally later in the course we'll see that horizontal margins do not collapse or overlap. A collapsing vertical margin makes a lot of sense if you think about it because the margin for our list items will make sure that we push away from the top and the left of our window, but when we have a list item in between two other list items, we still want that same consistent 5 pixel margin. We don't want both margins to add up to 10 pixels and push these further apart than they are from the left and the top, so vertical margins collapse and we like that behavior. The largest margin between these two elements will set the space between them.

Width

So far the boxes we've been creating with list items have extended to the far right of the screen, which is how most elements want to behave unless you constrain them with the width. You can set the width of an element inside of a rule, it's a simple property value, and like most measurements you can specify pixels, millimeters, inches, or relative width like 50%. Keep in mind that when you specify an exact width, like 250 pixels, you are specifying the width of the content area. Any border, margin, or padding that you add will add to the width and require more space. On the box we're looking at, if we say the margins are 5 pixels on all 4 sides, and we've set the content to be 250 pixels in width, then the total size of that box that is rendered is going to be 260 pixels because we add the 5 pixel margin on both the left and right sides. In the demo application let's see what happens when we set the width of our list items to be 180 pixels. You can see that shortens them up quite a bit, and as I mentioned earlier, you can also set this to a percentage, so I could say that they should be only 40% of the page. Well let's go back and let's set this to about 150 pixels and that's just enough space to fit everything inside of here and then I could also go in and say the text alignment, all the text inside of these, should be centered. We'll talk more about text and typography in the next module. And now imagine that I wanted to create an image that would align perfectly with the box that we've created. How can I calculate the total width of the box? Well I'd have to take the width of the list element content, which is 150 pixels, I'd have to remember that there's a left border of 10 pixels, that puts us at 160. There's also a 5 pixel margin on the left and the right, that adds another 10 pixels. Sometimes doing these computations can be a little bit tedious. Fortunately the developer tools that I showed you in an earlier module that are available in Chrome and Firefox and Internet Explorer, they will have a place where you can see, for any given element, what sort of layout or metrics it has. Here inside of Chrome I can see that a list item content is 150 pixels, there's 10 pixels of padding, 10 pixels of border, and 5 pixels of margin on the left. There are also 5 pixels of margin on the right and I could add all those numbers up to get the total width of that box.

Display and Visibility

Every HTML element has a display and visibility property. The display property is generally set to block, inline, or none. Block elements always sit on top of each other by default. Elements that have a block behavior by default are elements like div, P, and LI. These elements always move down to a new line when they display. Inline elements are elements like span, they don't create more vertical space unless they're forced to. I'll show you an example of this in just a moment. Visibility on the other hand, this is another property you'll come across quite often in web programming, the LI elements we've been working with are visible by default, but let me show you what happens when you set visibility to hidden. Let's go back to using the nth child trick to manipulate some display and visibility properties of the third element that's in our list, the one that says Blueprint. First what I'm going to do is say display: none, to see what impact this has. And refresh the page and you can see that that particular list item is just removed from the page, not only does it not render, but there's also no space there to render it in. So both the content and the space it had reserved, the box had reserved, are both gone. And that's a little bit different than saying, visibility: hidden. In this case, the list item will not display, but the space is left behind for it to appear. Visibility: hidden is quite often used in combination with JavaScript to make an element appear and then go away, but without impacting the layout of the page, it doesn't make the whole page bounce up and down because the space is still there to display it when it comes back on. Let's remove the nth child bit and turn our attention to all list items. What I'm going to say now is display, and you can see that I can choose from block and inline, I'm going to set the display inline and refresh the browser and now all of our list items are back and displaying and there's a couple things to notice, first of all, they didn't create themselves on new lines, in other words, they display inline and will stretch across the page until you run out of space and then they'll bounce down to the next line. Now let me go in and set the zoom out a little bit so this is a little easier to see and of course the second thing to notice is that the width setting is no longer in effect. We have some elements displaying that are obviously less than 150 pixels and this is because you cannot set the width of an inline element. They are going to expand so that all their content fits inside in the padding and the border, but that's it. If we wanted these elements to be inline, but still have block characteristics like being able to set the width, that's another display setting, which is inline-block. So now we can have the width and we can also have these list items arrange themselves horizontally as long as there is enough space to display them all on one line. And hopefully this gives you an idea of the flexibility you can achieve in your presentation just using CSS and markup.

Styling a menu

Before we leave this module, let's go ahead and complete what could be a top level navigation widget, that is almost like a menu control that's sitting up here with all of the different URLs to different areas inside of a web application or a website that I've built. There's just a few things that I want to clean up to make it look a little more like a menu and behave a little more like a menu. For instance, I like the black border on the bottom of these entries, but I don't like anymore the red border on the left hand side of these entries, so let's get rid of that. And also it's a little bit ugly to have both the black border on the bottom of a list item and also an underline under the hyperlink. That underline comes from the user agent stylesheet, most links are underlined by default. If I wanted to get rid of that what I could say is that the anchor tags inside of a list item should have a text-decoration of none as opposed to underline, which would be the default for an anchor tag. Now if this were really inside of a web page there's a couple other structural changes I would probably make too. For instance, I wouldn't want all list items to display in this fashion, I'm probably targeting specific list items that have some navigational content and that's why I'd probably use a class here to indicate something like this particular list is actually a menu and then instead of selecting all unordered lists, or UL tags, I can go out and say, I'm looking to modify things that have a class of menu or list items inside of a thing that have a class of menu. And we could also do that here, say anchors inside of list items inside of a element with a class of menu. And while we're here let's also change the color here just to be black, which should make it a little easier to see and we could make this flush with the top and the left by getting rid of the margin, but we could also align everything that's inside of here to the center of the browser window using a text align. And finally for one last effect, let's go in and select list items that the mouse is currently hovering over. This is one of those pseudo classes that I told you about in the first module, it's not a specific class name that you attach to an element, it's something that exists because of where the element is positioned or how the user is interacting with that element at any particular time, so when someone hovers the mouse over an element, what I'd like to do is set the background color to this color and now I should be able to refresh and you can see that we get a slight effect whenever the mouse moves over one of these entries. I will caution you that this particular CSS doesn't work in old versions of Internet Explorer. It does work in Internet Explorer 9, but if you need to make this same effect work in something like Internet Explorer 6, you'll probably need to fall back to JavaScript.

Summary

In this module we looked at the box model of CSS and HTML and now we know every element is inside of a box. Using CSS you can control the width of the box as well as the margin, padding, and border around all four sides. We've used some of the basics that we've learned about CSS so far to take a bulleted list and transform it into a navigational control that renders horizontally across the top of the browser window.

Styling Text with CSS

Introduction

Hi this is Scott Allen and this module is going to cover fonts and texts in CSS. It's inside of this module where you will find out how to specify the font that you want to use for the textual content of your site, like a monospaced font or a font with a serif typeface and also how to set the font properties to make that text italicized or bold. In addition, we'll also be looking at text properties including the ability to indent and align texts and we'll be styling a page of textual content as a demonstration.

Font Collections

In CSS you can select the font you want to use for the text inside of a particular element by setting the font-family property in a style rule, but before we get into the details of font family, we'll have to understand how to use font names because when we create a web page we don't always know what our users will use to view that web page. It might be a PC, a smartphone, even a refrigerator and we cannot be sure that all of the devices that users will have will have the same set of fonts available because specific fonts are not defined by web standards and the web browser is going to rely on the user's system, like Windows or OSX, to provide them with the fonts that we request. What the CSS standards do define is a collection of five font categories. The first of these categories is the categories of serif fonts. An example of a specific font that would fit in this category would be Times or Times New Roman as well as Baskerville, Century, and Schoolbook. Those are specific font names, like Baskerville, Century, and Schoolbook, those are the font names that you might see in a dropdown list when you're selecting a font inside of a word processor because the word processor knows exactly what is installed on your machine. But to CSS all these fonts would fall under the category of serif fonts. There is also a set of sans-serif fonts and a popular font that falls into this category is Arial. In case you didn't know, serif, so there are small lines at the end of a letter, they make the letters look a little bit fancy and embellished. You can see the difference in the R letters that are on the screen now. The R on the left is an Arial, which is sans-serif, literally without the embellishments, while the R on the right is Times New Roman and it contains a few extra little strokes here and there. Generally speaking, people find the sans-serif fonts easier to read on an LCD display. There are also categories for cursive fonts, which look a bit like handwritten letters and fantasy fonts, which can look like anything, but most designs are not going to make heavy use of these fonts because they can be hard to read and also a little bit unpredictable, we're not exactly sure how they're going to display on the end user's system. Finally, there is a category for monospaced fonts. These are the fonts that give equal spacing to every letter and that's probably the font that you want to use in your text editor that you use to create CSS and also the font that you would want to use on a web page to display code in a web page. Let's take a look at these fonts in a browser.

Using font-family

Here inside of Visual Studio I have all of the markup that is required to display that page that is on the right hand side of the screen and currently there are no styles defined except for two different background colors and what I'd like to do throughout the rest of this module is use some of the things that we're learning about fonts to improve the display of this page. Right now it feels like the header is a little bit too large, it feels like the code that is in the middle of that page is hard to read because it's not formatted the way I expect. Those are the types of changes we're going to be making, but before we do that, I want you to take a close look at that web page and tell me what category of font is being used. I think it's a serif font. It's the default font actually that both Chrome and Internet Explorer would use to display this web page unless I style it differently. And remember that I said that generally speaking, a sans-serif font is going to be easier to read, so that's one of the first changes we're going to make, however, that is a subject for debate. What you could do is go around to some of your favorite sites and see what type of font they're using and come to a decision about what you think is the cleanest. I think what's interesting is that if you look at sites like Twitter, they're obviously using a sans-serif font. If you look at a site like Google, the search results and the text on the side, that's all sans-serif, but when you go to a website that behind it is a company that is used to print media, for instance Smithsonian.com, Smithsonian.com is the website for the Smithsonian magazine, they are using a font that would fall in the serif category. Also the Washington Post, which is a newspaper, they're using a font that would match what they use in print. Serif fonts are very popular in print. I do think this might be something particular to news articles, what's interesting is if you go to a site like Yahoo, then a lot of the text that you see throughout their site is a sans-serif font. But when you drill down into a news article on the yahoo site suddenly they changed over into a serif font. Let's take a look at what happens on our page if we change the font here. So I'm going to go in and say that the font-family and I'll pick sans-serif and you can see the different feel that that gives to the web page. Let's also try monospace. The code looks better, but everything else looks worse. Let's also try fantasy. That's very difficult to read. And let's try cursive. And I think, given those options, let's go back to this setting for right now.

Font Selection

Many CSS designs will set the font family property to the name of a specific font like Arial. However, there's always a chance that the specific font you want, like Arial, that it's not available on a given user's system. And that's one of the reasons you can specify as many fonts as you'd like in a comma delimited list and the browser will use the first one that matches. In this sample on the screen we have a 1st choice, a 2nd choice and if neither of those fonts are available we are falling back to the generic sans-serif because even if the browser doesn't have any of the other fonts it must provide a default font for that category. Of course this raises the question of what fonts are safe to use for web design. Generally speaking, you can't go wrong with Arial, Verdana, Times, and Courier, but I've also provided a list here of other fonts that are commonly available on multiple platforms. Helvetica is a common font that you're see listed in many stylesheets, but Helvetica is not available on Windows. Windows provides Arial instead. Let's take a look at the font settings on these websites that we visited earlier. I'll start with Twitter. Inside of Chrome what I can do is select an area like this and right click and say Inspect element. That'll bring up the Developer tools that we look at earlier in this course and that will tell me what font-family is in effect. And it's down here at the bottom for Twitter, the font is Lucida Grande or sans-serif depending on what is available and we'll talk about what the .75 em in just a bit. Let's also look at Google. Its search results are displaying with a font-family of Arial or basically then any sans-serif font that's available. Let's take a look at the Smithsonian magazine. Its font is Georgia or Times New Roman or any serif font. You can see that Times New Roman when you have spaces in the font-family name, you can enclose that in quotes. And finally let's look at the Yahoo News, we'll inspect this element, and we can see here the font-family is Georgia or Times or Times New Roman or then just a font from the serif category. Something else that you might have picked up is that we talked about inherited properties earlier in this course and font-family is one of those properties that is inherited. So if I set it at the body level, then the paragraphs and the divs and the anchors that are inside of that body will pick up that font-family by default unless I override it. So for instance, this header here, this h1 tag, it is inheriting the font-family from the body, but it doesn't have to be that way. I can say that for the h1 the font-family should be Times New Roman, Serif and refresh the page and you can see that it has a different style than the rest of the web page. And what does happen when you specify a font that doesn't exist? And let me make sure that I put Times New Roman inside of a quote here, and we'll ask for a fontthatdoesntexist, you can see that's not going to produce an error on the page, the browser is simply going to move on to the next font and try to pick it up.

Font Styling

Another characteristic of fonts that you'll want to control is the size and when setting the size you'll first have to pick between absolute or relative units. Absolute sizes are commonly specified in pixels and pixels give you very precise control over the font size. Relative units, like percentages or the keywords larger or smaller or ems, they allow you set a font size based on some baseline, so 2em would be twice the size of the baseline and 0.8em would be 80% of the baseline. Many people today use ems for relative font sizing. One reason is relative sizes allow a user to scale up the text using their browser and this is something they might want to do because the text is hard for them to see. Relative sizes allow that to work. Some of the other properties of a font that you can set are the font style to force an italic font or the font weight to make a font bold. And the font variant property will bring in a typeface that uses only capital letters. Let's use what we've learned so far to improve the look of this page. The first thing I want to do is deemphasize that h1 element a little bit. It needs to be an h1 element because it is the primary header for this page, but it's a little bit too large for my taste. So let me set the font size to 1.2ems. That will bring that down a little bit because typically an h1 tag is going to be well beyond 1.2ems. The next issue I want to address is the code listing here. I want this to be a monospace font and there's a couple different wants to achieve this, for code there is something specific that I would do, but before I get there, let me show you this approach, which is I have a class on this paragraph element, class=code and that would allow me to come into styles.css and say code should have a font-family of monospace and let's make the font size 0.9ems just to reduce it a little bit and now it's looking a little more like code, but notice that versionInfo =, that particular block of text should really be on the next line. And in fact if we look at the source code here we can see that versionInfo = actually is on a different line than the line that ends here with a semicolon. But web browsers generally don't respect things like tabs and new lines and carriage returns inside of HTML code. What they're trying to do is take that text and get as much of it on one line as possible. What I really want to do here is I want to use a pretag, which means this is preformatted text and please don't try to manipulate it. I could still use a class of code, but let's just leave it just as a pretag to see how that looks and you can see that it gets indented and it follows the formatting that I have here precisely, in fact it doesn't even break that second line even if there isn't enough room to display it, so the user is going to have to scroll or open their browser wide enough to be able to see that entire line of code. And most user stylesheets will style a pre-element with a monospace font, so that's been taken care of for us. I also just want to come into the content area and add some padding because it's been kind of bugging me that everything is right up against the border. We know from the last module that padding will push things inward away from the border. And let's also come in and style the ordered list that appears here. What I could say is that for an ordered list I want a font of and just specifying font allows you to shortcut setting font-family, font-size, font-style. What I could say is I want this to be an italic 0.9em using a serif font, and you can see how that changed. I don't actually want to override this piece of it, so another way to write this would be to say font-style is italic, font-size is 0.9em and now it's still going to inherit the font-family setting from the body area. And lastly there is our copyright notice, which should be in a very small font. So let's come in and put a class of copyright on that particular div and I can come into my style rules and say that for a copyright the font weight should be bold or bolder, let's just go with bold, and the font size 0.8em.

Text Properties

Finally, there are properties you can set to influence the look and layout of textual content regardless of what font you use. These are the text properties. They allow you to indent the leading text of a paragraph, add spacing between letters and words, add underlines or strikethroughs across characters, and perform some alignment and transformations. Let's use some of these properties inside of our application. Inside of our page let's try out a few of these text properties. I'll start by demonstrating text indent, we'll set it here inside of the content area to a fairly dramatically large value of 3ems just so we can see what it's doing and what it's doing is it's pushing over the first line of text inside of any given element. So the first line of text inside of a paragraph, but also unfortunately the first line of text inside of our code and also inside of our list. So let me remove that for right now. So the page will go back to looking a little bit better and the next thing we'll play with is letter-spacing. Now this again is one of those things that you can use for special effects sometimes. So this is going to be on the header element. When I refresh you can see that all of the letters have been pushed apart. You can also specify word-spacing, which is the space between words and you can see that using and FileVersionInfo are now much further apart than they were by default. There is also text decoration, so you can do things like underline, that underlines our header. You can also have multiple settings inside of here, so in addition to underline I could also specify a line through the text and that gives me that strikeout look or we could go with just line through or we could remove the text decoration altogether. And then there's text align, we've used this before. What I want to do is center that h1 tag so it's in the middle of the page even as the user adjusts the width of the browser and you can also use text align to do things like move things to the right or to justify text so that the right edge of the text is always as close to the margin as possible. If I refresh you can see how it spaced some of those sentences out, but some sentences are just too short to be able to stretch out the whole way by adding an occasional space here and there. And finally, another text property that you'll commonly see inside of stylesheets, some people will set the line-height and what I'll do is set this to 1.2ems inside of our content and let's try 2.2ems, you can see how those lines move apart. And again that's something that you can use for an effect on the occasional web page.

Summary

In this module we've come to understand how to specify fonts using CSS and the difference between a specific font like Arial and the generic categories of web fonts like serif and sans-serif. We also talked about font sizing and the pros and cons of relative sizes versus absolute sizes and finally we used the font properties and the text properties to style the content of a web page.

Layout with CSS

Introduction

Hi this is Scott Allen and this module is going to show you how to perform page layouts with CSS. Layout design is all about positioning content so that your users and customers can find the information they need and navigate around the site. We'll be looking at the different positioning options in CSS, like relative, static, absolute, and fixed, and we're also going to look at floating content to the left and right and building two and three column layouts, which are quite popular on the web today.

Position

The first layout property we'll look at is the position property. You can position any HTML element with one of four values. The default position value is static. Static positioning stacks up elements one after the other down the page. Since this is the default, this is what we've been using throughout the course. Each paragraph or div we create will appear underneath the previous div or paragraph. That's what we call the flow of a document, elements just flow downward. Also keep in mind the difference between block elements and inline elements. We talked about that in a previous module in the box model module and remember inline elements, like a span, just appear in a line, they don't create a new line. So when I say static positioning stacks elements down the page, then I'm talking about block level elements like div, h1, p and so fourth. Relative positioning, on the other hand, moves an element from its default position in some direction. So for example, if I have a div and I set its position relative, then I can set values for the top and left property to move a relative div a number of pixels away. One thing we'll see in our demo is that once we start to position elements, we have to make sure they have space because you can position an element on top of another element. Sometimes that's the effect you want, but if not you'll have to mind the space. Two other positioning values are absolute and fixed. These values remove an element from the flow of a document, from that flow we talked about earlier. Absolute positioning moves an element to a specific position relative to the body of the document, meaning you can take an element and say, I want this to appear 20 pixels from the top and 50 pixels from the left and regardless of where this element appears in the markup, it's going to move to that position inside of the page. Fixed positioning is very similar, except the positioning is relative to the window itself, so if I want something with a fixed position, 20 pixels from the left and 20 pixels from the top, it is going to stay at that position even if the user scrolls the window down, that element will always be 20 pixels from the top. Let's see the position values in action.

Relative Positioning

Inside of Visual Studio I have some markup in place that renders the page we're looking at here on the right hand side of the screen. There's a fair amount of content in here because I want to use this content to manipulate the layout and positioning of elements. The only styles that are currently applied are styles that will set the margin around these three buttons that we see and a style that will set the width of each of those buttons to be exactly 120 pixels. Since inputs are inline elements, we can see those three buttons stacking up against each other and moving out horizontally. Their position is static, so they're following the normal flow of the document and let's change that up a bit. I want to change this actually just for the last button, so I'm going to say buttons input:last-child and what we'll do to the last child inside that series of buttons, button #3, is we'll set, first of all let's set the position to be relative, and if that's all I do it doesn't make any changes on the page, but now that I've set it relative, what I can start doing is start manipulating the top and left properties or the bottom and right properties. So I can say top 20 pixels, left 50 pixels. And if you watch this button, what it's going to do is move down an additional 20 pixels and left an additional 50 pixels from where it should have been. So it's moving relative to where it should have been in the normal flow of the document. You can also specify bottom and right. So if I do that, we move over from the right by 50 pixels and up from the bottom by 20 pixels. But now if I change this back to the way it was, I also want to point out that you can use negative units here, so I can say, move up 20 pixels and left 50 pixels, same effect. And in fact if I move to the left oh about 185 pixels then I can position this button almost in the middle between button 1 and button 2. Now you might be looking at this and saying, isn't this sort of similar to margin and padding? Aren't you moving things around by pixels and can't you do the same thing with margin and padding? And the answer is yes, in many cases if all you want to do is move something to the left just a few pixels, you could do that with a margin, you could do that with a padding. The difference with relative positioning is that you can actually have something like button 3 sitting directly on top of button 2 and completely ignoring padding and margin and things like that. So if we remove top and refresh, now button 3 is literally overlapping button 1 and button 2. And let's try something else different, let's try putting top back in, but making this 220 pixels and now you can see that button 3 is floating over a paragraph that's lower in the page. So you have to be careful with positioning and make sure that you have enough space around your elements if you start moving them around so that they're not overlapping, unless that is the effect that you're trying to achieve. This positioning can be applied to any element. So let's remove the manipulation that we're doing with buttons, put those back in place, and let's go into the page and find this paragraph that is here below the box model heading and I'm going to give it a class, just so I can identify it, let's just call the class temp and inside of styles.css I could say temp, position:relative, move down from the top 50 pixels, move in from the left 20 pixels, and now when I refresh this paragraph should jump. And you can see it's actually overlapping the image that appears below. I could also move it in the opposite direction. And again you can see that you have to be careful that you have the space to be able to move an element. Unlike applying padding or margin when the paragraph moves, it's not going to try to push anything out of its way, it's literally just going to jump and sit on top of the other thing.

Absolute and Fixed Positioning

Now let's look at absolute positioning and I'm going to pick on the last child in that series of input buttons that we have on the page. Now I'm going to set the position to be absolute and let's say down from the top 20 pixels, over from the left 100 pixels, and now when I refresh the page that last button should jump up here, almost over the h1 heading that we have, and indeed that's where it goes. And now that is its absolute position in this document. So even when I scroll, that button stays up there over top of that h1, when we come back up it's still there and that's a little bit different than fixed positioning. So if I set position fixed and save and refresh this page, now that button is in a fixed position relative to the window itself. So it should always be 20 pixels from the top of the window and over right 100 pixels so that even when I scroll that button stays in place. Sometimes a website will used fixed positioning to give you a toolbar somewhere on the web page and always have some buttons available for you. Of course, you have to be very careful using fixed positioning or even absolute positioning because you can never be sure exactly what content it might cover up.

Float and Clear

The float property is another property that you'll see used extensively in modern web layouts that use just CSS for the layout. A floating element can shift to the left or to the right of a line and then allow content to flow along beside it if there's enough room. For example, I can take a div that is statically positioned and tell it to float left. That would move the div to the left side of the line and will allow another div to float up and appear to the right of that div. I can also tell a div to float clear and the div will go back into the normal flow. All these effects are best seen in a demo. Here inside of the web page I want to draw your attention to the image that's in the center of the right hand side of the screen. An image has a display value of inline block. You might remember we used that a couple modules ago. And that means that the element can have a specific width, like we can give the image a width, and it doesn't force the following content to be on a new line. It can display inline like a span, but it has a specific width like you could typically give to a block element like a div and that just means that if I remove this break tag that's here, the caption that's inside of a strong element can move up beside the image. And let me give this a little more space and refresh and you can see that caption is trying to move up there, if it has enough space, the entire caption can be beside the image. And this is interesting, but what if I wanted this caption to appear at the top of this element? In fact, what if I wanted to move additional content up beside the image? There are a couple different ways to do this. One is to do this with align, so I could say align this image to the left, but it turns out that align is actually a deprecated attribute on the image element. We're not supposed to use it anymore, we are supposed to use styles and specifically the float property. So this image has an ID, it's figure1, let's go into styles.css and say that figure1 should float left, that means move as far left as you can and allow other content to resume their flow beside you. I'm going to put that in there and remove the align and now we have the same effect. And of course I can also go in and say float to the right, and the image moves as far to the right as it can and allows other content to resume flow on the left hand side. But for now, let's bring the figure back to the left hand side. You can see that the text is right up against the image so we want to give this a little bit of a margin. Maybe margin to the right of 5 pixels. That'll move the text away a little bit. And now what would happen if I didn't want all of this content to be floating to the right of this? If I wanted to push, let's say, the next paragraph down below the image, that's when I have to come in and clear something. Now clears can be a little bit tricky because sometimes you do have to introduce a new HTML element whose only purpose is to clear out content. In other words, one way to solve this would be to use an inline style and I could say clear both sides, essentially don't let things float around and now that paragraph is pushed down. And if you do not like using inline styles like this, another option would be to put a div in here and we're going to give it a class of clear and then I can define a style, anything that has a clear class should clear on both sides. And that has the same effect. And what would happen if I said, clear just on the right side? The paragraph would come back up because it's only looking to clear things that have been floated to the right. If I said clear left, now it is going to push down because it's not going to bump up against anything that is floated to the left. Many times people will just use clear both to make sure that thing do not float up against something that is floated either left or right.

2 Column Layout

Now that we know a little bit about positioning, let's take our page and give it a two column layout. That is, I have a content div that has all of this text and some images and I also have an author info area that is currently in the flow of the document so it appears at the bottom of the page. What I'd like to do is take this author info and put it on the top left and then have all of the content on the right hand side of the page and it looks like there's two columns coming down the page. So to do this we're going to use styles. The first thing I'll do is come into the body and just reset all the padding and margin so that we don't have to worry about that messing up any of our calculations and then I'm going to take the author info and what we're going to do is take it out of the normal document flow. So we know one way to do that is to give it an absolute position and let's say it's going to be absolutely positioned from the top by 5 pixels and from the left by 5 pixels and if I save this and refresh, we now have about the author in the top left where I want it, but it's also overlaying the content area. We'll fix that in just a bit. Before we do that, I want to point out that it's often quite useful when you're trying to achieve some sort of specific layout to give your block some background color so it's very easy to identify them. So let's give author info a background color of yellow and let's give the content area a background color of red. This is going to be very ugly, but it's also going to be temporary. And now I can see, there's my author info block. One of the first things that jumps out is that I'm going to have to constrain the width of this so that content can appear on the right hand side, I need to know exactly how wide that block is going to be. So let me go ahead and set the width to 300 pixels. One of the reasons I want that to be 300 pixels is because I know that this about the author is an image that has a width of 300 pixels, I want everything to match up. And now that I have author info under control, what I want to do is make sure that my content appears on the right hand side of the page and that it's not overlapping or underlapping where the author information is and the easiest way to do that is with a left margin. So I'm going to say margin to the left should be 310 pixels to make sure it's away from author info and also has a little bit of margin and we refresh that, we can see that is now moved over there. Let's get rid of the colors and see how this looks. And that would be a two column layout. So any additional information that I add into the author info area that would load down here on the left hand side of the document. Any additional information I add to the content area, that's flowing down here on the right hand side of the page. If we wanted the about the author section to appear on the right hand side of the page instead of the left, then you can imagine that we can just flip around some of the work that we did in the last piece. So I can take the absolute positioning and apply it to the content and then take the author info and push it to the right a certain number of pixels. Now the content typically is going to need a little more space than the author info does so let's set this to 400 pixels and then author info can be moved to the left, let's say 405 pixels and now you see that the two have reversed themselves, but still the image bleeds over a little bit into that right hand side area, so we'll have to be really careful with this particular type of layout and make sure that images have enough space to display fully and actually this looks like it's going to be maybe about 600 pixels over, so let's make the width 600 and set this to be 605 pixels and that looks like we have more than enough space now actually. For the next layout trick let's say that we wanted to take this bit of information, this h1 tag, the author name, and the date and make it into a header that spans across both columns across the top. In order to do this I'm going to make a slight structural change. I'm going to have to pull out that h1 and the paragraph tag outside of the content div and put it into another div. So I'm going to create a div called header, paste that h1 and p inside of there, if we just do that and refresh the page, you can see that things are starting to overlap. So this is one of the things you'll get used to in CSS, what we see is that we have a header up top and to make it easier to see, let me go ahead and say that a header has a background color of red and we'll go ahead and set a background color on our content again. You can also use the Developer tools to do this to some extent, but sometimes it's just easier if you can see it right there on the screen. So what's happening is we have the header here up top, that's in the flow of the document, the author information is fine because it's in the flow of the document, it's statically positioned, it's going to appear after that header, but the content area is not in the flow of the document, it's being absolutely positioned so we're going to have to make sure that we absolutely position it so that it's not overlapping the header. One way to do that is to give our header a specific height so we know exactly, down to the pixel, how much space it's going to occupy. So that's going to be 100 pixels and now I can come down to the top and say, I really want to push this part, this content area, down 115 pixels and now that's looking a little bit better. Let's make it 120 and let's take the background colors off so that it doesn't hurt the eyes. And now I have an independently header. So inside of here I could say, text-align to the center and that header sort of spans our two columns, it's appearing right in the middle of the page.

3 Column Layout

Now that we've seen how to do a two column layout, let's move up one level and try a three column layout. For the three column layout I've introduced an additional div, it's full of links, and what I expect to do is have the links appear on the left hand side of the page as one column, the content to appear in the middle of the page as a second page, and then the author information stays out here on the right as a third column. But in order to do this I'm going to need to do a couple things. I'm going to need to take the links and move them out of the flow of the document so that I can pull them up here using absolute positioning to the top. So let's go into styles and let's say for links I want them to be positioned absolute. I want it to be down 120 pixels from the top, remember we still have to leave room for the header. Left, we'll just move them over 0 pixels and let me also set the background color so we can see exactly where they're showing up. Background color of yellow and there you can see our links. That's about the position that we want them. But I do want to make sure that this has a specific amount of width. We don't need a lot of width, but we can expect they'll be some textual content in there that will need some space to display. And now what I can do is this middle bit of content, I'm actually going to switch it over to allow it to be in the flow of the document and take the author info out of the flow and push it up to the right so author info now has an absolute position. It also has to be down from the top 120 pixels, but now off of the right we're going to position it right up there on the right hand side and we can still make the width of 300 pixels, the content area, we'll fix that up next. Let me take the background color though and set it to red just to see how things are overlapping. So you can see now we have some absolutely positioned content here, some absolutely positioned content here. What I really need to do is squeeze this content area into the middle and I can do that by setting a margin left and we can bring it over 105 pixels to make sure that the links area can display and a margin-right, we'll set this to 305 pixels and refresh and now you can see that content is squeezing in there nicely between those two. So let's get rid of the background colors and refresh and we have a three column layout. One of the questions you might be asking when you look at this three column layout is, could I achieve the same effect using floats? And the answer is yes, generally speaking, you can. It all depends on what trade-offs you're willing to make. It turns out that the three column layout is one of the holy grails of CSS design. Everybody wants one, but we can never get 100% there, we can never have it behave exactly like we want it to behave under all circumstances. For example with the absolute positioning that we've used, we can still run into a situation where if the user sizes the browser just right you can have this image, which is very wide, start to overlap on some of the texts or other content that's in the author info area. Now there are things you could do to try to solve that problem, like setting a specific width to that content area and then making sure all the images you have fit into that width. But for everything that you do to try to fix this, you lose something else. So if someone has a very large monitor and they're viewing this full screen and you've set the width of this content area to 450 pixels, then the page starts to look very small and there's a lot of white space on the screen. So everything that you do in CSS design, there's always some sort of trade-off when it comes to layout. But let's see how this would look using floats. What I want to do is say that the lengths are no longer using an absolute positioning, instead we want to float them to the left, also the author information we're not going to use absolute positioning. We're also going to have it float to the left and then finally the content area, we'll no longer need these margins, we want it to float to the left. And if I refresh the page right now what we're going to discover is the content area is up here taking up all the space, down here at the bottom page we have the author information in the links and they are reversed, I want the author information on the right. So of course when you're floating things, the order in which the particular divs or things that you're floating, the order in which they appear in the document becomes significant. So if I want the links to be on the left, I need to place that div ahead of the content and now if I save everything and refresh, we come to the top, there's our links, but notice that the content has not loaded up beside it. And it's always instructive to set background colors to try to figure out some of the stuff. So let's set the background color of content to yellow. Let's set the background color of author information to be aqua. Let's set the background color of the links to be red and refresh the page and you can see that part of the problem is that this content area is taking up 100% of the width. So let's do this, let's say that links should have a width of 100 pixels, content should have a width of let's make it 600 pixels, and then author info we know needs a width of 300 pixels and now I refresh and as long as the page is large enough, everything can appear next to each other. One of the advantages to the floats here is that if you have an image and it's very large, it's never going to bleed over into the author information on the right, but at the same time, just using floats if there's not enough space on the screen, you might have something that moves to the bottom of the page. Now there are some great sources on the internet if you just go out to your favorite search engine and do a search for CSS three column layout, you'll find dozens and dozens and dozens of approaches to trying to achieve a three column layout and typically you can run the code that someone is presenting when they're showing how to do a three column layout and you can pick one of the layouts that works best for the content you're trying to display.

Summary

In this module we looked at positioning elements and we discovered the position property has one of four values, static, relative, absolute, or fixed. The default value is static, which keeps an element in the flow of a document. Relative positioning allows you to move an element from its default position, whereas absolute and fixed positioning essentially allow you to place an element in a specific position in the document or in the window. We also looked at floating, which allows an element to move to the left or to the right and for other content to then float up beside it in a normal flow. Clearing removes that float and prevents an element from floating and then finally we looked at a number of different layout techniques for two and three column layouts. You can use absolute positioning to achieve this, you can also use floating elements. The choice you make really comes down to the content you need to display and how you want the page to behave.