W4 Introduction

Overview

* **Tasks:**Learn about using CSS to improve the styling of our site.
* **Purpose:**This week's focus will be in improving our CSS coding skills.

Objectives

* Become familiar with the basic vocabulary for CSS
* Learn how to apply CSS to a website
* Learn about the role of Backend Web Developer

[Notes]

Remember that CSS is less forgiving than HTML about errors. If you are making change to your CSS and not seeing a resulting change in your web page...[Validate!](https://jigsaw.w3.org/css-validator/)

Preparation Materials

CSS Units of Measurement

CSS units are used to express length for widths, margins, font-sizes, etc. It is usually a number followed by a unit of measurement like px, em, or the percent sign. When coding unit measurements there is no space between the number and the unit. And you might see values of zero where a unit is omitted.

Some units are absolute meaning they are always the same size regardless of the page size. px or the pixel unit of measurement is an example of an absolute measurement. Other units are relative meaning their size can change based on something else like the page or font size. em or % are relative unit measurement examples. Relative units will scale better between different devices. This means for example if we give an image a percentage value for example 100% width then it would be 100% for that part of the page on a phone, tablet, laptop, etc. The width would change depending on what device they were using to render the page. This is called responsiveness. Whereas if we gave that image a unit measurement of 800px it would be a fixed size on all those devices. So relative measurements scale better and are good for responsive web

pages.

The default browser font size is 16px but you can change that default for the entire page by setting a font size on the element. And the child elements will inherit their parent body element’s font size.

em is also a relative measurement. Ems allow you to change the size relative to the size of the text in the parent element. em can be used on more than just fonts though you can use it for margins, padding, line-height etc. But it’s still going to be based off the inherited font-size. **If you always want to base your measurements off the root or font-size in the body regardless of the immediate parent’s font size, you can use rem or ‘root’ em instead of em.** 1em would mean 100% of the default font size and 1.5 em would be one and half times bigger or 150% and 2em would be twice the size, and so on.

1em = 100% of default font sized

1.5em = 150%

2em = twice the size

There are also other types of units of measurement like vw and vh for viewport width and viewport height referring to the width and height of the user’s device. W3schools.com has a more complete list <https://www.w3schools.com/cssref/css_units.asp> of units of measurement. **We will mostly be using pixels, percentages, and em for this course.**

vw (viewpoint width)

vh (viewpoint height)

Remember we talked about all elements having an invisible box around them. They are sized just big enough to hold their contents. We can adjust the size of these boxes with the width and height properties. Notice how this banner or hero image is quite large, it is actually going off the screen and we don’t even see the right part of it. We can change that width to 100% so it will now take 100% of the size of its parent container and not default to however many pixels the image was. We can also set widths on any element such as this div. Because right now it has no content it has no height. We can set a height to make it taller. We can also set a max-width or min-width. This is different from width because with pages that expand and shrink with the user’s screen such as percentages and ems there comes a time when a device might be so small that the page is not legible anymore or a screen so large that the page or element width appears much too big. Use max-width to ensure there is a set point when it shouldn’t grow anymore and min-width to set a point where it won’t appear too narrow. So, for example we can make sure that at a certain point the main section doesn’t get larger than 1500px for example. It’s always a good idea to check your site on different devices to make sure it looks good.

<body>

<div id = “background” /div></div>

<body>

#background (

background-color! Lightblue

height: 500px;

max-width: 1500px;

}

**BYU-Idaho Online Learning**

Video Transcript

Block vs. Inline Elements

*[The screen shows a blank background with text saying “Block and Inline Elements”.]*

**Female:** Block and inline elements. Remember, every element is inside its own invisible box. This box will either be a block or an inline element. Block level elements by default take up 100% of the space that they have, stretching from right to left as far as they can.

This means they will not allow other elements to share the line that they are on. So they appear to start on a new line in the browser window. These include these types of elements.

*contains “div”, “header”, “main”, “footer”,*

*“nav”, “p”, “ul”, “li”, “section”,*

*“article”, “form”, “h1 - h6”.]*

Inline elements by default, will allow other elements to share the same line as long as there's room, then they will only wrap to the next line, once it's full.

*[The text and image of block elements is replaced by an image example of inline elements. The rectangles this time aren’t as long and are side by side as long as there is room. If there is no room they go to the next line.]*

They will only take up as much width as necessary. These include these type of elements:

*“a”, “img”, “span”, “button”, “label”*

The h1 element in all the paragraphs here are block level elements.

*[The text and image is removed and replaced with an article. The title is “Ways to Stay Warm Outdoors”. Below the title are four paragraphs that give tips about how to stay warm. The first and third paragraph have a background color of gray.]*

If I place a border around them, you can see that they take up all the space they can.

*[A red border encircles each individual element. The paragraphs with a gray background color have their backgrounds completely contained in the boxes.]*

However, if we place some images side-by-side, as long as there's room, they will share the horizontal space available.

*[The image of the website changes to one that is lower down on the website. Only the last two paragraphs are shown. Below the paragraphs there are four images. The images are a blanket, a fire that is on a wheely table, hand warmers, and a parent sledding with their kid. The images are all on the same line.]*

You can change the defaults of inline and block elements with the display property. If I wanted to make a block element into an inline element, I would target it in CSS and use the display in my declaration.

* Step 1 – Have a warm blanket
* Step 2 – Get an outdoor heater
* Step 3 – Have plenty of hand warmers
* Step 4 – Stay active outside

#warm\_list li {

Display: inline;

}

Step 1 – Have a warm blanket Step 2 – Get an outdoor heater Step 3 – Have plenty of hand warmers Step 4 – Stay active outside

*[The image is removed and replaced with three images. The first one has a bulleted list that has four steps on how to stay warm. The list is vertical. The second image shows CSS for an element that has the id “warm\_list” and is an <li> element. Inside the curly braces is the code “display: inline;”. The third image shows the list again, but this time it is all on the same line.]*

If I wanted to make an inline element into a block level element, I would use the display block declaration.

Main img {

Display: block;

}

*[The images are removed and replaced with two images. The first image is CSS that selects elements that are in <main> and are an <img>. Inside the curly braces is the code “display: block” The second image is the four images of staying warm in winter that were on the website. This time they are all  on their own line.]*

There's also one more display property value, inline-block. It's kind of a mix between the two. It would cause a block level element to flow like an inline element, but it will retain other features of a block level element, like allowing you to set widths and heights on the element and top and bottom margins would work more predictably. The declaration you would use is display inline-block. Let's demonstrate the difference here. Here we have three spans, one inline, one inline-block, and one block.

Inline-block

Display: inline-block

*[A new image of a website appears on the screen. On the left side is HTML and CSS that was used to make the website. The title of the website is “The display Property”. Beneath the title are examples of inline, inline-block, and block displays. All the different displays have the same width, height, border, and background color. The inline example has the words “Aliquam” and “venenatis” highlighted and it contains a border. The height and width of the words follow the height and width of the other words in the paragraph. They are the same line as the rest of the paragraph. The inline block example has the same words in the same part of the sentence highlighted and bordered. Except, the width and height are what was set in the CSS. They are on the same line as the rest of the paragraph. The block display example has the same words highlighted and bordered. Except, the words have the width and height that was set in the CSS and they are each in their own line.]*

Notice the difference between the three. Even though the widths and heights are all set the same between the three, the in-line width, heights, and margins, padding and bottom are not always what is expected. Inline block is sometimes a better choice for a simple way to align a few items side-by-side. Because of these different display values. Centering elements on our page works differently with different elements. Two common ways to center elements are text-align center, margin 0, auto. Text-align center will center the contents of the box. So if I put text-align center on the H1 or the p, then the content will get centered inside that invisible box or container.

Margin 0 auto will center the box itself. The first value refers to the top and bottom margin. In our case, we'll just leave them 0. The second value refers to the right and left. It will automatically give an equal gap on each side of the box. With margin 0 auto, if the width of the element takes the whole space, you won't see the centering. There needs to be a width set, so there is space on each side of that side to automatically give an equal measurement on each side. Or in other words, the box needs a width. Otherwise, we'll take up the whole width of the page. So if I give the h1 and p a width, the contents are still centered inside the box with the text-align center. But I could use margin 0 auto to center the box itself.

main p, main h1{

text-align: center; (content of the box)

}

If I tried text-align center with an image which is an inline element, it doesn't appear to center.

main p, main h1{

text-align: center;

width: 800 px;

margin: 0 auto; (centers the box)

boarder: 2 px solid red;

}

main img {

text-align: center;

boarder: 2 px solid red;

}

That's because its border is tight around the image. The content is the image, and it's already centered inside its type border. But we can change the image to display block, give it a width, then we can center it with margin 0 auto.

main img {

width: 300 px;

display; block;

margin: 0 auto;

boarder: 2px solid red;

Or we could place the image inside of a block level element and then use text align center to center the contents of the element, which would include that child element.

<div id=”img\_container:

<img src=”image/blanket.jpg” alt=”warm blanket’>

</div>

#img\_container {

Text-align; center;

}

*[The images change again and a new image is added. The blanket is still centered in the window, but there is a new border that goes along the whole width of the window. The new image has text that is using HTML. The text shows the image element is inside a <div> element with the id of “img\_container”. The third image shows the CSS for the div. It selects the div using “#img\_container”. The code inside the curly braces is “text-align: center;”.]*

*[End of video.]*

**BYU-Idaho Online Learning**

Video Transcript

CSS Selectors

*[The text “CSS selectors” is shown.]*

**Instructor:** CSS selectors and specificity. There are many different types of CSS selectors that allow you to target specific elements in HTML.

*[Text saying “div { height: 400px; }” and “.logo-box { background-color: blue; }” is shown.]*

So far we've learned how to target by using element names like p or H1. We've also seen how we can target class and ID names as well, using, for example, “#content” for an ID or “.icon” for a class. Then it's more specific, and instead of targeting every type of element, like all the divs or all the IMG or images. We can be more specific in which ones to target. This is called **specificity.** We saw a little bit of specificity when we talked about the order of CSS and how that matters. **If the same exact selector had the same rule applied, the one later in the CSS file would be applied and overwrite the earlier one.** Think of specificity as a score or rank that will determine which style declaration will be the one applied. For example, **if I declare a font family for the whole page, each child element inherits the font throughout the page.**

*[An image of a website is shown. All the fonts are the same. An image shows some CSS text. The text selects the body element and sets the font family and font size of all the elements.]*

But if I later declare my H1 to have a different font family later in the CSS file, it's more specific, and all my H1’s ones will have a different font.

*[The website image is shown again. The font of the H1 text is now different. The CSS image is shown with text selecting all H1 elements and setting their font family and font size.]*

The body rule applies to all the text on the page, but the H1 rule is more specific. So it's the one that's going to be applied. So I might have all my images set to a **width of 100%.** But I find that some of my images are just too big.

*[The website is shown. The instructor scrolls down the website. Some of the images are too big to fit both the height and width comfortably on the screen.]*

So I need to have a smaller percentages for those. So I might target just those images that I need smaller with their class name, for example, “.heat” to make just those images have a more specific rule set.

*[An image showing HTML is shown. The images that were too big for the screen now have a class name of “heat”. CSS text selecting the heat class is shown. The CSS changes the image’s width to forty percent. The website is shown again, this time the images are a comfortable size.]*

Even though the width of 100% is still applied to all images, including them, the smaller width percentages will win out and will be applied because it's more specific. And here's where it gets interesting. Even if that rule came before the IMG rule, it would still apply because a class selector ranks higher and in specificity than an element name selector.

Every selector will have a ranking and the specificity hierarchy, inline styles, as we saw earlier, have a very high ranking. But we're only going to be using an external CSS file. So for us, the next more specific one is the id selector, like “#logo-link”. Then it would be classes and pseudo-classes like “.card-img” or “.card-img:hover”, which we're going to talk about here in just a minute. Then lastly, the least specific are the element selectors with the element names like p, H1, IMG, et cetera. **Keep this in mind when a rule seems to not work, you might have a more specific role targeting that same element and overwriting your rule.** You may see some elements and classes used together in CSS. For example, the class dot heat might be placed with the IMG tag using both the element name and the class, like this.

specificity hierarchy

inline styles

id selector like “#logo-link”

classes and pseudo-classes like “.card-img” or “.card-img:hover”

element names like p, H1, IMG

*[An image of CSS is shown. The selector is “img.heat”.]*

“img.heat” no spaces. This is just saying to target all the IMG elements that also have the class of heat. Or I might want all the a tags with the class of highlight.

img.heat {

width: 40%’

display: block;

margin: 0 auto;

*[An image of CSS is shown. The selector is “a.highlight”.]*

a.highlight {

background-color: lightyellow;

}

With “a.highlight”. This would come in handy if you use the class highlight with different types of elements and want to specify which element with that class name you want to target.

We can also list a number of element names, classes, IDs, et cetera, that have the same declaration applying to all of them by separating them with commas. For example, “h1, h4” this would mean that all the H1s and all the H4s would then have that font family, the same one, and wouldn't need a separate rule for each one.

h1, h4 {

font-family: ‘Lemon’, ‘Bradley Hand’, sans-serif;

font-size: 45px;

color: #4b4b2a

}

*[An image showing CSS is shown. The selector is “h1, h4”. Text above the image says “h1 and h4”.]*

We can get even more specific with descendant CSS selectors. For example, maybe we just want all the paragraphs inside of our article element to have a line-height.

*[An image showing HTML is shown. Elements are inside an article tag. An image of CSS is shown. The selector is “article p” the text inside sets the line height to 1.5em.]*

So we target just paragraphs that are children or descendants of an article element like this “article p”. The descendant would be the last element. So this means that all the p's that are inside of an article element would be affected.

Article p {

line-height: 1.5em;

Or maybe we just want the images inside of our product gallery like this.

#product\_gallery img {

Width: 50%;

}

*[The text “#product\_gallery img” is shown.]*

So just the images inside of “#product\_gallery. We can also target images like we had before with the width of 40 percent, but without giving them all class names and just selecting them by either all the images in the main section or all the images inside the “img\_container img”.

main img {

width: 40%;

} (or)

#img\_container img{

Width: 40%;

}

*[The images that had the heat class name now do not have a set class name. Two CSS selectors are show, “main img” and “#img\_container img”.]*

Visit [W3 schools](https://www.w3schools.com/cssfer/css_selectors.asp) for even more specific CSS selectors. Let's also talk about pseudo selectors and pseudo classes. One example of a pseudo-selector is the nth of type selector. If you have a number of the same type of element, it will let you select the first or the second, or the third or the fourth, et cetera, of those elements.For example, if I have a number of list items, I could select just the second one without having to give it a class or an ID by just using the end of type like this.

li:nth-of-type(2) }

background-color: #ccc;

}

*[An image of CSS is shown. The selector is “li:nth-of-type(2)”. Inside, a line changes the background color to a gray. An image of a list is shown. The second item in the list has a background color of gray.]*

If I wanted the fourth one, I'd use a four instead of a two.

li:nth-of-type(4) }

background-color: #ccc;

}

Pseudo-classes allow you to change the appearance of an element when a user is interacting with it. For example, if I want the a links in my nav to have a different background color or the text a different color when the user hovers over them, I can use the colon hover after the selector and give it the declarations I want to happen when the user hovers over it.

nav a:hover {

color: black;

background-color: rgba[255,255,255, 0.3];

}

button:hover {

background-color: #4b4b2a9d;

}

*[A CSS image is shown. Two selectors are shown. They are “nav a:hover” and “button:hover”. The website is shown next to the image. When the mouse hovers over a button that is also a link, the background color changes and the text changes color.*

There are other pseudo-classes like active, focus, and visited. See W3 schools. For more examples.

*[End of video.]*