WEB Development Learning Path Notes

CSS

## Using the <Style> tag

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

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

<head>

<style>

</style>

</head>

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

<head>

<style>

p {

color: red;

font-size: 20px;

}

</style>

</head>

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

## Using the css file

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

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

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

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

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

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

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

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

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

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

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

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

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

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

## Tag Names

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

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

p {

}

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

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

## Class Name

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

For example, consider the following HTML:

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

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

.brand {

}

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

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

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

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

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

.green {

color: green;

}

.bold {

font-weight: bold;

}

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

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

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

## ID Name

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

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

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

#large-title {

}

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

## Classes VS I’d’s

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

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

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

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

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

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

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

h1 {

color: red;

}

.headline {

color: firebrick;

}

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

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

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

ID 🡪Class 🡪Tag

ID will be applied with highest priority Tag with the lowest

## Chaining Selectors

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

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

h1.special {

}

The code above would select only the h1 elements that have a class of special. If a p element also had a class of special, the rule in the example would not style the paragraph.

## Nested Elements

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

<ul class='main-list'>

<li> ... </li>

<li> ... </li>

<li> ... </li>

</ul>

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

.main-list li {

}

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

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

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

For instance, consider the following CSS:

p {

color: blue;

}

.main p {

color: red;

}

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

## Important

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

The syntax of !important in CSS looks like this:

p {

color: blue !important;

}

.main p {

color: red;

}

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

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

## Multiple Selectors

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

For instance, the following code has repetitive style attributes:

h1 {

font-family: Georgia;

}

.menu {

font-family: Georgia;

}

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

h1,

.menu {

font-family: Georgia;

}

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

# Advanced Design With CSS

## The Box Model

Browsers load HTML elements with default position values. This often leads to an unexpected and unwanted user experience, while limiting the views you can create. In this lesson you will learn about the *box model*, an important concept to understand how elements are positioned and displayed on a website.

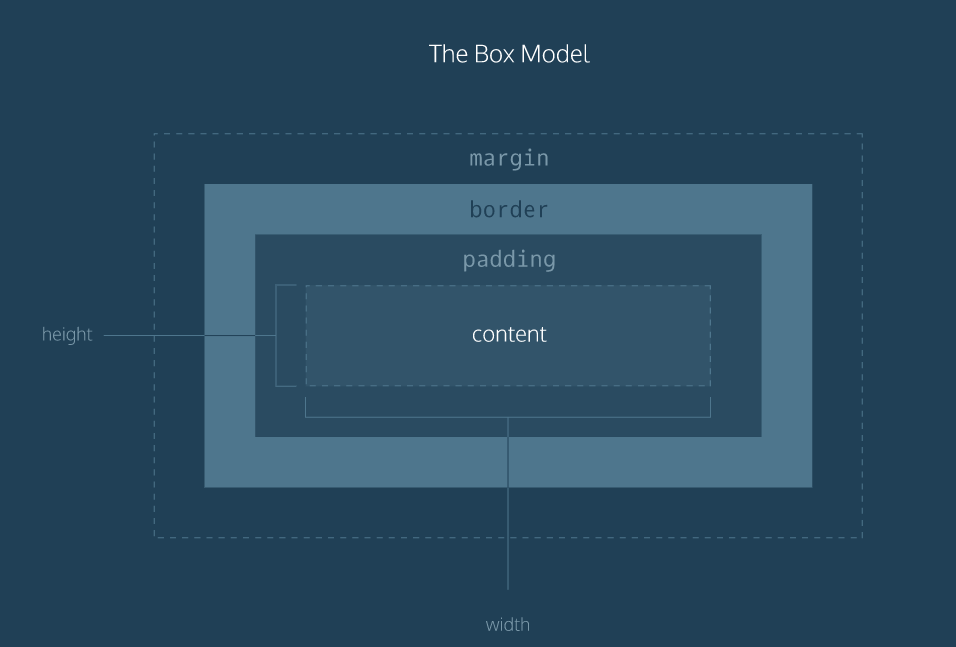
If you have used HTML and CSS, you have unknowingly seen aspects of the box model. For example, if you have set the background color of an element, you may have noticed that the color was applied not only to the area directly behind the element, but also to the area to the right of the element. Also, if you have aligned text, you know it is aligned relative to something. What is that something?

All elements on a web page are interpreted by the browser as “living” inside of a box. This is what is meant by the box model.

For example, when you change the background color of an element, you change the background color of its entire box.

In this lesson, you’ll learn about the following aspects of the box model:

1. The dimensions of an element’s box.
2. The borders of an element’s box.
3. The paddings of an element’s box.
4. The margins of an element’s box.



The box model comprises the set of properties which define parts of an element that take up space on a web page. The model includes the content area’s size (*width* and *height*) and the element’s *padding*, *border*, and *margin*. The properties include:

1. Width and height — specifies the width and height of the content area.
2. Padding — specifies the amount of space between the content area and the border.
3. Border — specifies the thickness and style of the border surrounding the content area and padding.
4. Margin — specifies the amount of space between the border and the outside edge of the element.

### Height & Width

An element’s content has two dimensions: a height and a width. By default, the dimensions of an HTML box are set to hold the raw contents of the box.

The CSS height and width properties can be used to modify these default dimensions.

p {

height: 80px;

width: 240px;

}

In this example, the height and width of paragraph elements are set to 80 pixels and 240 pixels, respectively — the px in the code above stands for *pixels*.

Pixels allow you to set the exact size of an element’s box (width and height). When the width and height of an element are set in pixels, it will be the same size on all devices — an element that fills a laptop screen will overflow a mobile screen.

### Borders

A border is a line that surrounds an element, like a frame around a painting. Borders can be set with a specific width, style, and color.

1. width — The thickness of the border. A border’s thickness can be set in pixels or with one of the following keywords: thin, medium, or thick.
2. style — The design of the border. Web browsers can render any of [10 different styles](https://developer.mozilla.org/en-US/docs/Web/CSS/border-style" \l "Values" \t "_blank). Some of these styles include: none, dotted, and solid.
3. color — The color of the border. Web browsers can render colors using a few different formats, including [140 built-in color keywords](https://developer.mozilla.org/en-US/docs/Web/CSS/color_value" \t "_blank).

p {

border: 3px solid coral;

}

In the example above, the border has a width of 3 pixels, a style of solid and a color of coral. All three properties are set in one line of code.

The default border is medium none color, where color is the current color of the element. If width, style, or color are not set in the CSS file, the web browser assigns the default value for that property.

p.content-header {

height: 80px;

width: 240px;

border: solid coral;

}

In this example, the border style is set to solid and the color is set to coral. The width is not set, so it defaults to medium.

#### Border Radius

Ever since we revealed the borders of boxes, you may have noticed that the borders highlight the true shape of an element’s box: square. Thanks to CSS, a border doesn’t have to be square.

You can modify the corners of an element’s border box with the border-radius property.

div.container {

border: 3px solid rgb(22, 77, 100);

border-radius: 5px;

}

The code in the example above will set *all four corners* of the border to a radius of 5 pixels (i.e. the same curvature that a circle with radius 5 pixels would have).

You can create a border that is a perfect circle by setting the radius equal to the height of the box, or to 100%.

div.container {

height: 60px;

width: 60px;

border: 3px solid rgb(22, 77, 100);

border-radius: 100%;

}

The code in the example above creates a <div> that is a perfect circle.

### Padding

The space between the contents of a box and the borders of a box is known as *padding*. Padding is like the space between a picture and the frame surrounding it. In CSS, you can modify this space with the padding property.

p.content-header {

border: 3px solid coral;

padding: 10px;

}

The code in this example puts 10 pixels of space between the content of the paragraph (the text) and the borders, on all four sides.

The padding property is often used to expand the background color and make content look less cramped.

If you want to be more specific about the amount of padding on each side of a box’s content, you can use the following properties:

1. padding-top
2. padding-right
3. padding-bottom
4. padding-left

Each property affects the padding on only one side of the box’s content, giving you more flexibility in customization.

p.content-header {

border: 3px solid fuschia;

padding-bottom: 10px;

}

In the example above, only the bottom side of the paragraph’s content will have a padding of 10 pixels.

Another implementation of the padding property lets you specify exactly how much padding there should be on each side of the content in a single declaration.

p.content-header {

border: 3px solid grey;

padding: 6px 11px 4px 9px;

}

In the example above, the four values 6px 11px 4px 9px correspond to the amount of padding in a clockwise rotation. In order, it specifies the amount of padding on the top (6 pixels), right (11 pixels), bottom (4 pixels), and left (9 pixels) sides of the content.

When using this implementation of the padding property, we must specify a padding value for all four sides of the element.

However, if the top and bottom values for padding will equal each other, and the left and right values for padding will also equal each other, you can use the following shortcut:

p.content-header {

padding: 5px 10px;

}

The first value, 5px, sets the padding value for the top and bottom sides of the content. The second value, 10px, sets the padding value for the left and right sides of the content.

### Margins

So far you’ve learned about the following components of the box model: content, borders, and padding. The fourth and final component of the box model is *margin*.

Margin refers to the space directly outside of the box. The margin property is used to specify the size of this space.

p {

border: 1px solid aquamarine;

margin: 20px;

}

The code in the example above will place 20 pixels of space on the outside of the paragraph’s box on all four sides. This means that other HTML elements on the page cannot come within 20 pixels of the paragraph’s border.

If you want to be even more specific about the amount of margin on each side of a box, you can use the following properties:

1. margin-top
2. margin-right
3. margin-bottom
4. margin-left

Each property affects the margin on only one side of the box, providing more flexibility in customization.

p {

border: 3px solid DarkSlateGrey;

margin-right: 15px;

}

In the example above, only the right side of the paragraph’s box will have a margin of 15 pixels. It’s common to see margin values used for a specific side of an element.

What if you don’t want equal margins on all four sides of the box?

A similar implementation of the margin property is used to specify exactly how much margin there should be on each side of the box in a single declaration.

p {

margin: 6px 10px 5px 12px;

}

In the example above, the four values 6px 10px 5px 12px refer to the amount of margin around the box in a clockwise rotation. In order, it specifies the amount of margin on the top (6 pixels), right (10 pixels), bottom (5 pixels), and left (12 pixels) sides of the box.

When using this implementation of the margin property, the margin value must be specified for all four sides of the box.

Just like the padding shortcut, when you’re certain that the top and bottom values for margin will equal each other, and that the left and right values for margin will also equal each other, you can use the following shortcut:

p {

margin: 6px 12px;

}

The first value, 6px, sets a margin value for the top and bottom of the box. The second value, 12px, sets a margin value for the left and right sides of the box.

#### Auto

The margin property also lets you center content. However, you must follow a few syntax requirements. Take a look at the following example:

div {

margin: 0 auto;

}

In the example above, margin: 0 auto; will center the divs in their containing elements. The 0 sets the top and bottom margins to 0 pixels. The auto value instructs the browser to adjust the left and right margins until the element is centered within its containing element.

The div elements in the example above should center within an element that fills the page, but this doesn’t occur. Why?

In order to center an element, a width must be set for that element. Otherwise, the width of the div will be automatically set to the full width of its containing element, like the <body>, for example. It’s not possible to center an element that takes up the full width of the page.

div.headline {

width: 400px;

margin: 0 auto;

}

In the example above, the width of the div is set to 400 pixels, which is less than the width of most screens. This will cause the div to center within a containing element that is greater than 400 pixels wide.

#### Margin Collapse

As you have seen, padding is space added inside an element’s border, while margin is space added outside an element’s border. One additional difference is that top and bottom margins, also called vertical margins, *collapse*, while top and bottom padding does not.

Horizontal margins (left and right), like padding, are always displayed and added together. For example, if two divs with ids #div-one and #div-two, are next to each other, they will be as far apart as the sum of their adjacent margins.

#img-one {

margin-right: 20px;

}

#img-two {

margin-left: 20px;

}

In this example, the space between the #img-one and #img-two borders is 40 pixels. The right margin of #img-one (20px) and the left margin of #img-two (20px) add to make a total margin of 40 pixels.

Unlike horizontal margins, vertical margins do not add. Instead, the larger of the two vertical margins sets the distance between adjacent elements.

#img-one {

margin-bottom: 30px;

}

#img-two {

margin-top: 20px;

}

In this example, the vertical margin between the #img-one and #img-two elements is 30 pixels. Although the sum of the margins is 50 pixels, the margin collapses so the spacing is only dependent on the #img-one bottom margin.

It may be helpful to think of collapsing vertical margins as a short person trying to push a taller person. The tall person has longer arms and can easily push the short person, while the person with short arms cannot reach the person with long arms.

### Minimum and Maximum Height and Width

Because a web page can be viewed through displays of differing screen size, the content on the web page can suffer from those changes in size. To avoid this problem, CSS offers two properties that can limit how narrow or how wide an element’s box can be sized to.

1. min-width — this property ensures a minimum width of an element’s box.
2. max-width — this property ensures a maximum width of an element’s box.

p {

min-width: 300px;

max-width: 600px;

}

In the example above, the width of all paragraphs will not shrink below 300 pixels, nor will the width exceed 600 pixels.

Content, like text, can become difficult to read when a browser window is narrowed or expanded. These two properties ensure that content is legible by limiting the minimum and maximum widths of an element.

You can also limit the minimum and maximum *height* of an element.

1. min-height — this property ensures a minimum height for an element’s box.
2. max-height — this property ensures a maximum height of an element’s box.

p {

min-height: 150px;

max-height: 300px;

}

In the example above, the height of all paragraphs will not shrink below 150 pixels and the height will not exceed 300 pixels.

What will happen to the contents of an element’s box if the max-height property is set too low? It’s possible for the content to spill outside of the box, resulting in content that is not legible. You’ll learn how to work around this issue in the next exercise.

#### Overflow

All of the components of the box model comprise an element’s size. For example, an image that has the following dimensions is 364 pixels wide and 244 pixels tall.

* 300 pixels wide
* 200 pixels tall
* 10 pixels padding on the left and right
* 10 pixels padding on the top and bottom
* 2 pixels border on the left and right
* 2 pixels border on the top and bottom
* 20 pixels margin on the left and right
* 10 pixels margin on the top and bottom

The total dimensions (364px by 244px) are calculated by adding all of the vertical dimensions together and all of the horizontal dimensions together. Sometimes, these components result in an element that is larger than the parent’s containing area.

How can we ensure that we can view all of an element that is larger than its parent’s containing area?

The overflow property controls what happens to content that spills, or overflows, outside its box. It can be set to one of the following values:

* hidden - when set to this value, any content that overflows will be hidden from view.
* scroll - when set to this value, a scrollbar will be added to the element’s box so that the rest of the content can be viewed by scrolling.
* visible - when set to this value, the overflow content will be displayed outside of the containing element. Note, this is the default value.

p {

overflow: scroll;

}

In the example above, if any of the paragraph content overflows (perhaps a user resizes their browser window), a scrollbar will appear so that users can view the rest of the content.

The overflow property is set on a parent element to instruct a web browser how to render child elements. For example, if a div’s overflow property is set to scroll, all children of this div will display overflowing content with a scroll bar.

### Resetting Defaults

All major web browsers have a default stylesheet they use in the absence of an external stylesheet. These default stylesheets are known as user agent stylesheets. In this case, the term “[user agent](https://en.wikipedia.org/wiki/User_agent)“ is a technical term for the browser.

User agent stylesheets often have default CSS rules that set default values for padding and margin. This affects how the browser displays HTML elements, which can make it difficult for a developer to design or style a web page.

Many developers choose to reset these default values so that they can truly work with a clean slate.

\* {

margin: 0;

padding: 0;

}

The code in the example above resets the default margin and padding values of all HTML elements. It is often the first CSS rule in an external stylesheet.

Note that both properties are both set to 0. When these properties are set to 0, they do not require a unit of measurement.

### Visibility

Elements can be hidden from view with the visibility property.

The visibility property can be set to one of the following values:

1. hidden — hides an element.
2. visible — displays an element.

<ul>

<li>Explore</li>

<li>Connect</li>

<li class="future">Donate</li>

<ul>

.future {

visibility: hidden;

}

In the example above, the list item with a class of future will be hidden from view in the browser.

Keep in mind, however, that users can still view the contents of the list item (e.g., Donate) by viewing the source code in their browser. Furthermore, the web page will *only* hide the contents of the element. It will still leave an empty space where the element is intended to display.

**Note:** What’s the difference between display: none and visibility: hidden? An element with display: none will be completely removed from the web page. An element with visibility: hidden, however, will not be visible on the web page, but the space reserved for it will.

### Changing the Box Model

The last lesson focused on the most important aspects of the box model: box dimensions, borders, padding, and margin.

The box model, however, has an awkward limitation regarding box dimensions. This limitation is best illustrated with an example.

<h1>Hello World</h1>

h1 {

border: 1px solid black;

height: 200px;

width: 300px;

padding: 10px;

}

In the example above, a heading element’s box has solid, black, 1 pixel thick borders. The height of the box is 200 pixels, while the width of the box is 300 pixels. A padding of 10 pixels has also been set on all four sides of the box’s content.

Unfortunately, under the current box model, the border thickness and the padding will affect the dimensions of the box.

The 10 pixels of padding increases the height of the box to 220 pixels and the width to 320 pixels. Next, the 1-pixel thick border increases the height to 222 pixels and the width to 322 pixels.

Under this box model, the border thickness and padding are added to the overall dimensions of the box. This makes it difficult to accurately size a box. Over time, this can also make all of a web page’s content difficult to position and manage.

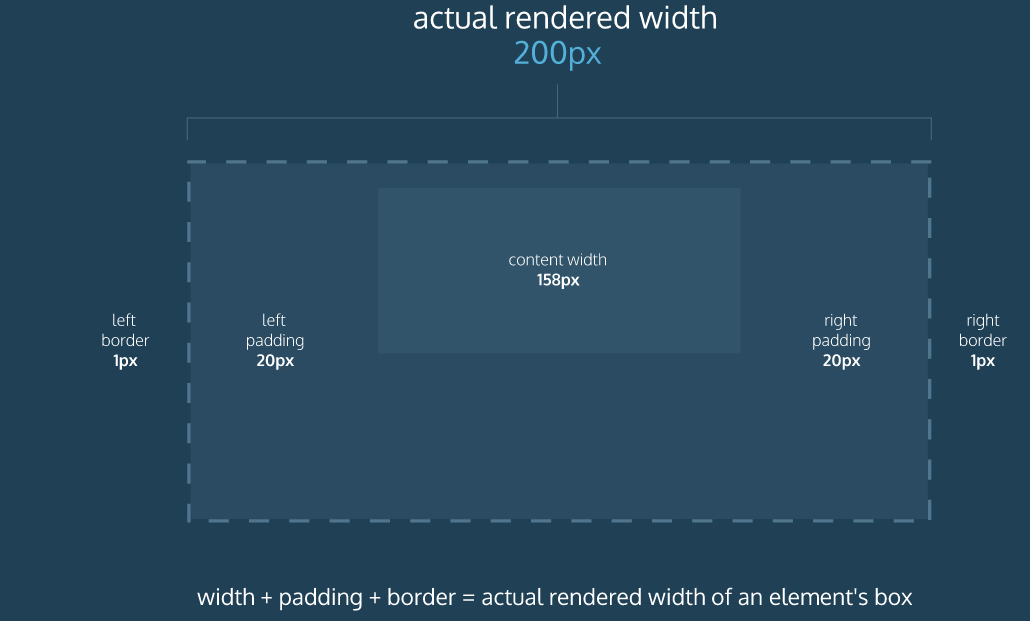
In this brief lesson, you’ll learn how to use a different technique that avoids this problem altogether.

Many properties in CSS have a default value and don’t have to be explicitly set in the stylesheet.

For example, the default font-weight of text is normal, but this property-value pair is not typically specified in a stylesheet.

The same can be said about the box model that browsers assume. In CSS, the box-sizing property controls the type of box model the browser should use when interpreting a web page.

The default value of this property is content-box. This is the same box model that is affected by border thickness and padding.



### Border Box

Fortunately, we can reset the entire box model and specify a new one: border-box.

\* {

box-sizing: border-box;

}

The code in the example above resets the box model to border-box for all HTML elements. This new box model avoids the dimensional issues that exist in the former box model you learned about.

In this box model, the height and width of the box will remain fixed. The border thickness and padding will be included inside of the box, which means the overall dimensions of the box do not change.

<h1>Hello World</h1>

\* {

box-sizing: border-box;

}

h1 {

border: 1px solid black;

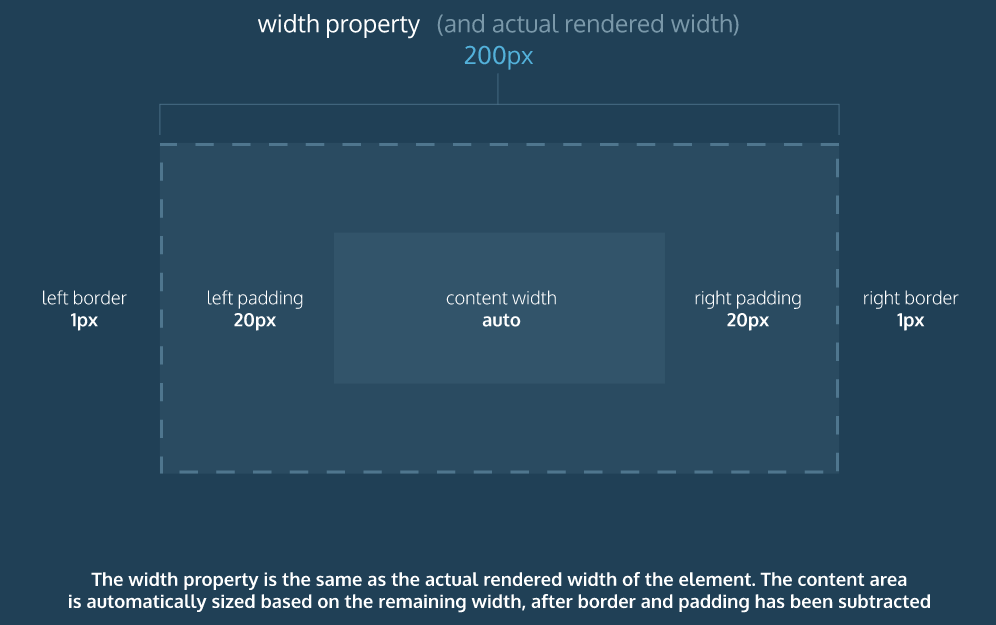
height: 200px;

width: 300px;

padding: 10px;

}

In the example above, the height of the box would remain at 200 pixels and the width would remain at 300 pixels. The border thickness and padding would remain entirely *inside* of the box.



Now that you know about the new box model, let’s actually implement it in the browser.

\* {

box-sizing: border-box;

}

It’s that simple! In the example above, the universal selector (\*) targets all elements on the web page and sets their box model to the border-box model.

## Flow of HTML

A browser will render the elements of an HTML document that has no CSS from left to right, top to bottom, in the same order as they exist in the document. This is called the *flow* of elements in HTML.

In addition to the properties that it provides to style HTML elements, CSS includes properties that change how a browser *positions* elements. These properties specify where an element is located on a page, if the element can share lines with other elements, and other related attributes.

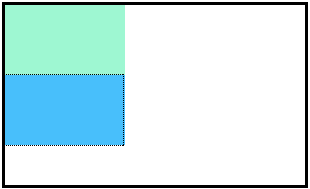
In this lesson, you will learn five properties for adjusting the position of HTML elements in the browser:

* position
* display
* z-index
* float
* clear

Each of these properties will allow us to position and view elements on a web page. They can be used in conjunction with any other styling properties you may know.

### Position

Take a look at the *block-level* elements in the image below:



Block-level elements like these boxes create a *block* the full width of their parent elements, and they prevent other elements from appearing in the same horizontal space. The boxes in the image above were created with the following CSS:

.boxes {

width: 120px;

height: 70px;

}

and the following HTML:

<div class="boxes"></div>

<div class="boxes"></div>

Notice the block-level elements in the image above take up their own line of space and therefore don’t overlap each other. In the browser to the right you can see block-level elements also consistently appear on the left side of the browser. This is the default *position* for block-level elements.

The default position of an element can be changed by setting its position property. The position property can take one of four values:

1. static - the default value (it does not need to be specified)
2. relative
3. absolute
4. fixed

In the next few exercises, you’ll learn about the values in items 2, 3, and 4 above. For now, it’s important to understand that if you favor the default position of an HTML element, you don’t need to set its position property.

#### Position Relative

One way to modify the default position of an element is by setting its position property to relative.

This value allows you to position an element *relative* to its default static position on the web page.

.box-bottom {

background-color: DeepSkyBlue;

position: relative;

}

Although the code in the example above instructs the browser to expect a relative positioning of the div, it does not specify where the div should be positioned on the page.

.box-bottom {

background-color: DeepSkyBlue;

position: relative;

top: 20px;

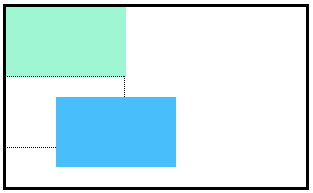
left: 50px;

}

In the example above, the <div> has been positioned using two of the four *offset properties*. The valid offset properties are:

1. top - moves the element down.
2. bottom - moves the element up.
3. left - moves the element right.
4. right - moves the element left.

In the example above, the <div> will be moved down 20 pixels and to the right 50 pixels from its default static position. The image below displays the new position of the box. The dotted line represents where the statically positioned (default) box was positioned.



Units for offset properties can be specified in pixels, ems, or percentages. Note that offset properties will not work if the value of the element’s position property is the default static.

#### Postion Absolute

Another way of modifying the position of an element is by setting its position to absolute.

When an element’s position is set to absolute all other elements on the page will *ignore* the element and act like it is not present on the page. The element will be positioned relative to its closest positioned parent element.

.box-bottom {

background-color: DeepSkyBlue;

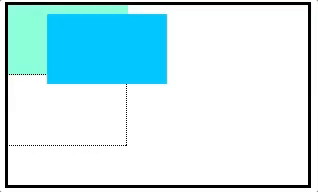
position: absolute;

top: 20px;

left: 50px;

}

In the example above, the .box-bottom <div> will be moved down and right from the top left corner of the view. If offset properties weren’t specified, the top box would be entirely covered by the bottom box. Take a look at the gif below:



The bottom box in this image (colored blue) is displaced from the top left corner of its container. It is 20 pixels lower and 50 pixels to the right of the top box.

In the next exercise, we will compare the scrolling of absolute elements with fixed elements.

#### Position Fixed

When an element’s position is set to absolute, as in the last exercise, the element will scroll with the rest of the document when a user scrolls.

We can *fix* an element to a specific position on the page (regardless of user scrolling) by setting its position to fixed.

.box-bottom {

background-color: DeepSkyBlue;

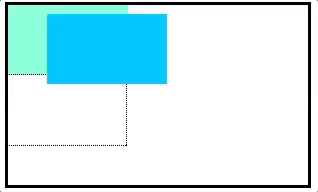
position: fixed;

top: 20px;

left: 50px;

}

In the example above, the .box-bottom <div> will remain fixed to its position no matter where the user scrolls on the page, like in the image below:



This technique is often used for navigation bars on a web page.

### Z Index

When boxes on a web page have a combination of different positions, the boxes (and therefore, their content) can overlap with each other, making the content difficult to read or consume.

.box-top {

background-color: Aquamarine;

}

.box-bottom {

background-color: DeepSkyBlue;

position: absolute;

top: 20px;

left: 50px;

}

In the example above, the .box-bottom <div> ignores the .box-top <div> and overlaps it as a user scrolls.

The z-index property controls how far “back” or how far “forward” an element should appear on the web page when elements overlap. This can be thought of the *depth* of elements, with deeper elements appearing behind shallower elements.

The z-index property accepts integer values. Depending on their values, the integers instruct the browser on the order in which elements should be displayed on the web page.

.box-top {

background-color: Aquamarine;

position: relative;

z-index: 2;

}

.box-bottom {

background-color: DeepSkyBlue;

position: absolute;

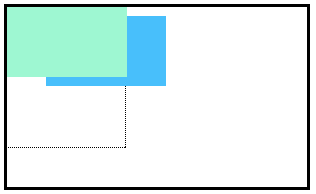
top: 20px;

left: 50px;

z-index: 1;

}

In the example above, we set the .box-top position to relative and the z-index to 2. We changed position to relative, because the z-index property does *not* work on static elements. The z-index of 2 moves the .box-top element forward, because it is greater than the .box-bottom z-index, 1. See the example image below:



In the image above, you can see the top box is moved in front of the bottom box.

### Display

Every HTML element has a default display value that dictates if it can share horizontal space with other elements. Some elements fill the entire browser from left to right regardless of the size of their content. Other elements only take up as much horizontal space as their content requires and can be directly next to other elements.

In this lesson, we’ll cover three values for the display property: inline, block, and inline-block.

#### Inline Display

The default display for some tags, such as <em>, <strong>, and <a>, is called inline. Inline elements have a box that wraps tightly around their content, only taking up the amount of space necessary to display their content and not requiring a new line after each element. The height and width of these elements cannot be specified in the CSS document. For example, the text of an anchor tag (<a>) will, by default, be displayed on the same line as the surrounding text, and it will only be as wide as necessary to contain its content. inline elements cannot be altered in size with the height or width CSS properties.

To learn more about <em>inline</em> elements, read <a href="#">MDN documentation</a>.

In the example above, the <em> element is inline, because it displays its content on the same line as the content surrounding it, including the anchor tag. This example will display:

To learn more about inline elements, read [MDN documentation](https://developer.mozilla.org/en-US/docs/Web/HTML/Inline_elements).

The CSS display property provides the ability to make any element an inline element. This includes elements that are not inline by default such as paragraphs, divs, and headings.

h1 {

display: inline;

}

The CSS in the example above will change the display of all <h1> elements to inline. The browser will render <h1> elements on the same line as other inline elements immediately before or after them (if there are any).

#### Block Display

Some elements are not displayed in the same line as the content around them. These are called block-level elements. These elements fill the entire width of the page by default, but their width property can also be set. Unless otherwise specified, they are the height necessary to accommodate their content.

Elements that are block-level by default include all levels of heading elements (<h1> through <h6>), <p>, <div> and <footer>. For a complete list of block level elements, visit [the MDN documentation](https://developer.mozilla.org/en-US/docs/Web/HTML/Block-level_elements).

strong {

display: block;

}

In the example above, all <strong> elements will be displayed on their own line, with no content directly on either side of them even though their contents may not fill the width of most computer screens.

#### Inline-Block

The third value for the display property is inline-block. Inline-block display combines features of both inline and block elements. Inline-block elements can appear next to each other and we can specify their dimensions using the width and height properties. Images are the best example of default inline-block elements.

For example, <div>s in the CSS below will be displayed on the same line and with the specified dimensions:

<div class="rectangle">

<p>I’m a rectangle!</p>

</div>

<div class="rectangle">

<p>So am I!</p>

</div>

<div class="rectangle">

<p>Me three!</p>

</div>

.rectangle {

display: inline-block;

width: 200px;

height: 300px;

}

In the example above, there are three rectangular divs that each contain a paragraph of text. The .rectangle <div>s will all appear inline (provided there is enough space from left to right) with a width of 200 pixels and height of 300 pixels, even though the text inside of them may not require 200 pixels by 300 pixels of space.

### Float

So far, you’ve learned how to specify the exact position of an element using offset properties. If you’re simply interested in moving an element as far left or as far right as possible on the page, you can use the float property.

The float property can be set to one of two values:

1. left - this value will move, or float, elements as far left as possible.
2. right - this value will move elements as far right as possible.

.boxes {

width: 120px;

height: 70px;

}

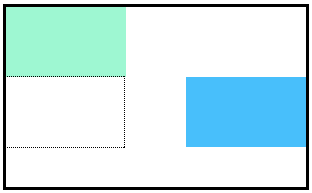
.box-bottom {

background-color: DeepSkyBlue;

float: right;

}

In the example above, we float the .box-bottom element to the right. This works for static and relative positioned elements. See the result of the code below:



Floated elements must have a width specified, as in the example above. Otherwise, the element will assume the full width of its containing element, and changing the float value will not yield any visible results.

### Clear

The float property can also be used to float multiple elements at once. However, when multiple floated elements have different heights, it can affect their layout on the page. Specifically, elements can “bump” into each other and not allow other elements to properly move to the left or right.

The clear property specifies how elements should behave when they bump into each other on the page. It can take on one of the following values:

1. left — the left side of the element will not touch any other element within the same containing element.
2. right — the right side of the element will not touch any other element within the same containing element.
3. both — neither side of the element will touch any other element within the same containing element.
4. none — the element can touch either side.

div {

width: 200px;

float: left;

}

div.special {

clear: left;

}

In the example above, all <div>s on the page are floated to the left side. The element with class special did not move all the way to the left because a taller <div> blocked its positioning. By setting its clear property to left, the special <div> will be moved all the way to the left side of the page.

## Color

CSS supports a wide variety of colors. These include *named colors*, like blue, black, and LimeGreen, along with colors described by a numeric value. Using a numeric system allows us to take advantage of the whole spectrum of colors that browsers support. In this lesson, we’re going to explore all the color options CSS offers.

Colors in CSS can be described in three different ways:

* *Named colors* — English words that describe colors, also called *keyword colors*
* *RGB* — numeric values that describe a mix of red, green, and blue
* *HSL* — numeric values that describe a mix of hue, saturation, and lightness

We’ll learn about and explore the benefits of each of these in depth. Using only named colors, you may feel like you’re picking labeled crayons out of a box. By the end of this lesson, you’ll feel like a painter mixing paints on a palette.

Before discussing the specifics of color, it’s important to make two distinctions about color. Color can affect the following design aspects:

1. The foreground color
2. The background color

Foreground color is the color that an element appears in. For example, when a heading is styled to appear green, the *foreground color* of the heading has been styled.

Conversely, when a heading is styled so that its background appears yellow, the *background color* of the heading has been styled

In CSS, these two design aspects can be styled with the following two properties:

1. color - this property styles an element’s foreground color.
2. background-color - this property styles an element’s background color.

h1 {

color: Red;

background-color: Blue;

}

In the example above, the text of the heading will appear in red, and the background of the heading will appear blue

### Hexadecimal colors

One syntax that we can use to specify colors is called hexadecimal. Colors specified using this system are called hex colors. A hex color begins with a hash character (#) which is followed by three or six characters. The characters represent values for red, blue and green.

DarkSeaGreen: #8FBC8F

Sienna: #A0522D

SaddleBrown: #8B4513

Brown: #A52A2A

Black: #000000 or #000

White: #FFFFFF or #FFF

Aqua: #00FFFF or #0FF

In the example above, you may notice that there are both letters and numbers in the values. This is because the hexadecimal number system has 16 digits (0-15) instead of 10 (0-9) like you are used to. To represent 10-15, we use A-F. [Here](https://developer.mozilla.org/en-US/docs/Web/CSS/color_value) is a list of many different colors and their hex values.

Notice that Black, White, and Aqua are all represented with both three characters and six characters. This can be done with hex colors whose number pairs are the same characters. In the example above, Aqua can be represented as #0FF because both of the first two characters are 0 and the second and third pairs of characters are both Fs. Keep in mind that all three character hex colors can be represented with six characters (by repeating each character twice) but the same is not true in reverse.

You can include hex colors just as you would include named colors: background-color: #9932cc;.

### RGB Colors

There is another syntax for representing RGB values that uses decimal numbers. It looks like this:

h1 {

color: rgb(23, 45, 23);

}

Here, each of the three values represents a color component, and each can have a decimal number value from 0 to 255. The first number represents the amount of red, the second is green, and the third is blue. These colors are exactly the same as hex, but with a different syntax and a different number system.

In general, hex and decimal color representations are equivalent. Which you choose is a matter of personal taste. That said, it’s good to choose one and be consistent throughout your CSS, because it’s easier to compare hex to hex and decimal to decimal.

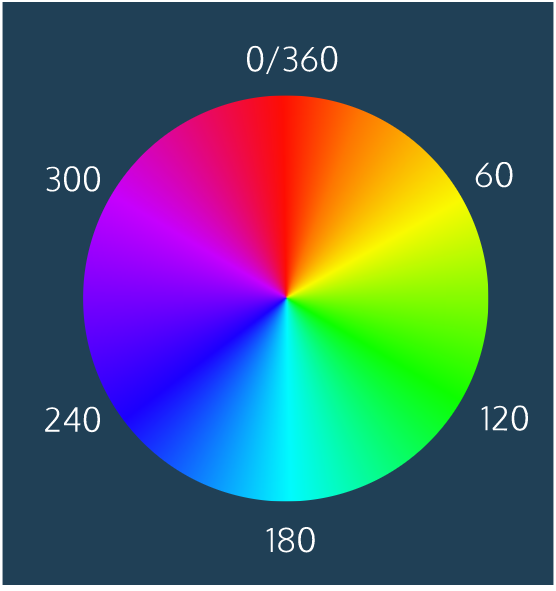
### Hue Saturation Lightness

The RGB color scheme is convenient because it’s very close to how computers represent colors internally. There’s another equally powerful system in CSS called the hue-saturation-lightness color scheme, abbreviated as *HSL*.

The syntax for HSL is similar to the decimal form of RGB, though it differs in important ways. The first number represents the degree of the hue, and can be between 0 and 360. The second and third numbers are percentages representing saturation and lightness respectively. Here is an example:

color: hsl(120, 60%, 70%);

*Hue* is the first number. It refers to an angle on a color wheel. Red is 0 degrees, Green is 120 degrees, Blue is 240 degrees, and then back to Red at 360. You can see an example of a color wheel below:



*Saturation* refers to the intensity or purity of the color. If you imagine a line segment drawn from the center of the color wheel to the perimeter, the saturation is a point on that line segment. If you spin that line segment to different angles, you’ll see how that saturation looks for different hues. The saturation increases towards 100% as the point gets closer to the edge (the color becomes more rich). The saturation decreases towards 0% as the point gets closer to the center (the color becomes more gray).

*Lightness* refers to how light or dark the color is. Halfway, or 50%, is normal lightness. Imagine a sliding dimmer on a light switch that starts halfway. Sliding the dimmer up towards 100% makes the color lighter, closer to white. Sliding the dimmer down towards 0% makes the color darker, closer to black.

HSL is convenient for adjusting colors. In RGB, making the color a little darker may affect all three color components. In HSL, that’s as easy as changing the lightness value. HSL is also useful for making a set of colors that work well together by selecting various colors that have the same lightness and saturation but different hues.

### Opacity and Alpha

All of the colors we’ve seen so far have been opaque, or non-transparent. When we overlap two opaque elements, nothing from the bottom element shows through the top element. In this exercise, we’ll change the *opacity*, or the amount of transparency, of some colors so that some or all of the bottom elements are visible through a covering element.

To use opacity in the HSL color scheme, use hsla instead of hsl, and four values instead of three. For example:

color: hsla(34, 100%, 50%, 0.1);

The first three values work the same as hsl. The fourth value (which we have not seen before) is the *alpha*. This last value is sometimes called the opacity.

Alpha is a decimal number from zero to one. If alpha is zero, the color will be completely transparent. If alpha is one, the color will be opaque. The value for half transparent would be 0.5.

You can think of the alpha value as, “the amount of the background to mix with the foreground”. When a color’s alpha is below one, any color behind it will be blended in. The blending happens for each pixel; no blurring occurs.

The RGB color scheme has a similar syntax for opacity, rgba. Again, the first three values work the same as rgb and the last value is the alpha. Here’s an example:

color: rgba(234, 45, 98, 0.33);

Alpha can only be used with HSL and RGB colors; we cannot add the alpha value to color: green color: #FFFFF.

There is, however, a named color keyword for zero opacity, transparent. It’s equivalent to rgba(0, 0, 0, 0). It’s used like any other color keyword:

color: transparent;

## Typography

### Font Family

If you’ve ever used a formatted word processor, chances are that you probably also used a feature that allowed you change the “type of font” you were typing in. The phrase “type of font” refers to the technical term [typeface](https://en.wikipedia.org/wiki/Typeface), or font family.

To change the typeface of text on your web page, you can use the font-family property.

h1 {

font-family: Garamond;

}

In the example above, the font family for all main heading elements has been set to Garamond.

When setting typefaces on a web page, keep the following points in mind:

1. The font specified in a stylesheet must be installed on a user’s computer in order for that font to display when a user visit the web page. We’ll learn how to work around this issue in a later exercise.
2. You’ve probably noticed that we haven’t been specifying a typeface in previous exercises of this course. How exactly does the browser know what typeface to use when displaying the web page? The default typeface for all most browsers is [Times New Roman](https://en.wikipedia.org/wiki/Times_New_Roman). You may be familiar with this typeface if you have ever used a formatted word processor.
3. It’s a good practice to limit the number of typefaces used on a web page to 2 or 3.
4. When the name of a typeface consists of more than one word, it must be enclosed in double quotes (otherwise it will not be recognized), like so:

h1 {

font-family: "Courier New";

}

### Font Weight

You’ve probably noticed **bold** text in websites you use, especially in news or text-heavy sites. It’s common to bold important headings or keywords. In CSS, we can style bold text with the font-weight property.

If we want to bold text in a web page, we can set the font-weight to bold.

p {

font-weight: bold;

}

If we want to ensure that text is not bold, we can set the font-weight to normal.

p {

font-weight: normal;

}

By default, the font-weight of most text elements is set to normal. Some elements, like headers, have built-in bold styling. A good approach is to check to see if the text element has any default styling, and use the font-weight property accordingly.

The font-weight property can also be assigned a number value to style text on a numeric scale ranging from 100 to 900. Valid values are multiples of 100 within this range such as 200 or 500.

When using numeric weights, there are a number of default font weights that we can use:

1. 400 is the default font-weight of most text.
2. 700 signifies a bold font-weight.
3. 300 signifies a light font-weight.

Let’s take a look at an example of how numeric fonts are used.

header {

font-weight: 800;

}

footer {

font-weight: 200;

}

Here, the header would appear as a deep bold, while the footer would appear rather light.

It’s important to note that not all fonts can be assigned a numeric font-weight. You can look up the font you are using to see which font-weight values are available.

### Font Style

You can also italicize text with the font-style property.

h3 {

font-style: italic;

}

The italic value causes text to appear in italics. The font-style property also has a normal value which is the default.

### Word Spacing

You can also increase the spacing between words in a body of text, technically known as word spacing.

To do so, you can use the word-spacing property:

h1 {

word-spacing: 0.3em;

}

The default amount of space between words is usually 0.25em. In the example above, the word spacing is set to 0.3em, which represents an increase of only .05em in word spacing.

It’s not common to increase the spacing between words, but it may help enhance the readability of bolded or enlarged text. Note, again, that the preferred unit is ems.

### Letter Spacing

You’ve learned how to increase the spacing between lines of text and words, but it’s possible to get even more detailed: increasing the spacing between individual letters.

The technical term for adjusting the spacing between letters is called tracking. Tracking can be adjusted with the letter-spacing property in CSS.

h1 {

letter-spacing: 0.3em;

}

Like word spacing, it’s not common to increase the tracking in text, but sometimes it enhances the readability of uppercase text.

### Text Transformation

Text can also be styled to appear in either all uppercase or lowercase with the text-transform property.

h1 {

text-transform: uppercase;

}

The code in the example above formats all <h1> elements to appear in uppercase, regardless of the case used for the heading within the HTML code. Alternatively, the lowercase value could be used to format text in all lowercase.

Since text can be directly typed in all uppercase or lowercase within an HTML file, what is the point of a CSS rule that allows you to format [letter case](https://en.wikipedia.org/wiki/Letter_case)?

Depending on the type of content a web page displays, it may make sense to always style a specific element in all uppercase or lowercase letters. For example, a website that reports breaking news may decide to format all <h1> heading elements such that they always appear in all uppercase, as in the example above. It would also avoid uppercase text in the HTML file, which could make code difficult to read.

### Text Alignment

No matter how much styling is applied to text (typeface, size, weight, etc.), text always appears on the left side of the browser.

To move, or align, text, we can use the text-align property.

h1 {

text-align: right;

}

The text-align property can be set to one of the following three values:

1. left - aligns text to the left hand side of the browser.
2. center - centers text.
3. right - aligns text to the right hand side of the browser.

Later in the course, you’ll learn exactly how the browser positions HTML elements by default, which will help you understand how the browser “aligns” text, since “align” is a relative term. For now, it’s enough to know that text can be moved to the left, center, or right side of the web page.

### Line Height Anatomy

Another property that we can set for text is line-height. This property modifies the leading of text.

The diagram to the right helps illustrate exactly what the terms “leading” and “line height” mean.



We often modify line-height to make text on a web page easier to read. When text is styled to appear larger, the vertical spacing between lines of text can decrease, creating text that is difficult to read, particularly in paragraphs.

We can use the line-height property to set how tall we want the line containing our text to be, regardless of the height of the text. Line heights can take one of several values:

1. A unitless number, such as 1.2. This number is an absolute value that will compute the line height as a ratio of the font size.
2. A number specified by unit, such as 12px. This number can be any valid CSS unit, such as pixels, percents, ems, or rems.

Generally, the unitless ratio value is the preferred method, since it is responsive and based exclusively on the current font size. In other words, if we change the font size, a unitless line-height would automatically readjust, whereas the pixel value would remain static.

p {

line-height: 1.4;

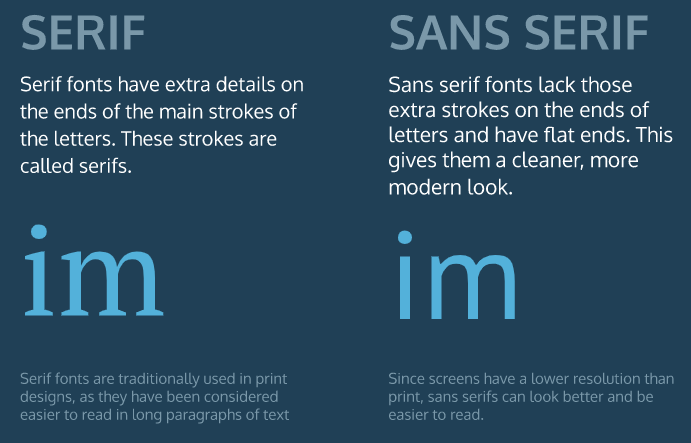
}

### Serif and Sans Serif

You’ve learned a lot of properties to modify text on a web page!

In the next exercise, you’ll set some text to be serif and some text to be sans-serif. What exactly do these words mean?

1. Serif — fonts that have extra details on the ends of each letter. Examples include fonts like Times New Roman or Georgia, among others.
2. Sans-Serif — fonts that do not have extra details on the ends of each letter. Instead, letters have straight, flat edges, like Arial or Helvetica.



### Fallback Fonts

What happens when a stylesheet requires a font that is not installed on a user’s computer? Most computers have a small set of typefaces pre-installed. This small set includes serif fonts like Times New Roman and sans-serif fonts like Arial.

These pre-installed fonts serve as *fallback fonts* if the stylesheet specifies a font which is not installed on a user’s computer.

To use fallback fonts, the following syntax is required:

h1 {

font-family: "Garamond", "Times", serif;

}

The CSS rule above says:

1. Use the Garamond font for all <h1> elements on the web page.
2. If Garamond is not available, use the Times font.
3. If Garamond and Times are not available, use any serif font pre-installed on the user’s computer.

The fonts specified after Garamond are the fallback fonts (Times, serif). Fallback fonts help ensure a consistent experience for the diverse audience of users that visit a site.

### Linking Fonts

With the number of fonts available with modern typography, it is unrealistic to expect users to have all fonts installed on their computers. New fonts are often centralized in directories made available for public use. We refer to these fonts as non-user fonts.

[Google Fonts](https://fonts.google.com/) is one such directory of thousands of open-source fonts, available for free use. Google Fonts gives us a way to retrieve the link for a single font, multiple fonts, or multiple fonts with the font-weight and font-style properties.

When we have the link to the font of our choice, we can add the font to the <head> section of the HTML document, using the <link> tag and the href.

Let’s take a look at a few examples:

1. A single linked font, using Droid Serif as an example:

<head>

<link href="https://fonts.googleapis.com/css?family=Droid+Serif" type="text/css" rel="stylesheet">

</head>

2. Multiple linked fonts, using the Droid Serif and Playfair Display fonts as an example:

<head>

<link href="https://fonts.googleapis.com/css?family=Droid+Serif|Playfair+Display" type="text/css" rel="stylesheet">

</head>

3. Multiple linked fonts, along with weights and styles. Here Droid Serif has font weights of 400, 700, and 700i, while Playfair Display has font weights of 400, 700, and 900i:

<head>

<link href="https://fonts.googleapis.com/css?family=Droid+Serif:400,700,700i|Playfair+Display:400,700,900i" rel="stylesheet">

</head>

Once a font is linked, we can create CSS selectors to target elements, just as we do with other fonts.

### Font-Face

There are other ways to link non-user fonts that don’t require the use of the <link> tag in the HTML document. CSS offers a way to import fonts directly into stylesheets with the @font-face property.

To load fonts with the @font-face property:

1. Instead of using the font’s link in the HTML document, enter the link into the URL bar in the browser.
2. The browser will load the CSS rules. You will need to focus on the rules that are directly labeled as /\* latin \*/. Some of the latin rules are on separate lines. You will need each of these.
3. Copy each of the CSS rules labeled latin, and paste the rules from the browser to the top of **style.css**.

It is important to stress the need to copy the @font-face rules to the top of the stylesheet for the font to load correctly in the project.