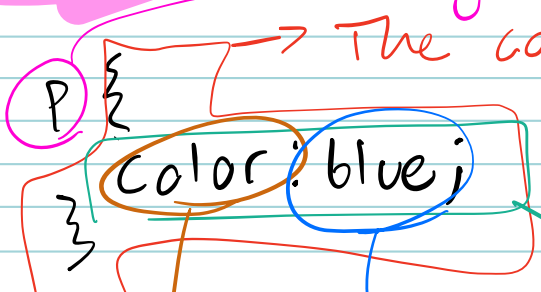
CSS

Lesson 1 Setup and Syntax

- CSS stands for cascading style sheets and is a language used to style HTML content
- There are two ways to write CSS code

- Both ways contain common features in their anatomy
  - such as declaration
    - declarations are the core of CSS
    - They apply a style to a selected element

Ruleset



- This is the selector used to target what element will be styled
- The code between the curly braces that contain styling (The declaration block)
- A certain

style applied (The declaration

The value of the declaration which is the value of the property

The property which is the first part of the declaration which signifies which visual characteristic is being manipulated

CSS inline style

property

 Hello World

value

declaration

This line is an HTML line and style is an attribute of CSS

Opening tag of the HTML element

- So, inline styles is when you write CSS in HTML
  - If you want multiple things to happen to an element then you need multiple semi colons within the HTML style attribute

Ex:

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

- Internal style sheet
  - You can avoid making a CSS file by using a <style></style> tag inside the head tag of an HTML file
  - Inside we can create normal looking CSS

Ex:

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

- External Style Sheet Link
  - In order to attach a CSS file to an HTML file we'll need to have a <link> tag inside the head
  - Link is a self closing tag and requires the following attributes:
    1. -href → the value of this attribute must be the address/path to the CSS file
    2. -rel → This attribute describes the relationship between the 2 files and is set to = "Style Sheet"

Ex:

```
<link href='./style.css'
rel='stylesheet'>
```

→ Inside head

Lesson 2 Selectors

- Type
  - So we know what the selector is in the CSS code which is on the begining of a declaration
  - We can set this part of CSS to a specific HTML type such as <p>

Ex:

```
p {
  color: green;
}
```

  - This I believe would set all paragraph types in the HTML file green
  - This could also be refered to as the tag name or element selector
- Universal
  - The universal selector selects all elements of any type
  - The use cases of this idea is strange, so don't worry about that right now

- The * is the universal selector

Ex:

```
* {
  border: 1px solid red;
}
```

• Class

- A common way to target HTML stuff is to have HTML elements with attribute class with your choosing of a value

Ex: In HTML

```
<p class='brand'>Sole Shoe
Company</p>
```

In CSS

```
.brand {

}
```

- We can add multiple classes to a single HTML attribute with simply adding a new word

Ex: class = "brand bold green"

(This element would now belong to 3 classes

• ID

- So say we want to style just one tag in HTML, well we'll be able to do this by giving that tag an id attribute
- Then just like with anchors we use a # in the CSS file to describe the id attribute:

Ex:
```
#id-name{
}
```

• Attribute

- So in CSS we can even describe every HTML tag that has an attribute because in CSS we can select the attribute

- We would do this with brackets

Ex:

```
[href]{
   color: magenta;
}
```

- so attributes have values right, we can even describe specific attributes with certain values with the syntax in css as type [attribute*=value]

Ex:

```
<img
src='/images/seasons/cold/winter.j
pg'>
<img
src='/images/seasons/warm/summer.j
pg'>
```

HTML

```
img[src*='winter'] {
  height: 50px;
}

img[src*='summer'] {
  height: 100px;
}
```

CSS

- Pseudo-Class
  - Selectors in action! situations like this could be like
    - clicking on a link and it turns purple
    - The submit button is grayed out until all the elments are filled
  - These are examples of pseudo classes such as
    - :focus
    - :visited
    - :disabled
    - :active
  - A pseudo class can be attached to any selector. It is always written as a colon : followed by a name

Ex: for example once you hover over a h1 here it'll turn red!

```
h1:hover {
    color:red;
}
```

- Classes and IDS
  - CSS classes are meant to be reused over many elements
  - Also IDs override the styles of types and classes

- Specificity
  - The order by which the browser decides

which css styles will be displayed
- A commonality in CSS is to style elements using the lowest degree of specificity to easily override it

Order from most specific to least

- ID
- class
- type

- Chaining
  - We can require an HTML element to have two or more CSS selectors at the same time
  - Basically if we want to choose type + class elements such as

Ex: h1s which have a class of "special"

```
h1.special {

}
```

- Descendent Combinator
  - So say we have a parent html tag and its childrens we want to do stuff with, we can combine the parent's class with the child's type

Ex:

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

HTML

```
.main-list li {

}
```

CSS

- Chaining Specificity
  - The more specific shit is will win so like .main p beats p in CSS

* You can be specific in ways such as "All the h4 within li's" with CSS selector

```
li h4 { ...
```

- Multiple selectors, so you can put commas in the selectors to have more than 1

Other → This is my div element with the class headline

```
div.headline {
  width: 300px;
  margin: 0 auto;
}
```

Lesson 3 Visual Rules

- Here we will learn about the fundamental visual changes we can make with CSS

- Font Family
  - Font in CSS refers to the technical term typeface or font family
  - To change the typeface in the website you'll need to use the font family property

Ex:
```
h1 {
    font-family: baramond;
}
```

  - Keep in mind that the font must be on the user's computer or downloaded with the site
  - Web safe fonts are a list of fonts covered by most os and browsers
  - When the font is more than 1 word its best practise to put in quotation marks

- Font size
  - To change the sizes of our font we use

font-size

Ex:
```
p {
  font-size: 18px;
}
```

- Font Weight
  - font weight is how bold or thin some text appears

Ex:
```
p {
  font-weight: bold;
}
```
- Another value for font-weight is "normal" which is basically not bold

- Text Align
  - Text will always appear on the left of its container, to align text we would use the text-align property
  - Values of text-align could be
    - left
    - center
    - right
    - justify → spaces out text to align with right and left side of container

What?

- Color and Background Color
  - There are two types of color:

Ex: h1 {

```
color: red;
background-color: blue;
}
```

- Opacity
  - The measure of how transparent an element is. This is measured from 0-1 with 1 being 100% aka fully visible and opaque and 0 being 0% or invisible

EX:
```
img {
  opacity: 0.5;
}
```

- If something is behind something else that has opacity then the thing behind can be seen

- Background Image
  - We can make the background of something an image with the background-image

EX:
```
.main-banner {
  background-image:
url('https://www.example.com/image
.jpg');
}
```

↙ The url could have diff syntax for internal files

- Important
  - !important is syntax that can be applied to specific declarations
  - This will override any style no matter the specificity
  - This means it should almost never be used

EX:
```
p {
  color: blue !important;
}

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

→ here all p elements will be blue even if a more specific → p here says it'll be red

- We can use the important property to override stuff when we use multiple style sheets...

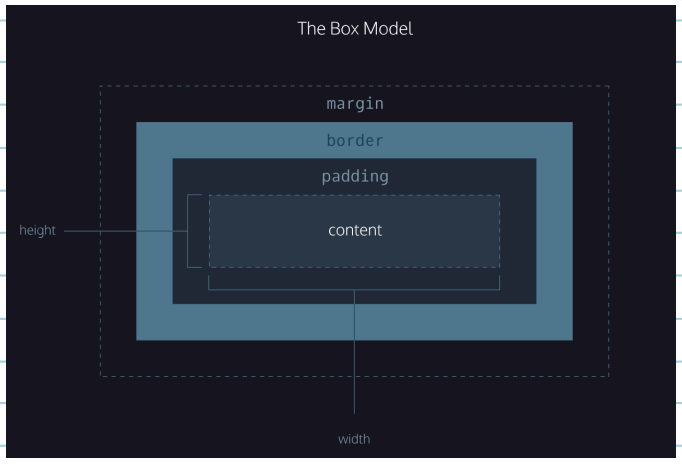
Lesson 4: The Box Model

Lesson 9 The Box Model

- Box model is a concept to understand how elements are positioned on a page
- We will answer questions to why the background color of an element effects the side or relative to what is a text centered?
- All elements on a webpage are living in a box
- This lesson will teach us about
  - dimensions
  - borders
  - paddings
  - margins

The Box Model

- The box model is the set of properties that define parts of an element that take up space on a webpage
- This model includes
  - dimensions → width and height
  - borders → The thickness and style of border
  - paddings → The amount of space between the content area and border
  - margins → The amount of space between border and outside edge of the element



Height and Width

- We can difine a css declaration to have width or height be in px

Ex:
```
p {
  width: 10px;
}
```

- When we define height or width in pixels, it will be the same on

all devices

- An element that fills a laptop screen will overflow a mobile device

- Borders
  - Borders can be a specific width, style and color
    - width
      - The thickness of a border can be set in px or "thin", "medium" or "thick"
    - style
      - The design of the border
      - There are a couple types and some include "dotted", "none" and "solid"
    - Color
      - Yeah

Ex:

```
p {
  border: 3px solid coral;
}
```

- The default value for border is "medium none color" where color is the current color of the element

- so, if I didn't put 3px here it's default value would be medium

- Border Radius
  - so apparently you can add another declaration called border radius to make the border more circular
  - A border would be a perfect circle if the width = height and set border radius to 50%.

Ex:

```
div.container {
  height: 60px;
  width: 60px;
  border: 3px solid blue;
  border-radius: 50%;
}
```

Padding

- The space between the box content and the borders
- We can add our own padding with the padding declaration in px
- This padding property is mainly used to expand the background color and make the content look less cramped
- You can be more specific with padding using the following declarations:
  - `padding-top`
  - `padding-right`
  - `padding-bottom`
  - `padding-left`

- We could write out each padding side individually or we could shorthand property

Ex:

```
p.content-header {
  padding: 6px 11px 4px 9px;
}
```

top right bottom left

- we could also write only 2 px amounts where the first would stand for top and bottom while the 2nd will stand for right and left

Margin

- The margin property is used to specify the size of the space outside the box
- Given this code: we will place 20px of space on the outside of the

```
p {
  border: 1px solid aquamarine;
  margin: 20px;
}
```

paragraphs box on all 4 sides, that means other elements can't come near

- just like padding, margin has top, right, bottom and left specific declarations
- Margins also have margin shorthand which is the same as padding
- something cool is 3 values would account for top, left + right, bottom

• Auto
- The margin property can let us center content

EX:

```
div.headline {
  width: 400px;
  margin: 0 auto;
}
```

This line has auto which instructs the browser to adjust the left and right margins until the element is centered within its containing element

Wait whats going on?

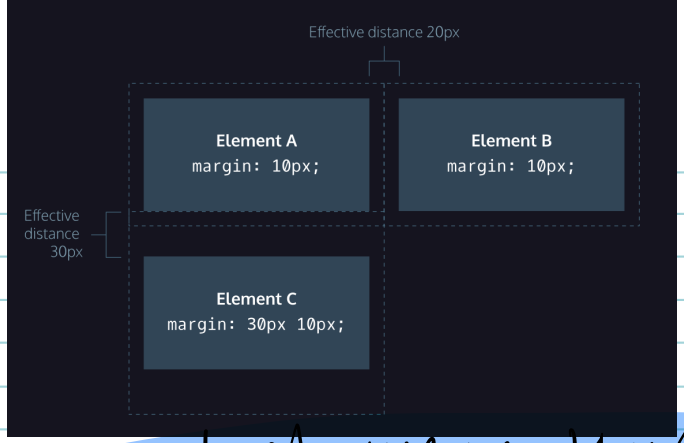
• Margin Collapse
- So remember padding is space added inside an elements border while margin is space added to the outside
- Another difference is that top and bottom margines aka vertical margins collapse while top and bottom padding doesn't

- So this basically means that you know when you have 2 elements next to each other with margins, the total margin inbetween them is both of them combined

while vertical margins on the other hand don't add, the bigger one dominates the other

This idea is collapsing

Ex:



Minimum and Maximum Height and Width

- since webpages are diff on diff screens css has limits on things:

- for width, min-width ensures minimum width of an elements box while max-width ensures a maximum width of an elements box

Ex:

```
p {
  min-width: 300px;
  max-width: 600px;
}
```

→ This will ensure that text stays legible same for height when you shrink the browser!

Overflow

- so sometimes parent elements don't contain enough space for children so we'll need to use the overflow property

- This property controls what happens to content that spills outside the box

- common values for overflow is
  - hidden
    - overflow will just be hidden
  - scroll
    - a scroll bar will be given to the elements box
  - visible
    - shit will just spill

Ex:

```
p {
  overflow: scroll;
```

- so overflow is supposed to be used on parents to instruct how to render child elements

- Resetting Defaults
  - So, all webpages have default stylesheets they use in absence of an external stylesheet
  - These default style sheets are known as user agent stylesheets

problem
  - These user agent style sheets have default padding and margin stuff which could complicate stuff
  so...

solution
  - We'll reset our defaults with:

```
* {
  margin: 0;
  padding: 0;
}
```

  - The code above is often the first rule in external style sheets

- Visibility
  - Elements can be hidden from view with the visibility property
  - The values for this property are
    - hidden
    - visible
    - collapse - This also hides space of where element would be

EX:

```
.future {
  visibility: hidden;
}
```

↓

display: none; also does this

- Also, if stuff is hidden from the user they can still see the source code stuff

Lesson 5 Changing Box Model

problem

- so, the box model is whack because when we have an element it would add the px of the border, height and padding to an awkward and difficult to manage number

solution:

change it

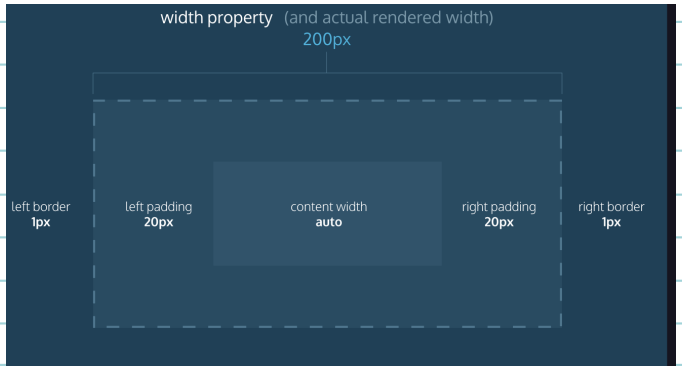
- Box Model - Content Box
  - Many values already have a default value in a stylesheet
  - In css the box-sizing property controls the type of the box model the browser should use when interpreting a webpage
  - The default value for this property is content-box which is the box effected by stuff
- Box Model - BorderBox
  - so, we can change our box-sizing property to border-box:

ex

```
* {
  box-sizing: border-box;
}
```

- This box model says no to the issues from before
- The idea here is we'd change the size of the content since

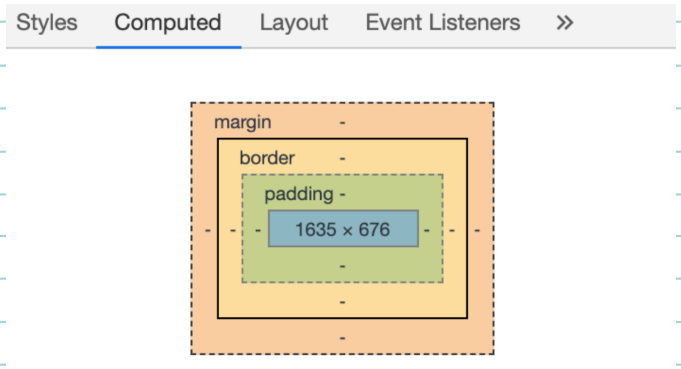
the declared width and height will be fixed so only content size changed:



Article: Box Model in Dev Tools

- dev tools is that f12 display
- You open it on mac with "command + option + i"
- You can click on what you want to inspect and click inspect
- Then you can look at computed section in dev tools:

↳



-You can make changes in dev tools to test stuff out...

Lesson 6: Display and Positioning

- Flow of HTML
  - -A browser will render the elements of an HTML document like normal from top to right
  - -CSS includes properties that change how a browser positions elements
  - -This means where the element

on the page, sharing lines etc

- This lesson will cover 5 properties for adjusting properties
  - position
  - display
  - z-index
  - float
  - clear

Position

- The position property can take one of 5 values:
  - static → Default behavior
  - relative
  - absolute
  - fixed
  - sticky

- Relative

- This value for position requires an accompanying offset property that will move the element away from its static position
  - top → moves down from the top
  - bottom → moves up from bottom
  - left → moves to right
  - right → moves to left
- We can have offset values be in px, ems or percentages or more

Ex:

```
.green-box {
  background-color: green;
  position: relative;
  top: 50px;
  left: 120px;
}
```

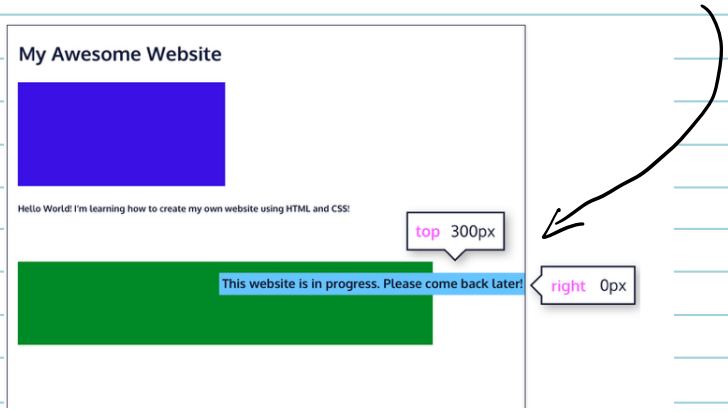
→ This green box will be moved down 50px and to the right 120px

- Absolute

- When an element is absolute all

other elements will ignore the element and act like its not on the page
- so it seems like this would be used for putting elements above other elements

Ex:



- Fixed
  - So we can have an element stay with the user as they scroll down by setting its position to fixed and accompanying it with offset properties
  - A typical offset property pair is top 0px and left 0px and I'd assume right 0px as well

Ex:

```
.title {
  position: fixed;
  top: 0px;
  left: 0px;
}
```

- If we only have the above code however it will block whats at the top of the page behind the fixed element
- But its still not working properly...

At this point our nav is fixed and our welcome under that is relative to make space

- Sticky
  - Okay wait so apparently this

guy sticks the element to the screen as the user scrolls

Ex:

```
.box-bottom {
  background-color: darkgreen;
  position: sticky;
  top: 240px;
}
```

- Nah the code is fucked up on codecademy, but its okay as long as we get the general idea

• Z-index

- So this element controls which element is on top of another aka the depth
- The z-index does not work on static elements
- The value of a z-index is an integer
- The less the number the further back it will be to another element which also has a z-value or not
- The default value is 0 for my element

Now that our nav has greater z-value it stays on top

• Inline display

- So display has 3 values
  - inline
  - block
  - inline block
- Display is the property that dictates if it can share horizontal space
- The default display for some elements is called inline and is when a box wraps tightly around its content

which only takes up nessesary space

- Inline elements cannot be altered by the CSS height and width properties

EX

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

- so, I'm a bit confused on what exactly this inline stuff is, but we can say this em element is inline because it displays its content on the same line as the content surrounding it

- display: inline; makes any element line and here:

EX

```
h1 {
  display: inline;
}
```

→ The brawser will render h1 elements on the same line as other inline elements immediately before or after

Display Block

- Block level elements are not displayed in the same line as the content around them

- These elements fill the entire page by default, but their width property can also be set

- Examples of block levels by default are headers, paragraphs divs and more

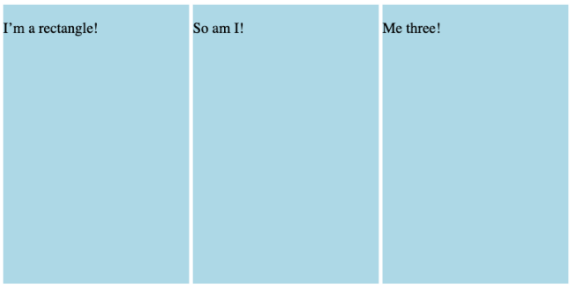
```
strong {
  display: block;
}
```

EX:

Display: Inline-Block

- Display: Inline-Block
  - This display combines both inline and Block ideals
  - These displays can be next to each other and we can specify their dimensions using width and height

Ex: output:



HTML:

```
<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>
```

CSS:

```
.rectangle {
  display: inline-block;
  width: 200px;
  height: 300px;
}
```

- These divs appear next to each other with 200px x 300px

- Float
  - If you want to set an element as far left as possible in a container, we can use the float property
  - The float property is used for wrapping text around an image
  - However moving elements left

or right is better suited for tools like CSS Grid and flexbox

- The values for float are left or right
  - left: moves element as far left as possible
  - right: moves elements as far right as possible
- Floated elements must have a width specified

-Clear

- The float can be used on multiple elements at once but when they have different heights it'll be weirdly laid out
- This situation is where the clear property comes in and this can take different values
  - left
    - The left side of the element will not touch any other element within the same containing element
  - right
  - both
  - none
    - The element can touch either side

EX: In the code below the special didn't move all the way to the left because of another div, so clear div left through

```css
div {
  width: 200px;
  float: left;
}

div.special {
  clear: left;
}
```

Lesson 7 Color

- We'll talk about colors, and 3 ways to describe colors is
  - named colors - like "blue"
  - RGB - red, green, blue
  - HSL - hue, saturation, lightness

• Hex Colors

- We can specify colors with hexadecimal
- They represent values for red, blue and green in 2 places

EX: #A52A2A

(A52A2A → r, b, g)

- Sometimes colors are like having the same num for the 1s and 16s place of a num, so if all 3 of rgb is like this we can just use 3 nums

EX: #000000 → #000

- Letters can be upper or lowercase

• RGB

- We could also write rgb values in the form:

EX:

```
h1 {
  color: rgb(23, 45, 23);
}
```

- Values must be from 0 -> 255

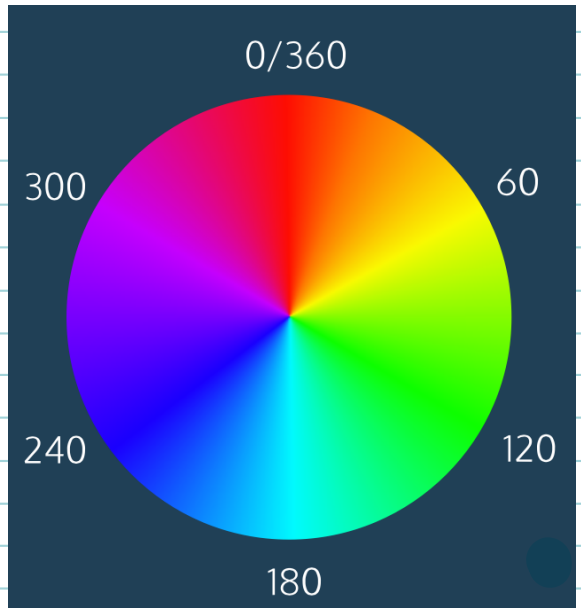
- HSL

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

the degree of the hue

percent of saturation (intensity of color)

percent of lightness



- There is also this thing about opacity but I do not really care...

slide 7/8

Lesson 7 Typography

- Typography is the art of arranging text on a page

Font family

-When we have multi word fonts we want to put it all in ' ' marks.

Ex:

```
h1 {
  font-family: 'Times New Roman';
}
```

-You can have fall back fonts which means that if your font is not available then it will fall back on something else

Ex

```
h1 {
  font-family: Caslon, Georgia,
'Times New Roman';
}
```

There, if Caslon isn't available then Georgia or TNR will work

Ex:

```
h1 {
  font-family: Caslon, Georgia,
'Times New Roman', serif;
}
```

If nothing is available it'll use serif font on system

Font Weight

-We know that this controls how bold or thin some text is

-We can use numerical values for this which is 1 - 1000

Ex:

- Btw font weights might not work all the time

```css
.left-section {
  font-weight: 700;
}

.right-section {
  font-weight: bold;
}
```

Font Style

- Doesn't say anything other than that we can make text italic

```css
h3 {
  font-style: italic;
}
```

Text Transformation

- We can make text all uppercase or all lowercase with the text transform property:

```css
h1 {
  text-transform: uppercase;
}
```

Text Layout

- Now we'll learn how text can be laid out within an element is container

Letter Spacing

- The letter-spacing property is used to set the horizontal spacing between characters in an element
  - We would use this for words in cases where the font is weird and we need to change the spacing a bit

Ex:

```
p {
  letter-spacing: 2px;
}
```

- There is another idea like this, but for words
- You may need that to help the readability of bolded or big text

- You could use px or em

Ex:

```
h1 {
  word-spacing: 0.3em;
}
```

Line Height

- The line height property changes you guessed it, and can be set in unitless ratios such as 1.2 or lengths in px, %, or em

Ex

```
p {
  line-height: 1.4;
}
```

Text Alignment

Ex

```
h1 {
  text-align: right;
}
```

Web Fonts

- Something cool is you can take free fonts from google fonts or something and link it to your HTML file
- We'd search for a font and get the link, then in our HTML code we'd

write

Ex:

```
<head>
  <!-- Add the link element for
Google Fonts along with other
metadata -->
  <link
href="https://fonts.googleapis.com
/css2?
family=Roboto:wght@100&display=swa
p" rel="stylesheet">
</head>
```

- Then you can simply go to your css file and use the font

Ex

```
p {
  font-family: 'Roboto', sans-
serif;
}
```

- Web fonts can also be introduced using @font-face
- fonts come in different files like
  - OTF
  - TTF
  - WOFF
  - WOFF2

  * These are in order of standards being the most compatible with WOFF2
- After you download your font files you can then use the @font face rule set

Ex:

```
@font-face {
  font-family: 'MyParagraphFont';
  src: url('fonts/Roboto.woff2')
format('woff2'),
       url('fonts/Roboto.woff')
format('woff'),
       url('fonts/Roboto.ttf')
format('truetype');
}
```

You then use this name when defining your font-family