Kalob.io

CSS Masterclass

By Kalob Taulien

Table of Contents

Introduction

Hello, welcome to the CSS Masterclass by Kalob.io. In this course we'll be focusing on, and learning, CSS and CSS3 from the ground up, including responsive design.

We suggest you know some HTML, but it's not a requirement to complete this course.

You'll learn everything you need to know about CSS, you'll get live examples, quizzes, the source code from each lesson, helpful resources and projects to practice on.

Whether you're looking to become a better front end web developer, or you're interested in becoming a front end developer, this course is for you.

About The Author

I had initially planned on leaving this part out, after all, this is a book (or course if you're taking the CSS Masterclass on Kalob.io) about learning to code. But there have been a few people who have asked me to share my credentials prior to getting started with CSS.

My name is Kalob Taulien, I'm a 28 year old web developer, teacher and entrepreneur. My first website was in 1999 and it was ugly. All websites back then were ugly. But each developer helped turn the internet into what it is today. There weren't tablets or mobile browsers. Laptops weren't even popular yet. In fact, there were just a couple browsers and Google didn't take over the world yet. It was a simpler time.

Since then I've made hundreds of websites, freelanced, worked as a contractor and as an employee. I've started web companies, been brought to other countries to code, travelled the world and helped build web services that have grossed over USD $1,000,000/year. My background is a full stack web developer (I can write code for browsers and servers). And I'm currently employed as a front end web developer in a web development company that creates award winning websites.

My first course was The Complete Web Developer course (which spawned the name "The Complete _____ Course" name trend), and have taught over 110,000 people worldwide. That course, however, is old (launched in late 2012) so I've been slowly making more courses, writing books, and I've been coaching junior web developers. I've been spending the last couple years gathering data on *what* people want to learn and *how* they prefer to learn it. And now I'm on a mission to help educate new web developers *properly.* There are a lot of new web developers who have been taking bad advice from other junior devs who have very little experience in the web development industry. That's like taking money advice from a homeless person -- it *might* be good, but that's highly improbable.

In the last 18 years of web development I've seen trends come and go. I've seen good technologies die and bad technologies win. I've seen the majority of the internet evolve from ugly plain-text websites, to the design marvels they are today. So you see, I'm a great resource to learn from -- so please, mine my brain for all the information you can absorb!

Getting the most from this course

Before we begin we'd like to establish a few guidelines for getting the most of out this course so you'll walk away knowing more about CSS in the standard amount of time.

Example Code

Throughout this course you'll find links and references to example code. This course has all its sample code on CodePen.io. The examples can be modified and played with. **Please use these**. Don't be afraid to wreck the code, it's there for you to experiment and play with.

Don't Just Copy & Paste

This course will give you a lot of code you can just copy and paste, and with live example code it's really easy to just copy and paste bits of code into your project. **Please do not do this**. Copying and pasting is convenient, but it's detrimental to your learning experience. You *need* to write out examples on your own. Writing code the long way, instead of copying the code, will help you with working memory (because you're experiencing the code) and helps with muscle memory -- which lets you remember how to code without having to think about it too much.

Do The Quizzes and Projects

It's easy to skip over quizzes and projects, and it's easy to say "I'll do it later." But **don't skip them!** Quizzes and in-video questions help you learn through a process called Recall, which makes you think really hard about what you just learned and solidifies the knowledge in your brain. And the projects will give you experience with the code -- because knowing how things *should* work versus knowing how things *actually work* are drastically different. Don't be that person that thinks they know everything, but has limited experience.

This only applies to the Kalob.io CSS Masterclass course. If you're reading through the documents instead of talking the course, you'll miss out on the vital learning experience that'll make you a great developer.

Ask Questions

If you run into problems, feel free to ask questions in the Learning to Code Facebook Group. If there are already questions, definitely read those first -- someone might have asked your question already.

Proper Etiquette

When asking or answering a question, *please* be kind to your fellow colleagues and *please* try to use proper sentence structure while being clear about your question. Questions or comments that are hard to read or impossible to understand may be marked as spam by your colleagues or your teachers.

Don’t Binge

Many people will want to “binge” a course. In other words, they think it’s more efficient to take this entire course in just one day. **Don’t do that.** You won’t *actually* learn the content in this course by just watching videos, or reading this e-book, in one sitting.

Take your time, practice for 25 minutes every day, and within a couple weeks you’ll be *very* good at CSS. Any other course that tries to sell you “Learn a new skill in 1 hour” is lying to you, our brains don’t work that way.

Just remember, slow and steady wins the race. That’s especially true when learning a new skill.

What is CSS

CSS stands for Cascading Style Sheets. It's the *only* way to add styling such as a color, decorations, different font sizes, weights and variants, and with the power of CSS3 you can even add animations, which we'll cover in this course.

Basically, CSS is how your website looks -- it's the design for your website.

CSS is considered a Markup Language. It is *not* a programming language. It's used on every website today. And there are no alternatives to writing CSS -- in order to make a beautiful website, you *need* to know CSS.

The more you know, the better equipped you'll be to secure a job as a web developer, to start freelancing, or to increase your net worth as a developer and ask for a raise (or charge more as a freelancer)

In this CSS Masterclass course by Kalob.io, you'll learn everything you need to know from the ground up. You will need to know some basic HTML, but since CSS can overwrite every property of a static HTML page, you don't need to know very much right now, and by the end you'll know how to make a beautiful and expertly looking website.

Let's dig in!

How to add CSS to your page

Before we get started we need to know how to add CSS to our web pages. There are 3 methods we can use and you'll learn all 3, but we'll be sticking with just 2 of these methods throughout the rest of this course.

External CSS

The first, and best, way to add CSS to your web page is through an HTML `<link />` tag. Here's an example:

```
<!DOCTYPE html>
  <head>
    <title>....</title>
    <link type="text/css" rel="stylesheet" href="http://yourwebsite.com/style.css"
/>
  </head>
...
```

This method lets you create a .css file, attach it to your web page (or several web pages) and any changes you make can take immediate effect on every page that uses this method.

Internal CSS

The second method, while less preferred, is a good way to add styling to one specific page. Ideally, you'd use this method for rendering any CSS above the fold.

> *"Above the fold" is the part of the website that you see as soon as it loads -- if you scroll down at all, everything below this section is "below the fold".*

Internal CSS is wrapped in a `<style></style>` HTML element and can be anywhere on your page, but should ideally be in your <head>. Below is an example:

```
<head>
  <style>
    ...
  </style>
</head>
```

Inline CSS

The third, and less preferred method, is the inline your CSS in your html elements with the style attribute. The reason we don't like this method is because it's not reusable, we can't use pseudo selectors (more on that later) and it makes the web page file size larger than it needs to be. Long story short: you end up writing more code and slowing down how fast the page can be loaded.

This is the method we'll generally be avoiding in this course. With that said, this is a very quick way to test some of your CSS. Many developers use this method to quickly test basic styling, and when they're finished they'll immediately put the styling into an external CSS file.

```
<body>
  <h1 style="color: red; background-color: blue;">
    Sample Inline Styling
  </h1>
</body>
```

Selectors

CSS is a markup language that lets you select which HTML elements to style, and how they should appear. There are several different ways to select an HTML element, including, but not limited to:

- It's HTML tag
- It's class attribute
- It's ID attribute
- Child elements
- Sibling elements
- First of it's kind in a group
- … and more.

Right now we're only going to cover the 3 major selectors: tag, class and ID.

Selecting by Tag

```
<body>
  <style>
    h1 {
      color: red;
    }
  </style>
</body>
```

The *h1* in the code above is a tag selector. It selects all <h1> tags in your page.

Likewise, if you wanted to select a paragraph <p> tag, you'd write:

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

The CSS tag selector will always match the exact HTML tag.

Selecting by Class

Instead of selecting an HTML element, you can select by it's class attribute.

```
<body>
  <style>
    .red { color: red; }
  </style>
  <p class="red">Red text</p>
  <p>Normal text</p>
</body>
```

> *Notice how the CSS was written in one line? That's allowed. But if you have more than one declaration (styling rule) you should put your CSS on multiple lines.*

The above code snippet will turn the text in the first <p> tag red, but the second paragraph will remain normal. That's because the `.` before "red" told CSS to look up any HTML elements that had "red" in the `class=""` part of your `<p>` tag.

You can add multiple classes to one HTML element to stack the CSS effects as well.

```
<body>
  <style>
    .red { color: red; }
    .large { font-size: 30px; }
  </style>
  <p class="red large">Red text</p>
  <p>Normal text</p>
</body>
```

Now your first paragraph element will have large red text, because we told CSS to style the paragraph with "red" and "large" classes.

> *Note: Classes can be used on multiple HTML elements. There's no limit to how many times you apply a CSS class in your HTML.*

Selecting by ID

ID's in CSS are represented by the # symbol. Each HTML element should only ever have one ID and every ID needs to be unique. Think of ID's like a class that can only be applied once.

Modern browsers will let you use the same ID more than once, but it's frowned up and may result in unexpected styling with older browsers. It's best to stick with the ID rule that one element gets one ID.

Here's an example of an ID selector in CSS.

```
<body>
  <style>
    #custom-id {
      background-color: black;
      color: white;
    }
  </style>

  <p>Welcome to a new page</p>
  <div id="custom-id">
    This section will have a black background
    and white text inside it.
  </div>
</body>
```

Notice the difference between the class and ID selectors:

- Classes use a period (.) in front of the class name, and the class name must match the class name in the HTML *class* attribute.
- ID's use a number sign (#) in front of the ID name, and the ID must match the ID name in the HTML attribute *and and ID attribute can only ever have one uniquely named ID.*

Properties and Values

So far we've written a little CSS and some HTML. But you don't actually *know* what the parts of a CSS block are called. So let's cover that now.

```
.selector {
  property: value;
}
```

The above snippet tells you exactly what we've been working with. And this entire snippet is called a *declaration*.

But let's break this down.

Selectors

As you know by now, a selector is how you select the HTML to style.

Properties

A property is *what* you want to change. Example of these would be: color, font-size, and background-color. There are hundreds of these, and most of them are named the way you would assume. For example, in CSS3 (which we'll cover later) you can add a shadow to your text -- that property name would be "text-shadow" (without the quotes).

Values

The value of a property is what you're changing the style to. If we wrote:

```
.new-color {
  color: red;
}
```

We'd be setting the text color to the color red on any HTML element that had `class="new-color"` set.

Declarations

The entire snippet it called a declaration. But truthfully, hardly any developers actually call it that. We tend to just call this a "style".

Parents, Children and Grandchildren

In both HTML and CSS there's a hierarchy of elements. In HTML, this is known as the Document Object Model (DOM), and CSS uses this model to traverse through elements. There are names for elements inside elements inside elements, and these names will be used throughout this course (and in your future as a web developer) quite often, so it's time to get get familiar with these terms.

An **element** is a stand alone HTML tag. It can be inside another element, or by itself. Regardless, it's just called an element.

```
<div>...</div> <!-- Basic element -->
```

When you put another element inside an element, you've started to create a family.

```
<div> <!-- This is the parent element -->
  <p> <!-- This is the child element --> </p>
</div> <!-- End parent element -->
```

The child <p> element above can also have children elements, like so.

```
<div> <!-- This is the parent element -->
  <p> <!-- This is the child element -->
    <span> <!-- This is the grandchild element --> </span>
  </p> <!-- End child element -->
</div> <!-- End parent element -->
```

This goes on forever. But don't get lost when talking about great great grandchildren, that's too much ambiguity for a human conversation. Instead, you can talk about a specific tag and talk about it's relative parents, grandparents, child or grandchild in relation to said specific tag. Here's an example:

My paragraph tag has a child span element and the parent element is a div.

Instead of starting from the top or bottom element, we just picked one (it happened to be the middle element) and referred to it's child and parent elements in relation to the element we're focused on.

Specificity

There will be a time when you need to drill through your HTML and style a specific element a certain way. Sometimes an ID is already set with styling, but it's applied on other pages -- so you can't change it for this one scenario. You *could* add more classes to it, that'd work perfectly fine. Or… you can tell CSS to drill through your HTML for you.

> *Warning: Too much specificity can lead to future styling problems. We'll explain this later in this course.*

How to Specify

First, let's look at an example.

style.css

```css
#list { background-color: red; }
.first,
.second,
.third {
  background-color: blue;
  font-weight: bold;
  color: #fff;
}

#list li {
  border: 2px solid black;
}
```

index.html

```html
<ul id="list">
  <li class="first">First</li>
  <li class="second">Second</li>
  <li class="third">Third</li>
  <li>No class applied</li>
</ul>
```

A few things to notice.

1. We have these in 2 separate files and we're assuming index.html is using `<link />` to attach the style.css file.
2. We've added multiple selectors in the second declaration by separating the selectors with commas. The new lines don't matter -- we could have written it all on one line and

the browser wouldn't care. It's the commas that are important. That means the styling for .first, .second and .third will all be the same.

3. Lastly, we have a specified selector written as `#list li`

Point #3 above is the specific selector we're learning about right now.

What `#list li` means is it's looking for an HTML element with the ID of "list" and any child elements (elements inside the #list element) and it will apply the styling from the properties we've written.

From the example above, this would select the <ul> element and every <li> element under it and apply the black border. Here's what this page would look like:

Not very attractive. But this is a good start because you're now familiar with selectors and how to apply them to your page.

To view the live code for this lesson, click here (Codepen.io)

Vendor Prefixes

Often the developers behind major browsers such as Chrome and Firefox will release updates with support for certain CSS properties before they are officially supported across all major browsers.

To access these CSS properties for certain browsers, we need to tell CSS to *try* accessing the property when using a specific browser.

That description was ambiguous, so here's a look at what prefixes look like.

```
-webkit-  /* Safari, Chrome and browsers using the Webkit engine */
-moz-     /* Firefox and other browsers using Mozilla's browser engine */
-ms-      /* Internet Explorer (but not always) */
-o-       /* Opera */
```

Those go in front of the CSS property *only if the feature is still considered experimental*.

> *How can you tell if you should use a vendor prefix in front of a property? You can check here (shouldiprefix.com)*

Over time, these experimental features become standardized and no longer require the prefix. Here's an example of when the `border-radius` feature was first introduced.

```
-webkit-border-radius: 10px;
-moz-border-radius: 10px;
-o-border-radius: 10px;
```

Border radius allows CSS to make rounded corners. This used to be very difficult to do, but CSS3 has made it easy for us. And the above snippet was how we used to write rounded corners with CSS. As you can see, we wrote it 3 times: once for webkit (Chrome and Safari), once for Mozilla (Firefox) and once of Opera.

Remember, this feature is *most likely* to be implemented fully in the future, so we should future proof this by adding the following line:

```
border-radius: 10px;
```

With all 4 lines we're now targeting the major browsers *and* future proofing so when the feature comes out of prefix mode it'll be used normally.

But to be honest, this example is a good example, but it's old. That's why we chose it. The border-radius feature hasn't been in prefixed for several years and is now a very common

occurrence on almost every website. So we can ignore these prefixes and just write `border-radius: 10px;`

If you'd like to see this code in action, we've created a CodePen. Feel free to check it out.

Specificity Problems

This is a lesson that, as of 2017, almost nobody ever tells you: **CSS Specificity is a major problem.**

A couple lessons ago we went over specificity -- how to target html elements inside html elements. There's another name for this: nesting.

Nesting

Nesting is nice at first because it gives you *a lot* of control. But eventually it will lead to heartbreak. And to make this just a bit worse, if you ever need to work with another developer or work with messy code from a different developer, you'll immediately realize that working with their CSS is *always difficult*, especially when it comes to nesting.

Not only does nesting get complicated to read, but because you can have a project with thousands of selectors, it gets difficult to keep track of. But it gets just one step worse.

Nesting makes overwriting styles difficult. Here's a simple example:

style.css

```
.parent .child {
  background-color: blue;
}
.child {
  background-color: red;
}
```

index.html

```
<div class="parent">
  <div class="child">
    I should have a red background
  </div>
</div>
```

If you read this code, it should be straightforward: your .child div element will have a red background color.

But it doesn't.

If you run this example you'll see the background color is still blue. Even though we told CSS to make the .child background color red. We even played by CSS's Cascading rule.

> *CSS will always render styling from the top of the file to the bottom of the file, and the first files can be overwritten by newer files.*

So why on Earth did this not work? Because specificity, that's why.

What if changed the .child selector so it included it's parent, like the first declaration, like so?

```
.parent .child {
  background-color: blue;
}
.parent .child {
  background-color: red;
}
```

Because both selectors are 2 elements deep (the nesting is the same), can we assume the background color will now be red? Or do you think it'll still be blue? Think about it for a quick 2 seconds, remember your answer, and then look at our example here.

How to Overwrite

Is it possible to overwrite the styling without nesting the same way? Well of course it is. But it's not a solution you should ever rely on. Let's learn about this solution first, and then we'll dive into the reason why it's bad to use.

In order to overwrite any styling that's being stubborn (maybe due to nesting, maybe the CSS is so messy you can't figure out why it's not changing for you), you can use `!important` in your CSS values. Essentially, this tells your CSS to "ignore everything else, just apply this" and "never change this value, it's too important."

Following our example from above, you can apply this new rule to our red background.

```
.parent .child {
  background-color: blue;
}
.child {
  background-color: red !important; /* Notice the !important */
}
```

Example

> *Anything in between the /* and */ will not execute. These are called comments, and they are very regular in code. You'll see them everywhere, and it's generally a good idea to add comments to your own code, too.*

That fixes our red background problem. Kind of. What if we wanted to overwrite that color again? Well… !important means “ignore other rules for this value” so we can’t overwrite it, can we? Let’s try.

```
.parent .child {
  background-color: blue;
}
.child {
  background-color: red !important;
}
.parent .child, /* Notice the 2 selectors for one property */
.child {
  background-color: yellow;
}
```

Go ahead and try this yourself. If you get stuck, refer to the example we’ve written here.

Did that work? Of course not. We knew it wouldn’t. But now we’re stuck with a red background… and perhaps the page we’re on should be yellow instead. How do we change this?!

There are two options:

1. We can use `!important` again and get stuck in !important trouble, or…
2. We can rewrite the original CSS so we don’t get stuck with this nesting and !important problem.

Here’s a fact about front end web development: nobody enjoys this situation. It’s often more complex than this, and can cause a lot of frustration. The right way to fix this is to rewrite your CSS. But if you have a deadline you might not have time to do that. Or maybe you have a project that has so much CSS it’s impractical to rewrite this -- in that case you’ll want to use !important. But a good general rule of thumb is: do not use `!important` unless you absolutely have to.

There’s a better way

In web development and programming in general, there are always better ways to do anything. What we’re offering right now is just one of many solutions available: BEM.

BEM: Block Element Modifier (getbem.com)

This course isn’t going to teach you about BEM, but if you want to avoid this type of CSS problem, a great method is to follow the BEM standard.

The short version of this is to *never nest unless you absolutely have to*. If you don't nest, you'll never run into specificity problems. This often means writing more CSS, but it also means less frustration in your future.

We're not suggesting you adopt this style, that's entirely up to you. But it is one of several great solutions that's become popular and supported by the development community.

http://getbem.com/ is the website. Maybe tuck this away in your back pocket for a tool to use in the future.

The Box Model

CSS comes with multiple ways to add spacing between sections. The tricky part about working with spacing is that you *cannot* see it -- it's the invisible space between two objects.

The best way to learn about the Box Model is to see a diagram of it, then get your hands dirty by trying it out.

The Box Model

The above image has several components:

1. The outside is the spacing outside the border, represented by the `margin` property
2. The border is the section between the margin and the padding, represented by the `border` property
3. The padding is the spacing inside the border, represented by the `padding` property
4. The content is where your next HTML element will go, it can be text or HTML, either way it will be cushioned by the padding, border and margin.

Here's a live example of how these work:

style.css

```
.box {
  border: 2px solid black;
  padding: 50px;
  margin: 50px;
}
.content {
  background-color: yellow;
}
```

index.html

```
<div class="box">
  <div class="content">
    There's padding outside and inside the black box.
  </div>
</div>
```

Live Example

By reading the above CSS you can see that there is a 2 pixel wide border around the .box element, 50 pixels padding inside the border and 50 pixels outside of the border. That means this element will be pushed away from the edges of it's parent element (in this case the parent element is the <body> tag, which will go to the edges of your browser by default), and the content (the code inside .box) will be pushed in 50 pixels as well -- because padding pushes code inwards.

Here's the link to the live example: https://codepen.io/Arkmont/pen/JNvxza?editors=1100
Open that up and edit it, change the padding and margin, and notice how the margin affects the outside and padding affects the inside of the div.box element.

> *Just a side note: starting now we'll be referring the HTML elements by their CSS selectors. This is good practice for when you need to talk to other developers about a certain element and styling associated with it.*

Specific Padding and Margins

We've looked at margins and paddings and how they both add spacing to your HTML elements. However, the padding and margins we've used so far add equal spacing to all sides. While that's visually pleasing when elements are alone, in the real world you'll need different spacing for different sides of an element.

Let's look at an example:

```
.marginExample { margin: 10px; }
.paddingExample { padding: 10px; }
```

Which is the same thing as writing:

```
.marginExample { margin: 10px 10px 10px 10px; }
.paddingExample { padding: 10px 10px 10px 10px; }
```

And that's essentially telling your browser this:

```
.marginExample { margin: top right bottom left; }
.paddingExample { padding: top right bottom left; }
```

The order of those are always top, right, bottom left (clockwise).

Now that you know you can write `margin: { 5px 10px 15px 20px; }`, let's see what it actually does with an over exaggerated example.

CSS

```
.paddingExample {
  display: inline-block;
  border: 1px solid black;
  padding: 10px 20px 30px 40px;
}
```

HTML

```
<div class="paddingExample">
  &larr; &rarr;
</div>
```

Live Code Example

We have some padding at the top, a bit more on the right (although it's hard to tell with the arrows), 30 pixels of padding on the bottom and 40 pixels on the left.

Ok, that's acceptable, but that still limits our control over the padding. Say, for example, you only wanted to add padding (or a margin) to the top of your element -- what then?

Well it just happens to be that

```
.paddingExample { padding: 10px 20px 30px 40px; }
```

Is the shorthand method, and your browser doesn't see it this way. Your browser actually sees the follow:

```
.paddingExample {
  padding-top: 10px;
  padding-right: 20px;
  padding-bottom: 30px;
  padding-left: 40px;
}
```

> *Note: you can use margin the same way we're using padding here. Simply switch out the word "padding" for "margin".*

Let's look at one more example:

```
.paddingExample {
  display: inline-block;
  border: 1px solid black;
  padding-top: 100px;
}
```

Live Code Example

Look at that! 100 pixel padding on the top, and no padding on the sides or the bottom.

Colors

Colors (or Colours outside of the U.S.) can be written in CSS in different ways. Up until now we've used names, but names only go so far. For example, the color grey has different shades… as a front end developer, you'll want access to more colors than just grey -- you'll want a spectrum of colors so wide you'll never use every color available to you.

Names

Names are fine to use when it's basic and there cannot be any other option. Grey was a good example -- there are different shades of grey, but there's only one white and only one black.

```
.color1 {
  color: white;
}
.color2 {
  color: black;
}
```

Hex Values

Hex codes give you a much wider range of colours. From #000000 (black) to #ffffff (white) and everything in between #000000-#999999 (greys) and #aaa-#fff (colors).

> *Note: don't remember too many hex values, they aren't important, there's too many, and they can be accessed using color picker tools.*
>
> *Also, you can write shortcuts. If you have the same values in the first 3 characters as you do in the last 3 characters, you can write the short version. Example:*
> *#000000 is the same as #000*
> *#FFFFFF is the same as #FFF*

```
.color1 {
  color: #fff; /* White */
}
.color2 {
  color: #000; /* Black */
}
.color3 {
  color: #8c1515; /* Arkmont Red */
}
```

RGBA

The last method we'll cover is RGBA: Red, Green, Blue, Alpha. Alpha is the last setting that lets you change the opacity. This is *great* for making colors lighter, or translucent.

You'll need to know the red, green and blue values of a color to use this though. Again, a simple color picker tool will help you with that. A quick google will find you a color picker that works best on your operating system (Windows, Mac or Linux).

```
.color1 {
  color: rgba(255, 255, 255, 0.5); /* 50% white */
}
.color2 {
  color: rgba(0, 0, 0, 0.4); /* 40% black */
}
```

The RGBA function with named arguments looks like this:

rgba(red, green, blue, alpha)

The *alpha* parameter is a percent as a decimal. Remember that math you learned in school that you thought you'd never use? That's not this -- this is just a simple number between 0 (completely transparent) and 1 (completely opaque).

Here's an example that shows all of these in one page.

Styling Links

By now you've most likely created a link using HTML. But by default, links are styled blue by default, and they turn purple once you've opened them.

That's not ideal. In fact, that's pretty much the easiest way for someone to judge your website as old, irrelevant, or just an eyesore. Let's fix that.

We'll begin by setting up a simple HTML link.

```
<a href="https://kalob.io/">
  Kalob.io
</a>
```

That will look something like this: Kalob.io

That's not pretty. But we can fix that with some very easy CSS and a couple pseudo selectors, which we'll use in this lesson but we'll formally learn about them in the next lesson as well.

To change your link color, write a regular CSS tag selector.

```
a {
  color: black;
}
```

That will turn all your links black. Remember, you can use hex values or RGBA values as well.

The Hoverstate

The Hoverstate is when you put your mouse over a link and it changes color. We do that with CSS through a selector modification (term used loosely) called a pseudo selector. Pseudo selectors cannot be used with inline CSS, which is why we've been adding CSS to a separate file, or keeping it all within the `<style></style>` element.

CSS was designed to be simple, so naturally the hover pseudo state would be called `hover.`

```
a {
  color: black;
}
a:hover {
  color: yellow;
}
```

Well that looks different! There's a colon before the hover keyword and that's how CSS knows this is a pseudo selector. As soon as you put your mouse over the link, it will turn yellow.

More Pseudos

There are 3 more pseudo selectors you need to be familiar with when it comes to links. If you think this is a lot to remember right now… then stop stressing out. You'll end up writing so many of these it'll become second nature by the end of your first project.

The 4 pseudo selectors are:

1. `:link` -- To define the link styling
2. `:visited` -- To define the styling when a link was already clicked
3. `:hover` -- To define the styling when you put your mouse over a link
4. `:active` -- To define the styling when you've clicked on the link, but haven't let go yet

Let's look at a live example (the link to the interactive example is below the code)

```
a {
  color: black;
}
a:link {
  color: red;
}
a:visited {
  color: blue;
}
a:hover {
  color: yellow;
}
a:active {
  color: green;
}
```

Here's an interactive example of how this all works:
https://codepen.io/Arkmont/pen/qmYvZv?editors=1100

Styling Lists

Ok, we haven't *really* written very much CSS. Let's take a break from learning *about* CSS, to learning *how to write* CSS. And we're going to start with an unkind example: lists.

Front end developers know the power behind lists. Lists, ordered or unordered, create clean sections of HTML that can be easily arranged and styled with CSS. Since CSS lets us reset every HTML element and shape it the way we need it, we *could* just use a plethora of tags, but that's not exactly helpful for search engines. With a list, at least the search engine can *easily* realize it's a list by the <ol> or <ul> tags, and save that information appropriately. That's good for SEO.

Let's look at a regular unordered list.

```
<ul>
  <li>First</li>
  <li>Second</li>
  <li>Third</li>
</ul>
```

It looks like this:

Wow. Such bore. Much lame. Let's spice this up using just CSS (the HTML won't change *at all*.)

```
ul {
  background-color: #ccc;
  padding: 10px;
  margin: 0;
}
```

That gives us the following:

That's actually worse than the original list. But sometimes you need to take a step backwards to move forwards (at least in design that can be true).

We need to get rid of those bullet points.

```
ul {
```

```
  background-color: #ccc;
  padding: 10px;
  margin: 0;
  /* Remove bullet points */
  list-style: none;
}
```

That's better. Not great, but better. But let's be *real* web developers and iterate through this a bit faster. Below will be a couple steps to demonstrate how this list can get nicer and nicer. **Note: If you don't understand some of the CSS that's coming up, don't give up! We'll cover *all* of it very soon!**

```
ul {
  background-color: #ccc;
  padding: 10px;
  margin: 0;
  /* Remove bullet points */
  list-style: none;
  width: 50%
}
li {
  display: inline-block;
  width: 32%;
}
```

- We made the width of the <ul> 50% (it's a block element by default)
- Each <li> has been inlined, so they can exist beside each other
- Each <li> is 32% the width of it's parent <ul> container (the <ul> has a default width of 100%; but we changed it to 50%)

```
...
li {
  display: inline-block;
  width: 32%;
  font-family: 'Arial', sans-serif;
  text-align: center;
  opacity: 0.5;
}
li:hover {
  opacity: 1;
}
```

- Changed the font to Arial (if Arial isn't on the users computer, use any sans-serif font)
- Aligned the text of each <li> to be centered
- Changed the default opacity of each <li> to 50%
- When you hover over any <li> it will fade in by changing the opacity to 100%

Now let's purposely get a head of ourselves a little bit, just to keep things interesting. Let's add a fadeIn transition when you mouse over one of the 's.

```
li {
  display: inline-block;
  width: 32%;
  font-family: 'Arial', sans-serif;
  text-align: center;
  opacity: 0.5;
  transition: opacity 500ms linear;
}
li:hover {
  opacity: 1;
}
```

Now there's a 500ms (0.5 seconds) transition set on the opacity property. Anytime the opacity property changes, it'll take half of one second to fade in and fade out. This is a CSS3 property, so be aware that this won't work on older browsers.

This is what it looks like (without the hover states):

Admittedly, this <ul> list isn't exactly beautiful, but it's much more functional and doesn't look like a regular list anymore. This is just the very beginning -- the smallest change we can make to a simple set of HTML elements.

Pretty cool, right? The interactive code can be found here so feel free to experiment with it.

Blocks and Inlines

HTML elements have particular standards set to them by default. With CSS we can overwrite all the default settings for an HTML element, but we need to know what options are available to us.

Introducing block, inline-block, and inline settings.

Block

Block elements are elements that take up 100% of the viewport width, even if you can't see it.

> *The viewport is the section of your browser that displays the website. It doesn't take into account your toolbar, bookmarks, minimize buttons, etc.*

A few very common examples of block elements are:

- `<h1> - <h6>`
- `<p>`
- `<div>`

Let's look at some code to help understand this.

```
<h1>Large title</h1>
<p>Paragraph element</p>
<div>Regular div element</div>
```

And the preview looks like this:

Notice how each HTML element in the preview is on it's own line? That's the block setting taking effect.

Inline

Inline elements are able to exist side by side and cannot take on extra margin or padding. In other words, they take up the least amount of space possible. Some examples of inline elements are:

- `<span>`
- `<a>`
- `<i><em><b><strong>`

Let's look at how these work with some CSS applied.

style.css

```
.b-red { border: 1px solid red; }
```

index.html

```
<span class="b-red">This is a span</span>
<a href="#" class="b-red">beside a link</a>
<em class="b-red">Beside italics</em>
```

Preview:

Unlike block elements, they exist beside each other without any problems. Nice, right? But if you wanted to apply a width, padding or margin, you're out of luck.

```
.b-red {
  border: 1px solid red;
  width: 100%;
}
```

Preview:

Nothing changed. That sucks.

So *block* takes up an entire line, and *inline* can't have a set width… then we need to mix these two together and create…

Inline Block

Yes, it's exactly how it sounds. The CSS property naming conventions so far are pretty exciting, aren't they? That was a semi-sarcastic joke because these names are so boring. But boring in this sense is good -- everything is easier to remember that way.

Let's look at a new example. Say we have 1 block and 2 inline elements but we want all three of them to be side-by-side.

style.css

```css
.b-red { border: 1px solid red; }
.one-third {
  display: inline-block;
  width: 32%;
}
```

index.html

```html
<h1 class="one-third">Large title</h1>
<a href="#" class="b-red one-third">beside a link</a>
<em class="b-red one-third">beside italics</em>
```

Preview:

Look at that, they're all beside each other now. Perfect! But this raises the question: when will you ever use this?

That's a great question you *should* ask yourself all the time. You'll actually use this *a lot,* and in such scenarios like:

- Creating a page header,
- Creating a navigation menu,
- Setting up custom columns,
- But most importantly… you'll use this in responsive design *all the time*.

All the code for this lesson can be played with on CodePen.

Sizes and Overflow

We have yet to talk about width, height and content that breaks out of our defined dimensions. Now is a good time to go over that.

We know that width, height, padding and margins can only be set on *block* and *inline-block* elements, and that *inline* elements don't let us change these properties. Immediately, we know we need to use block or inline-block properties.

CSS

```css
.box {
  display: block;
  border: 1px solid black;
  width: 50%;
  height: 50px;
  margin: 0 auto;
}
```

HTML

```html
<div class="box">
  Lorem ipsum dolor sit amet, consectetur adipiscing elit. Sed in suscipit mauris. Suspendisse imperdiet varius lobortis. Aliquam erat volutpat. Nunc rhoncus id elit vel mollis. Fusce egestas diam ac tellus mattis finibus. Fusce tincidunt, metus sed tempor ornare, ex ex fringilla sem, in ullamcorper leo leo sit amet risus. Mauris in mattis enim. Vivamus et vehicula nunc, sed bibendum est. In cursus, leo commodo fringilla ornare, ante mauris pulvinar risus, ut posuere sapien risus in nisl. Nam rutrum metus sed tempor viverra.
</div>
```

In the above code, we've set a to be a block element (it's block by default, but now we can change to any other element), added a black border so we know the size of the box, set the width to be 50% of the viewport, added a height of 50 pixels, and centered the box with a margin.

> *To center a block element with CSS you want to set the margin-left and margin-right properties to auto, like so:*
> `margin: 0 auto 0 auto; /* Values are: top, right, bottom, left */`

Here's what it looks like so far:

See how the black box is only 50 pixels tall, but the text keeps flowing past it? We can't have that, it looks unprofessional. The remedy to this problem is nice and simple -- we give the .box an overflow property.

The overflow property takes one of many different values.

```
overflow: visible | hidden | scroll | auto | initial | inherit;
```

Instead of allowing the text to flow through it's box container, we want to either hide the text with `overflow: hidden;` or make the box scroll.

Making the .box hide it's external contents is sometimes the exact solution you need.

```
.box {
  display: block;
  border: 1px solid black;
  width: 50%;
  height: 50px;
  margin: 0 auto;
  overflow: hidden;
}
```

Perfect, the extra content has been hidden. But it's not accessible, and you always want text to be visible *somehow*. Let's modify the overflow property value to "scroll".

```
.box {
  display: block;
  border: 1px solid black;
  width: 50%;
  height: 50px;
  margin: 0 auto;
  overflow: scroll;
}
```

Ok, looks like we can scroll now. But there's a scrollbar at the bottom! Is it necessary to scroll left and right? Well that depends on your design, but generally that's not the case. We want to remove the scrollbar and make it only scroll up and down.

To do that, we change the scroll value to "auto", which will let the browser detect what kind of scrollbars .box needs.

```
.box {
  ...
  overflow: auto;
}
```

Perfect! No scrollbar on the bottom, no overflowing content, a scrollbar on the side, and the user can still access the hidden content.

Try it out with interactive code on CodePen.

Opacity

In a previous lesson we used opacity in an example. This become available to us with CSS3 and it's a life saver. We used to use translucent images to create opacity, but those days are long gone!

We can set the opacity to almost every element, too, and that makes this extremely exciting because it allows us to create visible layers without needing to use images or other tricks.

CSS

```css
.parent {
  width: 200px;
  height: 200px;
  border: 1px solid black;
  margin: 0 auto;
  background-color: red;
  padding: 50px;
}

.child {
  width: 200px;
  height: 200px;
  opacity: 0;
  background-color: white;
  border: 1px solid black;
  transition: opacity 250ms ease;
}

.parent:hover .child {
  opacity: 1;
}
```

HTML

```html
<div class="parent">
  <div class="child">

  </div>
</div>
```

Now we're starting to get fancy with our styling. We've created a .parent element to contain the .child and it has a red background. The .child seemingly has a white background, but if you were to preview this right now you wouldn't see it because it's opacity is set to zero

(transparent). Then we have one more selector, the `.parent:hover .child { }` selector. We know what each part of this does if we break it down.

1. Select the .parent element…
2. When the mouse hovers over it…
3. Select the .child element and style it with the properties between the brackets.

But it just happens to be we added `transition: opacity 250ms ease;` to .child, meaning this will smoothly fade in and fade out when you hover over the .parent element. Here's the final animation when you mouseover the .parent element.

Alright, that's still not very beautiful to look at. But we're making progress considering we didn't use a red border (just 2 black borders) and this looks quite a bit like a picture frame.

Let's run with this and add a stock photo of some wood where the red is.

```
.parent {
  ...
  padding: 50px;
  background-image:
    url('https://images.pexels.com/photos/172292/pexels-photo-172292.jpeg?w=200');
}
```

Image provided by pexels.com

Here's what we have now:

Now if we add a background image to the .child selector...

```
.child {
  ...
  transition: opacity 250ms ease;
  background-image:
    url('https://images.pexels.com/photos/9135/sky-clouds-blue-horizon.jpg?w=200');
  background-size: cover;
  box-shadow: inset 0 0 15px rgba(0, 0, 0, 0.5)
}
```

Now when we hover over the wood box, we'll see the image to the right.

We did all that with just CSS. At this point, we're almost ready to start making our own seek-and-find game.

The full code can be found here.

One last thing before we continue. In the last snippet of CSS we see 3 properties we haven't explored until now:

1. background-image
2. background-size
3. box-shadow

Background image is the image in the background. We used this for the picture of the sky and the wood panels. Background-size lets us adjust how the background image sizes should be handled. In this scenario, we told it to cover the entire element, even if it needs to be stretched out a little bit. And box-shadow let us add that little shadow inside the .child element and gives us a slight feeling that the .parent element is in front and casting a shadow on to it.

Again, we're not artists by any means, but these are just a few more examples of how we can utilize CSS instead of images such as .gifs and how we no longer rely on flash animations at all.

Positioning Elements

Sometimes you'll see a web page that has a strangely positioned element that doesn't act like the elements we've written so far. Like a navbar that never goes away when you scroll, an element that looks like it's floating above another element, or a poll or contact person at the bottom right of a page. These are all done by changing the position property, and there are 4 main settings you'll work with as a front end developer.

Static

The `position: static;` setting is the default setting, it's what every element up until now has used.

Relative

By using `position: relative;` you can change the position of an element, relative to where it would normally be.

CSS

```css
.box {
  border: 1px solid red;
}
.relative {
  border: 1px solid blue;
  position: relative;
  top: 12px;
}
```

HTML

```html
<div class="box">
  <span class="relative">
    Relative (12px down from where it normally sits)
  </span>
</div>
```

Live Code Example

This can be extremely useful for making a web page pixel perfect.

Absolute

You can position an element in the exact position you want with `position: absolute;` but there's one caveat: it *has to* be a child element to a relatively positioned parent element. In other words, you can only position an element exactly where you want it to be if the container element is using a relative position. Remember, *static* is the default position, so you'll need to change both elements.

CSS

```css
.parent {
  display: block;
  width: 200px;
  height: 200px;
  margin: 0 auto;
  border: 1px solid black;
  position: relative;
}

.child {
  position: absolute;
  top: 0;
  left: 0;
  width: 50%;
  height: 50%;
  background-color: yellow;
}

.child2 {
  position: absolute;
  bottom: 0;
  right: 0;
  width: 50%;
  height: 50%;
  background-color: purple;
}
```

HTML

```
<div class="parent">
  <div class="child">

  </div>
  <div class="child2">

  </div>
</div>
```

Live Code Example

To the left is what you'll see when you run the code above. The .parent class was set to relative, and the .child and .child2 classes were set to absolute.

If you look at the CSS in this example, you'll notice we used height, width, top, bottom, left and right properties. These properties do exactly what they suggest.

The top, bottom, right and left properties tell the browser where the absolute element should be positioned. You can set all four at the same time, too.

Fixed

A position: fixed; style will make the element sticky -- as in it will stick to wherever you tell it to stick and never move. A great example of this in use is the floating navbar. We see this in practice every day on Facebook.

CSS

```
.navbar {
  position: fixed;
  top: 0;
  left: 0;
  width: 100%;
  background-color: green;
  padding: 20px;
  color: #fff;
  font-family: 'Arial', sans-serif;
}
```

HTML

```
<div class="navbar">
  WEBSITE NAME
</div>
```

Live Code Example

When you run this code, you'll only see a green bar across the top of your browser. It doesn't look any different. But if you added any code after the closing `</div>` tag, you'll find the green bar sits on top of the other code (you won't see it).

In the Live Code Example there's another class selector called .content and we gave it `margin-top: 60px;` -- that was purely for the fixed navbar. Also, we've added a bunch of lorem ipsum text so you can scroll down the example and observe how the navbar doesn't move at all.

Layers

There will come a time when you need to layer an element on top of another element. Using absolute or fixed positions works… until you need more than 2 layers.

If you've ever worked with PhotoShop or any design program, or even PowerPoint, you're familiar with how you can move sections forward and backward. This is the same concept.

But let's dive into an example.

CSS

```css
.container {
  position: relative;
  display: block;
  width: 200px;
  height: 200px;
  background-color: rgba(0, 0, 0, 0.2);
  border: 1px dashed rgba(0, 0, 0, 0.5); /* RGBA ;) */
  /* Center the block */
  margin: 0 auto;
}

/* Grouping selectors, we'll overwrite .layer2 later */
.layer1,
.layer2 {
  position: absolute;
  top: 25%;
  left: 25%;
  height: 50%;
  width: 50%;
  border: 1px dotted #000;
  background-color: green;
  opacity: 0.7;
}

/* Overwrite layer2 properties */
.layer2 {
  background-color: red;
  top: 35%;
  left: 35%;
  opacity: 1;
}
```

HTML

```
<div class="container">
  <div class="layer1"></div>
  <div class="layer2"></div>
</div>
```

Live Code Example

First off… wow, that was more CSS than we've written throughout this entire course so far. And most of it should have made sense to you by now. Any parts that didn't make sense… well, that's OK, you don't need to know *everything* right now -- this all comes with time and practice.

In this example there are 3 layered elements: the background (grey), .layer1 (green) and .layer2 (red). While this seems to be working fine, there's a question lingering about… How do you make the green layer sit on top of the red layer?

z-index

If you remember graphs from school, you might remember the X-axis, Y-Axis and possibly the Z-Axis (if you did any complex math or 3D design).

> ***Quick refresher:***
> *X-Axis goes left and right, Y-Axis goes up and down, Z-Axis goes in and out (depth)*

We've set .layer1 and .layer2's X and Y axis by setting the left and top properties. But now we need to change the depth (Z Axis). To do that, we simply add the `z-index` property to our layers.

But we can't add the same z-index value to both layers, otherwise nothing will happen -- they'll stay the same or become unpredictable in other browsers. Let's be very clear about making .layer1 move to the top.

CSS

```css
/* Add z-index to .layer1 */
.layer1 {
  z-index: 2;
}
```

Live Code Example

Look at that, it's a thing of beauty! The green layer moved forward because we gave it a higher z-index than it's default value (default is either 0 or 1; assuming 1 is your default is safer for browsers that might count from 1 instead of 0)

We *could* change .layer2 to have a higher z-index than .layer1, and bring the red layer forward again. It's that simple.

Visibility

There will be a time when you need to hide an element. Sometimes you need to hide an element for a better responsive experience, sometimes you just want to hide some content so it can be animated in with JavaScript or CSS. But there's more than one way to do this.

We're already familiar with the opacity property -- you can set `opacity: 0;` to pretty much any element and hide it. But does it actually hide it? Let's find out.

CSS

```
.red-block { background-color: red; }
.blue-block { background-color: blue; }
```

HTML

```
<div class="red-block"> </div>
<div class="blue-block"> </div>
<div class="red-block"> </div>
```

Live Code Example

That's exactly what we'd expect. Let's change the opacity of .blue-block to zero.

CSS

```
.blue-block {
  background-color: blue;
  opacity: 0;
}
```

Great, we made the blue block disappear! But there's still a space where it used to be. That's because we didn't actually hide it, it's just not visible. Those are 2 different things.

If we wanted to completely hide the .blue-block, we can change it's display property value to "none".

CSS

```css
.blue-block {
  background-color: blue;
  opacity: 0;
  display: none;
}
```

That looks strange, but expected. We removed the .blue-block from displaying at all. Technically it's opacity is still set to 1, but we told the browser, "Do not display this block". So now it's hidden.

Here's the Live Code Example so far.

But there's *another* way we can change visibility too, and that's with the visibility property.

Visibility

You might end up in a situation where you want to keep the assigned space of an element, but you don't want to remove the block from eye sight. Instead of changing the opacity to zero, we can change it's visibility to hidden.

CSS

```css
.blue-block {
  background-color: blue;
  opacity: 0;
  visibility: hidden;
}
```

Live Code Example

Your First Project

Welcome to your first CSS project. We've covered a lot in this course so far, but we've only just scraped the surface of possibility when it comes to writing CSS.

Instead of having an on-going project throughout this course, we've gone through the first module and we're going to play catch-up a bit -- with the goal of using our memories to build our first template.

We're going to use what we've learned to create a basic website template. It's not going to be easy, but it's not difficult either. Every resource you need to make this template is in this module.

Try not to simply copy and paste code -- use your memory for as much as possible! If you get stuck on anything for more than 2 minutes, then use the completed lessons as reference.

Below is a mockup of the template I want you to make. Feel free to create a *free* account on CodePen.io and write your HTML and CSS in there. You can save your progress and share your work with CodePen.io as well.

Do your best with this template because we'll be using it in the next module to really spice up your website, and we'll be making it responsive after that.

Requirements:

- The blue navbar at the top needs to be sticky (fixed)
- The left navigation menu must use at least one <ul> or <ol> using CSS to make it look natural, and not like a list.

- The body of your page needs to include a <p> with both <a> and ’s as child elements.
- Each colored section in the image above needs to have a different background color.
- The footer text needs to be centered using CSS.

Height

Setting a height with CSS isn't as simple as you would expect it to be. When you set a width to a non-inline element, it takes effect pretty easily. But with height, there are *real* limitations.

For example, if you wrote `width: 80%;` on any block element, you can see that it'll use 80% of the full width. That's not true when it comes to height.

CSS

```css
.set-height {
  background-color: #ccc;
  height: 100%;
}
```

HTML

```html
<div class="set-height">
  This should be 100% tall
</div>
```

Live Code Example

The above image (with the grey background) is not 100% height. We set it to be 100% tall, but it doesn't listen.

And this is not uncommon with heights in CSS. But we *need* to get around this somehome. Luckily there are a few different ways we can do this *without JavaScript*.

Exact Heights

If you know the exact height you need, you can set a pixel value to dictate the height.

CSS

```css
.set-height {
  background-color: #ccc;
  height: 100% 90px;
}
```

That worked nicely, but honestly… you won't *know* the exact height most of the time because, like a lot of things in life, web development is based more or less on relative dimensions.

Browsers have different heights and widths and come in different flavours of operating system and devices. What looks good on your laptop might not look good on a tablet (adjusting for that is called *Responsive Design* and we'll be learning about that in a future module… hang tight!)

So the next thing we need to do is figure out a relative height. There are 2 ways to do this:

1. Relative to the container it's in (the parent element), or…
2. Relative to the viewport

Relative to the container

CSS

```css
.parent {
  background-color: blue;
  height: 90px;
}
.set-height {
  background-color: #ccc;
  height: 50%
}
```

HTML

```html
<div class="parent">
  <div class="set-height">
    This should be 90px tall
  </div>
</div>
```

Live Code Example

By setting the parent elements *exact* height, we can set the child elements relative height as a percent.

But you don't always know the exact height. Luckily with this method you don't need to. You can know a *rough estimated* height, and keep everything transparent -- then set the styling to your child element.

But if you want to make a nice cover page where the introduction of the page takes up 100% of the browsers height when you load the page (called a viewport), you'll need to use a different method.

> *There also another method that we've learned already: absolute and fixed positions. Those allow for percent heights because the viewport has a set height*

Viewport Height

The viewport is the box inside your browser where the website is displayed. It does not include tabs, your toolbar, your bookmark bar, extra settings or any of that other stuff that comes with a browser. It's just the rectangular area where the website is displayed.

CSS

```css
.vh50 {
  height: 50vh;
  background-color: #8c1515;
}
```

Live Code Example

Go ahead and give that a shot. If there are no typos, you should see a dark red section that's 50% the immediate height of the page. Fun fact: this is also responsive, since you can grow or shrink your browser and it will always be 50% of the viewport height.

Width

We've written some CSS using the with property already, so this lesson should be review for you by now. But let's go over it one more time.

If you didn't read about height, or watch the video on height, there was a section that mentioned viewport heights. Width works the same way, but instead of typing:

```
.class { height: 50vh; }
```

You can type:

```
.class { width: 50vw; }
```

Live Code Example

Regardless of your screen size, this will always be 50% of your viewport. This differs from using a percent because a percent is based on its parent element's total width, whereas viewport width is based on the most senior element on your website: the browser.

> *Just a quick note about the example screenshots. As we get into more complex CSS we'll be leaving out some of the code that makes the example images distinct. For example, the image above has a background-color on the <body> element, but we didn't show that in this lesson. But don't be scared, we're keeping all that "extra" code in the CodePen examples and we'll continue to link to them on a regular basis.*

Widths can also be set using pixels, like we've been doing so far, and percents (px and %, respectively).

Calc

CSS3 introduced us to a new concept called `calc()`. This beautiful feature allows us to do basic math calculations in CSS, but it handles the difficult parts.

If I asked you to tell me how many pixels are in 50% of this document and subtract 32, could you easily tell me? Don't worry -- nobody can. And since browser sizes change so often that answer will constantly change. Fret no more, we can this with CSS, finally!

> *History lesson. We used to do this with JavaScript. And before JavaScript became powerful, we didn't do this at all -- we'd estimate the percent and hoped that would work.*

CSS

```css
.three-quarters-ish {
  width: calc(75% - 80px);
}
```

HTML

```html
<div class="three-quarters-ish">
  75% - 80px
</div>
```

Live Code Example

What calc() did was take 75% of the width (of a block element) and subtract 80 pixels from it. And when you resize your screen, it will always stay that way until there's nothing left to display (when your viewport reaches 80px in width).

We're going to leave this lesson open ended… kind of. Here's the link to the example code above, go play around with that code and try the following scenarios:

- A percent minus a percent
- A percent minus a pixel value
- A pixel value minus a pixel value
- A percent minus a pixel value plus another pixel value

Font Sizes

You've undoubtedly seen a plethora of different font sizes. From so tiny you can barely read it, to in-your-face borderline obnoxiously large text.

More often than not, there's a good reason for larger font sizes. Headlines are larger because people tend to read the headline and decide if the article is worth reading -- so make it stick out! And other places will use smaller text for their Privacy Policy or Terms of Service in order to make their long page of text feel unbearably long.

There's often a font size difference between a desktop site, tablet site and a mobile site. We'll tackle how to handle the responsive design in a later module, but for now we'll learn how to change font sizes. It's actually quite simple.

```
font-size: 30px;
```

That's it! Simple, right?

The default font size varies from browser to browser, and from project to project. But it's usually between 14px and 16px.

Using the font-size property, we can change *all* font sizes in every HTML element. Let's look at an example:

CSS

```
.small {
  font-size: 8px;
}

.large {
  font-size: 40px;
}
```

HTML

```
<h1>Regular H1 Font Size</h1>
<h1 class="small">Small H1 Font Size</h1>
<hr />
<h5>Regular H5 Font Size</h5>
<h5 class="large">Large H5 Font Size</h5>
```

Live Code Example

The only other thing you need to know about font-sizes, like all size and dimension related properties, is that `px` is not your only available option. With fonts and text in general, you can use pixels (px), em's and rem's.

We use pixels throughout this course because it's the general standard and most properties are broken down into pixels on the browser-level anyway.

Line Height

With font sizes comes line height. If you look at our previous example from the Font Sizes lesson, the large <h5> element seems to have some crazy spacing between the small <h5> above it.

That's due to a property called line height. Line height is the invisible height of each line of text on your website.

CSS

```css
.tall {
  line-height: 40px;
}
.short {
  line-height: 10px;
}
```

Live Code Example

The .tall class, assuming you're using the default font size, will have a lot of spacing between each line. And the .short class will squish the line together.

We're going to leave out the HTML for this example because it's boring lorem ipsum text. But the full version of this example can be found on CodePen.

Your line height will typically grow with the `font-size` unless you specify otherwise.

But if the line height will grow with the font size, why even set the line height? Well… some fonts have abnormal line heights, and sometimes you want more or less line spacing to make your design look cleaner.

Font Weights

A font “weight” is how you tell a font to be bold, or alternatively, you can make your font thinner (light).

CSS

```css
body {
  font-family: 'Roboto', sans-serif;
  font-size: 30px;
}

.weight100 { font-weight: 100; }
.weight300 { font-weight: 300; } /* font-weight: light;  <- Also works */
.weight400 { font-weight: 400; } /* font-weight: normal; <- Also works */
.weight500 { font-weight: 500; }
.weight700 { font-weight: 700; } /* font-weight: bold;   <- Also works */
.weight900 { font-weight: 900; }
```

HTML

```html
<link
href="https://fonts.googleapis.com/css?family=Roboto:100,300,400,500,700,900" rel="stylesheet">

<div class="weight100">Weight: 100 (thin)</div>
<div class="weight300">Weight: 300 (light)</div>
<div class="weight400">Weight: 400 (regular; default)</div>
<div class="weight500">Weight: 500 (medium)</div>
<div class="weight700">Weight: 700 (bold)</div>
<div class="weight900">Weight: 900 (blank)</div>
```

Live Code Example

We did a couple new things here:

1. We set a new custom font using the Google Fonts library. We'll learn about that in the next lesson.
2. We changed the "weight" several times and you see the difference when compared with the prior weight.

   A quick note about fonts and weights. Almost all fonts support regular (400) and bold (700). Many fonts will support light (300). And only some fonts will support all weights. That's why this example uses the Roboto font -- it has great support!

Different font weights are *very* common when you work with a designer or you're designing a highly customized page. It's probably a good idea to remember weights 300, 400 and 700.

Font Families

A font “family” is a how you tell your browser to use a different front. You’ve seen different fonts on different websites -- news websites tend to use a serif-font (a font with ticks) and everywhere else tends to use a sans-serif font (smooth looking).

It’s a good idea to set a default font family on your pages because there’s no standard default font for each browser.

To set a new font, you’ll want to specify a list of fonts to try. Your browser will go through each font, and if the first one is not available, it’ll try the next. And if *that* font isn’t available, it’ll try the next one.

```
body { font-family: ‘Roboto’, ‘Arial’, sans-serif; }
```

The above code will do the following:

1. It’ll look for the font “Roboto”, but if it can’t find that…
2. It’ll look for “Arial”, but if it can’t find that…
3. It’ll load any sans-serif font.

Two things to take note of: first, you can have as many font’s in this list as you like. Second, the sans-serif does *not* have quotations or apostrophes around it because it’s not a named font, it’s just a general font type.

Custom Fonts

When CSS3 was introduced, it allowed us to start using custom fonts. That meant using *any fonts* we could get our hands on, as long as we had access to to the font files.

Font files are files that end in:

- TTF/OTF
- EOT
- WOFF/WOFF2
- SVG

Before we can just use a custom font, the font needs to be on your server (or accessible through your CSS at least) and you need to declare it with @font-face. Let's look at some code.

CSS (style.css)

```
@font-face {
  font-family: customFontName;
  src: url('sample-font.woff');
}

p {
  font-family: customFontName;
}
```

In this example we've declared a new custom font that we can access in any of our CSS by using the name "customFontName" -- feel free to rename that to anything you want, but remember to stick with CSS friendly names. camelCasing your name is usually best. And we've used the src property to tell the browser where the font is located.

Assuming our CSS is in a file called style.css, we would need 'sample-font.woff' to be in the same directory. If you wanted to save your font files in a different directory you can traverse through directories the same way we traverse through the command line. If you're not familiar with command line traversing, the example below should provide some insight.

```
@font-face {
  src: url('../sample-font.woff'); /* Move up a directory */
  src: url('fonts/sample-font.woff'); /* Move into the "fonts" directory*/
}
```

To ensure that your font's are cross-browser friendly (meaning it works properly on all browsers) you'll want to make sure you use as many font files as possible -- because some browsers don't support certain file types (although that's subject to change at anytime).

To maximum browser compatibility, try this:

```
@font-face {
    font-family: 'Comfortaa Regular';
    src: url('Comfortaa.eot');
    src: local('Comfortaa Regular'),
         local('Comfortaa'),
         url('Comfortaa.ttf') format('truetype'),
         url('Comfortaa.svg#font') format('svg');
}
```

Code by Serj Sagan

There's no example for this lesson, but it's highly recommended you download a custom font (any font will work) and give try this out. There's nothing better than hands on experience!

Free Fonts by Google

Google, and it's parent company Alphabet, are insanely large. And they do *a lot* of good, especially for developers.

One of the *amazing* tools that Google gives front end developers are free fonts. Before you get too excited, they don't include specialty fonts like Helvetica for free, but they'll link to a source where you can buy a font. We're not here to buy fonts, we're here to use them!

If you open https://fonts.google.com/ you'll see a wall of different fonts. In this lesson we're going to pick out a font and apply it to a page.

- Step 1. Go to https://fonts.google.com/
- Step 2. Select a font you like. We'll search for Roboto Mono in the search bar.
- Step 3. Click into this font. (or click here)
- Step 4. Click the "Select This Font" button.

- Step 5. Select the "Customize" tab.

- Step 6. Select the following weights: 100, 300, 400, 500, and 700

- Step 7. Go back to the Embed tab.

- Step 8. Select the Embed Font HTML snippet, copy and paste it into your <head> on your page.

- Step 9. Use “Roboto Mono” in your `font-family` properties

Here’s what your code should now look like:

CSS

```
.normal {} /* Placeholder */
.roboto { font-family: 'Roboto Mono', monospace; }
```

HTML

```
<head>
  <link
href="https://fonts.googleapis.com/css?family=Roboto+Mono:100,300,400,500,7
00" rel="stylesheet">
</head>
<body>
  <p class="normal">
    This is a normal font
  </p>
  <p class="roboto">
    This is a Roboto Mono font
  </p>
</body>
```

Live Code Example

Voila! You are now using a custom font loaded by Google. Below is a picture of what the example code looks like in action:

Font Style

Font style is how you can change your font from regular to italic. The `font-style` property has 2 primary values you'll use almost every day: normal and italic. Considering the normal value is… well… normal, we'll skip right past that and demonstrate italics via CSS.

CSS

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

HTML

```html
<p>
  Italics are often used to bring <span class="italic">additional
emphasis</span> to your sentence.
</p>
```

Live Code Example

That's all there is to it! And if you have a style that's already italicized, you can return your font style back to normal with `font-style: normal;`

Apply what you've learned to your template

Remember that basic template you made last module? Open that up in your favorite editor (or CodePen if you're coding in the cloud).

Now find ways to use the following:

- Custom height
- Custom width
- Calc()
- Custom font with a font size (Google Fonts are OK to use)
- Font weight and font style

At this point it doesn't *really* matter *where* you use these, it's just important that you do. Try your hardest *not* to Google or refer to prior lessons -- instead, use your memory. Scientifically speaking, it's better for you to try and remember (even if it's *really* difficult) than it is to quickly refer to the answer. If you get stuck for more than a minute on any of the above properties then you should refer back to the appropriate lesson. And remember: you don't need to know it all off the top of your head right now -- you're still learning and it'll take time to remember everything.

Now go give it your best shot!

(If you need some ideas, we've been making the template with you. Check it out.)

Text Transform

Through CSS you have the ability to change how text is shown. You know how to make fonts bold, italicized and even how to apply custom fonts. But we're no longer talking about font manipulation -- we're talking about changing the text content itself.

Using the text-transform property we can force text to be lowercase, uppercase and capitalized. Let's look at all three values in one example.

CSS

```css
.upper { text-transform: uppercase; }
.lower { text-transform: lowercase; }
.capitalize { text-transform: capitalize; }
```

HTML

```html
<p class="upper">nothing but lowercase letters</p>
<p class="lower">NOTHING BUT UPPERCASE LETTERS</p>
<p class="capitalize">nothing but lowercase letters, again.</p>
```

Live Code Example

The lowercase HTML displayed as uppercase, and the opposite for the .lower class, while .capitalize made the first letter of *every word* uppercase.

If you ever need to reset these values, simply use `text-transform: none;` and your text will go back to the way you wrote it in your HTML.

> *Search engine lesson: search engines no longer "read" your code, they also look at your website, visually. If you have entire paragraphs in capital letters, it might be seen as somewhat spammy. Best to stick with how your language is naturally written, to the exceptions of logos, names, nav bars, call to action buttons and the likes.*

Text Align

Alignment is often important. In most designs you'll use left aligned text, and some centered text. However, there are times when you need to align your text to the right. Here's the latest example that comes to mind.

Features #1-3 (on the left) are aligned to the right. The main title is centered. And features #4-6 are aligned to the left. As you can imagine, this wouldn't look very nice if it was all aligned to the left (or centered).

CSS

```
.left { text-align: left; }
.right { text-align: right; }
.center { text-align: center; }
```

HTML

```
<div class="left">Left aligned text</div>
<div class="right">Right aligned text</div>
<div class="center">Centered text</div>
```

Live Code Example

Left, right and centered text, just like that! CSS is so simple it's almost beautiful.

Justify

Text justification will make your paragraph text line up against the left and right walls nicely, like a newspaper article. The following example continues from the example above.

CSS

```css
.justified { text-align: justify; }
```

HTML

```html
<h4>Left vs. Justified</h4>

<div class="left">Lorem ipsum dolor sit amet, consectetur adipiscing elit. Nullam et tempus magna. Ut pellentesque, dui quis molestie pulvinar, lorem diam cursus libero, nec scelerisque magna orci vitae est. Phasellus sed eros facilisis, hendrerit orci sed, interdum leo.</div>
<br />
<div class="justified">Lorem ipsum dolor sit amet, consectetur adipiscing elit. Nullam et tempus magna. Ut pellentesque, dui quis molestie pulvinar, lorem diam cursus libero, nec scelerisque magna orci vitae est. Phasellus sed eros facilisis, hendrerit orci sed, interdum leo.</div>
```

Live Code Example

It's hard to notice at first, but there is a big difference! The top paragraph aligns all it's text to the left (default behavior). Meanwhile, the bottom paragraph spaces the words out evenly on each line (except the last line) so the first letter of the each line reaches the left wall, and the last letter of each line reaches the right wall.

Text Shadow

With CSS3 we were given the ability to add shadows to our text. Prior to CSS3 we had to make an image with the text (usually in Photoshop) and give it a shadow in there. Images are larger than plain text and require another HTTP Request.

> *Fun fact: whenever your browser needs to get a file such as a font, advertisement, a .js file, a .css file or any other external file that it immediately uses, it does so using an **HTTP Request/Response Protocol**. You know those sites that load quickly, but while you're reading your content bumps down because something loaded? That's often a result of too many HTTP Requests.*

The `text-shadow` property accepts 4 parameters, in this order: *offset-x, offset-y, blur, color.*

CSS

```css
.shadow1 { text-shadow: 5px 5px 1px red; }
.shadow2 { text-shadow: 0 0 5px blue; }
.shadow3 { text-shadow: -10px -10px 5px green; }
```

HTML

```html
<p class="shadow1">Text Shadow #1</p>
<p class="shadow2">Text Shadow #2</p>
<p class="shadow3">Text Shadow #3</p>
```

Live Code Example

The o*ffset-x* and *offset-y* values can be negative numbers, which would move the shadow to the left and up, respectively, as seen in .shadow3.

The color value can be a name, a hex value, rgb or rgba, just like all other color-based styles.

And if you need to reset a text-shadow, simply write `text-shadow: none;`

Borders

Borders come in all colors and sizes. In this course we've used solid borders, like this:

```
.border { border: 1px solid black; }
```

By reading that, you know that there's a solid black border with a one pixel width. This is the shorthand method though, and your browser will end up breaking this down into 3 different components.

```
.border {
  border-color: black;
  border-width: 1px;
  border-style: solid;
}
```

The color and width should be straightforward for you now. However, border-style is a new property to us. Here's a list of available style options:

- `border-style: solid;`
- `border-style: dashed;`
- `border-style: dotted;`
- `border-style: double;`
- `border-style: groove;`
- `border-style: ridge;`
- `border-style: inset;`
- `border-style: outset;`
- `border-style: hidden;`
- `border-style: none;`
- `border-style: solid dotted dashed double;` -- Whoa. What's this?!

Lots of border styles to choose from. Click here to view these in action (demo image below). What's with the border style that has four options? Good question. This isn't a property you run into too often. In fact, the most common border styles are: solid, dashed, dotted, hidden and none.

If you open the example code, right click on "Multiple Borders" and select "Inspect" (sometimes called Inspect Element), you can see how your browser handles code. In the following image we're using Chrome.

It looks like our multiple border example is also shorthand for:

- border-top-style
- border-right-style
- border-bottom-style
- border-left-style

Did you notice how these options were presented in the top-right-bottom-left layout? This tells us the browser is looking at these options in a clockwise fashion. But we knew that already.

Instead of writing shorthand, you can write this out the long way and only apply a border to one side of your element.

And this goes a step deeper. Here's what the browser sees for our entire CodePen example:

It also applies the border-color property to each side, which means we can control those. All we need to do is specify the styling in our CSS and it will take effect immediately.

Did you miss the link to the example?
Here it is again.

And if we added a `border-width: 3px;` style, we'd see the following:

This is like CSS Inception. Styleception. How deep does it go?! Actually, this is as deep as it goes. Browsers like specific rules, but you can write shorthand. In fact, writing shorthand styles is often preferred because it'll keep your .css file smaller and it lets the users browser do more work by unravelling the shorthand into what it should be.

Border Radius

Another beautiful feature that CSS3 brought us is the ability to round corners. Back in the day rounding corners was an awful task -- so awful most of us turned to images for rounding corners, which it wildly inefficient for your HTTP Requests.

CSS

```css
.border {
  border: 1px solid #8c1515;
  width: 100px;
  height: 100px;
}

.round10 { border-radius: 10px }

.roundCustom { border-radius: 4px 10px 0; }
```

HTML

```html
<div class="border round10">
  10px radius
</div>

<div class="border roundCustom">
  Custom rounding (4px, 10px, 0, 10px)
</div>
```

Live Code Example

The first block has an equal 10 pixel radius on all four corners. But the .roundCustom class doesn't. In fact, it's missing a parameter!

Shorthand CSS for borders or corners will always look for 4 parameters (top, right, bottom, left). If you're missing a parameter, it will assume it should be the same as the opposite side.

```
/* Missing the bottom left radius value */
.borderRadius { border-radius: 4px 10px 0; }
```

Your browser will understand this as:

```
/* Autofills the 'left' value with the 'right' value */
.borderRadius { border-radius: 4px 10px 0 10px; }
```

Dealing with the corners

The example code used shorthand again but that's not helpful if we only wanted to add a single rounded corner. Let's look at what the browser sees:

Well, look at that! By using the browser's Inspect tool, we're given the answers to our inevitable question: *what are the individual properties for corner radii?*

If you're wondering which order the shorthand works with, since it can't be top-right-bottom-left, the browser image above tells us it's:

1. Top left
2. Top right
3. Bottom right
4. Bottom left

In other words, it's *still* clockwise, but starts at the top left (just like absolute positions).

Padding & Margin

In the last module you learned about paddings and margins, and how they add spacing inside and outside of your block or inline-block elements.

What we didn't go over was specifying the padding and margin values.

It's pretty rare when you get to add an even 10 pixel padding or margin around an element. Font sizes, line heights, sibling elements… they all play a part in how much padding or margin you should add.

For example, if you have a perfect square with any of the <h1>-<h6> tags or a <p> tag, you'll immediately notice additional margin spacing from these elements, and it pushes your inner content down a bit. Well now your even 10 pixels don't look even. Technically your inner content is still perfectly square, but the margin from the header will trick your eye into thinking it's not.

Specific margin and padding please come to the stage.

Side Specifics

Like our previous examples with border and border-radius, you can also specify the top, right, bottom and left margins and paddings.

Does this image look like the inner content is a square?

The top looks like it has just a few pixels more padding than the right and left. We can either remove the margin from the text inside, or we can offset the padding of the dotted box.

In this lesson we'll add additional padding.

CSS

```css
.square {
  width: 200px;
  height: 200px;
  text-align: center;
  border: 1px dashed black;
  padding-top: 0;
  padding-right: 8px;
  padding-bottom: 8px;
  padding-left: 8px;
}
```

HTML

```html
<div class="square"></div>
```

Live Code Example

If you run that code in your browser, you'll see a padding dotted box. Is it perfect? Probably not. Nor is it meant to be, it's just meant to be a quick fix.

> *Sometimes in front end development you just need to apply a quick fix. Just try not to make a mess, because you'll probably never come back and clean it up (even if you say you will; we all do it).*

The CSS above utilizes the top, right, bottom and left specific paddings. If you ever want to apply specific margins to an element, just swap out "padding-*" for "margin-*" (the * is just a placeholder we've written, it's not a selector we're working with.

Background Colors

By now you should be familiar with background colors and how to add them. We've background colors in *a lot* of examples so far. But this course is meant to be thorough, so let's formally learn background colors.

The Property

The official property for a background color is `background-color` and it can be applied to any element that's visible on the page (almost any HTML element inside the <body>) tag.

Here's a quick example snippet.

```
.newBg { background-color: red; }
```

By reading that you quickly *know* the background color is red.

Different Values

You don't need to stick with named colors. In fact, if you only used named colors your website will look terrible. There are 2 other common ways to tell your website which colors to use:

1. Hex values
2. RGBA values

Hex Values

Hex values look like #ABCDEF or #111222 or a mix of letters and numbers, with between 3 and 6 characters. Numbers are your grey colors and letters are your non-grey colors (basically).

Whenever you find a hex value that has 6 characters in a row that are the same, you can use a shortcut. Like this:

```
.bg {
  background-color: #444444; /* Six 4's */
  background-color: #444; /* Three 4's */
}
```

RGBA Values

Technically, you don't need the "A" in RGBA, but it's a nice feature and we'll talk about this in just a moment.

RGBA stands for Red Green Blue Alpha, which lets you pick 3 color attributes *and* it's translucency.

```
.rgba { background-color: rgba(1, 2, 3, 0.5); }
```

That means:

- 1 red
- 2 green
- 3 blue
- 50% opacity (0.5 is the same as 50% in decimal format)

While that background color *is* black, it's 50% see-through, meaning it'll look lighter *and* see-through.

But if you didn't want to add the alpha parameter you can ignore that (the browser will assume your color is 100% visible), but you need to drop the "A" in RGBA.

```
.rgb { background-color: rgb(1, 2, 3); }
```

For the hassle of writing the RGB values independently, you might as well add the alpha parameter, set it to 1 (default), and now you have a bit more flexibility in your future CSS.

Background Images

In today's internet-connected world, it's difficult to make a beautiful website without using background images.

In HTML, we can create <img /> elements and that will place an image on the page. But let's be honest -- that's extremely limiting. You can add an image, but can you place it where it *needs* to be, fit it to a containing box, make it use specific space or even just add an image with some text over it? Not easily.

But that's going to a problem you'll never face, thanks to background images. In this lesson we're going to learn how to add a background image. In the next lesson we'll learn how to manipulate that image.

Adding An Image

To add a background image via CSS, you'd write a declaration like so:

```
.bgImage { background-image: url('http://website.com/img.png'); }
```

The property is background-image, and the value is a URL with the world and brackets wrapped around it.

As an example, let's add a real image to the <body> element.

```
body {
  background-image:
url('https://s3.amazonaws.com/arkmont/courses/3/covers/the-ultimate-html-developer-course_lg.jpg');
}
```

Live Code Example

Now the entire page has this background image. We can add other HTML elements on top of it, like we normally do. Let's look at one more example.

CSS

```css
.box {
  background-image:
url('https://s-media-cache-ak0.pinimg.com/736x/5c/3c/a2/5c3ca219d7baf296c7e
442f1bcbb9676.jpg');
  height: 250px;
  width: 250px;
  color: red;
  font-size: 40px;
  font-weight: 700;
  text-align: center;
  font-family: 'Arial', sans-serif;
}
```

HTML

```html
<div class="box">
  Background Image
</div>
```

Live Code Example

In this example, you can select the text, but you now you have a background image. Granted this example isn't nice to look at, it *does* give us a good place to start. You can use fill images, or small images that repeat nicely.

What's Missing

Changing how the background repeats itself, it's size, how much space it covers, its position. There's a lot we didn't cover yet. The next lesson will cover these properties.

Background Repeating

Unless you're using a pattern for your background image, you won't want the background to repeat over and over in every direction.

```
body {
  background-image:
url('https://s3.amazonaws.com/arkmont/courses/3/covers/the-ultimate-html-developer-course_lg.jpg');
  background-repeat: no-repeat;
}
```

Live Code Example

To disable repeating background images, just add `background-repeat: no-repeat;` to your style and the image will stop repeating.

Actually, your background image will only show up once, like a normal image. But if the containing box changes shape, your background image won't change at all, which can be a nice effect depending on the design of your site.

Repeat in Directions

Sometimes you'll need to have a long but thin background image that only repeats left/right or up/down. We used to do this for adding box shadow, but CSS3 comes with box shadows, too. Since then repeating a background image across the x-axis (left/right) or the y-axis (up/down) has become less popular. Nevertheless, it's still sometimes used and good to know about.

```
body {
  background-image:
url('https://s3.amazonaws.com/arkmont/courses/3/covers/the-ultimate-html-developer-course_lg.jpg');
  background-repeat: repeat-x;
}
```

Live Code Example

```
body {
  background-image:
url('https://s3.amazonaws.com/arkmont/courses/3/covers/the-ultimate-html-developer-course_lg.jpg');
  background-repeat: repeat-y;
}
```

Live Code Example

The background image now repeats along the x-axis and y-axis, respectively.

Next we'll learn about background image sizing so we can control image size when it's set in a containing element.

Background Sizes

In our previous lessons we set the background image to the page <body>, which by default takes up the entire page. If you apply a background image to a with just one line of text it won't have very much image to show.

Common examples of needing to set a background image size are:

1. Image galleries
2. Banner/header images
3. Cover images (the large image above the fold with text over it)
4. Advertisements

Instead of showing you individual examples, this time we'll demonstrate what a "kitchen sink" is.

> *A kitchen sink is an area where you put all your work so you can see it together. This is fine in practice or for quality testing, but in a real project you'll want to avoid this because it clumps too much code together -- too much code can become unmanageable or exhausting to work with.*

CSS

```
.box {
  float: left;
  height: 300px;
  width: 300px;
  border: 1px solid black;
  background-image:
url('https://s3.amazonaws.com/arkmont/courses/3/covers/the-ultimate-html-developer-course_lg.jpg');
  background-repeat: no-repeat;
}

.box--half { background-size: 50%; }
.box--full-x { background-size: 100%; }
.box--50x50 { background-size: 50px 50px; }
.box--full { background-size: 100% 100%; }
.box--cover { background-size: cover; }
.box--contain { background-size: contain; }
```

HTML

```
<div class="box">Regular Box</div>
<div class="box box--half">Half Box</div>
<div class="box box--full-x">Full X</div>
<div class="box box--50x50">50x50</div>
<div class="box box--full">Full</div>
<div class="box box--cover">Cover</div>
<div class="box box--contain">Contain</div>
```

Live Code Example

By now you have every piece you need to put this puzzle together, metaphorically speaking. But we'll leave you with two tips in this lesson:

1. When there are two values (ie. 50px 50px) you're actually putting in the *x* and *x* axis values, in that order.
2. Get your hands dirty with the Live Code Example

Background Position

Positioning a background can be just an important as sizing it properly, and like all websites, your use depends on your project.

By giving a specific position value, you can tell your browser *where* to place the background image, and that's useful because up until now every background image has started at the top left of it's containing element.

Your background position *should* take 2 parameters, the x-axis and y-axis values.

Every one of those boxes above have the same background image with different background positions attached to them in another CSS class. The code and live example are below.

CSS

```
.box {
  float: left;
  height: 150px;
  width: 200px;
  border: 1px solid black;
  background-image:
url('https://s3.amazonaws.com/arkmont/courses/3/covers/the-ultimate-html-de
veloper-course_lg.jpg');
  background-repeat: no-repeat;
  background-size: 50%;
}

.box--left-top { background-position: left top; }
.box--center-top { background-position: center top; }
.box--right-top { background-position: right top; }

.box--left-center { background-position: left center; }
.box--center-center { background-position: center center; }
.box--right-center { background-position: right center; }

.box--left-bottom { background-position: left bottom; }
.box--center-bottom { background-position: center bottom; }
.box--right-bottom { background-position: right bottom; }

.box--custom1 { background-position: 10px 10px; }
.box--custom2 { background-position: -25px -25px; }
.box--custom3 { background-position: 50% 80%; }

.clear {
  clear: both;
  padding-top: 10px;
}
```

HTML

```
<div class="box">Normal</div>
<div class="clear"></div>

<div class="box box--left-top">Left Top</div>
<div class="box box--center-top">Center Top</div>
<div class="box box--right-top">Center Top</div>
<div class="clear"></div>
```

```
<div class="box box--left-center">Left Center</div>
<div class="box box--center-center">Center Center</div>
<div class="box box--right-center">Center Center</div>
<div class="clear"></div>

<div class="box box--left-bottom">Left Bottom</div>
<div class="box box--center-bottom">Center Bottom</div>
<div class="box box--right-bottom">Center Bottom</div>
<div class="clear"></div>

<div class="box box--custom1">Custom 1</div>
<div class="box box--custom2">Custom 2</div>
<div class="box box--custom3">Custom 3</div>
```

Live Code Example

There's a lot of code to go through, and yet there's not very much to actually teach beyond telling you that the background-position property should have an x and y value, that they can be percents or pixels or names (left/center/right).

Practice

Right now would be the perfect time to create your own banner section of a website using just background CSS properties. Here's what you *can* try to do:

1. Create a that's 100% wide and 200 pixels tall.
2. Fill that with a background image of your choice.
3. Set that background image size so it *covers* the entire banner.
4. Set that background position to be centered vertically and horizontally.

You should *try* to do this without referring to the live example or other lessons, but if you *really* need to, definitely use the material -- it's there to help. And here's the link to this lessons Live Code Example.

Box Shadows

A common theme among websites in 2016/2017 is to have this “flat” theme, where we don’t use too many visual cues that would tell our brains there’s some level, or depth, to a page. There are, however, some exceptions, and box shadows are one of those examples.

> *If you aren’t familiar with CSS Cards, we’ve created an example for you to checkout (with code). Google loves to use cards.*

To create a box shadow, we use the property called… you probably guessed it… box-shadow.

CSS

```css
.box {
  display: inline-block;
  width: 200px;
  height: 200px;
  border: 1px dashed #000;
  background-color: #d5d5d5;
  margin: 10px;
}

.box1 { box-shadow: 0 0 10px #000; }
.box2 { box-shadow: 10px 10px 1px #8c1515; }
.box3 { box-shadow: -10px -10px black; }
```

HTML

```html
<div class="box"></div>
<div class="box box1"></div>
<div class="box box2"></div>
<div class="box box3"></div>
```

Live Code Example

Let’s go through a couple of these examples.

In the CSS code above, .box1 says “0 0 10px #000”. There’s 4 parameters. If you glance at the image above you can probably figure out which params do what. But that’s no way to learn.

In “0 0 10px #000” we have, in this order:

1. The x-axis offset (none)
2. The y-axis offset (none)
3. The blur amount (10px)
4. The background color in shorthand format (#000 -- black)

When the x-axis and y-axis values are set to zero, the background box shadow won’t move, it’ll be perfectly behind the element, that’s why we added a 10px blur, so you can *actually* see the shadow.

The next line is .box2 and it has the same number of params, but it’s “10px 10px 1px #8c1515” value tells the browser to do this:

1. Move the shadow 10px to the right (x-axis)
2. Move the shadow 10px down (y-axis)
3. Blur the shadow by 1px
4. Use #8c1515 as the shadow color

And lastly, we have .box3, the box with the crisp shadow that’s flowing towards the top left. This one is different. It’s missing a parameter! It doesn’t have the blur value set. And as you can see, that’s OK. The browser will assume it doesn’t need to blue anything and let you have a shadow.

The value “-10px -10px black” means:

1. Move the shadow 10px to the left (negative numbers move the shadow in the opposite direction)
2. Move the shadow 10px up (again with the negative number)
3. *Skipped the blur value*
4. Use “black” as the color.

Don’t get too excited though! This is only half the fun.

Inset

Box shadows also give us the option to set an inside shadow. If that sounded strange to you then we’re on the same page. Although an “inside shadow” seems counterintuitive, it’s actually quite nice.

CSS

```css
.box1 { box-shadow: inset 0 0 10px #000; }
.box2 { box-shadow: inset 10px 10px 1px #8c1515; }
.box3 { box-shadow: inset -10px -10px black; }
```

Live Code Example

By adding the keyword “inset” to the list of box-shadow values, we’ve just inverted the shadow. The shadow is on the inside now!

Everything else stays the same. The direction, the blur amount, the color… nothing else changes -- the shadow simply moved to the inside of the element. And as you can see, it looks drastically different.

The 2 right examples with the crisp black shadow and the red shadow… Those are *not* friendly on the eyes and you’ll probably never use those in a real website. But the second box with the light inset shadow (10px blur) actually looks acceptable.

It’s hard to tell you where exactly you’d use these because each website is different, and sometimes an inner shadow just isn’t the answer. In fact, in most situations, *no shadow* is the answer. But it adds a nice effect, especially to CSS Cards, Lightboxes and Modals.

Filters

In the last lesson we learned that shadows can be blurred -- that's just the start!

We can blur, add brightness, add contrast, drop shadow, rotate hue, invest, change the opacity, saturate, sepia and even grayscale an image.

Instead of going through each filter one-by-one, we're going to look at all of them side-by-side (kitchen sink). The contract between each image will help you visually learn what each filter actually does!

CSS

```css
.filter-blur { filter: blur(5px); }
.filter-grayscale { filter: grayscale(100%); }
.filter-hue-rotate { filter: hue-rotate(90deg); }
.filter-drop-shadow { filter: drop-shadow(10px 10px 4px #8c1515); }
.filter-brightness { filter: brightness(150%); }
.filter-contast { filter: contrast(190%); }
.filter-invert { filter: invert(100%); }
.filter-opacity { filter: opacity(15%); }
.filter-saturate { filter: saturate(4); }
.filter-sepia { filter: sepia(100%); }
.filter-mix { filter: contrast(150%) brightness(200%); }
```

HTML

```html
<div class="box">Normal</div>
<div class="box filter-blur">Blur</div>
<div class="box filter-grayscale">Grayscale</div>
<div class="box filter-hue-rotate">Hue Rotate</div>
<div class="box filter-drop-shadow">Drop Shadow</div>
<div class="box filter-brightness">Brightness</div>
<div class="box filter-contrast">Contrast</div>
<div class="box filter-invert">Invert</div>
<div class="box filter-opacity">Opacity</div>
<div class="box filter-saturate">Saturate</div>
<div class="box filter-sepia">Sepia</div>
<div class="box filter-mix">Mix</div>
```

Live Code Example

Back in the day, adding these kinds of effects was *not a simple task*, and if you were one of the lucky people who had access to Photoshop, you still had load up the program, load the image, apply your filter, save the image and to top it off… your website is now loading a different version of this image -- so you *could* be loading 2 or more copies of the same image with a different filter.

Not anymore!

Some of these we've already learned about, like drop shadow (box-shadow) and filter: opacity; (opacity), and we've learned about box-shadow's "blur" value.

But the rest of these are new to this course. When you see these filters, the first thing you *should* think of, as a web developer, is "Instagram". There was a time when their filters weren't much more than this (they are much more complex now). If your brain isn't being flooded with ideas and excitement about what you can do with just CSS, just take a second and think about how much money $1 BILLION dollars is -- because that's what Facebook bought Instagram for -- and they started with simple filters.

Now we're not going to go through all these filters, just because there's so many and all you have to do it change the value (just be careful; they're not all percents).

Placeholder Styling

There are *a lot* of quick wins with CSS, mostly in the naming scheme, like how a background color uses the background-color property. Simple!

And then there are styles like: placeholders, button outlines and scrollbars. Each browser handles these differently so we need to consider each browser when styling a website, as we should do naturally by now.

In regards to placeholder text, we need to apply browser prefixes to make these work.

CSS

```css
/* Chrome/Opera/Safari */
::-webkit-input-placeholder { color: blue; }
/* Firefox 19+ */
::-moz-placeholder { color: blue; }
/* Firefox 18- */
:-moz-placeholder { color: blue; }
/* IE 10+ */
:-ms-input-placeholder { color: blue; }
/* Edge; double colon */
::-ms-input-placeholder { color: blue; }
```

HTML

```html
<input type="text" placeholder="placeholder text" />
```

Live Code Example

As you can see, to change the placeholder color wasn't exactly simple. There's very little standardization across the placeholder property, and so we use pseudo selectors *with* browser prefixes!

If you're looking at this CSS and thinking, "Wow, that's ugly," then we'd say you're correct -- it's not pretty to look at, it's not easy to remember, and we need to support different browsers from different times.

Outline

Remember when we learned about spacing, and how there's inner spacing (padding), outer spacing (margin) and a border between those? Just for fun, let's add one more layer -- because why not?

An outline sits on the outside of your border, like a secondary border. It's not a replacement for your borders, so don't start using `outline` instead of `border`.

CSS

```css
.box {
  width: 200px;
  height: 200px;
  margin: 50px auto;
  border: 1px solid black;
  outline: 10px solid #ccc;
}
```

HTML

```html
<div class="box"></div>
```

Live Code Example

The outline property is very similar to the border property in the sense you can change the width, color and even outline style (instead of solid you can have a dotted border, for example). If you don't remember what all those border options are, that's OK. Just go back to the lesson on borders and you'll find them all there!

Offset

In the event you don't want to have your outline directly touching your border, you can specify an offset value. The `outline-offset` property will add spacing between your border and your outline.

```
.box {
  width: 200px;
  height: 200px;
  margin: 50px auto;
  border: 1px solid black;
  outline: 10px solid #ccc;
  outline-offset: 10px;
}
```

Live Code Example

If your parent element has a background color, you can make it look like there's 3 borders:

1. The black "border"
2. The space between the border and the outline (colored by the parent element)
3. The "outline"

That's all there is to know about outlines! Because you were already familiar with borders, this lesson should be pretty straightforward for you.

Flexbox

CSS3 introduced an incredibly powerful featured called flexbox.

Flexbox is named from the Flexible Box Layout module and it's goal is, essentially, the provide a better way to lay out and align spacing among items in a container, even if we don't know the full height.

Back in the lesson about height, we learned that heights are not easy, unlike it's friendlier and better half: width. But flexbox allows us to work with heights in a new way, like create boxes of the same height, aligning boxes and even vertically aligning text (which, up until now, has been *very* difficult to do).

But flexbox goes one step further and tries to use the rest of the space it's been given, *and* lets you control if a box can grow and shrink, if it should wrap to multiple rows or put all boxes on one jammed line, and more!

In the next few lessons we're going to learn about flexbox and it's power, along with all of it's components that, whether we like it or not, make flex a bit more complex than we'd like.

Before we begin, you should also know that flexbox has specific rules -- and setting *some* properties, like the width property, will not affect a flexbox -- instead, we'll use a different property. More on this in the next few lessons.

Let's begin!

Getting Started With Flexbox

The first thing to know about flex is how to set it up and how it works.

To create a flexbox, you use the display property.

```
.boxes { display: flex; }
```

That will enable flexbox. And it will enable flex for all of its direct children, which means widths don't easily work with any immediate children elements of .boxes.

Each of the flex children (an individual box) is called an "item". Think of how <ol> is an ordered list, and each <li> is a list item. Each box in a flexbox is also known as an item.

The flex items can be re-ordered, though, unlike tradition list items. Let's look at an example.

CSS

```
.boxes {
  display: flex;
  border: 1px solid red;
}

.item {
  flex: 1 1 50%;
  border: 1px solid blue;
}

.item1 {}

.item2 { order: -1; }
```

HTML

```
<div class="boxes">
  <div class="item item1">
    Item 1
  </div>
  <div class="item item2">
    Item 2
  </div>
</div>
```

Live Code Example

Here's what we did with the above code:

1. Set the parent element to be a flexbox by setting `display: flex;`
2. The .item selector tells the browser each "item" can grow and shrink and should start at 50% width. More on this later.
3. .item2 stated the order should be -1. By default the first flex item will have an order value of 0, and by setting .item2 to `order: -1;` we told that entire HTML block to show up *before* the first .item block. But the order of the HTML didn't change!

So why is this important? Because in a world where every website needs to be responsive, you'll need to tell the browser which boxes go in which order on different screen sizes. We'll learn about responsive design in a future module.

Flex: Grow, Shrink and Default Widths

You can dictate which flex items should grow, shrink and what their default widths should be. But here's the amazing, and albeit difficult to understand at first, feature behind flex: you can set a default width and if you allow the item to grow, it can grow beyond that width value, and if you allow it to shrink, the item is allowed to go smaller than the specified width.

CSS

```css
.boxes {
  display: flex;
  border: 1px solid red;
  padding: 10px;
}

.item { border:1px solid blue; }

.item1 {
  flex-grow: 1; /* Is allowed to grow */
  flex-shrink: 1; /* Is allowed to shrink */
  flex-basis: 25%;
}
```

HTML

```html
<div class="boxes">
  <div class="item item1">
    Item 1
  </div>
  <div class="item item2">
    Item 2
  </div>
</div>
```

Live Code Example

The Item 1 (.item1) element was allowed to grow as far as it can (because .item2 did not have a specified flex-basis (flex width)) so it took up as much space as needed.

But if we changed just one little value, it'll drastically change.

CSS

```css
.item1 {
  flex-grow: 1 0; /* Is NOT allowed to grow */
  flex-shrink: 1; /* Is allowed to shrink */
  flex-basis: 25%;
}
```

Live Code Example

Item 1 now takes up a maximum of 25% of its parent width, while Item 2 acts like it's an inline element.

When we add another item to this flex container, and set Item 1 and Item 2 to be 50%, but allow them to shrink, they'll *try* to take up 50% of the width but then shrink down so they are evenly spaced out.

CSS

```css
.item1 {
  flex-grow: 0;
  flex-shrink: 1;
  flex-basis: 50%;
}
.item2 {
  flex-grow: 1;
  flex-shrink: 1; /* Toggle between 1 and 0 */
  flex-basis: 50%;
}
```

HTML

```html
...
  <div class="item item2">
    Item 2
  </div>
  <div class="item item3">
    Item 3
  </div>
...
```

Live Code Example

Item 2 has: flex-shrink: 1;

Item 2 has: flex-shrink: 0;

Changing flex-shrink to zero for .item2 basically told the browser, “50% is the minimum width, make that happen”. And because .item1 has flex-shrink: 1 the browser thought this, “.item1 has extra space, I’m going to borrow from that.” Meanwhile, Item 3 was ignored because we didn’t specify any rules.

These rules are important for making flexible layouts for your websites layout, right down to a basic column.

Equal Widths

Making boxes with equal widths are important. But what if we’re in a situation where there *could* be between 3 and 7 columns at any given time, but we don’t know how many there will be each time?

We need to write some CSS that will make these columns equal no matter what.

CSS

```
.item {
  Border: 1px solid blue;
  flex: 1 1 auto;
}
```

Live Code Example

Three Columns

Seven Columns

All columns are equally sized.

Flex Wrap

Take a look at the following image from last lessons example code:

Seven equal sized columns. If we added 7 more items we'd have a very squished user interface. By default, flexbox will *not* let your items move to another line. Now, in this case there's no specified width so the browser doesn't know when to move certain items to the next row.

In order for flex wrap (move items to a new row) we need 2 things:

1. The parent element needs the flex-wrap property set, and
2. The flex items need a width that's *not* "auto". Auto is the default, so it must be specified.

CSS

```css
.boxes {
  display: flex;
  border: 1px solid red;
  padding: 10px;
  flex-wrap: wrap;
}

.item {
  border: 1px solid blue;
  flex: 1 0 200px; /* Can grow, cannot shrink, basis of 200px */
  margin-bottom: 20px;
}
```

HTML

```html
<div class="boxes">
  <div class="item item1">Item 1</div>
  <div class="item item2">Item 2</div>
  <div class="item item3">Item 3</div>
  <div class="item item4">Item 4</div>
  <div class="item item5">Item 5</div>
  <div class="item item6">Item 6</div>
  <div class="item item7">Item 7</div>
</div>
```

Live Code Example

What we have is 7 columns that start at 200px, can grow to take up more space on each line, cannot go below 200px and any items that go beyond the red border (the parent element) will be wrapped to a new line.

Below are screenshots at different screen sizes:

5 items with 200px or more in width. Bottom 2 are >200px so they take up equal space.

4 items with 200px of more in width. Bottom 3 are >200px and also take up equal space.

What happens when you shrink your browser; each item gets its own row.

The reason for the equal space is because .item has flex-grow: 1; set in it's shorthand CSS.

```
.item { flex: 1 0 200px }
/* ^^ flex-grow: 1; flex-shrink: 0; flex-basis: 200px; */
```

If you changed the flex-grow value to zero, then the items must always be 200px, and we'll get results like this:

When would you use this? Well… you can use this instead of the float property and get better results. Or as a grid system for holding your content in place, like an image gallery or navigation bar.

Same Height Boxes

Prior to flexbox, if you want a box with the same height as another box you'd need to:

- Specify an exact height for both boxes, or
- Use a table (frowned upon)

But now you can use flexbox to get the same results. And it's so much easier.

CSS

```css
.boxes { display: flex; }

.item {
  border: 1px solid blue;
  flex: 1 1 50%;
  padding: 10px;
}
```

HTML

```html
<div class="boxes">
  <div class="item item1">Item 1</div>
  <div class="item item2">
    Lorem ipsum dolor sit amet, consectetur adipiscing elit…
  </div>
</div>
```

Live Code Example

Huh… look at that! The boxes are already the same height! We didn't do anything extra and it just did it for us. Isn't that nice!?

Vertical Text

Here's another feature that's always been difficult to achieve, but has been resolved with flexbox: aligning your text in the middle (vertical alignment)

```
.boxes {
  display: flex;
  align-items: center;
}
```

Live Code Example

By adding `align-items: center;` you're telling the browser to make each box align right through the vertical center point. See the image below.

If you remove the blue border, the left box (Item 1) would look perfectly centered along the y axis. It doesn't matter if you add more text to the left *or* the right box, it'll just center itself again. But by doing this you'll vertically center *all* your items inside a flex container. By aligning all items in the center you'll get this:

That's no good. What if, for whatever reason, you just wanted to align a single flex item in the center? Then you'd use align-self on the individual item.

Align Self

The align-self property only works on flex children (called items), there's no need to set this on the flexbox container (unless the flexbox container is also a flex item to its parent element; Flexeption?!)

CSS

```css
.boxes {
  display: flex;
}

.item {
  border: 1px solid blue;
  flex: 1 1 50%;
  padding: 10px;
}

.item3 {
  align-self: center;
}
```

HTML

```html
<div class="boxes">
  <div class="item item1">
    Item 1
  </div>
  <div class="item item2">
    Lorem ipsum dolor sit amet, consectetur adipiscing elit...
  </div>
  <div class="item item3">
    Item 3
  </div>
</div>
```

Live Code Example

Perfect, now you're centering just one flex item at a time!

Flexbox Resources

This course isn't designed to go over *every* component in CSS, but rather go over the important and common parts of everyday CSS that'll you'll need to know about before being able to land a front end developer job or to start freelancing and making beautiful websites.

As an addition to the flexbox lessons, we're also going to give you a few *great* resources to use. We may expand more on flexbox in the future and cover what these resources offer you, but getting outside of the classroom and reading code written by other people is also important.

- CSS Tricks: A Guide to Flexbox
- Free Code Camp: How Flexbox Works Explained with Gifs
- Free Code Camp: Understanding Flexbox
- Flexbox in 3 Minutes
- LearnLayout.com: Flexbox
- MDN Flexible Boxes

CSS2 vs. CSS3

When CSS1 jumped to CSS2 we saw some nice new features, but there weren't that many developers around the world yet, and so the CSS2 news was relatively quiet.

But CSS2 to CSS3 is a different story.

As of 2017, some CSS3 standards are still being worked on -- such as grids. But features like Flexbox and border-radius have been established and are now supported by all the major browsers. That's what we want. As developers we want more standardized features that ship with browser updates. There's just one little caveat: old browsers.

Welcome to Front End Web Development

We all wish coding was easy to learn and easy to do. And just to make things a bit more complex than they need to be, let's throw old browsers into the mix.

While modern browsers today can use CSS3 features like Flexbox without any problems, old browsers may still need vendor prefixes, or worse, they may not support a certain feature at all!

If we travelled back in time and used a version of Internet Explorer 7 (probably the worst use of a time machine… ever!) we'd realize it didn't support flexbox. That's OK because we had ways around that -- albeit our solutions back then weren't very elegant.

But what if we went back to 2003 and wanted to have rounded corners on an element? We'd need to use JavaScript, which wasn't as popular as it is today -- not even close to being as popular as it is today. So then we'd use images to create a rounded border, but the border wouldn't line up properly with a corner image -- we'd end up using images for the *entire border*. Seriously. As if the internet wasn't slow enough in 2003, adding several HTTP requests just to display a rounded border made loading websites even slower.

Why am I telling you this?

Because those browsers *are still in use*. I wish that was a joke; it isn't. We don't see many out dated browsers in countries like the UK, US, Canada, and other nations that have steady internet connections. But you need to consider the target audience of your website.

And alas, we get to the major learning point in web development: good coders code; great coders understand problems beyond code. We want you to be a *great coder*.

If you're ever making a website that's supposed to get the attention of people in a city in central Africa, where they have internet but it's incredibly slow, you can also assume they're using older

versions of browsers that don't support features like flex or border-radius. If you try to use these new features on a website where most people have old browsers, the site will look terrible for them and they'll probably find a different website.

So knowing *some* of the CSS3 features *and* your target audience can be a very powerful tool to tuck away in your back pocket.

Now, for a hint of relief.

There aren't many websites that *need* to support older browsers. You can typically get away with support the last 2 or 3 versions of a browser. An excellent resource for you to use is https://caniuse.com/ -- enter the word "flexbox" and you'll see which browsers support it and which browsers do not.

So if this isn't *really* a problem that comes up too often, why are you being told all of this? It's simple really: 2 or 3 years from now you'll be in this situation, where we'll all be using new CSS features but many people will be using browsers that were last updated in 2017 -- which means you *may* need to support our current browsers in the future, depending on how fast CSS standards can be written and supported.

3D Rotate

Rotate, a CSS3 feature, can happen in 3 directions: x, y and z.

> *A quick reminder: x is your left-right axis, y is your up-down axis, z is your in-out axis.*

Rotating an element actually uses the transform property, which lets you do more than just rotate. But since you'll be making beautiful and modern websites, you'll probably use rotate fairly often. Are you ready for some *fun* CSS?

CSS

```css
.flex {
  display: flex;
}

.item {
  flex: 0 0 100px;
  border: 1px solid black;
  height: 100px;
  margin: 20px;
  /* A flexbox inside a flexbox
     to create vertical text inside
     a regular flex item */
  display: flex;
  justify-content: center;
  flex-direction: column;
  text-align: center;
}

.rotate-x {
  transform: rotateX(45deg);
}

.rotate-y {
  transform: rotateY(45deg);
}

.rotate-z {
  transform: rotateZ(45deg);
}
```

HTML

```
<div class="flex">
  <div class="item rotate-none">No rotation</div>
  <div class="item rotate-x">Rotate X</div>
  <div class="item rotate-y">Rotate Y</div>
  <div class="item rotate-z">Rotate Z</div>
</div>
```

Live Code Example

Maybe just take a minute and re-read that CSS. Something funny (not ha-ha funny) happened in there. We added a flexbox inside a flexbox item in order to vertically center the text inside a flex item without resizing the flex item.

That's not the lesson though. Move along. Move along.

Above are 4 square boxes. Believe it or not, they *are* all square. But they are being rotated on different axis. It's hard to tell *why* the middle two are being squashed inwards, but it's like taking a piece of paper (flat) and rotating it while keeping it flat -- you'll see it from different angles.

Here's an example where you can hover your mouse over the element to see it's 3D transformation effect.

One last thing: you can mix and match your transformations.

```
.rotate-zx { transform: rotateZ(45deg) rotateX(45deg); }
```

That's a valid line of CSS! You can add more transformations, too!

Scale

A CSS3 transform value known as “scale” is the one that’s responsible for making your elements larger and trying to accurately scale the quality up or down.

CSS

```css
.flex {
  display: flex;
  flex-wrap: wrap;
}

.item {
  flex: 0 0 100px;
  border: 1px solid black;
  height: 100px;
  margin: 20px;
  /* A flexbox inside a flexbox
     to create vertical text inside
     a regular flex item */
  display: flex;
  justify-content: center;
  flex-direction: column;
  text-align: center;
  transition: transform 1.5s ease;
}

.scaled:hover { transform: scale(3, 2); }
```

HTML

```html
<div class="flex">
  <div class="item rotate-none">No scale</div>
  <div class="item scaled">Scale(3, 2)</div>
</div>
```

Live Code Example

The values that the scale() function takes are the width and height to scale. In our example of scale(3, 2) we told the browser to scale the right block by making the width 3x the regular width, and 2x the regular height.

This won't change your font size either -- your font size will stay the same. Although it looks bigger, you can think of scale() as a zoom-like feature, where you're zooming in. And like a microscope, when you zoom in, the image *looks* bigger but the actual size has not changed.

And in this example, scale() didn't scale the width and height proportionally (3x width; 2x height), so the width is stretching out the text making it look strange, instead of nicely scaled up.

One thing to take note of when using the scale() transform: occasionally, and depending on the browser, the text and images that you scale may not stay in focus -- in other words, the scale() function might have some side effects where the image and/or text becomes just a little bit blurry.

Animations

For the better part of the last decade we've used JavaScript for animations, and JS libraries like jQuery and jQuery UI -- but CSS3 allows us to write your own animations now, and they are *much* smoother than a JavaScript-based animation.

If you're ever faced with a decision to make for animations: JavaScript or CSS, choose CSS whenever you can. CSS is great at the simple animations like moved an element left and right, zooming in, scaling up, rotating and skewing (just to name a few). But JavaScript can handle the complex animations quite well. And libraries like jQuery come with some animated features built it -- and they support all the major browsers, whereas a CSS animation will only work on a CSS3 enabled browser. So if your target audience will most likely be using older browsers, JS is your answer. But if you need smooth and simple animations, like what you see on a mobile app, CSS is your answer (some exceptions apply, as always).

With a CSS animation, you can change as many properties as you want… as many times as you want! But there's one catch: you need to specify keyframes. Keyframes are like stopping points.

Say you're driving out to the lake this weekend. You'll want to stop at the grocery store for food and the gas station for fuel. You currently have three destinations before arriving at your final (third) destination.

1. The grocery store,
2. The gas station,
3. The lake.

And of course, your house, where you start your journey. At each stop you can change your CSS properties, and while you're in transit those properties are being animated.

Keyframes

Keyframes are a CSS rule, not a property, which means they start with @keyframes. We'll look at an example in a minute. In order for an animation to work, you must specify the keyframes, the name of your animation, and then bind that animation to your CSS declaration. If that was scary sounding, just relax, we'll go through a full example with live code.

Below is an example of a simple animation. It takes the "start" and "end" values as "from" and "to".

CSS

```css
@keyframes exampleAnimation {
    from {
      background-color: white;
    }

    to {
      background-color: #8c1515;
   }
}

.box {
  border: 1px solid black;
  background-color: white;
  width: 100px;
  height: 100px;
  animation-name: exampleAnimation;
  animation-duration: 2s;
  animation-iteration-count: infinite;
}
```

HTML

```html
<div class="box"></div>
```

Live Code Example

This transitions from white to dark red;
open the Live Code Example to see it in action

In our CSS there are 3 animation declarations inside .box, 2 declarations inside @keyframes and 1 @keyframe rule.

The @keyframe rule has a space and then the name of the animation. That's the name we use to bind the animation to the .box element.

Inside @keyframes we have the **from** and **to** declarations, which take the properties you want to change and well… change them.

Lastly we have:

- animation-name
- animation-duration, and
- animation-iteration-count.

Animation name is the name you specified right after the @keywords rule. Animation during is how long the entire animation should take. Animation iteration count is how many times this animation should play -- we specified infinite so the animation will never stop, but you can just as easily specify 1, or 100, or any other number you want.

Ready for a large and more advanced CSS example? We'll look at the code, then go over what it does in each step.

CSS

```
@keyframes exampleAnimation {
    0% {
      background-color: white;
    }

    25% {
      background-color: gold;
    }

    50% {
      background-color: black;
    }

    75% {
      background-color: black;
    }

    100% {
      background-color: #8c1515;
      color: #fff;
   }
}

.box {
  border: 1px solid black;
  background-color: white;
  width: 100px;
  height: 100px;
  animation-name: exampleAnimation;
```

```
  animation-duration: 4s;
  animation-iteration-count: infinite;
  animation-delay: 2s;
  animation-direction: alternate;
  animation-timing-function: ease;
}
```

HTML

```
<div class="box">Text</div>
```

Live Code Example

No preview available due to animation.
View Live Code Example.

In this example, we specified different percents in the animation instead of using the keywords **from** and **to**. You can set those percents to anything you want (between 0 and 100). This example is saying:

1. Start with a white background
2. When it's 25% through the 4 second animation (4s is from the animation-duration property), change the background to gold
3. At 50% change the background to black
4. And 75% don't do anything -- just keep the background black
5. At 100% turn the background dark red (#8c1515) *and* change the text color to white.

We didn't specify how to change the text color until the very end of the animation. By default, your browser will use the style from the .box CSS declaration (default is black). And because we didn't specify what to change the text color to *at all* between 0% and 99%, the browser will assume you want to animate the current color to the ending color (#FFF; white) but do it linearly.

In .box, we also set 3 new properties:

- animation-delay,
- animation-direction, and
- animation-timing-function

The animation-delay means the animation will *not* start for 2 seconds. You can set that to any value you need, in seconds. Animation direction is the direction the animation should play in. We specified it should "alternate", which means the animation will play forward, then backward, then forward, and continue alternating until the animation is complete. Lastly, the animation-timing-function is how you want the animation to run. The following values are all acceptable for the animation-timing-function:

- ease - Animate with a slow start, then speeds up, and ends slowly (default)
- linear - Animate with the same speed throughout the entire animation

- ease-in - Animation with a slow start
- ease-out - Animate with a slow end
- ease-in-out - Animate with a slow start and end
- cubic-bezier(n,n,n,n) - Define your own animation speed with a cubic-bezier function

I've thrown together a couple examples so you can get your hands dirty and play around with the code. Here's the Animation Kitchen Sink Example and here's an example of how you could apply an animation in real life (one that looks half decent).

Box Sizing

Whenever we add padding and a border to an element, it *adds* those values to the total width. For example, if you had an element that's 200 pixels wide with a 20 pixel padding, your entire element would *actually* be 240 pixels wide (200px + 20px + 20px; don't forget about both sides of your element).

This kind of behavior can create problems in your code. For example, if you need a menu system that can only be 250px wide, but should have at least a 10px padding, your menu will be wider than you expected and it might squish your other elements down.

To solve this problem, we can set a property called box-sizing to have the value of border-box. Let's look at an example to get a better about how this solution works.

CSS

```css
.box {
  width: 200px;
  height: 20px;
  border: 1px solid black;
}

.box--default {
  padding: 20px;
}

.box--border-sized {
  box-sizing: border-box;
  padding: 20px;
}
```

HTML

```html
<div class="box box--default">200px + 20px padding</div>
<div class="box box--border-sized">Border-box</div>
```

Live Code Example

The top box has a width of 200px, but it also has 20px on the left *and* on the right, which expanded the entire element.

Whereas the bottom box has a 200px width, but the padding *is included* in its total width.

How important is this to know? Very. This absolutely *will* be a problem in the future when working with pixel-perfect or near pixel-perfect designs.

A lot of CSS3 sites will set all elements to use *border-box* instead of its default value of *content-box*.

A quick shortcut to setting a property on *all* elements is to use the wildcard selector. But **be careful** when setting a value for *every* element on your page, it may end up causing unaccounted for affects.

To add this property with a wildcard selector, see the code below.

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

Gradients

There are three secrets about developers that they don't always like to admit, but are almost *always* true for every dev.

1. We're lazy. We don't like writing the same code more than once.
2. We use Google and Stackoverflow *a lot*. No shame -- technology moves faster than we can learn so we use the internet's "hive mind" to help us code.
3. We like tools that do a great job. Because reason #1, that's why.

Gradients are *super* cool and used all over the place. Back in ye olde day we needed to use images for a background gradient -- but those days are long gone, thank goodness! Rather, we use CSS, as you probably guessed.

But styling gradients isn't simple. It's a lot of work and a lot to remember, and depending on what you want to do it has browser compatibility problems. **in steps a cool tool**

The tool I'm about to introduce to you is a graphical interface for creating CSS gradients. It's probably not the *only* tool out there, but it's darn powerful. Here's the URL:

http://www.colorzilla.com/gradient-editor/

Now, I'm not about to leave you hanging. To get a better understanding about how gradients work, I'm expecting you'll read through the code it generates to get a better understanding of gradients and the hardship that can come with them.

SVG Images

SVG stands for Scalable Vector Graphic, and it's wildly different from a regular image. A regular image, if you scale it up or scale it too far down, becomes blurry, pixelated or generally unrecognizable. SVGs on the other hand do not -- they scale beautifully.

You've undoubtedly seen an SVG as an image by now. Small icons are often SVG's precisely because they scale so well -- you can scale the image using HTML and/or CSS to 150x150 pixels and it'll still look sharp.

Besides scaling nicely, SVG's tend to compress nicely too, which makes them an ideal candidate for images that *need* to be crisp at any size. But there's one caveat: SVGs are not made easily. It's not like opening an image in Photoshop and saving the image as an .svg, although with the right plugin that's probably doable.

Dissecting the SVG

Unlike regular images such as .jpg, .png and .gif, an .svg file is somewhat readable by humans. If you open an SVG and read the code, you'll see what I mean.

> *If you need a sample SVG, we have one hosted for you here.*

The code inside an SVG, especially our example SVG, isn't *easy* read, but it *is* easier than a typical jpg, and it comes with XML-like elements and attributes, too.

```
... <g display="none"><g display="inline"> ...
```

<g> is the element and "display" is the attribute, just like plain HTML. There's one attribute you'll want to pay particular attention to, and that's the "fill" attribute inside the <path> element.

```
<path fill="#000000" d="M434.889,279.48...
```

The value for the "fill" attribute looks pretty familiar. We know that #000000 (or #000 for short) is black. What would happen if you edited this SVG in your text editor and changed that value to, say, #f00? If you're feeling experimental, go ahead and give that a try!

Using SVGs

You can apply an SVG just like you apply an image.

```
<img
src="https://s3.amazonaws.com/arkmont/files/css-masterclass/css-icon.svg"
width="200" />
```

Live Code Example

Wow, it looks just like a normal image! How fun! **sarcasm**

Technically, using SVGs is part of the HTML5 spec, and not the CSS3 spec. *But* we *can* use SVGs in our CSS instead of images. Just use them as background images, and they'll continue to scale nicely!

CSS

```
.box {
  width: 200px;
  height: 200px;
  border: 1px solid black;
  background-image:
url('https://s3.amazonaws.com/arkmont/files/css-masterclass/css-icon.svg');
  background-repeat: no-repeat;
  background-size: contain;
  background-position: center center;
}
```

Live Code Example

It looks like a regular image, doesn't it? But if you change the height and width of the .box the image will scale with you! Also… you can apply CSS to an <img /> with an SVG and it'll scale nicely, too.

But if this is more of an HTML5 spec than a CSS3 spec, why are we learning this? Great question!

Because many CSS3 sites will try to take advantage SVGs small size and amazing scaling. And you should know to work with them.

Another part of the HTML5 spec that we didn't learn about in this lesson was the <svg> tag. If you put your SVG code directly into your page, you can use CSS to control some of the colors, too, by specifying the fill property. We're not going to go over that right now, but that'd be one amazing hands-on experiment to do on your own. **hint hint**

Pseudo Selectors

A pseudo class is a special state, like when you hover your mouse over a link. We've already used a few of these when writing links (hover, active, focus, visited). *Now* we can access even more.

But first, here's what a pseudo class looks like:

```
.selector:pseudo-selector {

}
```

Or in a *real* example:

```
/* Unvisited Link */
a:link {
  color: #fff;
  background-color: #0e0111;
}
```

> *Quick note: you can add a pseudo class to any CSS declaration.*

There are quite a few pseudo class states, and CSS3 introduced a lot of powerful pseudo's. But the truth is… you won't use most of them. And after this course, you'll most likely forget a number of them. So we're only going to learn the most common pseudo classes and lightly pass by the others.

Before

This pseudo class lets you create a style that will show up *before* your CSS selector. You can even add content through CSS.

CSS

```
.box {
  width: 200px;
  height: 200px;
  border: 1px solid black;
  margin: 30px auto;
}
```

```css
.box::before {
  display: block;
  width: 100%;
  height: 100%;
  border: 1px solid blue;
  content: ' ';
  transform: translate(-15px, -15px);
}
```

HTML

```html
<div class="box"></div>
```

Live Code Example

Our .box selector was just a plain box with a black border. By adding `::before` to the selector, you're able to create *more* styling. But there's 2 things to watch for:

1. We used the `content: ' ';` property and value. This was required to make the make the "before" section show up. This is also where you'd add text to your code.
2. We used 2 colons in a row. CSS2 used one colon, CSS3 uses the standardized double colon approach. Here's a CodePen of *just the text* in a real scenario.

After

The after pseudo works just like the before pseudo, with the exception that the styles you write come *after* the selector the pseudo class is appended to.

CSS

```css
.box::after {
  display: block;
  width: 100%;
  height: 100%;
```

```
  border: 1px solid red;
  content: '::after';
  transform: translate(15px, 15px);
}
```

Live Code Example

This time we added a red box *after* the black box and told it to move 15px right and 15px down. It looks like it moved *more* than 15px down, though. Why is that? Well, it's because the ::after box comes *after* the content inside the black box, which also takes up space. So it moved down 15px in relation to the content text "Regular".

> *Note about **before** and **after** pseudos: They act like real elements so they can conflict with each other. In other words, they don't take up the same space at the same time by default. To make them overlap you'll need to specify special styling.*

First Child

This pseudo will select the first child in a group of sibling elements. If that didn't make sense to you, that's OK, we'll work through an example and it'll make more sense.

CSS

```
ul li:first-child {
  background-color: black;
  color: #ccc;
  padding: 10px;
}
```

HTML

```
<ul>
  <li>Number one</li>
  <li>Number two</li>
  <li>Number three</li>
  <li>Number four</li>
  <li>Number five</li>
</ul>
```

Live Code Example

In this example, there are 5 sibling <li> elements (or you can say there are 5 child <li> elements) and one parent <ul> element.

Of those 5 's, only select the first one. That's what :first-child does; selects the first of it's kind between it and it's siblings.

Also note the *single* colon.

Last Child

Exactly the same as :first-child, but with the last child.

```
ul li:last-child {
  background-color: black;
  color: #ccc;
  padding: 10px;
}
```

Live Code Example

Again, watch for the single colon.

Even and Odd

To select every 2nd element we can use the :nth-child() pseudo with two keywords: even, or odd.

```
ul li:nth-child(even) {
  background-color: yellow;
}

ul li:nth-child(odd) {
  background-color: blue;
  color: #fff;
}
```

Live Code Example

Even and odd. Pretty straightforward. But that :nth-child() pseudo… that's new! And while it might look daunting, it means well. This is the way you select *any* specific child. You can swap out the keywords "even" and "odd" for a number, and it will only selec *that* child you enter.

The :first-child selector is simply a shortcut for :nth-child(1).

Every Other

There will be a time when you need to select every 3rd element but skip a few of them first. For this, we'd use the :nth-child(*an+b*) formula, where:

- *a* is the cycle length (2 would be even, 3 would be every 3rd, and so on…)
- *n* is the number counter which starts at 0
- *b* is the offset value. If you wanted to skip the first 2, *b=2*.

```
ul li:nth-child(3n+2) {
  background-color: #8c1515;
  color: #fff;
  display: inline-block;
}
```

Live Code Example

Here we are selecting every 3rd element, starting at the 2nd element. Or… :nth-child(3n+2);

This is a pseudo selector that makes a lot of people think two thoughts: “*oh great, math!”* and *“when will I ever use this?”*. Besides the calc() function, this is the most amount of math you’ll do in CSS, and for a reason to use this: patterns.

Live Code Example

Here’s a patterns that’s not uncommon for a designer to make, and you’ll need to make this work. You *could* write JavaScript to do this, but why write JavaScript when CSS works better with less effort?

All Other Pseudo Classes (Alphabetical Order)

Pseudo Selector	Example Use	Brief Description
:active	a:active	Select the active link
:checked	input:checked	Select every <input> element that's checked
:disabled	input:disabled	Select every <input> element that's disabled
:empty	div:empty	Select every element that have no children elements
:enabled	input:enabled	Select every <input> element that's enabled. Opposite of :disabled.
:first-child	li:first-child	Select every element that's the first of it's kind in it's group of sibling elements.
:first-of-type	div:first-of-type	Select every element that is the first element of its parent
:focus	input:focus	Select the <input> element that has been clicked into (when you're about to type).
:hover	a:hover	Changes the style when you put your mouse over an element
:in-range	input:in-range	Select <input> elements with a value within a specified range. Used with HTML5 form validation.
:invalid	input:invalid	Select all <input> elements with an invalid value. Used with HTML5 form validation.
:lang(*language*)	div:lang(en)	Select every element with a lang attribute value starting with "*en*".
:last-child	li:last-child	Select every element that is the last child. Similar to :first-child, but selects the last element.
:last-of-type	p:last-of-type	Select every <p> element that is the last <p> element of its parent
:link	a:link	Select all unvisited links
:not(selector)	:not(div)	Select every element that is not a element.

:nth-child(n)	li:nth-child(even|odd|*number*)	Select every element that is either the 2nd element (even or odd) or a specific number.
:nth-last-child(n)	li:nth-last-child(2)	Select every element that is the second child of its parent, counting from the last child
:nth-last-of-type(n)	p:nth-last-of-type(4)	Select every <p> element that is the 4th <p> element of its parent, counting from the last child
:nth-of-type(n)	li:nth-of-type(3)	Select every element that is the 3rd element of its parent
:only-of-type	p:only-of-type	Select every <p> element that is the only <p> element of its parent
:only-child	p:only-child	Select every <p> element that is the only child of its parent
:optional	input:optional	Select <input> elements with no "required" HTML attribute
:out-of-range	input:out-of-range	Select <input> elements with a value outside a specified range. Used with HTML5 form validation.
:read-only	input:read-only	Select <input> elements with a "readonly" HTML attribute specified
:read-write	input:read-write	Select <input> elements with no "readonly" HTML attribute
:required	input:required	Select <input> elements with a "required" HTML attribute specified
:root	root	Select the document's root element (top of the DOM)
:target	#post:target	Select the current active #post element .
:valid	input:valid	Select all <input> elements with a valid value. Used with HTML5 form validation.
:visited	a:visited	Select all visited links

Responsive Design Introduction

Welcome to responsive design! This is the part where you gain skills that are incredibly valuable as a front end web developer. Everything leading up to this point was necessary for creating a basic website. But it's not enough anymore.

If this was 2005, you'd be set to start applying for a front end development job and/or start freelancing. Unfortunately, those days are long gone and what's required from a front end developer is much higher due to new devices, namely mobile devices and large screens.

When you create a website, you *need* to consider how it will look and act on mobile devices of *all* sizes. What looks good on your laptop probably looks bad on a phone, and looks kind of strange on a tablet -- and requires users to pinch their screen and zoom in. With responsive design you can make your website look and feel more like an app -- that's the goal anyway.

To make matters more complicated, browsers tend to handle code differently. What works on Google Chrome on your laptop might look a bit different on Chrome on Android; likewise with Safari on Apple products.

Up until now we've built a website for a standard laptop with dimensions around 1366x768. But what happens if you shrink your page down to 320x500? Or 768x480? Or *any* other size? And *then* what happens when you look at your website on Chrome, Firefox, Safari, IE, Edge and Opera? What about the mobile versions? What about the default versions that come with some mobile devices, Internet or Dolphin? When you consider all these complexities and start building a website that looks great on every (well… almost every) device that's when you truly become a valuable front end developer. And don't worry, it's not as scary as you think.

In this module we'll explore browser considerations, the Inspect tool, different devices, who you should and should not build for, media queries, frameworks that make life easy for you and more.

Browser Considerations

When you build a responsive website you need to make sure it responds well to other browsers, not just mobile devices. The browsers we tend to consider *most* are:

- Google Chrome
- Mozilla Firefox
- Apple's Safari
- Opera
- Internet Explorer
- Microsoft Edge

Generally speaking, these are the ones you need to satisfy because most of the world uses these browsers.

It's an impossible task to consider *every* browser ever made, and frankly, if smaller browsers can't stay up to date with modern practices like Chrome or Safari, your users probably shouldn't be using them -- by using old browsers or browser that don't support what the big 6 browsers support you'll be missing out rich features like Flexbox and Grids.

If you *really* want to support more browsers by all means go for it. But after supporting the 6 major browsers you'll most likely be wasting your time. In fact, some developers have even stopped supporting Internet Explorer because Edge is its successor and has much better support for modern features.

The general front end rule, for the typical internet user, is to support the last 2 or 3 major versions of all 6 browsers (some exceptions apply with Internet Explorer, however, since it's now outdated and comes with a lot of painful quirks).

As a heads up, if you ever have to develop a website for Internet Explorer users, namely IE6-9, you'll most likely run into a number of problems that *only affect Internet Explorer*. Some developers, as mentioned, have stopped supporting IE due to security flaws, bad practices, wasted time fixing a tiny bug and incompatibility issues.

My suggestion is to do your best for Internet Explorer, but don't waste too many hours fixing small features -- IE is one of the least popular among to big 6. But definitely support Edge -- just because the two browsers are from the same family doesn't mean we should be biased against the newest version -- it's popular, and frankly, Edge is a pretty good browser. (Yes, I said it! Don't knock it till you try it).

Inspect Tools

These tools are life savers, especially when it comes to debugging JavaScript. For now we'll be sticking with CSS and how it can help us understand and test our live code a bit better.

When you're on *any* page, you can right click and select "Inspect" or "Inspect Element".

Once you open your Inspect tool, a window at the bottom (or the right) of the page will show up. It'll look something like this:

With this tool you can click around and see what every element is doing on your page. It's *very* helpful when it comes to viewing padding and margins, but you can also see everything the

browser can see, the way the your browser sees it. Oh, and you can change the HTML in your Inspect tools, too!

But wait, there's more!

You can change the CSS values, too. Select an element and you'll see all the CSS proeprties that are in use. Select one of those properties and you can change the value.

In the above example, you *could* enter almost any valid number and measurement. If you so desired, you could change the font-size of 20px to 170px.

If you're wondering what this has to do with responsive design… it has *nothing* to do with it. But your Inspector is a valuable tool for HTML, CSS and JavaScript development and when you start writing more complex responsive CSS you'll run into little problems that make you think, "Why is that happening?!", and your Inspector will help you understand what's going on and what you can tweak.

Media Queries

Media queries are CSS rules that tell the browser how to act in certain situations. For example, if you're viewing a website on your phone it might only be 375px wide (or less), and a 4K SmartTV might be 2560px wide (or more).

The average laptop is about 1366 pixels wide. The average phone is only 320 pixels wide. The average tablet is 768px-1024px wide. The average user uses all of these devices over time so you *cannot* ignore them. In fact, it'd be foolish to ignore them.

Instead of targeting "laptops" since their resolutions can greatly vary, you'll want to target the resolution.

> *Heads up! Media queries can make your CSS files very long, since it's overwriting some of the previous properties in a style.*

Here's what a typical media query looks like:

```
@media (max-width: 768px) {
  .selector1 {}
  .selector2 {
    ...
  }
}
```

Every media query starts with the @ symbol -- that's how CSS knows this is a rule and not a selector. We do the same thing with custom fonts, too. Then you need to specify your rule. In this case the rule was "a maximum screen width of 768px", but be careful, this isn't the viewer's screen resolution -- it's their browser viewport! Then we open the curly brackets and place regular CSS inside of it. It looks like CSS inside CSS.

Media queries aren't limited to just max-width either, they can be set on a min-width, orientation (like a sideways phone), if the page is being printed, and even the browser height. Those are just a few, and there are even more!

The three most common in web development are: max-width, min-width and print. Lets look at an example -- beware, this is going to seem like a lot of code but *please* take the time to read through it! Knowing how to read code, especially code from other devs, is absolutely vital to your success as a developer!

CSS

```css
.box {
  display: none;
  width: 200px;
  height: 200px;
  border: 1px solid black;
  flex-direction: column;
  justify-content: center;
  text-align: center;
}

/* Large query */
@media (min-width: 1025px) {
  .box--large { display: inline-flex; }
}

/* Medium query */
@media (max-width: 1024px) {
  .box--medium { display: inline-flex; }
}

/* Small query */
@media (max-width: 475px) {
  .box--small { display: inline-flex; }
}
```

HTML

```html
<div class="box box--small">Small (475 or less)</div>
<div class="box box--medium">Medium (1024 or less)</div>
<div class="box box--large">Large (1025 or more)</div>
```

Live Code Example

In this example you'll only ever see 2 of the 3 boxes, and if you're on a larger browser, you'll only ever see the large box.

We also used two query rules: min-width and max-width. The min-width parameter in the @media rule is essentially saying, "If the browsers viewport width is equal to or larger than *x-value*, apply the following CSS…". Likewise with the max-width parameter, except it checks for the browsers viewport width and applies it's CSS *if* the browser is smaller than the specified value.

Let's look at one more example that you'd see in real life. Be ready, here comes *a lot* of code!

CSS

```css
body {
  font-family: 'Helvetica', 'Arial', sans-serif;
  padding: 0;
  margin: 0;
}

.row { display: flex; }

.header {
  flex: 1 0 100%;
  border-bottom: 1px solid #ccc;
  font-size: 35px;
  height: 60px;
}

.col {
  padding: 10px;
  box-sizing: border-box;
  flex: 1 1 100%;
}

.col--left {
  background-color: #ccc;
  border-right: 1px solid #000;
  flex: 0 1 150px;
  padding: 0;
}

.col--right {
  flex: 1 1 50%;
  height: calc(100vh - 100px);
  min-height: 300px;
}

ul.nav {
  list-style: none;
  padding: 0;
}

li.nav-option {
  display: block;
```

```
  width: 100%;
  border-bottom: 1px solid #fff;
  padding: 5px;
}

.footer {
  text-align: center;
  border-top: 1px solid #ccc;
  color: rgba(0, 0, 0, 0.5);
  font-size: 80%;
  height: 40px;
}

/* Shrink your browser down until it changes */
@media (max-width: 550px) {
  .row { flex-wrap: wrap; }

  .col--left,
  .col--right {
    flex: 1 0 100%;
    min-height: auto;
    height: auto;
  }

  .col--right {
    padding: 30px;
  }

  ul.nav {
    display: flex;
  }

  li.nav-option {
    border-bottom: none;
    flex: 1 1 33%;
    text-align: center;
  }

  .footer::after {
    content: "SURPRISE, EXTRA TEXT!"
  }
}
```

HTML

```
<div class="row">
  <div class="col header">
    Logo
  </div>
</div>
<div class="row">
  <div class="col col--left">
    <ul class="nav">
      <li class="nav-option">Option 1</li>
      <li class="nav-option">Option 2</li>
      <li class="nav-option">Option 3</li>
    </ul>
  </div>
  <div class="col col--right">
    Page content in here
  </div>
</div>

<div class="row">
  <div class="col footer">
    &copy; 2017
  </div>
</div>
```

Live Code Example

The left image is the desktop version; right image is the mobile version.
Uses the same HTML.

For this, don't worry about reading *all* the code; the most important parts for this lesson are the media queries. Definitely open the CodePen example code and play with it, break it, and customize it! Try adding new media queries for different size screens, too.

Responsive Images and Videos

Images on desktops tend to look fine until you view the same web page on your tablet or phone, in which you'll often see your image *not* scale down -- it stays the same size!

Embedded videos, on the other hand, never scale nicely due to it's <iframe> element. To make content of an <iframe> use the entire allocated space, you need to make sure the page that's being iframed is absolutely positioned to each corner, and that will let the <iframe> be somewhat responsive, but there's more to it than that.

Responsive Images

You'll be pleasantly surprised at how easy it is to make an image responsive.

CSS

```css
.responsive {
  width: 50%;
  height auto;
}
```

HTML

```html
<img src="../file.jpg" alt="Normal Image" />
<hr />
<img src="../file.jpg" alt="Responsive Image" class="responsive" />
```

Live Code Example

The image on top will always stay the same size regardless of viewport width.

The image on the bottom will always be 50% of its parent element (in this case that's the <body>). When you scale down your browser the image will get smaller, and the aspect ratio will stay the same.

Responsive Videos

In the world of web development, there are two kinds of videos: <video>'s and <iframe>'s (embeds). If you want to scale a <video>, use the same method we used for responsive images. If you want to make an embedded video, like a YouTube video, responsive… well that's another story. We need to add a parent element as a container for the <iframe>.

CSS

```css
.video-container {
  position: relative;
  padding-bottom: 56.25%;
  overflow: hidden;
}

.video-container iframe,
.video-container object,
.video-container embed {
  position: absolute;
  top: 0;
  left: 0;
  width: 100%;
  height: 100%;
}
```

HTML

```html
<div class="video-container">
  <iframe width="560" height="315"
src="https://www.youtube.com/embed/sh6V1zMmbuU" frameborder="0"
allowfullscreen></iframe>
</div>
```

Live Code Example

If you look at .video-container, you'll notice we added a 56.25% bottom padding. That's to maintain the 16:9 video aspect ratio so the child<iframe> can maintain its original size. If you open up a calculator and do: 9 / 16, you'll get 0.5625 (56.25%).

Then we positioned all iframes, objects and embeds (the YouTube <iframe>) to fill the entire .video-container.

If the video is too big, you can change the size of .video-container and your <iframe> will adjustment accordingly.

Mobile First

Mobile is growing in importance every day. When you're developing websites you need to realize that mobile is, and will continue to, outgrowing desktop and laptop browsers. While almost all the CSS we've explored in this course has been for desktop first, I want you to do one more *really* hard thing: **build for mobile first**.

Building mobile-first websites is important for two reasons: mobile is important, but more importantly for you as a developer, it's harder to code a desktop site and make it fully responsive than it is to consider mobile-first code from the very beginning.

In the next lesson we're going to start exploring frontend web development frameworks, and they *all* put an emphasis on mobile first.

Just because a framework, or your codebase, is mobile first doesn't necessarily mean it's mobile-only. It just means you've considered the website will look and act on mobile devices before completing your entire desktop version.

Frameworks

A framework is more than just a single piece of code that helps you solve a problem. A framework isn't the potatoes on your plate; it's the entire meal. Frameworks include specific HTML structures, specific CSS rules and quite a bit of JavaScript. By adding premade rules for you to build your website with, you're actually speeding up your development cycle because it's less code you have to write, and the frameworks we're about to be introduced to don't just make simple navbars for you -- they'll take care of dropdown menus, accordion menus, modals (popup windows inside of a website), tooltips, but most importantly… they take care of the responsive design for you! And more often than not, these frameworks are completely open source so you can customize it as far as you need to finish a website and it won't cost you anything -- yup, they're free!

But that's not an excuse to not learn about media queries and become great at writing them. In fact, the next step after CSS (if you decide to master CSS) is learning SASS: a method of writing CSS in several files using variables, functions and mixins and compiling all your code into one .css file for you.

This course isn't going to cover how to use each of the frameworks you'll be introduced to. Instead, we'll just leave them with you and let you explore them. And we'll try to keep out any biases while being informative.

Bootstrap

http://getbootstrap.com/

If you haven't heard the term "bootstrap" before then welcome to web development! Not only is this a common term in web development and startups in general, but it's *the* name *the* most commonly used framework across the world. This framework is so popular because it came from a couple devs out of Twitter. It used to be called Twitter Bootstrap, but it's detached from the Twitter entity and is now a massive framework that'll help you do anything you want. It comes with a fully responsive grid system (hide and show elements based on screen size), custom icons, full CSS support, component (groups of HTML+CSS to make beautifully styled sections like forms or fields with an icon in them), and much much more.

In all honesty, Bootstrap is easy for new devs to learn -- the learning curve is relatively low and the docs are pretty clear, but there are a lot of questions and answers on StackOverflow. But the real power comes from it's ability to fully customize your theme inside your browser (see http://getbootstrap.com/customize/).

But there's one *major* downside to using Bootstrap: unless you drastically change your Bootstrap theme it's easily recognized as a Bootstrap site. That's nothing to be ashamed of, but if your client is looking for a unique site, they may not want one that looks similar to a lot of other sites. If you're not against a little swearing, here's a funny (and valid) site about Bootstrap websites, http://adventurega.me/bootstrap/. Don't let this sway your decision to learn or not learn Bootstrap. As a front end developer I'd highly recommend learning it because you'll end up using it anyway.

If you want to whip up great looking websites that use Bootstrap, you can buy and download free themes at https://wrapbootstrap.com/. And when you get *great* at Bootstrap, you can also use {wrap}bootstrap to sell your themes and make some extra side money.

Foundation

http://foundation.zurb.com/

Foundation is a lot like Bootstrap, but seemingly it's supported by a full company. Foundation is probably the second more common front end development framework, and for good reason!

Once you become familiar with Bootstrap you might want to take your project to the next level with additional customizability -- this is where Foundation really sticks out! It lets you easily control your themes through a CSS compiler like SASS and lets you bring outside tasks (such as compiling SASS into CSS) into your organization and that'll let you *and* your coworkers work on the same code easier.

Aside from that, it's default styling is a bit different from Bootstrap -- it has harder corners and a flatter design. As a front end developer, you'll most likely end up working with Foundation at some point in your career. If you don't decide to learn this framework it's still a good idea to read through their features. And just because it's not *the* most common framework doesn't mean it's not powerful and not better than Bootstrap.

UIkit

https://getuikit.com/

This framework is surprisingly amazing. It doesn't get a lot of attention, probably because the learning curve is quite steep. Honestly, you probably won't pick up UIkit first, and that's OK. But keep it on your radar because it's probably *the most powerful* framework I've seen yet.

It comes with *a lot* of features, uses modern CSS under the hood (lots of Flexbox), and it doesn't force very much basic styling on you -- meaning you're free to customize the heck out of

this framework and you'll run into very little problems. In face, they use a global variables.scss file where you can control almost every aspect of your site from a single SASS file. Their docs are pretty good, they have live examples of every feature, and it's clearly being worked and updated every few weeks. It takes a lot of the work you'll end up doing (like writing cards, grids, cover images, and absolute positions), mixes that with all the JavaScript you'll end up writing, and put it into fairly straightforward HTML markup.

While UIkit is very powerful and comes with a lot of features out of the box, the learning curve is much steeper than Bootstrap, but the control is greater than Foundation.

Should you learn UIkit? Well, that's up to you. I personally recommend learning Bootstrap, being mostly familiar with Foundation and then upgrading your front end framework skills to UIkit once you're more established and familiar with JavaScript (because it's good to know how things work).

Scalable Icon Libraries

When you're moving into mobile development, you'll start to notice that a lot of text links become icons. Obviously tablets and phones have less real-estate to work with, so we need to compress text into little images; icons.

For this to work well, you'll want either: a library of .svg files or a font that acts as a library of .svg files. Here we're going to learn about 3 common options. There *are* many more options, but to get you started as a front end responsive web developer you *need* to know about these three options.

Font Awesome

http://fontawesome.io/icons/

Font Awesome is probably the most common icon library in the world. If you browse through their vast library of icons you'll find a couple that immediately look familiar. And it's super easy to install. You can sign up for free and they'll give you a unique URL for your Font Awesome library, or you can use any other CDN (Content Delivery Network) that offers Font Awesome, and plug it into your HTML <head> like so:

```
<link
href="https://maxcdn.bootstrapcdn.com/font-awesome/4.7.0/css/font-awesome.m
in.css" rel="stylesheet"
integrity="sha384-wvfXpqpZZVQGK6TAh5PVlGOfQNHSoD2xbE+QkPxCAFlNEevoEH3Sl0sib
VcOQVnN" crossorigin="anonymous">
```

From BootstrapCDN

And to add a new icon to your page, just write a simple <i> tag with the name of the icon you want as a CSS class.

```
<i class="fa fa-github"></i>
```

Yes, leave the element empty, it'll fill itself in. Then you can change the color and size like you would a normal font (font-size and color).

It's that easy! And their library of icons is *still* growing! And it's free! You *can* buy into Font Awesome, but it's not necessary and they won't even harass you about it.

Glyphicons

http://getbootstrap.com/components/#glyphicons

If you're using Bootstrap and don't want to load in another icon library, Bootstrap comes with their own fonts. Albeit it's a bit limited, but it's nice to have when you're just getting started. Once you've installed Bootstrap, all you need to write is a simple element.

```
<span class="glyphicon glyphicon-search"></span>
```

From Bootstraps Docs

It works and acts the same as Font Awesome, but they have an entire framework to support so you won't get thousands of icons. Oh, and it's also free.

The Noun Project

https://thenounproject.com/

Looking for just one SVG file? Or maybe you want to create your own library of icons. Any time you want to add SVG icons to your site you can trust that The Noun Project has an icon you can use. You'll need to write your own CSS to contain the scalable vector graphic to a certain size, but that's easy for you now that you know CSS.

Resources & Tools

We've gone over a lot of material in this course. Knowing how to code is just one part of the battle. Knowing your tools is equally important. Below are some resources to help you out along the way.

- This e-book. Reference this at any time.
- All the Live Code Examples on CodePen.
- Feature support for different browsers: https://caniuse.com/
- A beautiful CSS animation library: https://daneden.github.io/animate.css/
- Gradient generator: http://www.colorzilla.com/gradient-editor/
- An SVG buffet: https://thenounproject.com/
- Flexbox Froggy Game: https://flexboxfroggy.com/
- Grid Graden: http://cssgridgarden.com/
- Mozilla Developer Network, CSS: https://developer.mozilla.org/en-US/docs/Web/CSS

If you know of other resources or tools that *absolutely need* to be listed here contact me and we'll explore adding your suggestions to the next version of this book.

Final Project

Your final project is simple but not easy. By this point you're equipped with all the tools you need to make a fully responsive and beautiful website. Although most of the Live Code Examples were pretty ugly to look at, they made it easy to see what you're looking at. Fee free to use the code example and any other resource you can.

Your mission: to make a nice looking, fully responsive home page for your own website. The template is up to you -- it can be simple or complex, just remember that every section will also need a media query for mobile devices (it should support large screens, laptops, tablets and phones).

The hard part: You *cannot* use a framework to help you. You *must* write the media queries by hand and make your site responsive using only CSS you've written. *If you can do this, you can get a job as a front end web developer.*

Again, feel free to use references and example code to help you build your welcome page. You're not being timed, but you'll be measured by the quality of your work. So take your time and *do it right the first time*.

If you're in the CSS Masterclass course on Kalob.io feel free to ask questions. If you have any code problems, put them into a CodePen and share the link -- it's nicer than looking at regular text on a normal website.

Next Steps

Do not, for any reason, *ever* ask a newbie group of developers (like those groups on Facebook) about what you should do next. They aren't very experienced, they have very little industry experience and they're sharing advice they read from an unknown and untrusted source.

Yes, that had to be in bold because it's so important!

At this point in your new career as a web developer you have several paths you can go down. There are no right or wrong answers; it all depends on what *you want to do*.

SASS

If you enjoy making websites beautiful, you might want to level up your CSS by learning SASS.

> *Sass is a scripting language that is interpreted into Cascading Style Sheets (CSS). SassScript is the scripting language itself. Sass consists of two syntaxes. The original syntax, called "the indented syntax", uses a syntax similar to Haml.*
> *- Wikipedia*

Or in a helpful context: SASS is a way of writing CSS in multiple files using variables, basic programming functions and a thing called "mixins", which you'll learn all about in your journey to learn SCSS.

Most *good* front end development companies will use SASS or LESS (SASS being the preferred CSS preprocessor). It makes remember things like hex colors easier, lets you adjust one setting to make changes across your entire site, and overall easy customization.

In my current role as a front end developer, I only write SASS -- it's very rare to only write pure CSS.

JavaScript

This is what every junior dev will tell you to learn. Is it important? Yes. Is it mandatory for making beautiful websites? Kind of. Many JS libraries are plug and play; add the .js file and change your HTML markup because the JS library will scan your page for features it should apply. Should you learn JavaScript? Absolutely, but *right now* might not be the best time, depending on what you want to do in your career as a web developer.

If you're interested in becoming a backend developer, you'll want to learn Linux, Python, Apache/Nginx, HTTP responses, APIs and so on. That doesn't require a lot of JavaScript, honestly, but knowing JavaScript (even just basic principles) is helpful.

Here's the truth: you *will* end up learning JavaScript. But if you'd rather focus on the beauty of a website instead of the user experience (UI vs. UX), you can safely put off learning JavaScript for a little while longer.

If you want to create a social network, a website that users can interact with or anything above and beyond a beautiful and responsive website, you should start learning some JavaScript now. Start with the basics, that'll get you *really* far.

If you go down the JavaScript path next, you'll also want to learn jQuery. It's assumed that if you know JavaScript you know jQuery. Even if you become an Angular or React.js developer -- everyone will assume you know jQuery, just like they assume you know HTML.

Git

Git is a command line tool (although it has graphical user interfaces, GUI's) that lets you put your code on services like BitBucket, GitHub and GitLab. Git is the official tool for all web developers who need to collaborate on a project.

In the real world, developers *don't* share their working files. Instead, we'll work on whatever we need to work on, and push those changes into a Git Repository (a special kind of storage facility for multiple developers to work on code at the same time).

I personally recommend learning Git next. You don't need to be an expert, but knowing about repo's (short for repositories), how to pull code from a repo, how to publish code and how to collaborate with other developers is absolutely *vital* to your success as a developer.

Up until now you've never needed this. But eventually you *will* work with either another developer *or* need to pull code from a repo for your project.

I recommend learning Git next because it's a necessary tool -- you cannot escape it, regardless of your expertise as a developer. Both frontend and backend developers use Git to collaborate. And even if it's just you working on a project, Git is the perfect way to safely store your code *and* save old versions of your code.

Backend

If you'd rather build web apps that store information (anything that accepts payments or has user accounts) you might want to consider learning a backend language. I have two recommendations that will get a lot of backlash from certain dev communities.

Python

Anyone who tells you to *not* learn Python is steering you in the wrong direction. Python has built services like YouTube and several other Google services. It's being used in Machine Learning and AI (the future of computers) and it's an elegant language to read. Oh, and employers will pay you more for Python than a language like PHP -- that's because *good* Python devs are hard to find. Python is also a great language for learning *how* computers work, and the skills are fairly transferable.

Python is in demand with bigger companies and government departments that want to work with lots of data. If you don't live in a city with either of those opportunities, Python is still a good language to learn, but it probably won't earn you very much. Node.js is the same way -- unless a company like Netflix is hiring devs in your city, it's more of a "cool, look what I can build!" language than it is a language that's in demand.

PHP

Here comes the backlash from newbie devs and any developer that has never written PHP. PHP is what fuels WordPress, which currently runs 27% of *all websites on the internet*. Think about that for a minute. How many websites exist today? Hundreds of millions, maybe even billions! 27% of that is using PHP -- love it or hate it, that's just a fact.

If you know PHP, which is fairly easy to learn (but very different from Python) you can find work as a freelancer *anywhere*. Small businesses *need* websites but can't afford the $15,000 invoice -- offer to build them a beautiful WordPress website for upwards to $1000, that's an acceptable rate for an ultra basic website. And you'll never run out of clients.

Is PHP an elegant language? Not really. Is it fast? Well PHP7 is. Can you transfer these skills? Somewhat, yeah.

Lots of people will say "PHP is a terrible language, I'd rather die than write it" (a real quote I've heard more than once). They simple don't understand economics; learning a skill that nobody wants isn't valuable in todays market.

Also, services like Facebook and Yahoo! Have scaled *massive* companies using PHP.