

# DevCamp

## Coding Foundations course

### Module 1.

#### HTML & CSS foundations

2024, December.

# Module 1. HTML/CSS foundations

## Index

- 1.1 **Objectives**
- 1.2 **Bottega Support Application Overview (Not included)**
- 1.3 **HTML CSS Course Intro**
- 1.4 **HTML / CSS: Definitions**
- 1.5 **Website We're Going to Build**
- 1.6 **HTML Structure Overview**
- 1.7 **devCamp VSCode Extension (Not included)**
- 1.8 **Using CSS Styles**
- 1.9 **CSS Cascading process**
- 1.10 **HTML links**
- 1.11 **DIV tag**
- 1.12 **HTML .class and #id**
- 1.13 **Flexbox**
- 1.14 **CSS Padding**
- 1.15 **CSS Margin**
- 1.16 **Custom Icons, logos,...**
- 1.17 **Child tag elements**
- 1.18 **CSS Grid**
- 1.19 **Importing Custom Fonts**
- 1.20 **HTML images**
- 1.21 **CSS code Refactoring**
- 1.22 **Flexbox inside Grid: Align**
- 1.23 **CSS animations**
- 1.24 **NavBar: Styling as a column**
- 1.25 **NavBar: Styles meeting Best Practices**
- 1.26 **Parallax Scrolling**

- 1.27 Parallax: Text overlays.**
- 1.28 HTML char./ASCII “&” icons**
- 1.29 :hover Animations**
- 1.30 ::nthChild selectors**
- 1.31 CSS Box Shadows**
- 1.32 Embedding content: Gmaps**
- 1.33 HTML Footer**
- 1.34 Flex Direction: Grid vs Column**
- 1.35 CSS image filters**
- 1.36 Footer styles**
- 1.37 Creating an About.html**
- 1.38 Skew Images in CSS**
- 1.39 CSS Float property**
- 1.40 HTML Square Grid**
- 1.41 CSS Grid>Flexbox: 2 columns**
- 1.42 Review of Code Organization Best Practices**
- 1.43 “Menu” page with 2 grids**
- 1.44 HTML Bullet Point and Numbered Lists**
- 1.45 HTML Anchor Tags**
- 1.46 ::Before ::After pseudoselectors**
- 1.47 contact.html creation**
- 1.48 Page Layout with CSS Grid**
- 1.49 HTML Form Elements**
- 1.50 Text Input styles with CSS**
- 1.51 Form textarea / Button styles**
- 1.52 Label and Form Element Drop Shadow Styles**
- 1.53 Custom Form Placeholder Text Styles**

**1.54 Form Labels Animations**

**1.55 Finalizing Contact Page Styles**

**1.56 CSS Media Queries**

**1.57 Responsive Styles**

**1.58 Implementing Responsive Styles to the Square Grid, Image Skew, and Form Elements**

# 1.1 Objectives

During this section of the coursework, the following objectives will be met. You will learn to:

Take advantage of these other development skills resources:

- Identify tools used to program websites.
- Describe basic HTML Website Structure
- Use HTML tags such as head tag, div tag, span tag, etc.
- Utilize HTML IDs and Classes
- Integrate HTML tables, forms, text input, checkboxes, radio buttons, etc.
- Implement Inline and Embedded CSS styles
- Use CSS best practices with style sheets
- Utilize CSS selectors.
- Incorporate CSS animations, borders, and styling of tables.

Please make sure you are **Coding Along** with the videos. A few guides from now you will start building out a website in HTML. You need to be coding along with the Instructor, creating the website on your own computer as the instructor creates it on theirs. To do this, you will need a text editor. You can use any text editor that you like, but for beginners we recommend Visual Studio Code. Other text editors you might want to use are Atom and Sublime.

If you are having any trouble as you code along, please reach out on the Support App. We have a team of **Mentors** who will be able to answer any questions and help you debug your code.

When you are done with this section, you will need to reach out with a ticket and have the Devcamp Fries Website passed off.

[Visual Studio](https://code.visualstudio.com/) <https://code.visualstudio.com/>

[Sublime](https://www.sublimetext.com/3) <https://www.sublimetext.com/3>

[Atom](https://atom.io/) <https://atom.io/>

# 1.3 HTML CSS Course Intro



Welcome to this HTML-CSS Bootcamp course, where you are going to learn how to build a real-world website, and along the way, you're going to be able to learn all about HTML and CSS.

My name is Jordan Hudgens, and I'll be your instructor throughout the course material. I'm currently the lead instructor and CTO of the Bottega Code School.

In addition to the work I do with Bottega, throughout the past 15 years, I've been building out web applications for companies like **Eventbrite** and **Quip**.

In this course, you're going to learn all of the key skills that you're going to need in order to start building out professional websites.

Some of the skills that you're going to learn are going to be **HTML5**, **CSS3**, **Flexbox**, **CSS Grid**, how to build animations, how to work with navigation, **fonts**, and also how to integrate images directly into your websites.

As we go through the course material, you'll notice that I'm going to teach you my own process that I use whenever I'm building out a website.

My goal for this entire course isn't just to teach you how to build a single website, but instead my goal is to be able to leverage this project build that we're going to put together, teach you the fundamentals as we implement every single feature, and then by the end of it, you're not just going to be able to build the single website and follow along with what I do, but you'll be able to build any type of website.

There are no technical prerequisites for going through this course. This is the perfect course to go through if you want to learn how to build websites completely from scratch.

The ideal student is someone who is dedicated, wants to learn, and also wants to learn how to build websites using the most modern and up to date technologies. So, thank you for spending the time and going through this material, and good luck with the coding.

## 1.4 HTML / CSS: Definitions

This guide will discuss the concept of cramming vs consistent study. And don't change the channel if you're not in school, if you're a developer or if you want to learn software development... the learning never ends.

What are HTML and CSS? Well, technically they're just tools that you can use, and they're syntaxes you can use that allow you to write code that can be interpreted by browsers. By default, a browser can only interpret and then render on the screen so many different types of code.

For example, if you were to take an Excel spreadsheet or a Word doc and you try to open it up in Google Chrome or in Firefox, it is not going to work. That's because the browser is not capable of taking a file like that and interpreting it into something that it can show on the screen.

HTML and CSS are capable of that. Their goal is to provide a syntax so that when the browser sees it, it can interpret it and then it can show exactly what you're wanting on the screen.

Literally, every web page in the entire world has HTML, and pretty much all of them also have CSS. We're going to talk about the different types of roles that each one of them supports.

HTML stands for hypertext markup language. Now, when you hear the word language, you may think programming language, but that's not quite accurate because it really is just providing you a syntax for writing code so that the browser's able to read it properly.

It's really more of just a markup language, whereas a programming language such as JavaScript or Java or Ruby allows you to have some extra behavior like conditionals and loops and those kinds of concepts.

With HTML, you're simply writing a static page, meaning there's not really a lot of behavior in it. You need to use other languages in order to make that happen.

HTML allows you to wrap a structure around content. If you imagine having some type of blog page, HTML allows for you to designate where the content is, where the title is, or where a video or an ad might be. Then you're able to then organize it and later on, style it.

That's where CSS comes in.

CSS stands for cascading style sheets. The style word in there may indicate its main primary objective. CSS is in charge of giving style to websites. You're able to take all of that HTML code that you organize and you put on the page.

CSS then goes in, and it adds all of the style elements. This can be anything from colors, to fonts, to animations. Anything that you look at on the page that has a type of style associated with it most likely is coming from CSS.

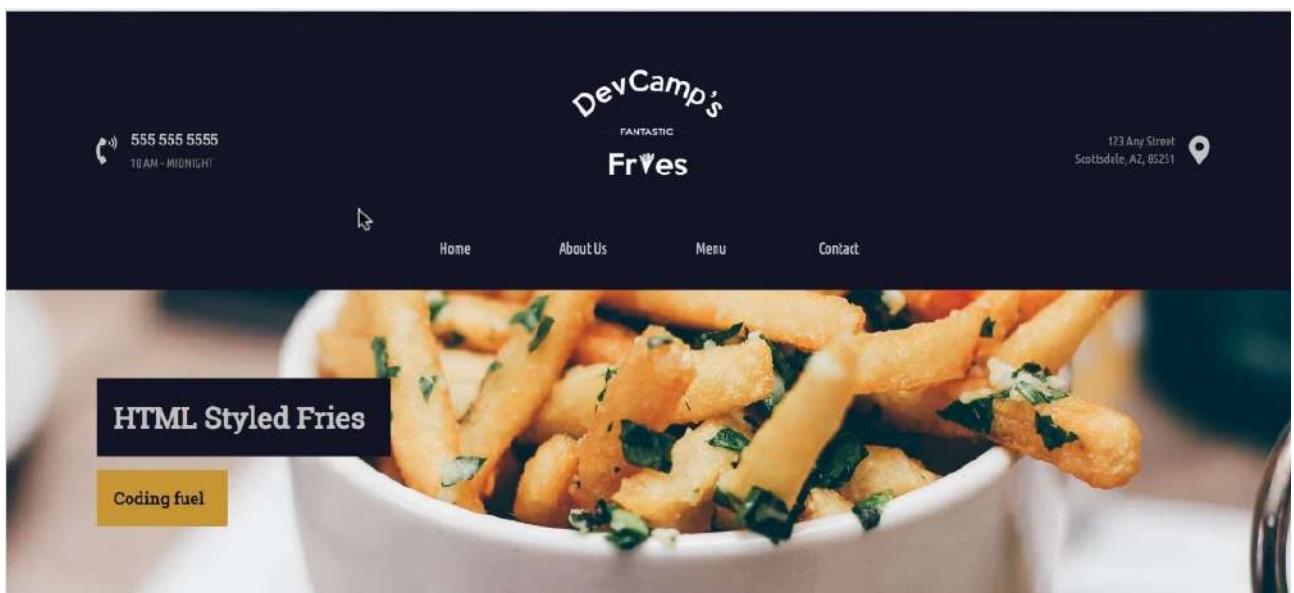
Let's walk through a real-world analogy that will hopefully help you understand the differences and the specific roles that HTML has compared with CSS.

Imagine that you're building a house. If you're building a house, the HTML is kind of like the foundation and the wood framing and the roof and the sheetrock.

The CSS, on the other hand, is more like the paint and the carpet and any kind of design accent that isn't really associated with the structure, but it is what allows a house to look good.

Now let's take a look at a website and see what happens when you have a website that has HTML and CSS compared with a website that only has HTML.

Right here you can see a fully built out website.



This has everything from a navigation bar, it has a parallax feature with background images, it has animations, it has a map and all of the different elements you'd expect in a website.

This website has HTML and CSS. Now, I duplicated this website, and if you want to take a look at it, what I did is I removed the CSS. I didn't make a single change to the HTML. All I did was I removed the calls that brought in the CSS styles, and this is what you would get.

## HTML Styled Fries

Coding fuel

We deliver!

Lorem ipsum dolor sit amet, consectetur adipisicing elit.

You can code from here!

Lorem ipsum dolor sit amet, consectetur adipisicing elit.

100+ types of fries!

Lorem ipsum dolor sit amet, consectetur adipisicing elit. Unde, dicta!



Now, if you scroll through it, it still has all the same content, and it even kept a few elements like the embedded map and some things like that, but because CSS is in charge of adding styles, what we're left with here is just plain HTML.

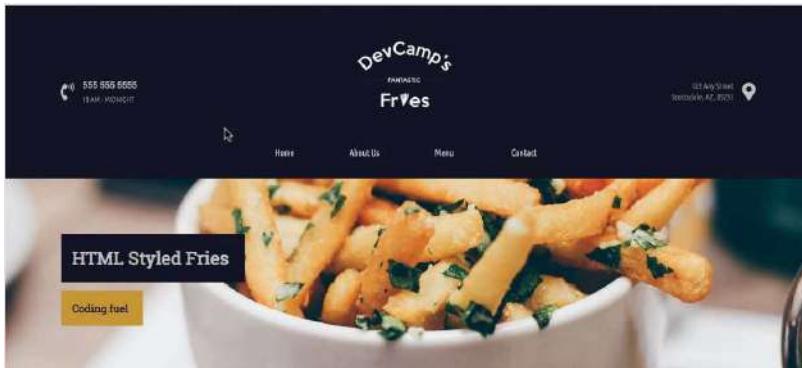
It has all of the same structure, but because it doesn't have the CSS, it doesn't have any kind of alignment. It doesn't have the colors. It doesn't bring in the background images, and it doesn't align the items in the proper way.

In review, that is a high-level overview of what HTML and CSS are. HTML is a markup language that web browsers are able to look at, and they're able to interpret. They're able to take in content and then show that on the screen.

It provides the structure; whereas, CSS is also a language that you're able to use, and you're able to write in the CSS syntax and then have that applied to the HTML code that you write.

# 1.5 Website We're Going to Build

As you can see here, we are going to build a restaurant website. By building out this project, you're going to learn all of the key fundamentals around HTML and CSS, and how you can build websites exactly like this using all of the modern tools that are available to you.



I'm going to take a very high-level approach in this video, and just show you some of the features that you're going to get to build. As you can see, you're going to get to work with images, such as logos and also background images.

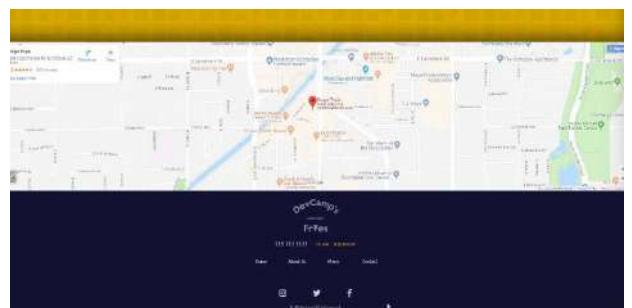
You'll be able to work with custom fonts, and learn how you can align every piece of an element on the page using tools that you're going to learn about, like **Flexbox** and **CSS Grid**. You're also going to learn how to build a responsive **navbar** that has different animations.

As you can see right here, as I'm hovering over each one of these elements, we actually are able to change the look and feel, based on the user behavior.

Scrolling down, you'll see that not only are you going to get to learn about images, but you'll also get to learn how you can implement features such as **parallax scrolling** as you can see right here. Where the image stays in the background and the scrolling simply goes past that.

Moving down, you'll learn about **grid layouts**, and how you can set up different components on the page, like we have right here, along with how you can combine that with animations.

Moving down, you can see that you'll be able to **embed and customize a Google Map**. This is a map that is fully responsive right here. Where a user is able to click on it and use it, exactly the way that they would use inside of the Maps application.



Then we're going to build a **footer** on the bottom that can combine images, different styles, another **nav bar**, and then links to social media. Right here, you're going to be able to learn how to work with **Font Awesome**, which is a great way of integrating different icons directly into your application.

Now, you'll also learn how you're able to implement links for navigation. Right here, if I click on the About Us section, you can see I'm taken to the About Us page, and then the menu, and then the contact.



Now, if you come back to any of these pages, you'll see that you'll learn **how to build a completely different layout** than the homepage.

You're also going to learn some more advanced techniques, such as how to build these kinds of designs, how you're able to implement a tool called **clip-path**, where you're able to implement a look and feel that is much more modern. These designs are the exact same type of designs that a professional developer needs to learn how to implement.

Moving down, you'll see how you can **integrate images** directly into content. Then you can also see how you're able to **set images right next to** other styled content, such as **headings** and **paragraphs**.

You can do that on multiple pages, and it's going to give you the ability to take any kind of program that you need to build out, any type of interface that you need to implement and be able to do that.

Moving to the last page we're going to build four pages throughout this application, you can see that we're also going to learn **how to build a modern-looking form**.

Right here, you can see we have a contact form we're going to build.

A screenshot of a contact form titled 'DevCamp's Fries'. The form has several input fields: 'Name' (with placeholder 'John Doe'), 'Email' (with placeholder 'john.doe@example.com'), and a large 'Message' area with a placeholder 'Type your message here...'. At the bottom right of the message area is a small 'Send' button.

If you click on that, you'll see that we even are going to be able to **use animations** like we're doing here, on each one of the elements, along with building out a **custom button with custom actions**, such as being able to **add drop shadows** and various animations like that.

All in all, this is going to be a very comprehensive course that is going to teach you HTML, CSS, and how you can use all of the modern tools associated with them. Right here, you can see that when I was putting this entire course together, I wanted to make sure that this covered all of the various techniques you need in order to build any type of website.

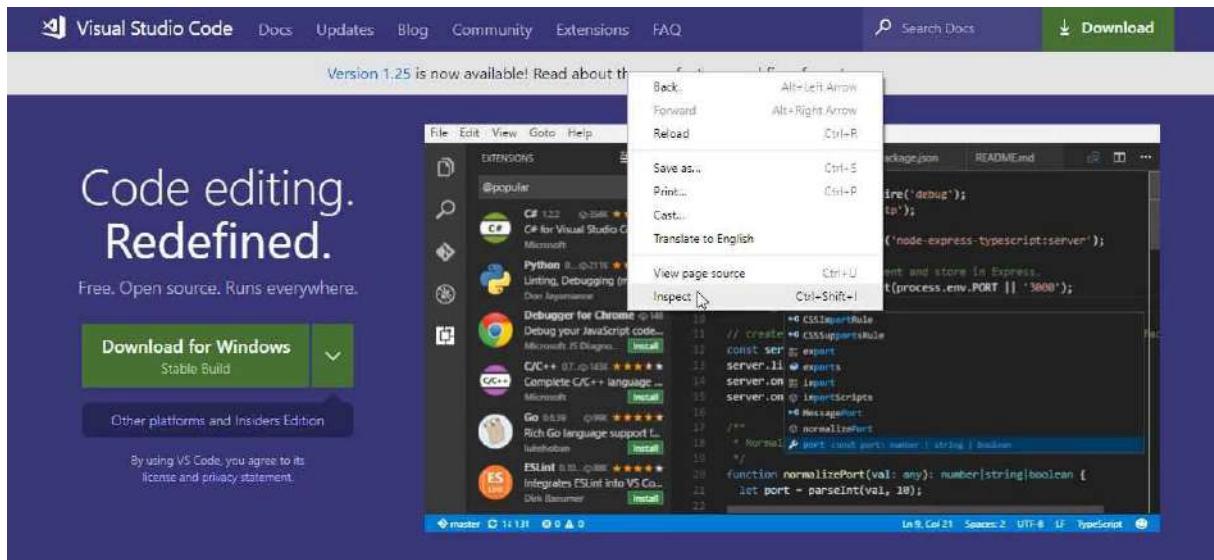
Each one of these features here is what we're going to cover as you go through the material. Now that you know what we're going to build, we're going to get into, in the next guide, some of the tools you're going to need in order to do that.

# 1.6 HTML Structure Overview

Now, there are two items that you're going to need no matter what kind of operating system that you're on. The first item is a browser, but since I'm assuming you're watching this video in a browser, I'm assuming you already have that.

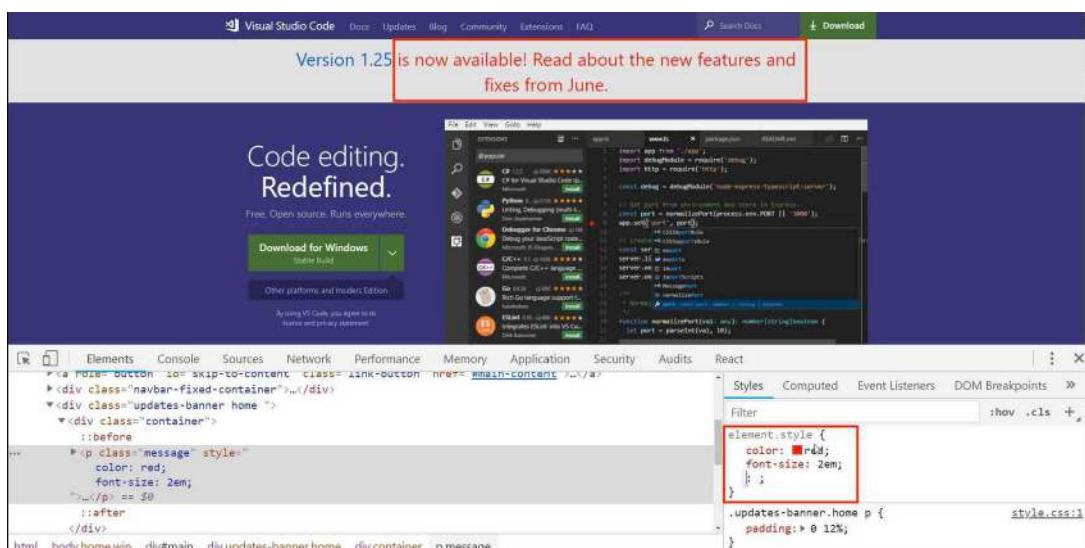
Now, the type of browser is also important. I personally prefer and recommend using Google Chrome, and the main reason for that is because Google Chrome offers some great developer tools.

Now, if you've never seen them before, the way that you can test them out is, if you come and right-click on any element on the page, then click on Inspect.



What this does is it brings up the actual HTML and CSS code right here, and you can play with it. You can see that on this Visual Studio code page, it selected this message element, and if I come here on Element Style, I can say, "Color," and then I can change it to red.

Then I could say font size and change it to 2em and you can see that I've actually altered the code right here and I'm able to see what this would look like in a different color and also a different font size.

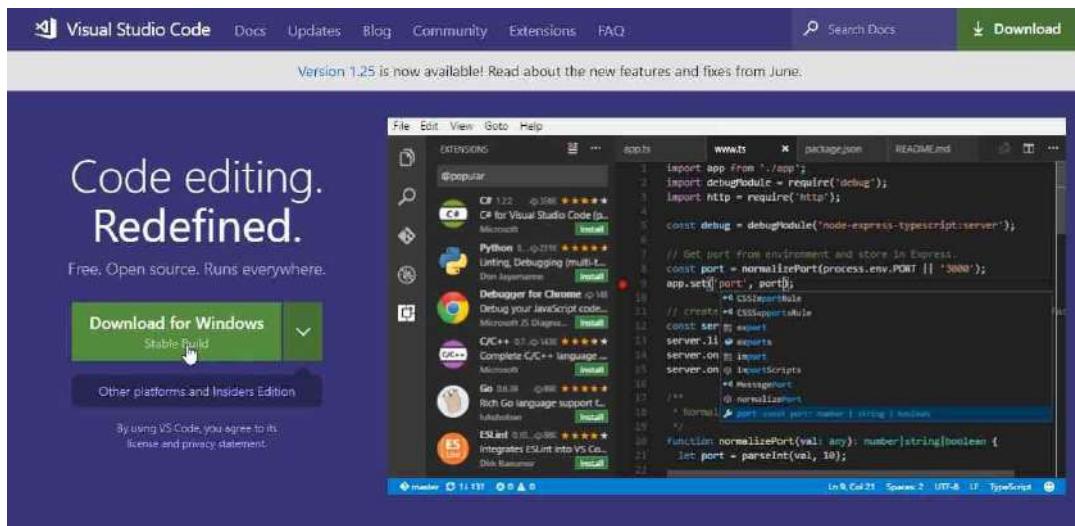


```
element.style {  
  color: red;  
  font-size: 2em;  
}  
  
.updates-banner.home p {  
  padding: 0 12%;  
}
```

Now, this does not change it on the visual code servers. If I hit refresh, you'll see that it goes back to normal. So what I just did was temporary, but this is very helpful when you're building out websites because it gives you the ability to make small changes and see what happens on the page. So, that's the reason why the browser that you choose is important.

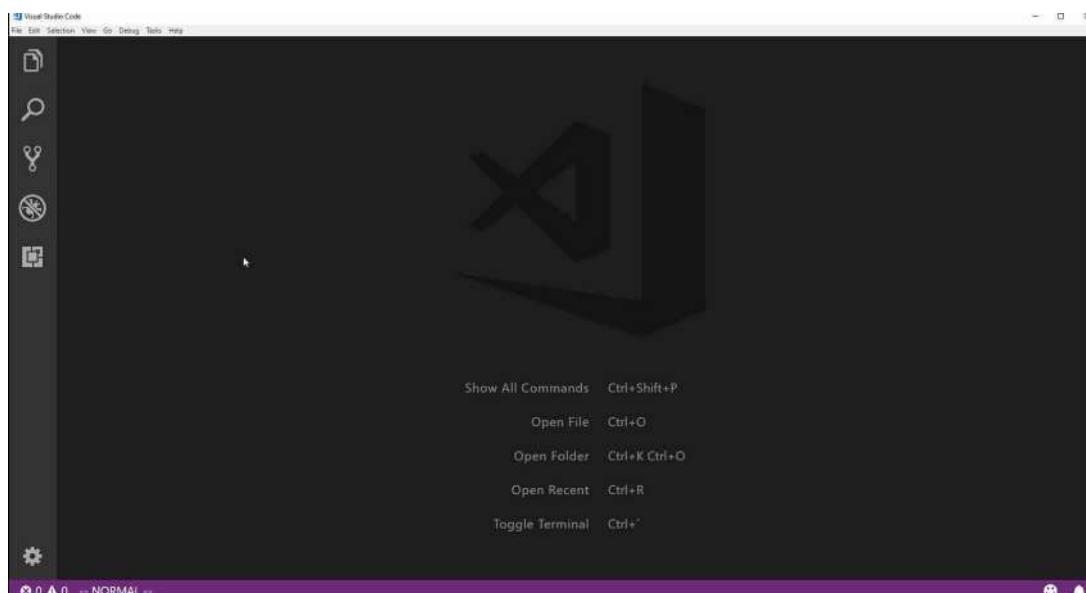
Now the next item that you need is a text editor. There are all kinds of editors on the market. Some are free, some are paid, some are what are called IDE's, which stands for a integrated development environment. What we are going to use is just a very basic one, but it has a lot of features, called [Visual Studio Code](#). It's completely free and it also works great on all the platforms.

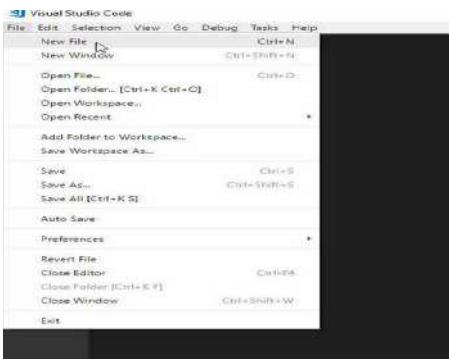
So if you're on Mac, Windows or Linux, you can use Visual Studio code. You can get it for free by going to [code.visualstudio.com](http://code.visualstudio.com) and then just click on the green download button.



Now, I already have it installed, but all you need to do is click on download. It's going to download on your system and then you can double click that installer, run through and just add all of the defaults, there's nothing special that you need to add.

So just install it on your system and then when you open it up, you should have a program that looks like this.



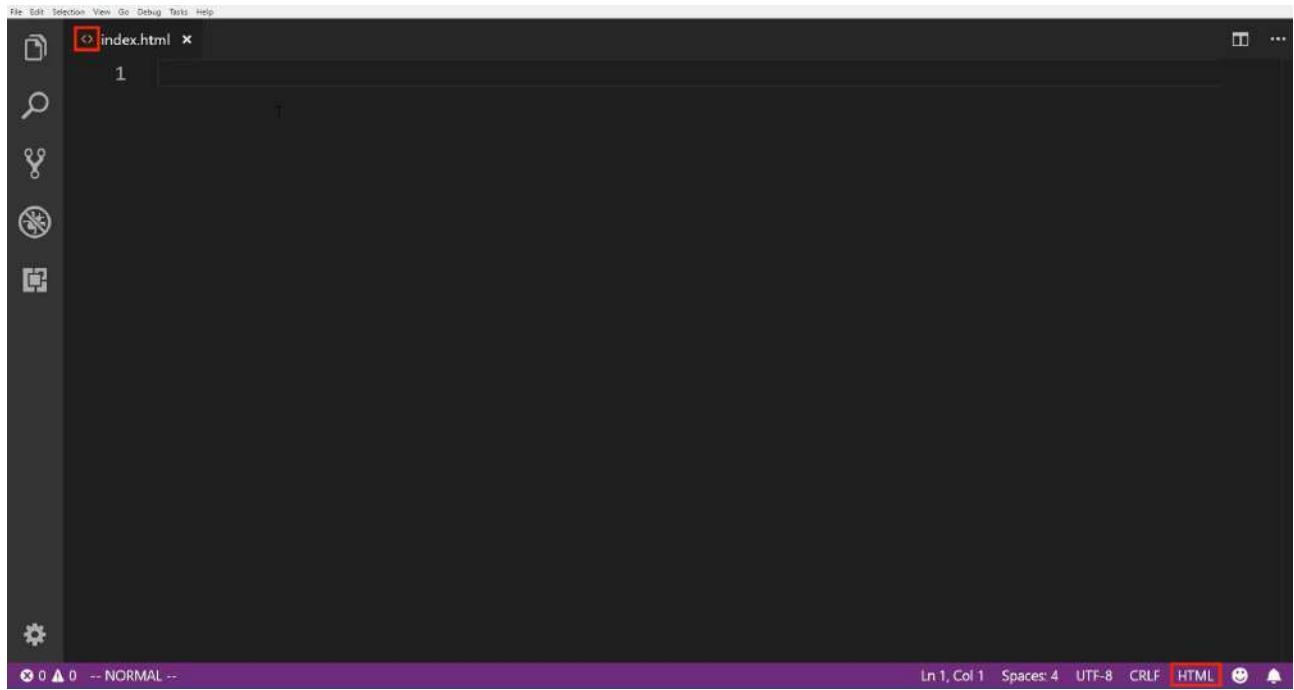


Now, if you're opening that up for the first time, you've never used it before, then there might be some other text and instructions, but you can just ignore those.

Right here, we're just going to create our very first ever HTML document and we're going to see how we're able to open it up in the browser. So the way you can do it is by going to file, new file.

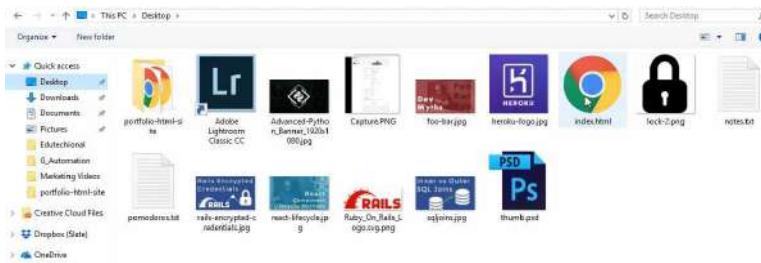
You can also hit **control + N**, or if you're on Mac, you can hit **command + N**. Then this is going to bring up a new file. Now, I'm going to hit save, so **control + S**. You could also go to file and then just go down to save or save as. And I'm going to save this on the desktop, you want to change the entire file name, you don't want this .text.

So I'm going to say **index.html**. Hit enter and now you can see that Visual Studio Code has actually changed the little icon here so that it's telling you that it understands this is HTML code. If you come down to the bottom right hand side where it says HTML, what this means is that Visual Studio Code is going to give some very helpful syntax highlighting.



And all that means is that as we're typing, it's going to pick up and give us some ideas for the type of code we want to write and it's going to show some of the different elements with different colors. It's a very helpful thing, especially as you're building out websites.

So I am going to build our very first website here. So I'm going to just say hi there. Now notice I'm not wrapping this in any other code. I'm just typing hi there and I'm going to hit save. Now, if I come back to the browser, I can hit **control + o** and that is going to open this up. If you're on Mac, it's going to be **command + o**.



Then go and find that file that we created. So, you can see right here.

I have an index.html file and if I hit open, you can see it says hi there.



So, we have created our very first web page.

Now, it's not very exciting and there's a little issue with it.

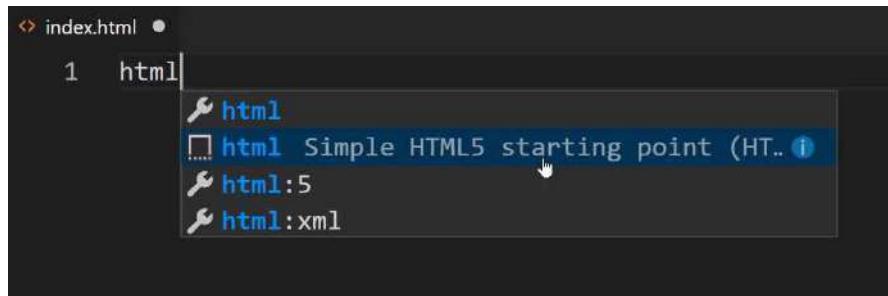
If you come to, let's just go to a demo site right here and click on view page source, you'll see that we have all kinds of code here.

It has all of these different kinds of tags. It starts at the very top where it says **DOCTYPE html**, then it has all of the code underneath that. Now, that's what is required for this to be valid html, what we wrote is something you can see here, but it's technically not HTML. The browser is just taking a glob of text and it's opening it.

```
C:\ [file:///C:/Users/devCamp/Desktop/index.html]
<!DOCTYPE html>
<html>
  <head>
    <meta charset="UTF-8">
    <title>DevCamp's Fries</title>
    <link rel="stylesheet" href="https://use.fontawesome.com/releases/v5.2.0/css/all.css" integrity="sha384-hvLj+0U1P9ftM47zNQFZj/3dpcbJ0E9tWY+qo5XQfNnOOGmJGgkH&lt;br/&gt;fcb27fb0ed0f15c1e944fb007911x#004080100" crossorigin="anonymous">
    <link rel="stylesheet" href="https://fonts.googleapis.com/css?family=Open+Sans:400,700" rel="stylesheet">
    <link href="https://fonts.googleapis.com/css?family=Open+Sans:400,700" rel="stylesheet">
    <link href="https://fonts.googleapis.com/css?family=Open+Sans:400,700" rel="stylesheet">
    <link href="https://fonts.googleapis.com/css?family=Open+Sans:400,700" rel="stylesheet">
  </head>
  <body>
    <div class="header">
      <div class="header-img" style="background-image: url('images/backgrounds/contact.jpg');"></div>
      <div class="header-wrapper">
        <div class="header-content">
          <div class="heading-wrapper">
            <h1>Contact Us</h1>
          </div>
          <div class="links-wrapper">
            <a href="index.html">Home</a>
            <a href="about.html">About Us</a>
            <a href="menu.html">Menu</a>
            <a href="contact.html">Contact</a>
          </div>
        </div>
      </div>
    </div>
    <div class="page-remainer">
      <div class="content-grid-wrapper">
        <div class="company-details">
          <div class="metadata-logs">
            <img alt="DevCamp Fries logo" data-bbox="340 810 610 830"/>
          </div>
          <div class="metadata-wrapper">
            <div>123 Any Street, Scottsdale, AZ, 85251</div>
          </div>
        </div>
        <div class="metadata-wrapper">
          <div>999 999 9999</div>
        </div>
      </div>
    </div>
  </body>
</html>
```

So let's fix this and let's build out our full, fully functional HTML page. So I'm going to get rid of this Hi there and if you're using VS code, it has a very helpful tool built into it called **Emmet**.

What that does is it gives you some auto complete features. So if you just start typing HTML five. Then what it's going to do is it's going to give us the ability to complete all of that base HTML code that you need. So right here you can see where it says HTML and so what you have here is it's called the simple HTML five starting point.



A screenshot of the VS Code interface showing the Emmet auto-complete feature. The file 'index.html' is open, and the cursor is at the beginning of the word 'html'. A dropdown menu appears with several options: 'html' (with a key icon), 'html Simple HTML5 starting point (HT.. 1)' (which is highlighted with a blue background), 'html:5' (with a key icon), and 'html:xml' (with a key icon). The background of the code editor shows the start of an HTML document.

So you should just have to type that in and this little pop up should come up and say okay. Would you like to use the simple HTML starting point? You also have a few other options here. So if you click on

HTML:5, it's going to be very similar.

Now, depending on the version of VS Code you have, these might look a little bit different and that really doesn't matter, what I'm doing is just a shortcut. After I click on that, you can see that it gives us all of this code automatically.



A screenshot of the VS Code interface showing the generated HTML code. The file 'index.html' is open, and the code is as follows:

```
1  <!DOCTYPE html>
2  <html>
3  <head>
4      <meta charset="utf-8" />
5      <meta http-equiv="X-UA-Compatible" content="IE=edge">
6      <title>Page Title</title>
7      <meta name="viewport" content="width=device-width, initial-scale=1">
8      <link rel="stylesheet" type="text/css" media="screen" href="main.css" />
9      <script src="main.js"></script>
10 </head>
11 <body>
12 |
13 </body>
14 </html>
```

Now, some of this you're not going to need whatsoever, and I'm going to walk through line by line exactly what's needed and what's not. If for some reason you don't have that emmet auto complete, then you can just pause the video right now and then go and type out all of this code. Or you can get it from the show notes.

Now, let's walk through what is happening here because this is very important in understanding HTML and seeing exactly what's on the page. So, at the very top, what we're doing, this is called the declaration. This is a HTML document and we are saying that it is going to use HTML code

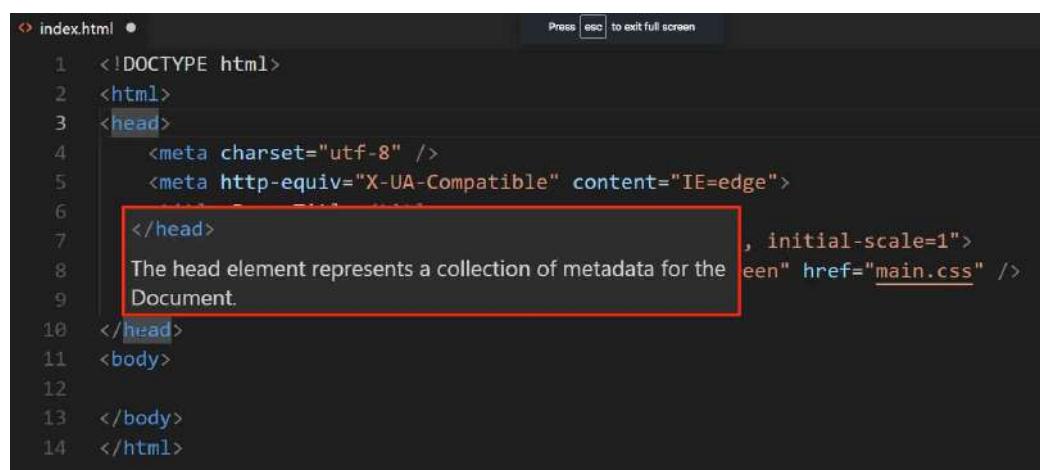
inside of it. Obviously as you could see earlier, the browser doesn't need it to render the text, but if you are going to be building a web page, this has to be your very first line.

The next one is where we're going to be including all of our HTML code inside. Now, whenever you see these little greater than and less than symbols in HTML, what this means and what it's called is a tag. So you're going to be using tags throughout any type of HTML that you build out, any web pages are all going to use tags. So you have a beginning and then you have what is called a closing tag.

So at the very beginning tag, you can see it's just using the less than symbol then the HTML word and then it closes that off with the greater than symbol. Then everything inside of this, all the way down to the bottom is going to be inside of that tag.

So everything, this head, this body, all these elements, they are what are called nested inside of the HTML. Now whenever you use a closing tag, it's going to look almost identical except you're going to have this slash right in front of it and that's how you know that it's a closing tag.

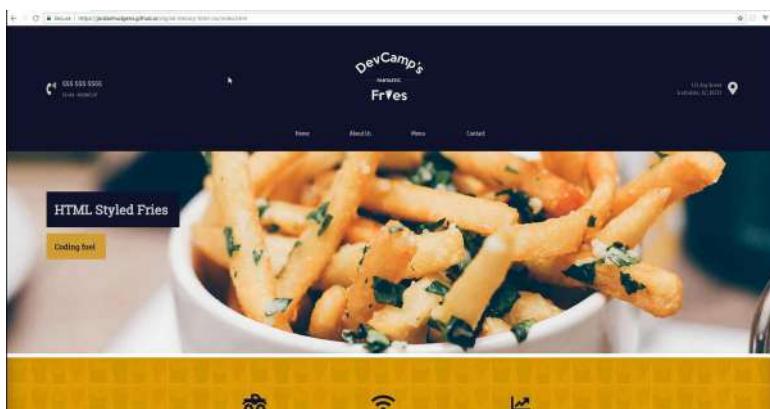
Now moving down, you can see that inside of the HTML, we have what is called a head tag. And we have the beginning of the tag and then we have the closing tag. Then if you hover over your VS Code gives you a little bit of its own tutorial.



```
index.html • Press esc to exit full screen
1  <!DOCTYPE html>
2  <html>
3  <head>
4      <meta charset="utf-8" />
5      <meta http-equiv="X-UA-Compatible" content="IE=edge" />
6
7      </head>
8      The head element represents a collection of metadata for the Document.
9
10 </head>
11 <body>
12
13 </body>
14 </html>
```

So it says the head element represents a collection of meta data for the document. Now if that

sounds weird to you and you don't understand it, don't worry. What that means is that all of the code that is placed inside of your head tag is, it's kind of invisible. It's invisible to the user.



So if you go to a regular website like we have right here, this has a head tag. However, you don't actually see it unless you go and you view the page source and you can see you have your head tag right here.

```
<!DOCTYPE html>
<html lang="en">
  <head>
    <meta charset="UTF-8">
    <title>DevCamp's Fantastic Fries</title>
    <link rel="stylesheet" href="https://use.fontawesome.com/releases/v5.2.0/css/all.css" integrity="sha384-HViflufxL6sNzntih27bfzkr27PmbbK/15vJ4e4+0uwXq79...+jsFkwG4b0Gb1QO" crossorigin="anonymous">
    <link href="https://fonts.googleapis.com/css?family=Roboto" rel="stylesheet">
    <link href="https://fonts.googleapis.com/css?family=Slab" rel="stylesheet">
    <link href="https://fonts.googleapis.com/css?family=UbuntuCondensed" rel="stylesheet">
    <link rel="stylesheet" href="styles/common.css">
    <link rel="stylesheet" href="styles/navigation.css">
    <link rel="stylesheet" href="styles/homepage.css">
    <link rel="stylesheet" href="styles/media-queries.css">
  </head>
  <body>
```

So this does things like gives you the ability to implement custom fonts and it calls your CSS files. So it's a way that you can set up the way the document is going to work, but the users don't actually see it. So that is what you use the head tag for.

Moving down to line four, you see where it says meta and then charset utf-8. What this is, is this is what's called a meta tag and this gives you the ability to define some more information and some rules that you want the document to follow.

So right here what we're saying is that the character set, meaning the content inside of this HTML document, is going to conform to this kind of encoding, utf-8 encoding. So what that means is that you have all kinds of different encoding out there. So if you want to use emojis or if you want to use foreign characters or anything like that, then this is where you're going to define the type of encoding that's used.

I would say that, assuming that you are building websites for a US based audience or a UK based audience, you're just using standard English, then utf-8 is perfectly fine. If you're having to use other encoding, then you're going to have to change it. But for everything that I use for the most part, this is the encoding that gets used.

Now, the next meta tag here is what is called a http equivalent tag. And what this is doing is, if you have ever seen a website that looks different in one browser than another, then that is what this meta tag is addressing, specifically internet explorer. So for years and years, when I've built out websites, starting a very long time ago, internet explorer kind of had their own rule set when it came to how it was going to interpret how a website looked.

Its gotten a lot better through the last few years, but about, I'd say, five, ten years ago or so, you'd have to almost build an entirely different website for Internet Explorer. You'd have to have all kinds of other tools that would be put in and other rule sets. So what this meta tag does here is, it just says that we want to use the compatible mode with Internet Explorer.

Now, I'm giving you that information just so you're aware of it. For the purposes of what we're doing right now, I'm just going to get rid of it. It's not necessary for what we're doing and I don't use it on most of my applications. But, I will tell you whenever I see anything like that, I'm going to give you a heads up because I'm assuming you're going to be working on other websites and you may need that. Or you may see it and wonder what it's there for. So I'll still mention it.

Now, the next line here is our title tag. Now you will notice that it starts off just like the HTML and just like the head tag. So you can see it has a beginning tag and a closing tag. So, if you want to say that this is My Cool Website, that is going to be the title for the document.

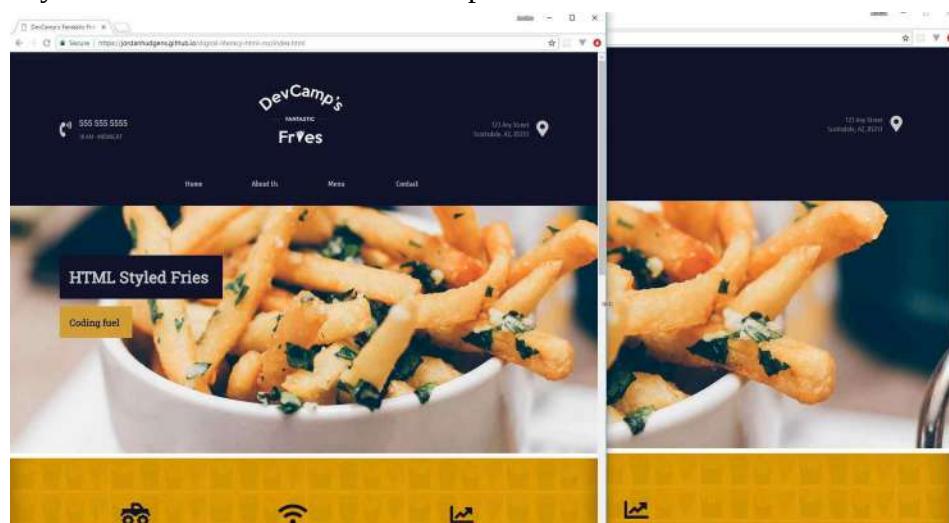
And if you go to the actual browser page, so now let's go back to our **index.html** and let's hit refresh right here. Do you see here at the very top where it says My Cool Website? That is getting pulled in from the title tag.



So if I want to say My Cool Website and then say again. And now come back, hit refresh, you can see it says My Cool Website Again. So whenever you want to change the title up here, this is what the user sees when they look up in their tabs, the title tag is where you are going to do it.

Now moving down to line six, this is the meta tag for the view port. Now what the view port is, is this gives us the ability to make our sites responsive. By responsive, what I mean is that it will look one way on a smartphone, but it will look completely different on a different device, such as being on the computer or a tablet. So if you want to see this application here, if I open this up in a new browser, you can see this is exactly how it looks when it's in desktop mode.

If I shrink it, this is the way it will look when it's on a tablet.



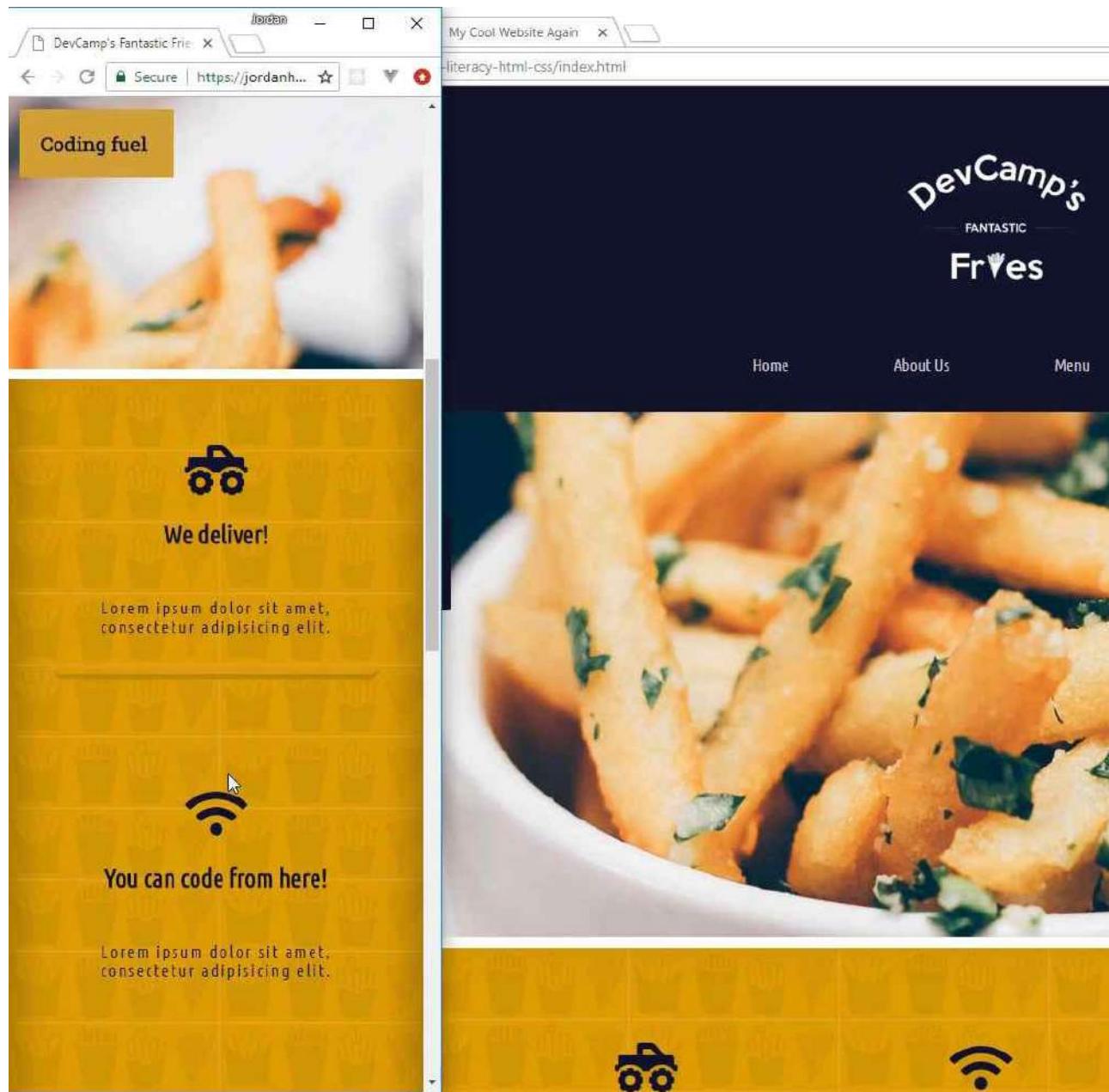
If I shrink it even more, you can see how the entire site adjusts and now the navigation bar is stacked on top of itself.

The image shows a web browser window with two tabs open. The left tab, titled "DevCamp's Fantastic Fries", contains the following content:

- DevCamp's**  
FANTASTIC
- Fries**
- [Home](#)
- [About Us](#)
- [Menu](#)
- [Contact](#)
- 555 555 5555**  
10 AM - MIDNIGHT
- 123 Any Street**  
Scottsdale, AZ, 85251

The right tab, titled "My Cool Website Again", displays a close-up image of a bowl of fries garnished with green herbs.

These items are stacked on top and now everything is sitting exactly like the way you'd want on a smart phone.



That is what the view port allows you to do, it's to implement the ability to know how wide and how tall the browser is so it can adjust if it's on a different device, so we're going to keep that one in there.

Now, the next two are ones that I'm going to remove because they're not needed, but we will later on get into what these represent. The very first one is a style sheet import link. So what this does, and you may notice this is what's called a self closing tag. So we have a link that starts out here, but instead of having something like we did with the title where we had a closing tag that was separate, the link tag actually is just one tag.

So it goes from all the way from the left all the way to the right. It doesn't have a closing one, it has a slash right before the greater than symbol. So this means, and it's what is called, a self closing tag. It's just using one.

And what this does, is it imports the style sheet. We're not going to worry about it now and we will get rid of that for the moment, but when you want to import a third party style sheet or you want to have a dedicated style sheet, this is the way that you can do it.

The same thing here, this is how you can import JavaScript code.

```
<script src="main.js"></script>
```

So now that we've done all this, we've cleaned up our head tag and nothing is on the page. So if I hit save and come back here, this is what we opened up. If I hit refresh, nothing's on the page and the reason for that is we have to add content.



---

Now, content goes inside of the body tag. So, if I open this up and I'm going to use a heading. So this is going to be big, so I am going to have a heading tag that says **Hey from HTML**. You may notice here that this heading tag has a start and a close and then I'm putting the content inside of that. If I hit refresh, now you can see it says hey from HTML.



So you're still rendering something out into the browser, but now it's actually valid HTML code because you have this doc type, you're letting the browser know that we're going to be opening up an HTML file. And then you have your head tag and then your body tag where all of the content is going to go inside.

So in review, we walked through the different tools that you need in order to get started with HTML and CSS and building out websites. And then also, the structure and the basic structure for building out a website.



## Coding Exercise

Make an "h1" tag that says "Hello World".

# 1.8 Using CSS Styles

Now that you know what an HTML document is, let's get into the role that CSS plays in building out websites. We're also going to build out our structure for the website that we're going to build throughout this course.

I'm going to open up the code file right here, and I still have it open. If you open it up in VS Code or whatever text editor you're using, we're going to talk about CSS and how you can implement it. CSS stands for **Cascading Style Sheets** and there are three main ways that you can use CSS code.

The first is going to be **inline**, the second is going to be what is called **embedded**, and then the third is **imported**. We're going to use all three just so you can become familiar with them, but then I'm going to show the best practice.

**Inline** means that we are going to add a style directly to the element. If you come and you have this **H1** right here. If you type **style**, then inside of here you can type some type of style rule. So I'm going to go with the basic one and just say **color:**, and then we can give this any color that we want.

I'm going to say **color: red;**, end it with a semi-colon. Now if I hit save, what is going to happen here, if I come back, hit refresh, you can see that, that is red.



Now, we're going to be using the developer tools quite a bit so let's start getting used to that. If I **right-click** on that element, click **inspect**, you can see that it says the element style is red.

The screenshot shows the Chrome DevTools interface. The Elements tab is active, displaying the DOM structure of a simple HTML file. An 

# element is selected, which has an inline style of `color: red;`. In the Computed tab of the DevTools, the computed style for the element is shown as `color: red;`, indicating that the inline style overrides the default `color` property from the user agent stylesheet.

So if I wanna change this to blue, now you can see the color has been changed to blue and the element itself has the inline style of **color: blue**.

The screenshot shows the browser window displaying the updated HTML content. The 

# element now contains the text "Hey from HTML" in blue. Below this, another screenshot of the Chrome DevTools interface shows the inline style for the element has been changed to `color: blue;`, demonstrating how inline styles can be modified.

So that is how you can use an inline style. Now I'm going to comment this out. The way that you can comment in VS Code, is if you're on a line of code, you can type control and then the slash character. So it'll look like **control + /**. Type those key bindings in, on whatever line you're on, and this will add a comment.

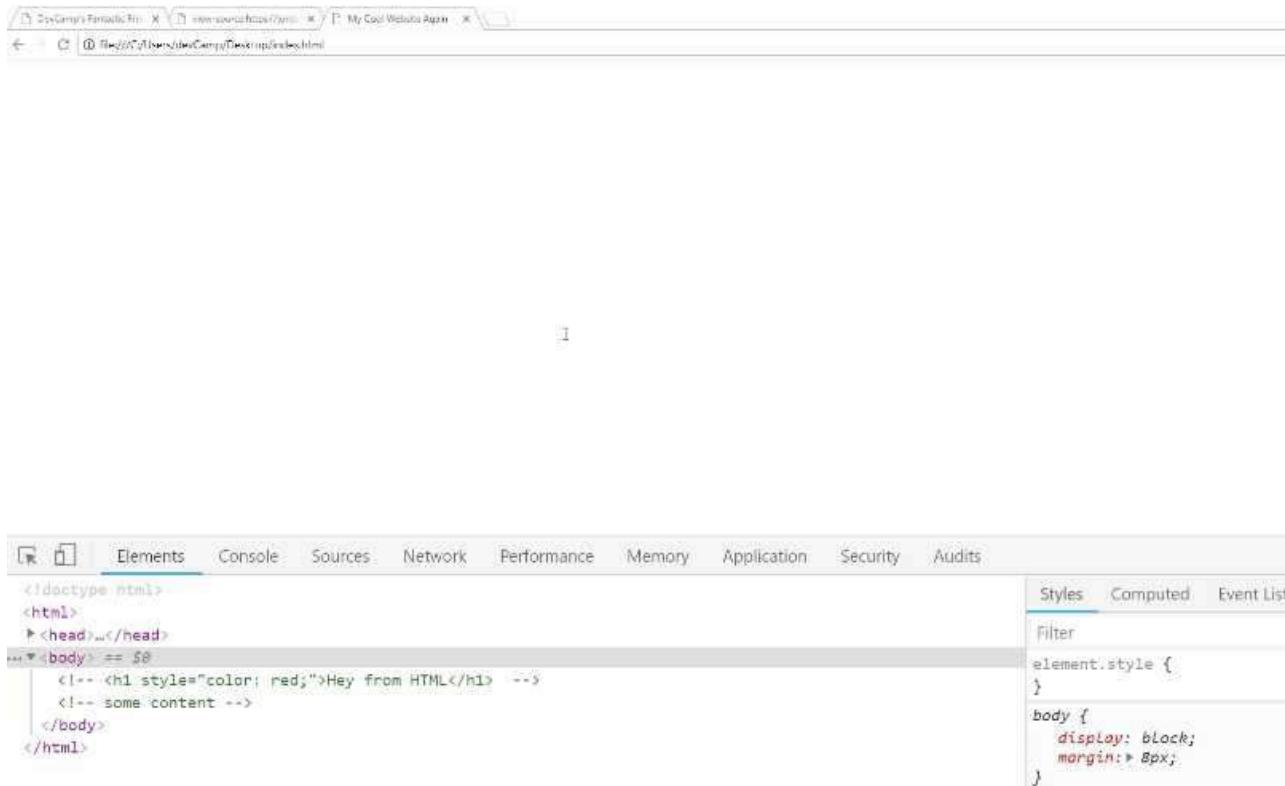
The screenshot shows a code editor window in VS Code with the file `index.html` open. On line 10, there is a comment block that wraps around the previous inline style declaration. The comment starts with `<!--` and ends with `-->`, effectively removing the inline style from the document.

```

<!DOCTYPE html>
<html>
<head>
    <meta charset="utf-8" />
    <title>My Cool Website Again</title>
    <meta name="viewport" content="width=device-width, initial-scale=1">
</head>
<body>
    <!-- <h1 style="color: red;">Hey from HTML</h1> -->
</body>
</html>

```

Now you also could just type this out manually, so you could just say `<!--some content-->`. What a comment is going to do is it's going to remove it from the page. So if you come back here, hit refresh, you can see that it's gone.



It's still in the code so you can see right, it's commented out. It is still in the code, but it's not going to be shown to the user. The reason why I'm doing that is because I want you to be able to see all three versions on how you can import and work with style sheets. I don't wanna delete them so you can kind of compare and contrast.

Now let's come down here, and I'm going to just copy that line of code. **Uncomment** this out. I'm going to remove this entire style rule. Now that is gone. We're back to just having an H1.

If I come up into the **head tag** here and now type `<style>`, what this is going to do is it's going to give me a **beginning** and an **ending** tag and we can embed style here. So any time that you want to select an item on the page, then you can just grab that element.

Right here, I want to grab the **H1 tags**, so I'm going to say

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

Then in curly braces, I'm going to add a style. Here we'll go with our same color **red**. Once again, hit save, and now if I come back, hit refresh, you can see we're back to it saying **Hey from HTML**.

The screenshot shows the browser's developer tools with the 'Elements' tab selected. The DOM tree on the left shows a simple HTML structure:

```

<!DOCTYPE html>
<html>
  <head></head>
  <body> == $0
    <!--> <h1 style="color: red;">Hey from HTML</h1> ==>
    <!-- some content -->
    <h1>Hey from HTML</h1>
  </body>
</html>

```

The right panel shows the 'Styles' tab with the following CSS rules:

```

element.style {
}
body {
  display: block;
  margin: 8px;
}

```

What we're doing here, and I'm walking through a few different concepts here all at the same time, so if this is confusing, I definitely recommend for you to go through it a couple of times just to familiarize yourself with it.

Because we are getting here, not only to embedded styles but the only way that an embedded style works is to work with **CSS selectors**. Now selectors are going to be something you're going to be doing throughout this entire course and for pretty much every website you ever build, you're going to have to use selectors.

We're going to walk through them quite a bit in the next set of videos, but what we have here is we're saying that we wanna have every **H1 heading** to have the color **red**. That's a reason why it was able to know that **Hey from HTML** should be **red** without us having to put inline.

The reason why this is really helpful is, imagine that you have some kind of scenario. Let's add another style here. You can add as many as you want, so if I want to say **text-decoration**, say **underline**. This is going to underline this text as you can see right here.



The reason why you want to go with this approach is, imagine a scenario where you have multiple headings on the page.

You could have all kinds of different paragraphs, and you want your headings to all to be the same.

The screenshot shows the browser's developer tools with the 'Elements' tab selected. The DOM tree on the left shows a simple HTML structure:

```

<!DOCTYPE html>
<html>
  <head></head>
  <body> == $0
    <!--> <h1 style="color: red;">Hey from HTML</h1> ==>
    <!-- some content -->
    <h1>Hey from HTML</h1>
  </body>
</html>

```

The right panel shows the 'Styles' tab with the following CSS rules:

```

element.style {
}
body {
  display: block;
  margin: 8px;
}

```

What you're able to do here is you're able to say, "Okay, I want to have every H1 to have these styles." If you use inline styles, you'd need to type this entire line of code every single time that you called the heading. That's not very scalable.

Imagine a scenario where you want to change the colors. So you wanna change it to blue or some other color, then what's going to happen is you're going to have to go and remember every spot that you typed that code and then change it.

As you can see here, if I just copy this and say **Hey from HTML again**, hit refresh, you can see that the same style is applied right here. This is a much more scalable way of doing it.

The screenshot shows a browser window with developer tools open. The address bar shows the URL: Hey/CrossSideCamp/Codecamp/index.html. The browser displays two red 'Hey from HTML' headings. The developer tools interface includes tabs for Elements, Console, Sources, Network, Performance, Memory, Application, Security, and Audits. The Elements tab shows the HTML structure: <!doctype html> <html> <head>...</head> ...<body>...</body></html>. The body contains an inline style for the first heading and another heading with a class. The Styles panel on the right shows the element.style rule (color: red) and the body style rule (display: block; margin: 8px;).

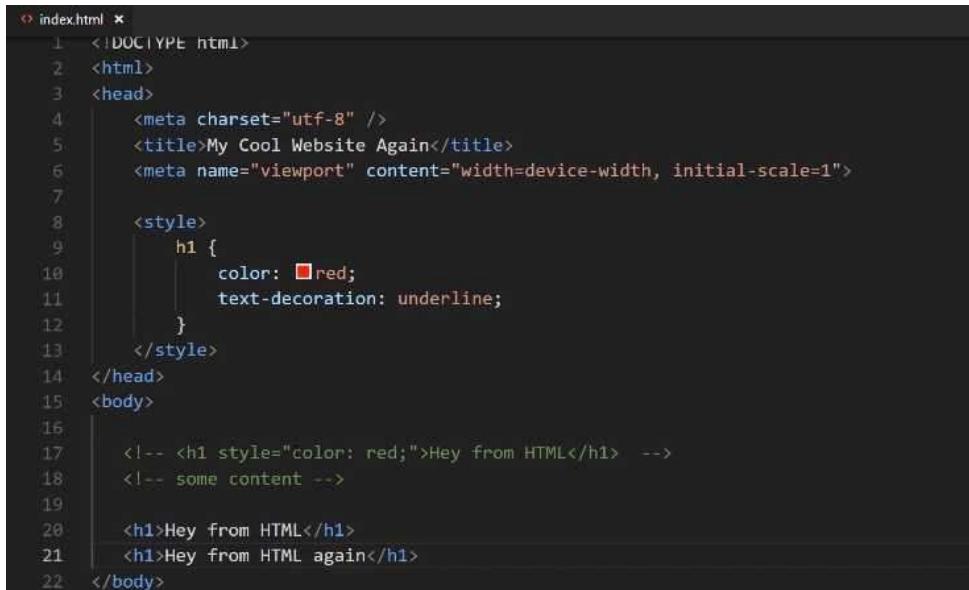
Now, so far we've gone through two ways, we've seen how we could use inline styles, that's where we add the tag directly to the element. Then we saw how we can use embedded styles, that's where we have more of this abstract concept of a style tag in the document.

Now it gets a little bit trickier when we go into the third option, which is to have external style sheets. The reason why you need external style sheets is, let's take a real-world example.

Say you're building out this website, which we're about to do. If you wanted to make some kind of style rule, such as the hover effect right here on the navigation items if you wanted to have that same effect and you wanted it to be on all of these other pages like we do right here.

If you use embedded styles then what that would mean is that you'd have to literally copy and paste all of the style information and all of that code into every single page of the entire site.

What this would mean is inside of this **style tag**, you would have literally thousands of lines of code all right here, and it would be on every single page.



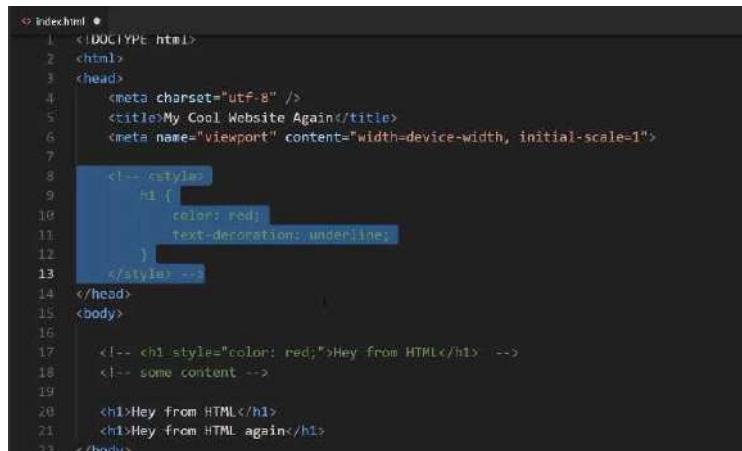
```
<!DOCTYPE html>
<html>
<head>
    <meta charset="utf-8" />
    <title>My Cool Website Again</title>
    <meta name="viewport" content="width=device-width, initial-scale=1">
<style>
    h1 {
        color: red;
        text-decoration: underline;
    }
</style>
</head>
<body>
    <!-- <h1 style="color: red;">Hey from HTML</h1> -->
    <!-- some content -->
<h1>Hey from HTML</h1>
<h1>Hey from HTML again</h1>
</body>
```

Then, kind of like the issue we had with the inline styles, what you'd have to do is if you had to make a change, so say that you wanted to change that kind of gold color to something else, you'd have to go and make the change in every spot in every file in your project.

Now, this is going to be a relatively small project, we're only going to have four pages. Imagine a scenario where you have hundreds of pages. That would start to get very unwieldy, very quickly. So instead, we have a third option, which is called **external style sheets**.

In this case, I'm going to be able to keep these here because the way that you select the items is going to be exactly the same and as we're doing with embedded. It's only how we're defining them that'll be different.

I'm going to comment all of this out. Once again, if you're new to VS Code, if you wanna comment out multiple lines of code, you can just select all of it and then hit the **control + /** again. That will comment all of that out.



```
<!DOCTYPE html>
<html>
<head>
    <meta charset="utf-8" />
    <title>My Cool Website Again</title>
    <meta name="viewport" content="width=device-width, initial-scale=1">
<!-- <style> -->
<!-- <h1> -->
    <!-- <h1 style="color: red;">Hey from HTML</h1> -->
    <!-- some content -->
<!-- <h1> -->
    <h1>Hey from HTML</h1>
    <h1>Hey from HTML again</h1>
</body>
```

Now, our third option of having our external style sheets, you're going to use a link. This is going to be **link**, a self-closing tag. If you're using VS Code, it should do all this for you automatically. If you type in **link** and then **tab**, this is going to give you a **stylesheet**.

```
index.html •
1  <!DOCTYPE html>
2  <html>
3  <head>
4      <meta charset="utf-8" />
5      <title>My Cool Website Again</title>
6      <meta name="viewport" content="width=device-width, initial-scale=1">
7
8      <link rel="stylesheet" href="">
9
```

What our **href** stands for is it's asking for a reference. What it means is it is saying, "I need you to tell me where to go find this other file." For here, just for ease of use, I'm just going to say **styles.css**.

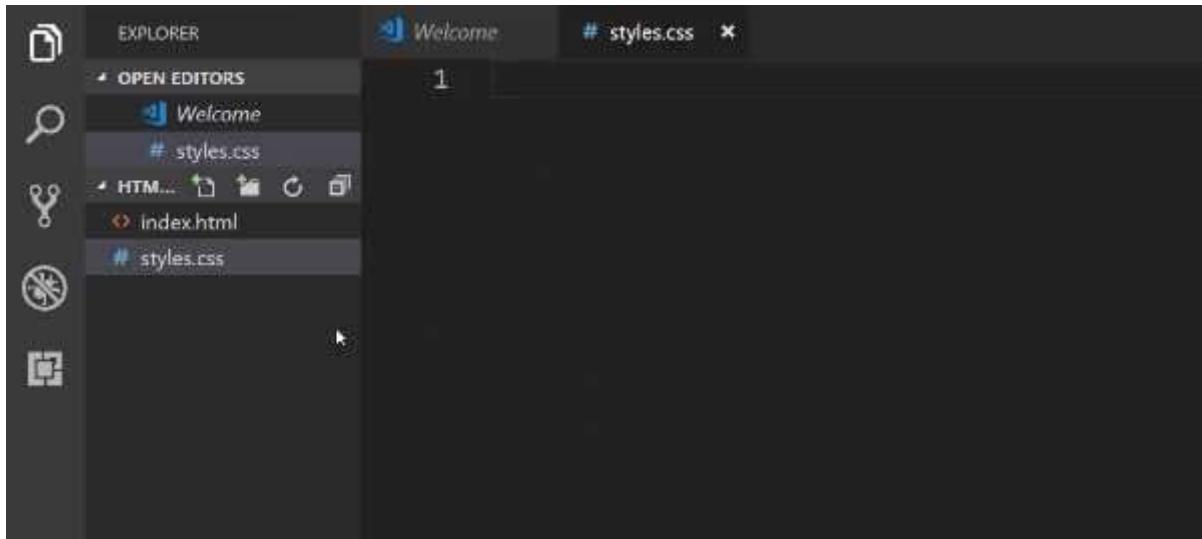
You give the full name to the file, hit save, and now we're not going to have any styles. If I come back here and hit refresh, you can see we don't have any styles. Technically there's going to be an error if you click on console.



You can see right here where it gives you this error. What it's saying is that it tried to go find this **styles** file, and it couldn't do it. We're going to fix that now. If you click on this little explorer here, then you can see that in the open editor we have the index file and we also do not have a folder open.

Let's create a project folder. I'm going to click **open folder**. We're here on the desktop, and now let's just right-click, say **new folder** and we're just going to call this **HTML - CSS**. Let's open this up, click on **select folder**, and this is now going to open up a project.

Now we wanna put our **index.HTML** file in there, so if you go down into the explorer or if you're on Mac you can go into the finder and then go to your desktop. You can just click and drag the **index** file directly into our new project folder. You can see it's right there. Now click on **new file** and say **styles.CSS**.



In here, let's use that same exact code. So **h1** and then inside of it we'll say **color: red;**. Then let's go with that **text-decoration** of **underline**

```
h1 {  
  color: red;  
  text-decoration: underline;  
}
```

Hit save, and now let's go back and see if that worked. If you come back, hit refresh. Oh and actually I made one mistake. Right here, you can see it's still looking for this file on the desktop. This is a good mistake to make because what it means is it'll just kind of reinforce understanding of how the path system works.

The way the browser is using this file and the way it's reading it is it's going into the hard drive, then it goes into the user's directory. Then into one I have called devCamp, then it goes to the Desktop then it tries to find this file. It can't do that anymore because we moved the file.

Your file was not found  
It may have been moved or deleted.  
C:\Users\deCamp\Documents\index.html

Elements    Console    Sources    Network    Performance    Memory    Application    Security    Audits

<!DOCTYPE html>  
<html dir="ltr" lang="en" i18n-processed>  
  <head>...</head>  
  <body id="t" style="font-family: 'Segoe UI', Tahoma, sans-serif; font-size: 75%" jstcache="0" class="neterror" style="font-size: 75%; background-color: #fff;">...</body>

Styles    Computed    Event  
Filter  
element.style {  
 font-family: 'Segoe U...  
}  
body {  
 background-color: #fff;

If you hit **control + o** and click on **html-css**, then click on **index** and open it up. Now you can see we have our file back again, and the styles are being applied.

In review, we now have our index file using an external style sheet. So instead of having to put this code in every single file of our entire project, what we can do is we can just call this one file, or you can even call other files, and then import those directly into any other page of your project.



## Coding Exercise

Using what you just learned about CSS styling, write CSS code that would make the text of the provided h1 tag the color green. *Don't use any HTML in your submission.*

```
<h1>Make me green!</h1>
```

# 1.9 CSS Cascading process

If you're new to building websites, the name cascading style sheet may a little bit confusing, so in this guide, I want to go through that just to give you an idea of how important it is.

I think that it's worth it to go through an entire guide just to understand **CSS** because I've seen this particular feature of CSS get very confusing, especially as developers get into more advanced programs.

Hey from HTML

Hey from HTML again



Now if you come back into the text editor, notice how the text was red.

If you want to override that, watch what happens here when I add different styles in below.

styles.css

```
h1 {  
    color: red;  
    text-decoration:  
underline;  
}  
  
h1 {  
    color: goldenrod;  
}
```

Hey from HTML  
Hey from HTML again

If you come back to the website and hit refresh, you can see that the color is now goldenrod.

What this means from a cascading perspective is it is perfectly fine, and it's also a very normal process to have styles overriding other ones. Now it doesn't make sense in this exact use case, 'cause if you wanted to change the style by overriding it, then you wouldn't usually put this directly in the same file.

Let's test out something a little bit more advanced. If I go into our file directory, and I create, say, **styles2.css**, let's go and grab this code from here, and just paste this in Styles2.

## styles2.css

```
h1 {  
    color: goldenrod;  
}
```

Next, inside of our `index.html` file, remember how I told you that you could import multiple files? Well, let's come and do that.

```
1  <!DOCTYPE html>  
2  <html>  
3  <head>  
4      <meta charset="utf-8" />  
5      <title>My Cool Website Again</title>  
6      <meta name="viewport" content="width=device-width, initial-scale=1">  
7  
8      <link rel="stylesheet" href="styles.css">  
9      <link rel="stylesheet" href="styles2.css">  
10  
11     <!-- <style>  
12         h1 {  
13             color: red;  
14             text-decoration: underline;
```

Hit save here and then come back in the browser.



Hey from HTML  
Hey from HTML again

You can see this works exactly the same way.

One way that you could imagine that the workflow for importing style sheets is, HTML loads the document, and when it gets to the `<link>` tags, it pulls in the styles.

It'll pull in the first `styles.css` and run everything in it, then right afterward it gets to this other `<link>` and it says, "Oh, okay. We need to pull in this other style sheet." And then it pulls that in and runs it.

## styles.css

```
h1 {  
    color: red;  
    text-decoration: underline;  
}  
  
p {  
    background-color: red;  
    color: white  
}
```

Now let's check out the browser.

You can see that it says "My Cool Content", and it has a background of red and the color of white, so even though the heading tag did get overridden, the paragraph tag styles still are working.

This means that if you import multiple files, just like we did here, only the overrides are going to take the place of the preexisting one, so, in other words, this Style 2 is only going to override the H1 tag. It's not like it wipes away all of the styles from those other files.

Now we can get even more specific here. I think this is going to help because this is where a lot of bugs come in later on if you're not used to it.

Let's go, and let's un-comment the styles in `index.html`, and we're going to change this color back to red and hit save.



Now it's not overriding the file at all. It is simply overriding the selector, which is important to also keep in mind.

If we were to come down on the bottom of `index.html`, and we're going to use what's called a paragraph tag, or `<p> </p>`. We'll get into this later, but it's pretty self-explanatory. It just gives you the ability to have paragraph content.

I'll say `<p>My cool content</p>` here, and hit save, and now I'll go into our main styles file and style this.

A screenshot of a code editor showing a portion of a CSS file. The code includes a line number column (10-17), a closing brace for a previous section, a new `<style>` block starting with `h1 {`, and then the properties `color: red;` and `text-decoration: underline;`. The code editor has a dark theme with syntax highlighting.

We have added our embedded styles back in and if I hit refresh, you can see that we are back to red again. This means, that our embedded styles are going to override the selectors in both the first call and the second one.

Now the order here is very important. If I cut the embedded styles out, and I go, and I put them above those imports and come back, you can see that we're back to that goldenrod color.

Hey from HTML

Hey from HTML again

My cool content

Hey from HTML

Hey from HTML again

My cool content

Now, this may seem a little confusing if you've never done it before, however, you can think of this as just being an order of operation. As soon as the **index.html** file calls this style tag, it is going to set this as a rule.

Then, if it gets another rule, then it's going to override it. Then if it gets another one for the same element, it will override it. And it will do that for every element on the page.

## Hey from HTML

Now we can get even more specific. Let's comment out our tags and use our in-line styles instead. Let's change this one to blue. If I hit refresh, you can see that it's blue.

This is the way your style rules are going to work with cascading style sheets. That's the whole reason for the name. Now you also have the ability to see this process inside of the inspector.

Now I'm going to show you these rules. You notice how it actually has our full history of that cascade process.

So starting at the very bottom, this is the base HTML set of rules. So when you use an H1 tag in HTML, it is going to have these rules associated with it. It's going to have a default font size, it is going to have all of these other things.

The screenshot shows the 'Styles' tab in a browser's developer tools. It displays a list of rules for the 'h1' selector across different sources: 'element.style' (blue), 'styles.css:1' (green), 'index.html:9' (orange), and 'user agent stylesheet' (grey). The 'element.style' rule has 'color: blue;' highlighted. The 'index.html:9' rule has 'color: red; text-decoration: underline;' highlighted. The 'user agent stylesheet' rule has 'display: block; font-size: 2em; -webkit-margin-before: 0.67em; -webkit-margin-after: 0.67em; -webkit-margin-start: 0px; -webkit-margin-end: 0px; font-weight: bold;' highlighted.

```
Styles Computed Event Listeners DOM Breakpoints >
Filter
element.style {
    color: blue;
}
h1 {
    color: red;
    text-decoration: underline;
}
h1 {
    color: red;
    text-decoration: underline;
}
h1 {
    display: block;
    font-size: 2em;
    -webkit-margin-before: 0.67em;
    -webkit-margin-after: 0.67em;
    -webkit-margin-start: 0px;
    -webkit-margin-end: 0px;
    font-weight: bold;
}
```

You don't have to worry about them right now. You're going to learn how all of these work as you go through the material.

Then, you have an H1 rule, and you can see that this was declared inside of the index file. So if you click on that, it even shows you exactly where this was defined. This was our embedded style. Now going back to elements, you can see the next one was defined inside of our **styles.css** file.

This is the very first external style sheet that we created. And you can see when it shows that's marked out, that means that that rule has been overridden.

Then from there, moving up, we go to **styles2.css**, which you can see right here with a color of goldenrod. And that also has been overridden. And lastly, we have the element style. This is the in-line style that was added directly here of color blue. So, this is very helpful.

I can tell you, this may seem a little trivial or contrived right now, but I promise you, when you start building out big applications and big websites, you're going to run into a number of issues where you think that you have applied a certain style to an element, and it's not going to be applied.

It's going to be very frustrating if you don't know how this process works. So, one of the very first things I do when I apply a style, and it's not reflected on the page, I will open up the inspector. And then many times, what I see is that it has a little slash through it where it has this strikethrough effect.

What that tells me is that there is another style that is overriding the one that I was trying to do. And then I know what to do in order to go fix it, which is to go change up the order and to be able to have something that gives the style I was looking for.

In a final review, what we walked through in this guide is understanding the cascading nature of CSS and how the order in which you call the styles is very important, how it gives you the ability to override elements, and how you can see this entire process inside of the browser's development tools.



## Coding Exercise

Make the text of the p tag pink and also give it an underline.

```
<p>Hey there! Make this text pink and add an underline!</p>
```

# 1.10 HTML links

Right here, you can see in the finished product, we have these navigation elements. If I click on them, they will take us to these other pages. Let's walk through exactly what needs to happen in order for that to work.



We're also going to add all of the pages that this entire project is going to use. We're going to start building this out. Now, I know that you may be getting a little bit frustrated because we've been doing quite a bit of work, and we still seem like we're a very far distance away from here.

If you're learning HTML and CSS for the first time, it is critical to have some of the fundamentals down. We are building out the project, but we're learning the fundamentals as we go. Let's get started on this. Let's create the other files that we need, and let's talk about links.

I'm going to switch back to the code editor here and I'm going to get rid of all of these sample code items. I'm going to get rid of **styles** too, and everything else here. I'm going to start by just saying **homepage**. I want to do another **h1** and I'll say **homepage**.

I'm also going to indent that. Just a little side note, the indentation that you use is very important. The browser doesn't care about indentation at all, so it's perfectly fine. You could technically have tabs all over the place and place this right here, and this is still going to work.

```
# styles.css      # styles2.css    index.html •
1  <!DOCTYPE html>
2  <html>
3  <head>
4      <meta charset="utf-8" />
5      <title>My Cool Website Again</title>
6      <meta name="viewport" content="width=device-width, initial-scale=1">
7
8      <link rel="stylesheet" href="styles.css">
9
10 </head>
11 <body>
12     <h1>Homepage</h1>
13 </body>
14 </html>
```

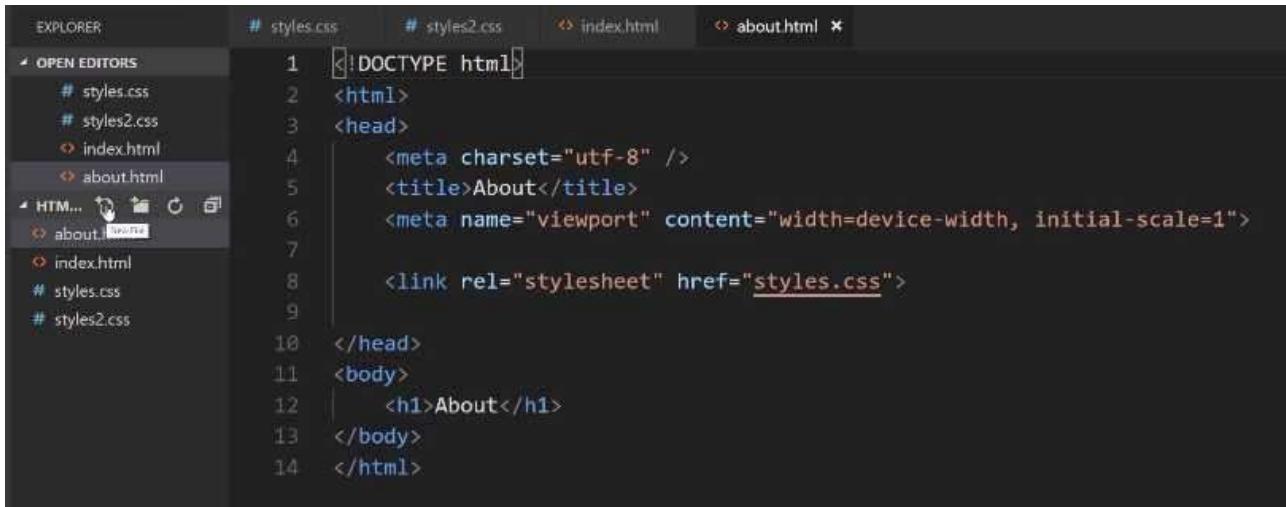
If you hit refresh, you can see it still says homepage, but from a best practice perspective, you really want to make sure that your code is aligned properly. As you go through the project, you'll see how it should be.

For right now, just know that if you have a **nested item** like this **h1 tag** is nested inside of this body tag, then it's probably a good idea to indent it. That way it's very clearly nested inside.

Now, with all that in place, let's also update the title. This is going to say **Homepage**. Later on, we're going to probably change it, but for right now, let's keep it at Homepage. Now, let's create those other files.

Let's copy everything that we have here, and now we're going to create an **About** page, a **Contact** page, and a **Menu** page. The way you can do that is by just coming over here in Visual Studio Code on the left-hand side, go to the menu bar, click on **new document** and say **about.html**.

Then you can paste that into the title. Let's say **About**, and then in the content itself, just so we can watch it all changing, we're going to say **About**. Now, let's copy this, and now go and create another document.



The screenshot shows the Visual Studio Code interface. The Explorer sidebar on the left lists files: styles.css, styles2.css, index.html, and about.html. The about.html file is currently open in the editor. The code in the editor is:

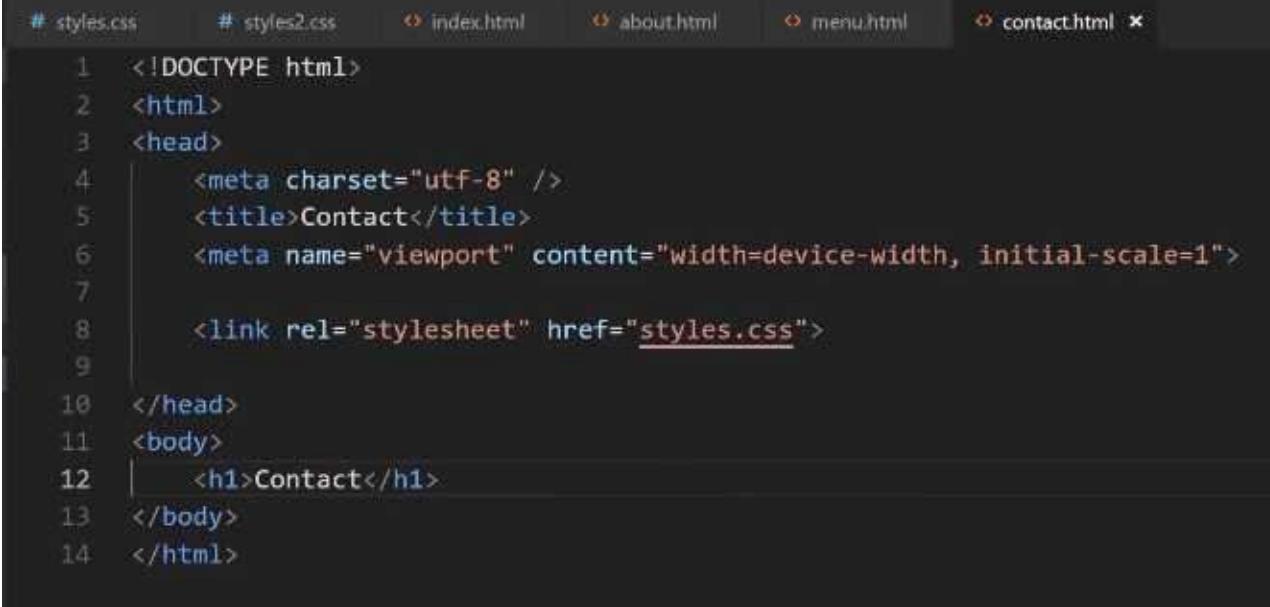
```
1 <!DOCTYPE html>
2 <html>
3 <head>
4   <meta charset="utf-8" />
5   <title>About</title>
6   <meta name="viewport" content="width=device-width, initial-scale=1">
7
8   <link rel="stylesheet" href="styles.css">
9
10 </head>
11 <body>
12   <h1>About</h1>
13 </body>
14 </html>
```

This one will say **menu.html**. Paste that in and as you may have guessed, change the title to say **Menu**, and then change that heading to say **Menu**. We just have one more left, so create one more document. We'll call it **contact.html**.

Now, you can call these files whatever you want, but I think it makes sense for them to have a very similar name just so you know exactly what is inside of each one of them. Say for example, and I have seen some students do this, where they'll create one file and call it **index.html**. That's a convention, your homepage should also be called **index.html**.

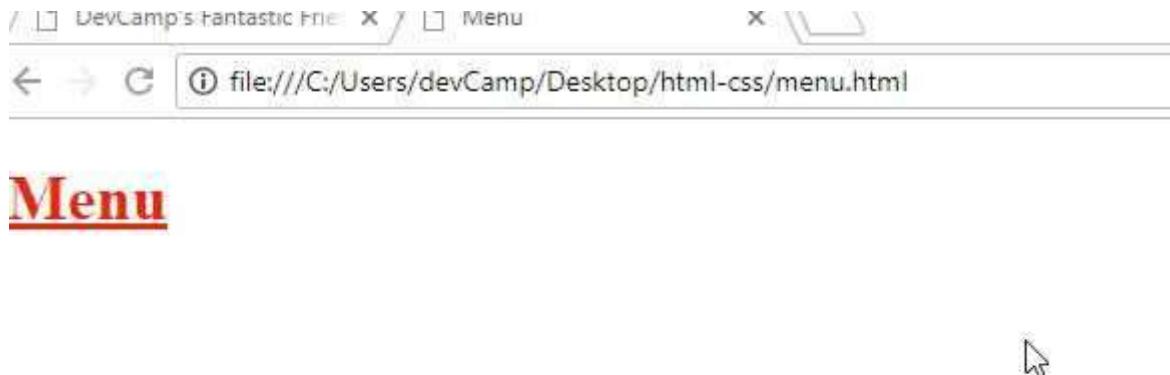
That's what browsers look for, but what the issue is I've seen before is something like having the index page there and then the other pages would be something like **2.html**, **3.html**, **4.html**, and **5.html**.

That's not very clear. If you have, say a hundred pages, you want to very clearly and quickly be able to look on the left-hand side, know what is in that file, and then it will make it easier for you later on when you want to go and make updates. The last one is going to be **Contact**, and hit save.



```
# styles.css      # styles2.css    index.html    about.html    menu.html    contact.html x
1  <!DOCTYPE html>
2  <html>
3  <head>
4  |   <meta charset="utf-8" />
5  |   <title>Contact</title>
6  |   <meta name="viewport" content="width=device-width, initial-scale=1">
7  |
8  |   <link rel="stylesheet" href="styles.css">
9
10 </head>
11 <body>
12 |   <h1>Contact</h1>
13 </body>
14 </html>
```

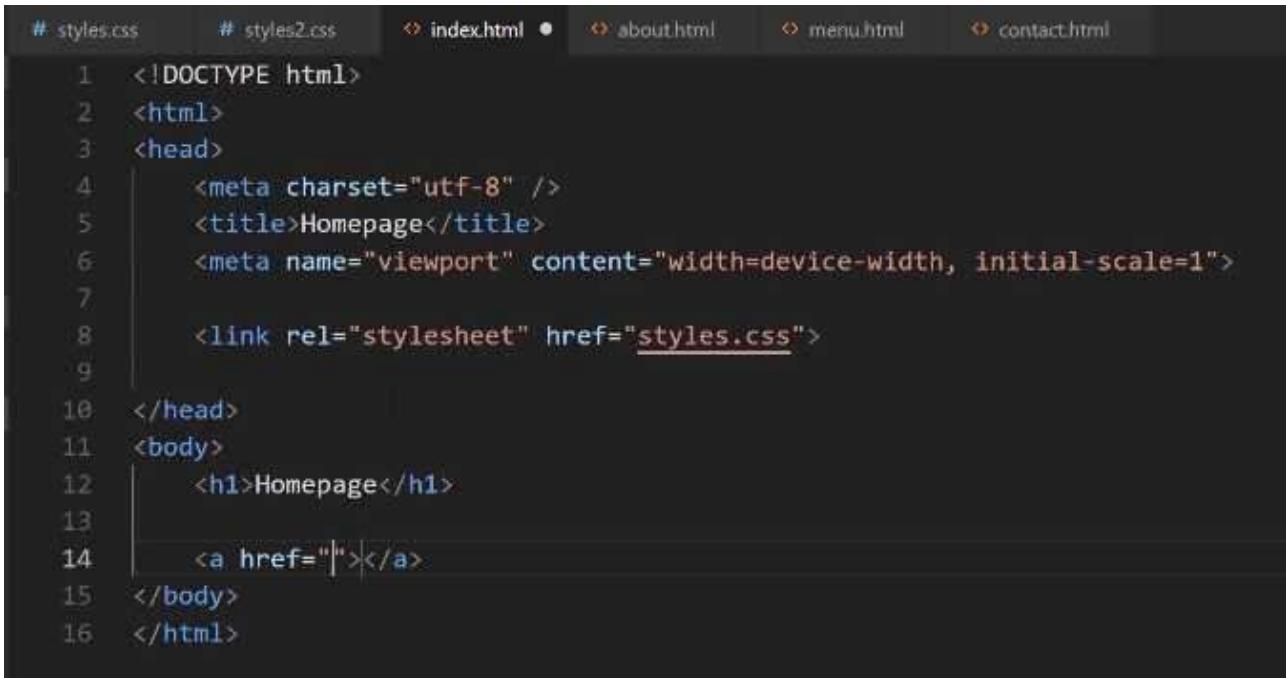
Now, if we come back here, right now we just have our Homepage, but if you come up here to the menu bar and you type say, **menu**, you can see it pulls up our menu document.



Then, here if I say **contact**, it's going to pull up the contact document. This is exactly the same process as if you hit **control + o**, and then you just went and you clicked on one of these. If I click on index once again, then we're back on the homepage.

All of our files that we want to use are all there. Now, how can we link to them? In the index, the way that you can do this, and this is going to be the start of our navbar, is we are going to use what is called an **a-tag**.

You can just start typing a, then hit tab complete, and it's going to look for an **href**. Now, this href, as you may have noticed, is also the same href we used up here for styles.



The screenshot shows a code editor with several tabs at the top: '# styles.css', '# styles2.css', 'index.html' (which is the active tab, indicated by a red dot), 'about.html', 'menu.html', and 'contact.html'. The code in the editor is as follows:

```
1  <!DOCTYPE html>
2  <html>
3  <head>
4      <meta charset="utf-8" />
5      <title>Homepage</title>
6      <meta name="viewport" content="width=device-width, initial-scale=1">
7
8      <link rel="stylesheet" href="styles.css">
9
10 </head>
11 <body>
12     <h1>Homepage</h1>
13
14     <a href="#"></a>
15 </body>
16 </html>
```

What the HTML document is saying is that we need to provide a path to the file that we're wanting to call. That's really all a link is. For a basic HTML site is a link to another document. Here, we can say **about.html** and inside of the tag itself we can say **About**.

## \*\*index.html

```
<a href="about.html">About</a>
```

Now, let's just copy this. Now, we're going to say **menu** and then **menu**. Then, we'll say **contact** and then **contact**. At the very top, let's also add one for our homepage. This is going to be **index.html** and this will just say Home.

```
<h1>Homepage</h1>

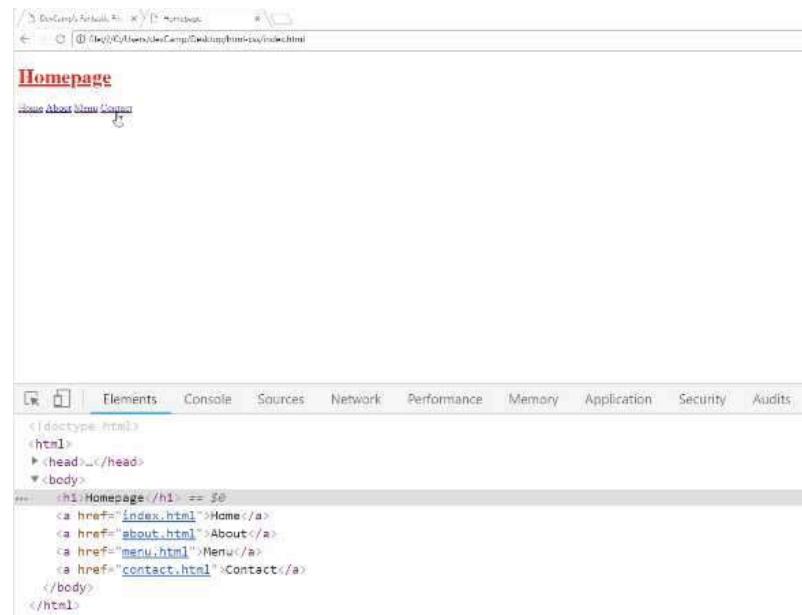
<a href="index.html">Home</a>
<a href="about.html">About</a>
<a href="menu.html">Menu</a>
<a href="contact.html">Contact</a>
```

Now, I do want to go back just very briefly and say I've kind of glossed over the reason why we use **index.html** and we use this naming convention. It's because many servers when they call for a file or they go into a directory, so kind of like, say that you have a server that goes and it hits this HTML/CSS directory. The very first thing it's going to show is a file called **index.html**.

If you don't set up any rules on the server or anything like that, it is going to assume that there is an **index** file, and it's going to try to show that. Once you get into bigger frameworks, like using tools like **React** or **Rails**, a lot of that is done for you and you don't have to do this part yourself, but that is the standard convention.

You're going to see the name **index.html** quite a bit as you build out websites and that's the reason. It's because that's what the server looks for. Okay, so now that you have that piece of information, let's switch back to the browser on our homepage and hit refresh.

Now, you can see we have our menu here. If I click on about, then you can see it takes us to the about page. I'm going to go back and if I click on menu, it takes me to the menu page. Contact takes me to the contact page.



That's all that links are is we are just telling HTML where to go find another document. In this case, it's just the documents that we have here locally. Now if you want to add this navbar to all of the other pages, then you have to copy this code and put it on each one of those pages.

Let's do that now, just to make it a little bit easier to navigate. I'm going to add it to **about.html**, **menu.html**, make sure you're saving it after each time, and then **contact.html**.

A screenshot of a code editor, likely Visual Studio Code, showing the 'menu.html' file. The code editor has a dark theme. On the left, there is an 'EXPLORER' sidebar with files listed: styles.css, styles2.css, index.html, about.html, menu.html, and contact.html. The 'HTML/CSS' section also lists these files. The main pane shows the HTML code for 'menu.html'. The code includes a header section with meta tags and a link to 'styles.css', followed by a body section containing a menu with links to 'index.html', 'about.html', 'menu.html', and 'contact.html'. Lines 1 through 18 are visible in the code editor.

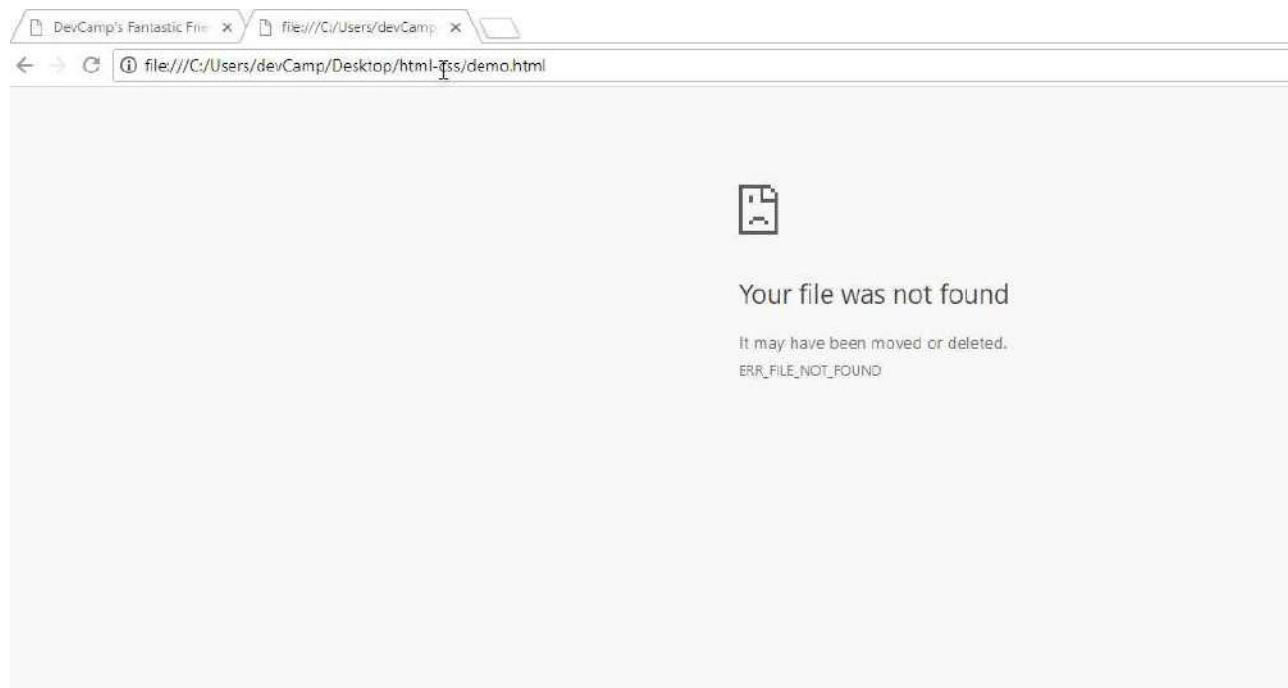
Hit save, and then come back and hit refresh. Now, if you click on any of these links, you'll see that our navigation menu's still there and it's because we copied all the links over. That gives us the ability to do that.

Now, I told you that this is a path to the **document**. It's important that it isn't just the **document** name. Let's add in a little demo here. If I click on **new folder** and say I have a folder here called **pages**. Inside of **pages**, I have another document and we'll just call this something like, let's say just **demo.html**, just like this.

In the **index.html** page, if I come down here and I create a new link and say **<a href="demo.html">Demo</a>**, hit save. Let's come and let's copy all of this content here just so we know once we've successfully reached the demo page. That's demo and then demo.

Okay, so I'm going to hit save. Notice I'm also keeping these nav elements here. It's for a reason I'm going to show you here in a second. Now, if we come back to the homepage, you can see we have this little demo link.

Now, if I click this, it's going to throw an error and the reason is because, and you can find this up in the URL bar, notice where it tried to find the document. It was still in the **HTML - CSS** directory, and then it just tried to find **demo.html**.



It's not just the file name. It's actually the full path that you have to provide. The way you could fix that is right here just say **pages/demo.html**. Now, if you hit refresh and go and click on this link, you can see that that works. That's part of it.



Now, there are a couple other things that it's important to understand when it comes to links. Do you notice how the styles are no longer being applied? The reason for that is because when we're importing the style sheet here in the demo, what is happening is it's all relative.

The **demo.html** file is looking for a **styles.css** file inside of the **pages** directory. That is very important to understand. If you don't understand that part, then you are going to run into a lot of bugs when it comes to understanding how the path works and how you can access and import files.

We saw how you can go and grab that demo link. Let me close off some of these other files and also, one very helpful little trick if you've never done this before. Let me close off everything.

I'm going to have demo open and then, let's also take a look at the index page right next to each other. You can right-click on a file and then, right here you can say open to the side.

Now, you can see both of these right next to each other. I'll close this. Now, the issue that we're having is that our styles is looking for it in the wrong place. What we can do here is go backwards. We're going to use a little bit different syntax.

Inside of our pages directory, when we want to move backwards we need to use two dots. In front of styles, put **../styles.css**. That is going to traverse us back up one directory. Now, if I hit refresh, you can see the styles are now being pulled in properly. That's how you can traverse that.

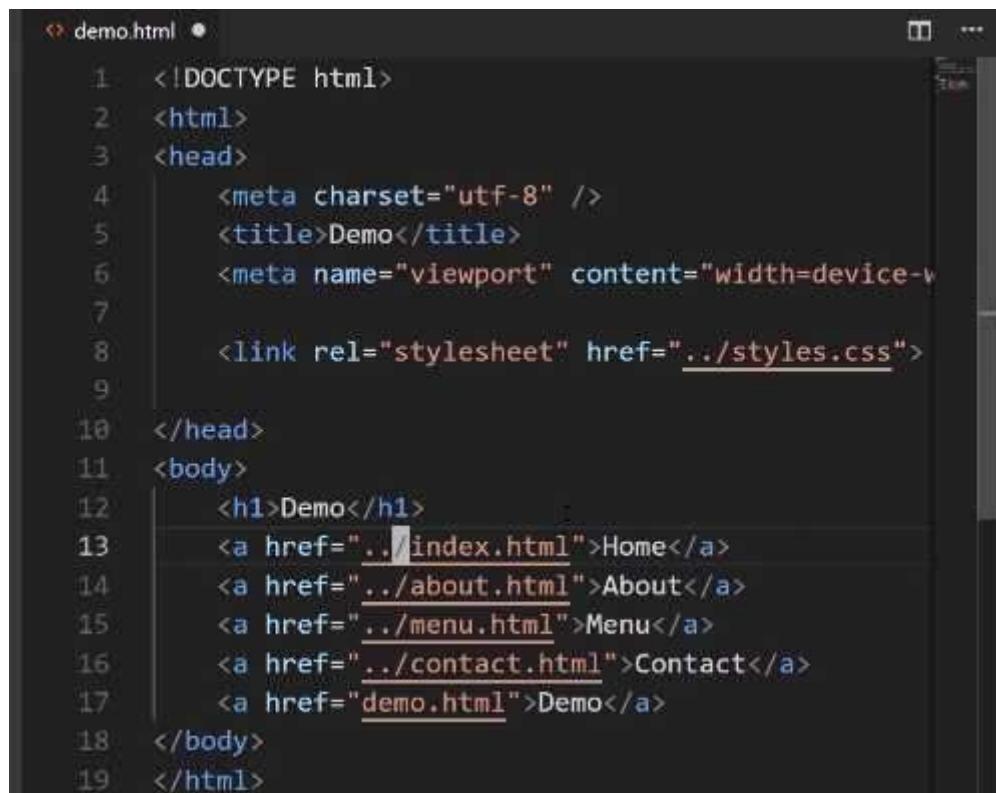


Now, we would also have to do the same thing right here. Let's just go and let's take this demo just so we have all the same exact links. Notice here, when you're in that demo directory, in the pages

directory, you do not want to say pages again or else then it's going to try to jump up into another page's directory.

We want to delete that, but now, if you hit refresh, all of these are going to be broken. If you click home, that's broken, about is broken, and hopefully by now it's starting to make sense why those are broken. It's because we need to move one level up.

The way we can do that is just to add that . . . in front of each one of these file calls. I'm going to say . . . /, hit save, and notice I did not do with the demo because that one is actually in the right directory.



```
demo.html
1  <!DOCTYPE html>
2  <html>
3  <head>
4      <meta charset="utf-8" />
5      <title>Demo</title>
6      <meta name="viewport" content="width=device-width, initial-scale=1.0" />
7
8      <link rel="stylesheet" href="../styles.css">
9
10 </head>
11 <body>
12     <h1>Demo</h1>
13     <a href="../index.html">Home</a>
14     <a href="../about.html">About</a>
15     <a href="../menu.html">Menu</a>
16     <a href="../contact.html">Contact</a>
17     <a href="demo.html">Demo</a>
18 </body>
19 </html>
```

Now, if I hit refresh, click on about, that works. Click on menu, click on home, go back to demo, demo still works when you click it, and then you can navigate to all of the other pages. That is how you can work with the path whenever you're wanting to link to other pages in the program.

Now, the very last element that I'm going to talk about, let's close off demo, is when you want to link to an outside service. Say that you want to link to Instagram. You can say <a> and then here, if you want to link to a outside website, you need to provide the full path.

Here, I can say <a href="https://Instagram.com/JordanHudgens">Instagram Profile</a> and we'll just say this is the **Instagram Profile**. Hit save, now come hit refresh, and now you can see Instagram Profile is here.

```

<!DOCTYPE html>
<html>
  <head>...</head>
  <body>
    <h1>Homepage</h1>
    <a href="index.html">Home</a>
    <a href="about.html">About</a>
    <a href="menu.html">Menu</a>
    <a href="contact.html">Contact</a>
    <a href="pages/demo.html">Demo</a>
    <a href="https://instagram.com/jordanhudgens">Instagram Profile</a>
  </body>

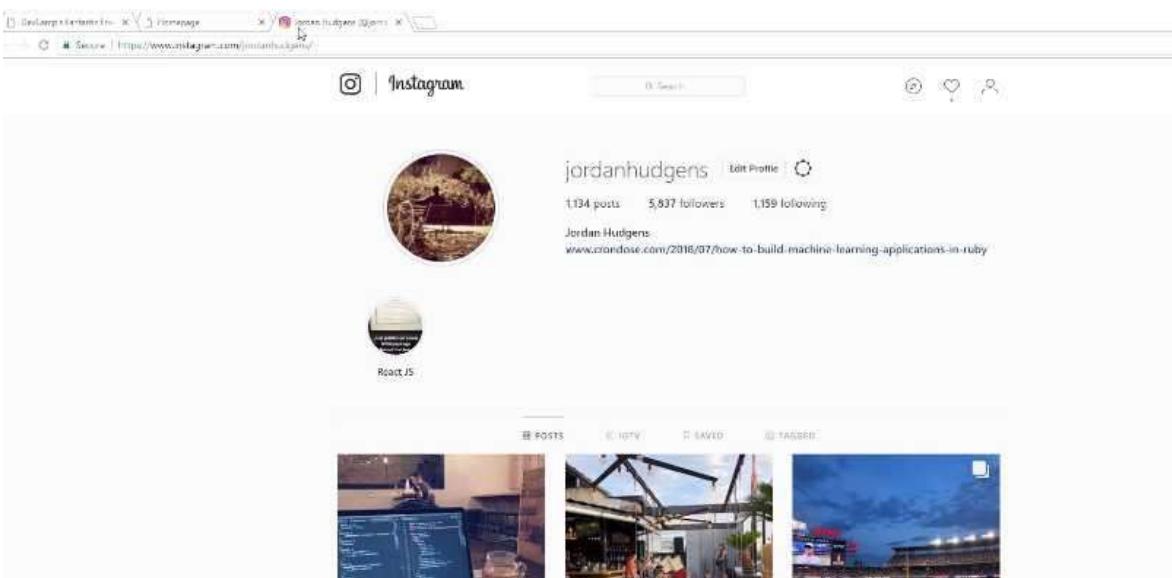
```

If I click on this, then it's actually going to open up Instagram just like that. You're able to not only link to your own internal documents, but you can also link to other websites or anyplace on the website that you want to navigate the user to.

Now, do you notice also how when I clicked on that, it actually took us away from the page? That may not always be what you want to do, so I think it's a good time to talk about some of the optional attributes that you have with links.

Right here, this is the most basic kind of link you're ever going to use. You have an `<a>` tag with an **href**, which is where you want the user to go if they click. Then, you have the content. Now, you also have the ability inside of that tag to say **target="\_blank"**.

The link is going to open up that target or it's going to open up that link in a new tab. Now, if you switch back, hit refresh, if you click on Instagram profile now, you'll see that it opens up in a new tab and it didn't get rid of the actual page that we were on.



Depending on the type of user experience that you're wanting to give your site visitors, you have both options available to you.

In review, we starting building out our navigation bar, we saw how links work, we saw how we can work with different path values, and how we can traverse up and backward. We also were able to see how you can link out and have some optional behavior, like targets, when you're wanting to connect to third party websites.



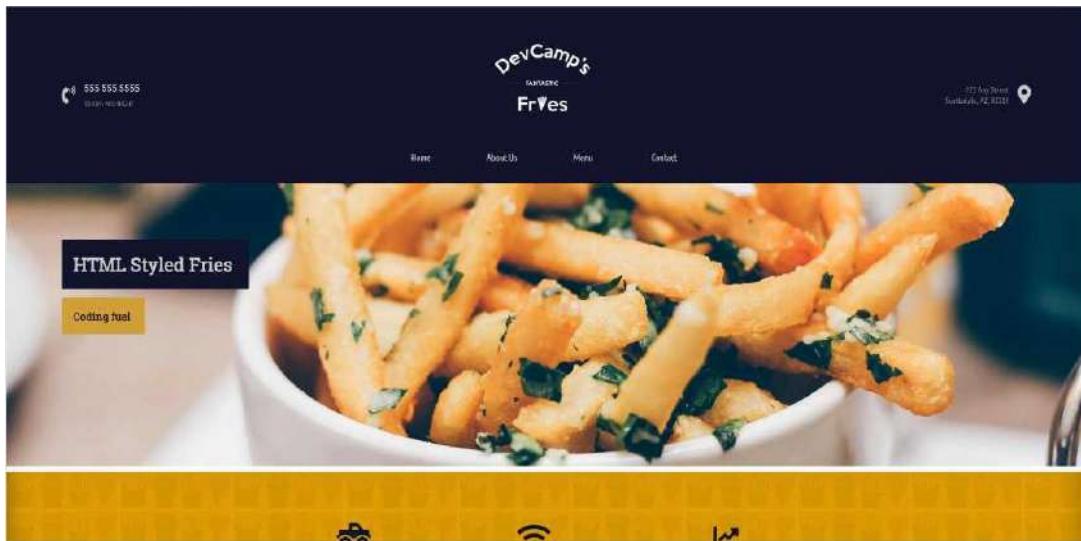
## Coding Exercise

Make a link with an href set to <https://devcamp.com/> and also add the target so that it opens in new browser tab.

# 1.11 DIV tag

When it comes to organizing elements on the page, you are going to use a very important tag called the div tag. Now, we're going to be using div tags throughout this entire course. Pretty much every element that we're going to be putting on the page, is going to be wrapped inside of a div tag.

It gives us the ability to select the item and be very specific about it, and also to organize and align the content however we need to. With that, we're going to be building out this nav bar.



Now we're not going to be organizing it like this in this guide. We're not going to move it to the right and left, and center it, and add in the styles. We're going to save that for a later guide.

For right now, I just want to get all of this content on there. Let's start off with the phone number. We know we're going to need to have this phone number. We'll add the icon later, as well as the hours of operation.

We're going to need our links, which we already have, and then we're going to add the address. We're going to use div tags in order to accomplish all of that.

Open up your **index.html** and in between videos, I removed some of the demo content, like some of the things we were just using for an example. Your file structure on the left-hand side should look exactly like what I have here with the about, contact, index, menu, and style files.

```
<!DOCTYPE html>
<html>
<head>
    <meta charset="utf-8" />
    <title>Homepage</title>
    <meta name="viewport" content="width=device-width, initial-scale=1">
    <link rel="stylesheet" href="styles.css">
</head>
<body>
    <h1>Homepage</h1>
    <a href="index.html">Home</a>
    <a href="about.html">About</a>
    <a href="menu.html">Menu</a>
    <a href="contact.html">Contact</a>
</body>
</html>
```

Now in this guide, we're not going to be actually changing a lot on the page besides just adding some content. That is going to be something we do in later guides where we're going to use CSS. For right now, this is all about structure.

Let's change some things in **index.html**

### index.html

```
<!DOCTYPE html>
<html>
<head>
    <meta charset="utf-8" />
    <title>Homepage</title>
    <meta name="viewport" content="width=device-width, initial-scale=1">
    <link rel="stylesheet" href="styles.css">
</head>
<body>
    <div>
        <a href="index.html">Home</a>
        <a href="about.html">About</a>
        <a href="menu.html">Menu</a>
        <a href="contact.html">Contact</a>
    </div>
</body>
</html>
```

Now, let's come back and hit refresh on our page,



You'll notice nothing changes. Each one of these elements is working exactly the same way as before. Now, we have some other ways that we want to use divs.

Whenever it comes to nesting elements and organizing them in code, using a div is going to be the best way to do that. Now we have this entire nav component right here. What we want

to do is we want to make that entire thing one big div.

We're going to create what I like to call a wrapper div.

### index.html

```
</head>
<body>
    <div>
        <div>
            <a href="index.html">Home</a>
            <a href="about.html">About</a>
            <a href="menu.html">Menu</a>
            <a href="contact.html">Contact</a>
        </div>
    </div>
</body>
</html>
```

Now, this may seem really weird to you if you've never worked with div's before, this should seem like a little bit of weird syntax. The reason why I'm doing this is that the more divs that you have, the more control you're going to have when it comes to selecting elements on the page.

Hopefully, that's going to make a little more sense the further we go along. Especially when we dive into CSS, but for right now let's just add in some comments here.

### index.html

```
</head>
<body>
    <!-- Navigation wrapper -->
    <div>

        <!-- Link wrapper -->
        <div>
            <a href="index.html">Home</a>
            <a href="about.html">About</a>
            <a href="menu.html">Menu</a>
            <a href="contact.html">Contact</a>
        </div>
    </div>
</body>
</html>
```

Now we have some spots where we can actually put the rest of our content. I know that we have that phone number and those hours, so let's add another div here.

### index.html

```
</head>
<body>
    <!-- Navigation wrapper -->
    <div>
        <!-- Hours + phone -->
        <div>
            555-555-5555
            10 AM - MIDNIGHT
        </div>

        <!-- Link wrapper -->
        <div>
            <a href="index.html">Home</a>
            <a href="about.html">About</a>
            <a href="menu.html">Menu</a>
            <a href="contact.html">Contact</a>
        </div>
    </div>
</body>
</html>
```

Let's come back to the home page.

555 555 5555 10 AM - MIDNIGHT

[Home](#) [About](#) [Menu](#) [Contact](#)

The next item we want to add is the address. I'm going to copy it from the design and add it in.

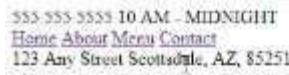
## index.html

```
</head>
<body>
    <!-- Navigation wrapper -->
    <div>
        <!-- Hours + phone -->
        <div>
            555-555-5555
            10 AM - MIDNIGHT
        </div>

        <!-- Link wrapper -->
        <div>
            <a href="index.html">Home</a>
            <a href="about.html">About</a>
            <a href="menu.html">Menu</a>
            <a href="contact.html">Contact</a>
        </div>

        <!-- Address -->
        <div>
            123 Any Street
            Scottsdale, AZ, 85251
        </div>
    </div>
</body>
</html>
```

Let's check the browser again.



As you can see, we have our content there, but we still seem like we're really far away from where we want to go, and that's fine. That is the natural progression, so don't let that discourage you whatsoever.

Just so you know, everything we implemented is actually almost all of the HTML code that is right here on the page believe it or not. CSS is what's going to give us the ability to structure the page, to place the elements and align them where we want, and add in those styles.

Now we're not quite done yet. I want to add a few other items. In addition to having things like these wrapper elements, the other thing that I like to do is, I like to have div's wrapping around the single elements themselves.

So take, for example, our nav links. I think that each one of these needs to have its own div.

## index.html

```
</head>
<body>
    <!-- Navigation wrapper -->
    <div>
        <!-- Hours + phone -->
        <div>
            555-555-5555
            10 AM - MIDNIGHT
        </div>

        <!-- Link wrapper -->
        <div>
            <!-- nav Link -->
            <div>
                <a href="index.html">Home</a>
            </div>

            <!-- nav Link -->
            <div>
                <a href="about.html">About</a>
            </div>

            <!-- nav Link -->
            <div>
                <a href="menu.html">Menu</a>
            </div>

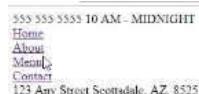
            <!-- nav Link -->
            <div>
                <a href="contact.html">Contact</a>
            </div>
        </div>

        <!-- Address -->
        <div>
            123 Any Street
            Scottsdale, AZ, 85251
        </div>
    </div>
</body>
</html>
```

So I'm going to get rid of that. And now right here, let's indent that. So this is going to be a wrapper div for the nav link. Let's just add a comment here. And we'll just say this is going to be the nav link. And then let's go and do it for each one of the other elements. So copy this, indent.

That's all we need. Each one of our links now has a div wrapper.

If you've been curious and you wonder what a div does besides just giving you some content, it also will place each item on its own line. So if I hit refresh, now you can see our nav links are all on their own line.



Now, this is not what we're going to have at our end result. But this is important because if you've never worked with div's before this might seem a little bit confusing on understanding exactly what a div does.

Now let's click on one of these div's, and it can be any of them, and if you come here on the right-hand side, do you see where it says, **div { display: block; }**?



What this means is anytime you see that something inside of the inspector tools is in italics, that means it's a base HTML default style.

Display block means that this element is going to just take up its own block space here on the page. That's the reason why before, all of our links were sitting right next to each other. But when we wrap them in div's, it moved them all on to their own lines.

That is what display block does. And this is going to be a perfect lead into what we're going to do in the very next guide.

In the next guide, we're going to be introduced to one of the most powerful tools in CSS, called Flexbox. And Flexbox is going to be what we can use to start arranging and aligning each one of these elements on the page.



## Coding Exercise

Place a div at the top that says "Welcome to my site!". Below that, make an [a](#) with the url set to `about.html` and have the link say "Get to know me".

## 1.12 HTML .class and #id

The reason for that is because we still have one piece of prerequisite knowledge that we need to walk through before we can get into Flexbox, and that is the process for working with **IDs** and **classes** in HTML and how to use those in CSS.

Let's review what we've done in regards to **selectors** up to this point. I'm going to open up the **style.css** file, and also I want to show you a cool little tool. Throughout this entire course, I want to give you some additional knowledge on how to work with tools like VS code.

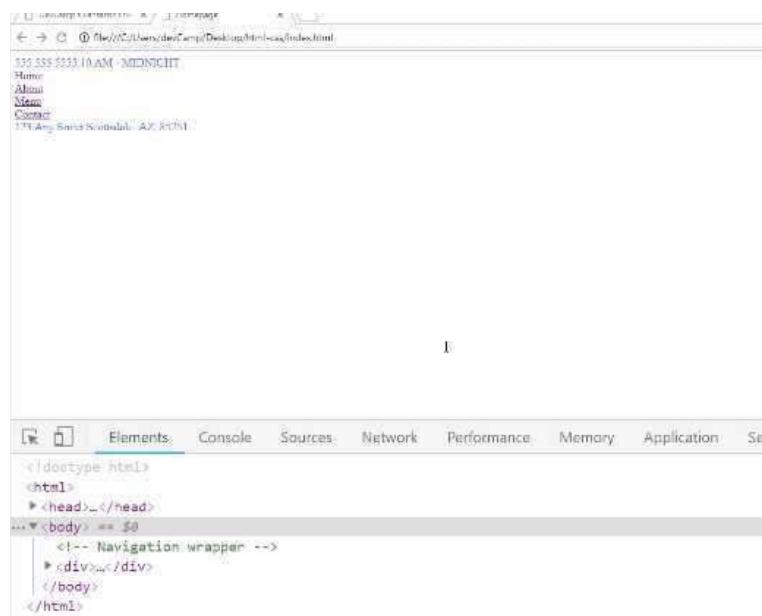
If you type **control + p**, this is going to bring up the ability to search for any file name in your entire project. Now, when you only have five files, this may not seem like a big deal, but if you're working with hundreds or even thousands of files on a large project, I can promise you that you will be using this tool quite a bit.

I want to open up the **styles.css** file here, and in between videos, I cleared it out just so we had a clean slate. We're not going to use any of that demo content anyway. Up until this point, we would select items on the page doing something like this:

**styles.css**

```
div {  
    color: royalblue;  
}
```

We'd give the tag name and then inside of that, we would provide whatever value we wanted to use. So let's use **royalblue**. Now if I hit save and then switch back to the homepage and hit refresh, you can see that our text, all of our **div** elements are now in **royalblue**. That's what you'd probably expect.

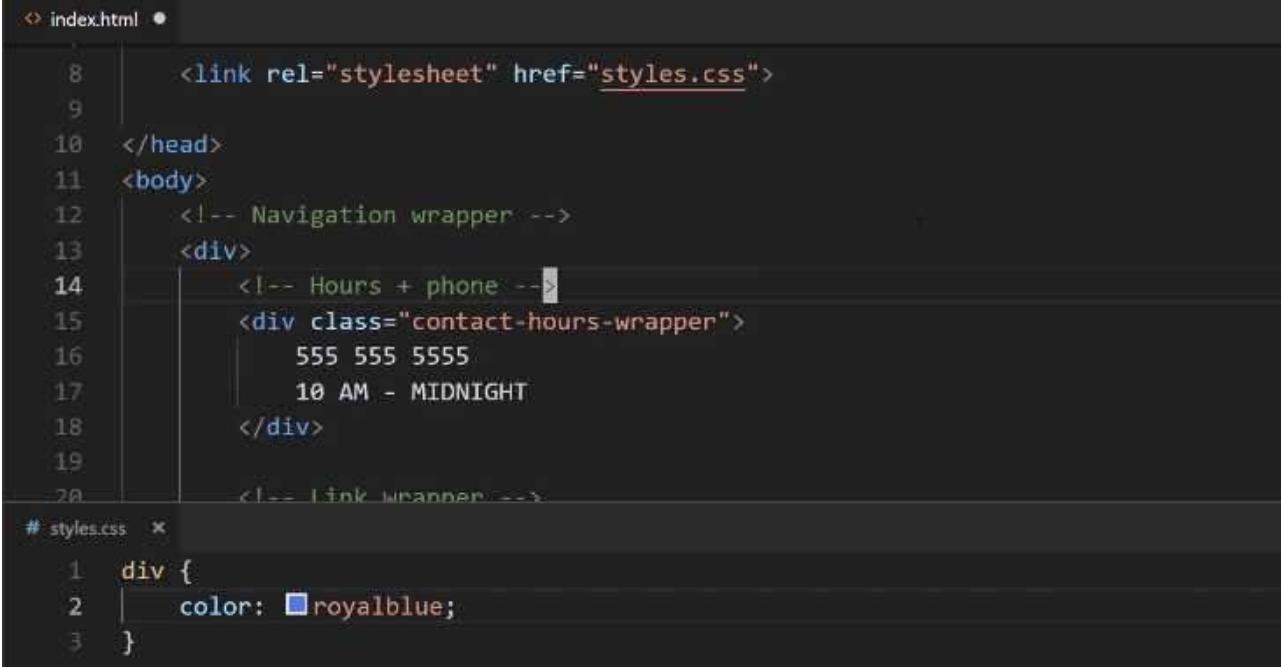


Now, what happens when you want to only apply one style to one of these elements, such as the hours and the phone number, but you don't want it to apply to the address? Well, what we can do is add a **class** that is going to allow us to do that.

Now let's look at this code at the same time. So I'm going to **right-click** on **style** here, and let's go down to **split down**. This is going to let us look at the files with HTML on the top, and the CSS on the bottom. I'm going to close out of the top one.

The way that you can implement a class in HTML is inside of the **div tag**, just say **<div class="">** and then give a name that you want to use. Right here I want it to be nice and descriptive. So I'm going to say **<div class="contact-hours-wrapper">**, just like that.

Notice how in our comments, we had to put comments here so that we would know what the contact was about or what the **div tag** was about. If I have a very good descriptive name like I have right here, I can actually get rid of that content, because now the class itself is very nice and descriptive.



The screenshot shows a code editor with two tabs: "index.html" and "styles.css".

**index.html:**

```
8     <link rel="stylesheet" href="styles.css">
9
10    </head>
11    <body>
12        <!-- Navigation wrapper -->
13        <div>
14            <!-- Hours + phone -->
15            <div class="contact-hours-wrapper">
16                555 555 5555
17                10 AM - MIDNIGHT
18            </div>
19
20        <!-- Link wrapper -->
```

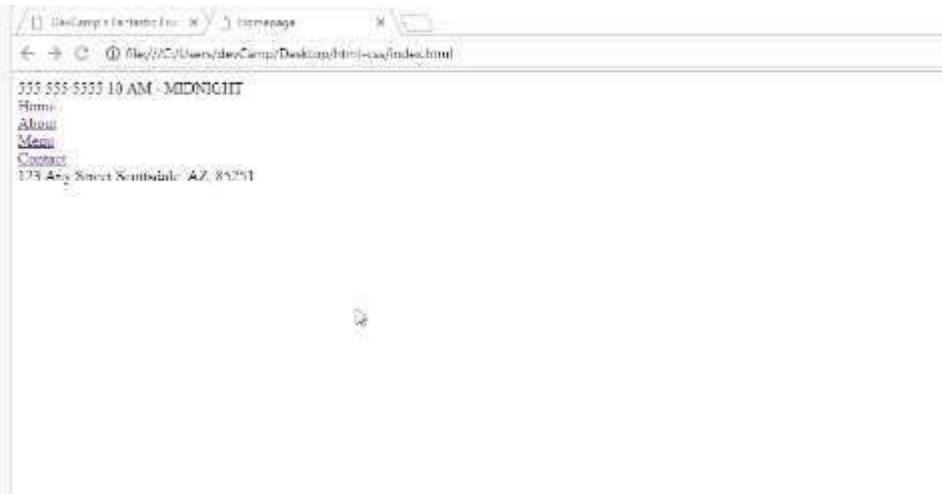
**styles.css:**

```
1 div {
2     color: royalblue;
3 }
```

I don't usually like having a lot of comments throughout my code. Usually, when I see comments, it usually means I didn't pick out the right class names because they're not descriptive enough. With all that being said, let's come down here.

Now in the **style.css** file, you are going to grab this class different than you did with the tag. So remember with the **div tag**, we selected it just by saying **div**. Well if you say **contact-hours-wrapper**, this is not valid CSS code.

If I hit save here and come back and hit refresh, you can see that our **royalblue** has not been applied. We want it to be applied to that one element.



The way that you select a class in CSS is by saying `.`, So you're going to say **.contact-hours-wrapper**, hit save. Now if you come back and hit refresh, now you have selected the right element on the page. That is how you can create a class and then call it using CSS.

The other idea that we need to understand is how to work with **IDs**. So an ID is very similar to a class, except it has a few differences and we'll walk through those here in a little bit. Now the syntax for creating them is pretty much the same.

You're just going to say `id=""`, and then inside of quotation marks, you're going to say what this value is. Here I'm just going to say, `id="address-wrapper"`, hit save. Now down below, let's try to make this red.

The way that you are going to select an ID by using the `#` symbol and then the name of the ID. So **address-wrapper** and now let's just say **color: red**.

### styles.css

```
.contact-hours-wrapper {  
    color: royalblue;  
}  
  
.address-wrapper {  
    color: red;  
}
```

Now if I save and come back, hit refresh, you can see that we have selected the address. So this is all working very well. The very first question that you might have is what is the difference? Because they seem to act pretty much the same exact way, and for our very tiny use case, they do.



For a larger application, the rule of thumb is that if you are ever, ever going to have multiple items on the page, so say that you're going to have this **contact-hours-wrapper** multiple times on the same page, you want to use a class. If you are only going to be using an element one time, then you can use an ID.

Now throughout this course, I will most likely be using a class for almost every single element that we're going to build. The reason for that doesn't really have anything to do with a project this size. The issue that you could run into is some of the larger frameworks like **React** or **Angular** or some of those, they actually can create some conflicts here.

Some of the frameworks will override the ID value. Because of the way that their system works and the way those frameworks work, sometimes they will set the ID dynamically. You don't even have any control over that or you don't without creating a lot of work arounds.

You could create code exactly like how we have right here and then your selector doesn't work, because technically your **address-wrapper** no longer exists because the ID got overridden. For the sake of just being safe, I just pretty much use a class all the time, and that is the recommendation from a best practice perspective.

I'm going to get rid of this comment because we don't need it anymore, and then I'm going to change the ID to a class. Right here, you can see we have all these **nav links** right here. Well, what we can do is we can actually get rid of the comments and then here, let's just add classes.



The screenshot shows a code editor window with the file 'index.html' open. The code is a snippet of HTML with line numbers 26 through 48. Lines 26-28 show a comment and an anchor tag with href='about.html'. Line 29 is a closing div tag. Lines 30-32 show another comment and a div tag. Lines 33-35 show an anchor tag with href='menu.html' and a closing div tag. Lines 36-38 show another comment and a div tag. Lines 39-41 show an anchor tag with href='contact.html' and a closing div tag. Line 42 is a closing div tag. The code uses standard HTML syntax with nested div elements and anchor tags for navigation.

Now it's going to be **class="nav-link"**. Let's go and do that for each one of them. I'm going to do it here for this one and the same thing for contact. Now, nothing's changed here. Now we just have some **nav-links**.

Once we get into the next guide, where we are going to get into **Flexbox**, you're going to see that we're going to add a few more classes so that we can select those items.

I wanted you to be able to see that when you use a class, it's very common to use it multiple times on the same page. Then you have the ability when you have that, to select specific elements instead of just being very generic with your selectors. Like where you just grab an entire `div`, because then you're going to affect every single div on the page.

If you try to apply one color for example, and then you just selected the `div`, then you'd be overriding every single div in the entire page, which is not probably what you want. That's where classes and IDs can be very helpful because they allow you to become much more granular, and you have more specific control on what you're selecting.



## Coding Exercise

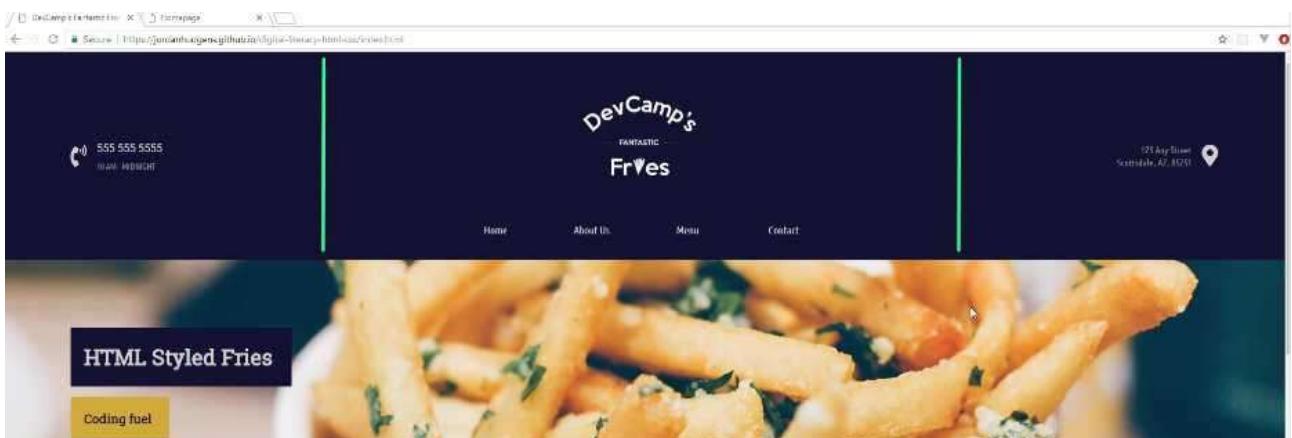
Write a div with an id of "header-wrapper" and another div with a class name of "content-wrapper".

# 1.13 Flexbox

In this lesson, we're going to start building out this navbar component.

This will be everything from the hours and phone number to these links, and then also the address. We're not going to have the logo here later. We're going to save that for a guide that talks specifically about how we can use images in HTML.

Whenever you have something like this that you want to build out, and you have a three column layout. Right here you can see. You can kind of visualize that each one of these elements is represented and contained inside of a column.



Years ago if you wanted to implement something like this, this took quite a bit of work. Thankfully because of the skill that we're going to learn in this guide with **Flexbox**, this is going to be a relatively straightforward system to build out.

Let's switch back to our code here and then open up the text editor. What we're going to do is we're going to start at the very top of the chain here. You can see that we have this **div** and then inside of it, that's where all of the navigation elements are.

```
index.html
10  </head>
11  <body>
12      <!-- Navigation wrapper -->
13      <div>
14          <div class="contact-hours-wrapper">
15              555 555 5555
16              10 AM - MIDNIGHT
17          </div>
18
19          <!-- Link wrapper -->
20          <div>
21              <div class="nav-link">
22                  <a href="index.html">Home</a>
```

You can see we even put that comment there. Well, just like we did before, I'm going to get rid of the comment and instead, I'm going to give it a class. This is going to have **class="navigation-wrapper"**.

Now let's come down into the **styles.css**, and at the very top here, I am going to add a comment. You can see the comments in CSS have a little bit different syntax where it's the **/\* \*/**, then whatever you want to put inside of it.

Here, I'm just going to say, "Common nav styles", and the reason why I'm doing that, it doesn't have anything to do with code. It's just so that later on when we want to reference this, we'll know exactly where the nav styles are.

Right here, I'm going to start by adding styles to the navigation wrapper. Inside of here, I first want to establish a **height**. Right now, the height is going to be determined by the content and I don't really want that.

If you look at the end goal, you can see that this has a predetermined height right here. Let's come over and let's add that height. We're going to use a height attribute, and the value's going to be **190px**.

### styles.css

```
.navigation-wrapper {  
    height: 190px;  
}
```

If you're not familiar with what a **pixel** is, it's pretty much what every kind of screen out there, every image, anything digital is made up of pixels. If you're looking at an image, is a great example, and you want to zoom in. If you can zoom in all the way to the pixel level, you'll see that that image is made up of squares.

What a pixel does is it is a unit of measure. Right here you'll see what we're going to do, is we're saying that we want **190** of those little squares, and that is what the height's value is going to be.

If you're coming from a non-coding background and you're just used to working with inches or millimeters or anything like that, a pixel is essentially the digital version of that unit of measure.

Now that we have a height, let's also add a **background-color**. So I'm going to say, **background-color**. Up until now, we've used these nice little names, and these are some built-in colors provided by CSS. We also have a few other options and we're going to talk about those throughout the rest of the course.

If you start off your color with a **#**, what this means is that you can give a **hexadecimal value**. I'm going to say **#11122B**. What this is going to give us is a nice dark blue color. That's going to be the background color.

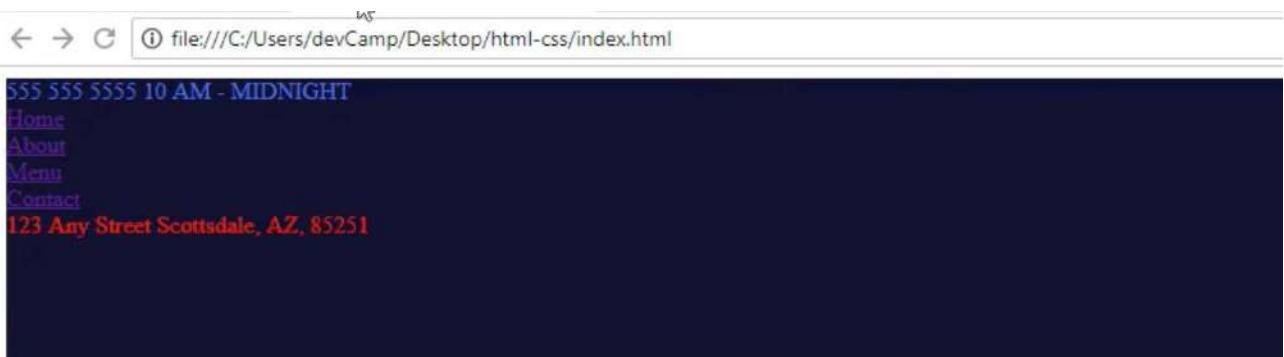
Now, we also want to have a color for the text and so whenever you want to have a background color, the attribute name is going to be **background-color**.

Whenever you just say **color**, that is referring to the text. Here I'm going to give another # value and it's going to be **#CFCFCF**, just like that, and that will give us a light gray color.

### styles.css

```
.navigation-wrapper {  
    height: 190px;  
    background-color: #11122b;  
    color: #cfcfcf;  
}
```

Let's hit save, and just see what that's given us so far. There you go. You can see that we now have that dark blue color, and you may also notice that we have some white around the edges. I'll show you how to fix that because we want our background just like this to be able to be nice and flush.



We're not going to worry about that right now. We'll do that later on. Now that we have that, we're going to get into using **Flexbox**. What Flexbox is, is it's built directly into CSS. You don't have to install anything or import any libraries or anything like that.

You can just use it if you're building CSS code. What it allows you to do is to align items on the page in a much easier fashion than before. If you were to see the way that we had to align items back 10, 15 years ago, it was really not fun.

You had to get a little bit tricky with how you arranged everything on the page, but with tools like **Flexbox** and **CSS grid**, it gives you a really nice way of organizing and aligning items.

Now, anytime that you want to have Flexbox used, the first thing that you need to do is type **display**, and then just say **display: flex**. That means that this navigation wrapper is now going to be what is called a **Flexbox container**.

I'm going to add a comment just right above here and say this is now making this a "Flexbox container". Don't worry about what this word means. We're going to talk about it, but it's a very important piece of terminology.

```
# styles.css •  
1  /* Common nav styles */  
2  .navigation-wrapper {  
3      height: 190px;  
4      background-color: #11122b;  
5      color: #cbcfc;  
6      /* Flexbox container */  
7      display: flex;  
8  
9  
10 }
```

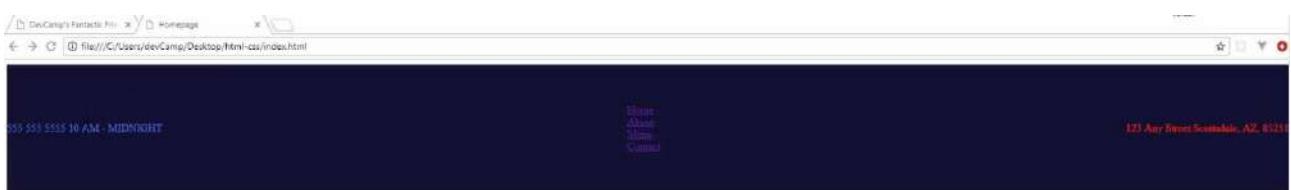
We have **Flexbox containers** and we have **Flexbox items**. As soon as you understand the difference between the two and how they work together, everything in Flexbox and how you can arrange items on the page is going to make a lot more sense.

Right here, we have now, because we've said `display: flex`, we've made the **navigation-wrapper** a Flexbox container. Let's also add a few Flexbox rules, so I'm going to say **justify-content**:. Let's make this one **space-between** and I'll talk about what that represents here in a second, and then **align-items: center**.

### styles.css

```
.navigation-wrapper {  
    height: 190px;  
    background-color: #11122b;  
    color: #cbcfc;  
    /* Flexbox container */  
    display: flex;  
    justify-content: space-between;  
    align-items: center;  
}
```

For right now, let's keep it just with that. We'll add a few more things here shortly. If you hit refresh, you can see that that has now got us a little bit closer to our goal. Now obviously, we still have long ways to go before it looks like this.

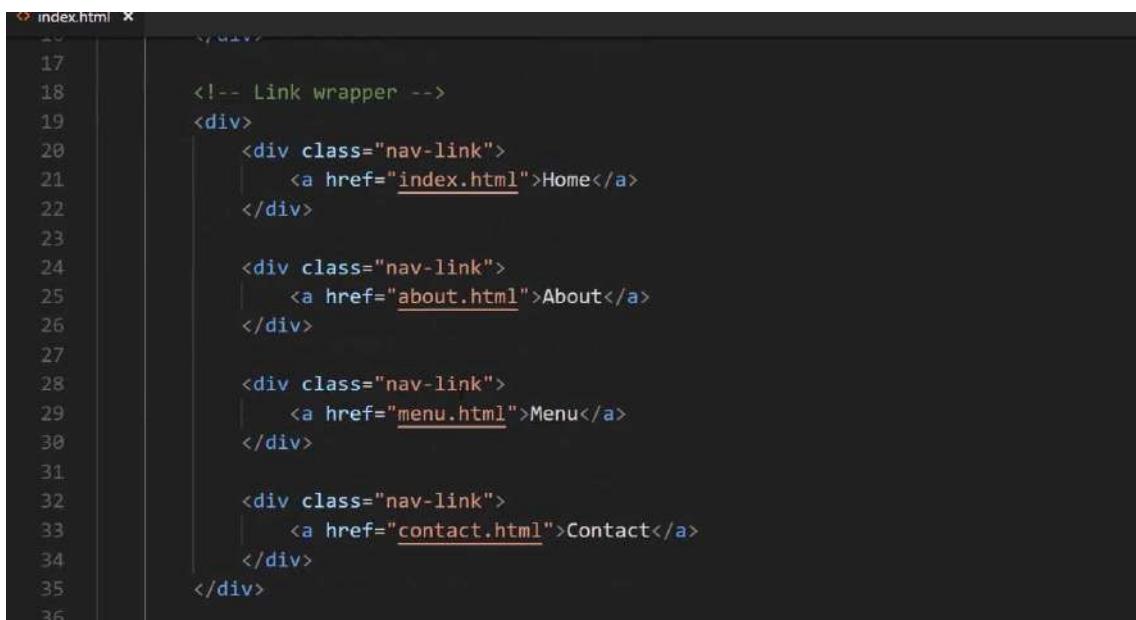


However, by using Flexbox you see how we were able to arrange the items on the page exactly where we wanted them. This is critical for understanding the way that the process works. What Flexbox does is we have made this entire unit here, so this entire nav bar is now a Flexbox container.

All of the items inside of it. So each **div** becomes a Flexbox item. The way that it works is we have now one container, and then we have three items inside of it. Let's look at the HTML code, and I'll close styles off here.

What this did is **navigation-wrapper**, when it became a Flexbox container, automatically with the way Flexbox works, is it took each element inside. It took this div of **contact-hours-wrapper**, then it took the **link-wrapper**, and then it took the **address-wrapper** and it made those all Flexbox items automatically.

Now, the way that Flexbox works is this is not a deep nested relationship. What I mean by that is that it only looks one level deep. So the **nav-links**, so each one of these links, are not Flexbox items.



```
index.html
17
18     <!-- Link wrapper -->
19     <div>
20         <div class="nav-link">
21             <a href="index.html">Home</a>
22         </div>
23
24         <div class="nav-link">
25             <a href="about.html">About</a>
26         </div>
27
28         <div class="nav-link">
29             <a href="menu.html">Menu</a>
30         </div>
31
32         <div class="nav-link">
33             <a href="contact.html">Contact</a>
34         </div>
35     </div>
36
```

We're actually going to use another tool for that later on. All that Flexbox does is it looks at the very next level. If you think I'm going and I'm kind of repeating myself here a little bit, I am because this part is so important.

When I was learning how Flexbox works, it didn't really click for me until I understood this one process: that there is a **parent-child relationship** where you have this container element and then, however many items are inside, those are the elements. Those are the Flexbox items.

I just want to show you this one more time. If I add another address here, the way it works is if you come back here and hit refresh, you see how it got added automatically here to the right-hand side.



The way it works is you have the first element, the second element, the third, and then the fourth. Then to go kind of line by line, we'll open up **styles.css**. We gave our height, we gave our background-color and our color.

Then inside our Flexbox rules, we displayed flex, which made this a Flexbox container. Then from there, what we did is we said, **justify-content**. Let's talk about this. What justify-content does is it tells the elements inside how to behave.

It says if you want to do **space-between**, which is what we did, that means that the way they're going to be separated is by the space. All of the potential space between them. In this case, it is going to go all the way from side to side, and then they're going to be separated by whatever available space that they have in between each one of those items.

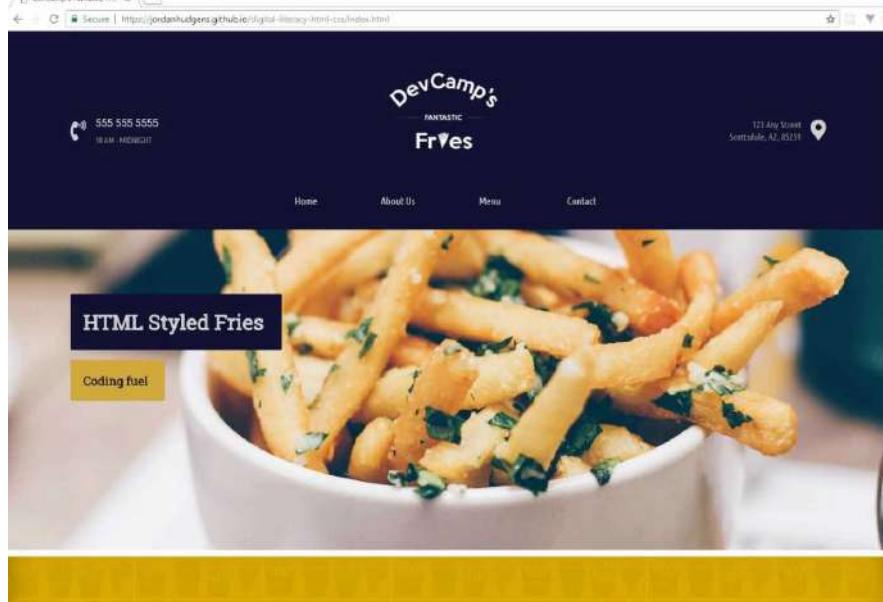


That space is flexible. If you were to come back here to **index.html** and let's add a few more of these, hit save, come hit refresh. Now, do you see how the space in between has shrunk?



What that means is with **space-between**, it just is going to allow for any of the available open space to be used. You can see here in the final version, this is exactly what we have going on. Where we have the space-between has all of this open area.

If we were to pop this open in a new browser window and shrink it, watch what happens. Do you see how the space shrinks depending on the size of the window? That's the way it works. It is not a hard-coded value.

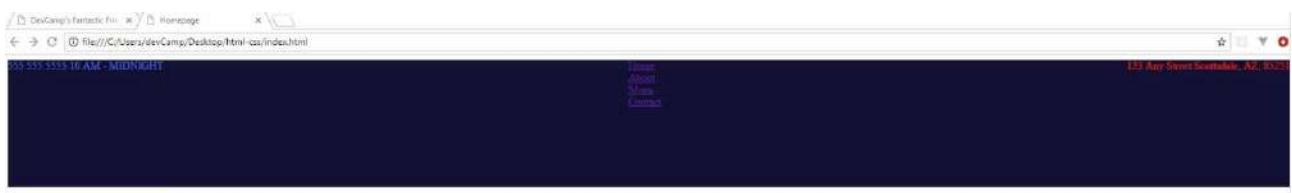


There are ways of doing hardcoded ones. If you do need the space to remain there, but in our case, I want this to be flexible. That's giving us exactly what we're looking for right there. Let's come and let's get rid of the addresses that we don't need. Hit save, hit refresh, and we're back to where we wanna be. That is how **justify-content** works.

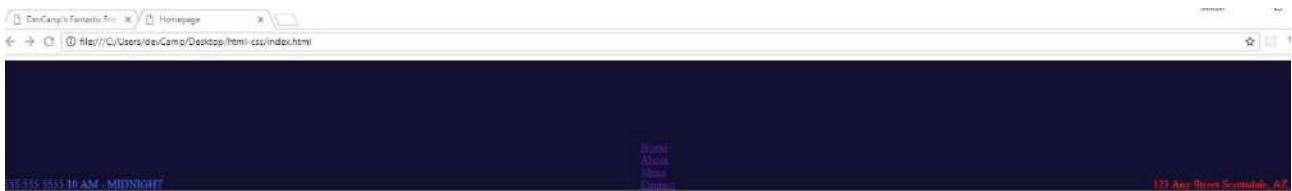
Now if you come down here, you can see that we also have **align-items**. The way align-items works, if justify content provides us the ability to tell Flexbox how we want these items to be aligned, what align-items does is it says how they should be aligned vertically.

When we said **align-items: center**, what we're saying is I want these items to be arranged at the very centermost part of that div element. That is going to be the way that they work. That's the way we want them to be because we want these to be centered.

If you wanted them to be at the top, then we would change this rule and we would say **align-items: flex-start**. Now, if you hit refresh, you can see that is at the top and let's also use these tools right here.



Let's use the browser tools. You can see we have **align-items** here. If we wanted to say **flex-end**, you can see they go down to the bottom.



The screenshot shows the Chrome DevTools Elements tab. The DOM tree on the left shows the structure of the page, including the `navigation-wrapper` class applied to a `div` element. The right side of the screen displays the Styles panel, which shows the CSS rules for the selected element. The `element.style` section contains a single rule: `element.style { }`. The `styles.css` section contains the following code:

```
.navigation-wrapper {  
  height: 190px;  
  background-color: #11122b;  
  color: #cfcfcf;  
  display: flex;  
  justify-content: space-between;  
  align-items: flex-end;  
}
```

I'm going to go back to center. This is where we want them to be. Now, we also could play with **justify-content**. If you want to see some of the other options here, you have **space-between**. You also have **center**. If you wanted all of the items aligned perfectly in the center, then you could use the center rule.

Now, you could use **flex-end** and they would all be at the right-hand side, **flex-start** and they'd all be on the left-hand side. You can also do **space-around**. You can see our space-around places all of the items here. It takes open space on the left-hand side, open space between the elements, and then open space on the right-hand side.

Now, I did contemplate using this. This would work for a number of situations, but later on, I'm going to show you, actually in the next guide, how we can hardcode some values. Because I would like to have some open values here.

I don't wanna use **space-around** on that side. I'd rather the middle be flexible, and then us to have enough open area here

on the right and left-hand side, just to make sure that we always have kind of that symmetrical look.

I don't care about this space, but I do care about the space on the right and left-hand side. So with all of that being said, great job. If you went through that, you now know how to implement a Flexbox container and how to align the items inside of it.



## Coding Exercise

Utilize flexbox with **space-between** and align the divs in the center, the same as you did in the video.

```
<div class="parent">
  <div class="child">Spread</div>
  <div class="child">These</div>
  <div class="child">Out</div>
  <div class="child">With</div>
  <div class="child">FlexBox!</div>
  1. </div>
```



# 1.14 CSS Padding

This is going to be a relatively short guide, but we're going to discuss a very important topic in CSS. That topic is padding.

Let's switch back to the code, and let's also revert this back so `align-items` is back to `center`. I'm going to say `align-items: center`. Then as we've already discussed, we want to have some value, we want to have some empty white space here on the right and left-hand side as you can see this value that we have here.

Now, I did not want to use the `space-around`. It's mainly because I want to have exactly the type of look and feel what we have right here. If we had `space-around`, and we can actually change this just using the devtools. With `navigation-wrapper`, if I remove the padding, as you can see, that's going to change a lot of things.

Let's ignore that for right now. If I did `space-around` instead of `space-between`, you could see it kind of gets us where we want, but it's not perfect. It's not exactly what the designer originally intended.



Instead, what I would like to do is I'd like to have some hard-coded values for the left and right here, and then also on the top and bottom. That's what padding is going to allow us to do.

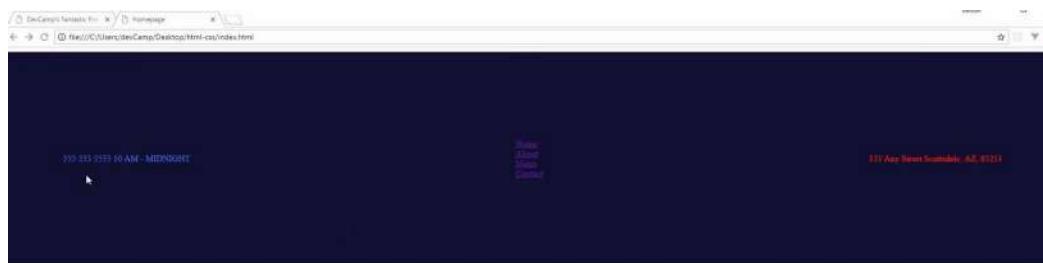
I'm going to get us back to where we want here, and now let's come back into the code, and let's see exactly what we need to do. I'm going to, at the very bottom here, I'm still in the `navigation-wrapper` style definition, and I'm going to say: `padding`.

## styles.css

```
.navigation-wrapper {  
  height: 190px;  
  background-color: #11122b;  
  color: #cbcbcb;  
  /* Flexbox container */  
  display: flex;  
  justify-content: space-between;  
  align-items: center;  
  padding:  
}
```

Now, there are a couple ways that you can declare padding. I'm going to show you a few different options, and then I'm going to explain which one that I personally like going with.

We could say something like `padding: 100px`. This is going to be kind of the sledgehammer. This is going to change the padding for the top, the bottom, the left, and the right. If I hit refresh now, you can see that this is way too much space.

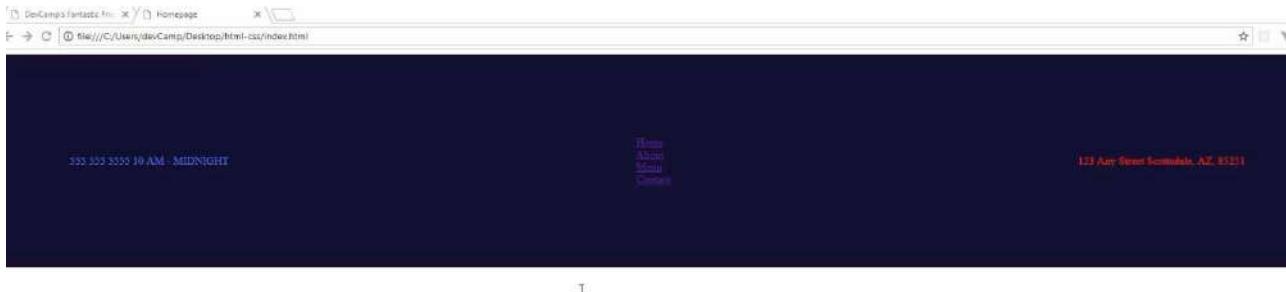


This is giving us padding all over the place. What I'd like to do is become a little bit more granular with it. Another option is I could go, and let me comment this out, I could create a rule for each side. I could say:

## styles.css

```
.navigation-wrapper {  
  height: 190px;  
  background-color: #11122b;  
  color: #cbcbcb;  
  /* Flexbox container */  
  display: flex;  
  justify-content: space-between;  
  align-items: center;  
  /* padding: 100px; */  
  padding-top: 60px;  
  padding-bottom: 60px;  
  padding-right: 100px;  
  padding-left: 100px;  
}
```

This is an approach that I will use from time to time, so if you hit refresh, you can see that that's giving us more the size that we're looking for.

A screenshot of the Chrome DevTools Styles tab. The left sidebar shows the DOM tree with the current node selected as '

'. The right panel displays the CSS rules for this element. The first rule is 'element.style { }', followed by '.navigation-wrapper { height: 190px; background-color: #11122b; color: #cbcbcb; display: flex; justify-content: space-between; align-items: center; padding: 60px 100px 60px 100px; }'. A small 'styles.css' icon is next to the selector.

We still have a few other options, so that gives us the ability to do everything all on one separate line, but you can also do it all together. I could say padding here, and then you could pass in either two or four arguments. I'm going to do four arguments at first.

It goes clockwise, so the way it works is I could say: the top is going to be `60px`. Then the next one is going to be to the right, so that's going to be `100px`. Next one's going to be on the bottom. That's `60`. Then left.

### styles.css

```
.navigation-wrapper {
  height: 190px;
  background-color: #11122b;
  color: #cbcbcb;
  /* Flexbox container */
  display: flex;
  justify-content: space-between;
  align-items: center;
  padding: 60px 100px 60px 100px;
  /* padding-top: 60px;
  padding-bottom: 60px;
  padding-right: 100px;
  padding-left: 100px; */
}
```

If you think of the way a clock looks, the way that you give these style values here work exactly the same way. You just start at the very top of the clock, then you go to the right where the three would be, then you go down to the bottom where the six would be, and then all the way around to the left where the nine would be, and that gives you all of those values.

71 / 355

If you hit Refresh, you can see nothing changes. This is giving us the exact same control as when we did it all on their own dedicated lines.

Now, if you ever run into a situation where you see this, where you want to have identical values for the top and bottom, and then identical values for the left and the right, there's even another short-hand syntax.

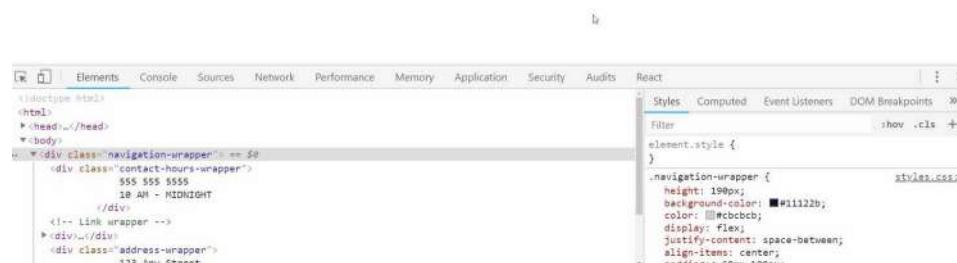
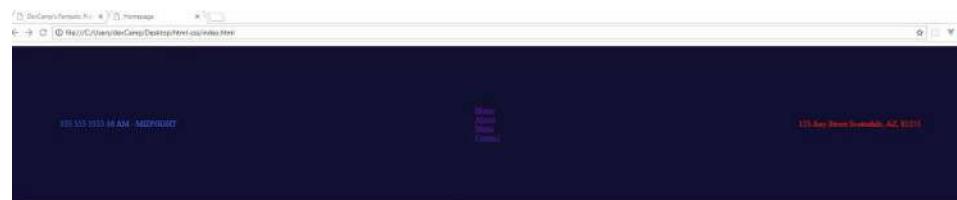
You can come here, and you can just use two values. What this will do is you'll say, the first one is going to work for the top and bottom, and the second one's going to be for left and right.

### styles.css

```
.navigation-wrapper {  
  height: 190px;  
  background-color: #11122b;  
  color: #cbcbcb;  
  /* Flexbox container */  
  display: flex;  
  justify-content: space-between;  
  align-items: center;  
  padding: 60px 100px;  
  /* padding-top: 60px;  
   padding-bottom: 60px;  
   padding-right: 100px;  
   padding-left: 100px; */  
}
```

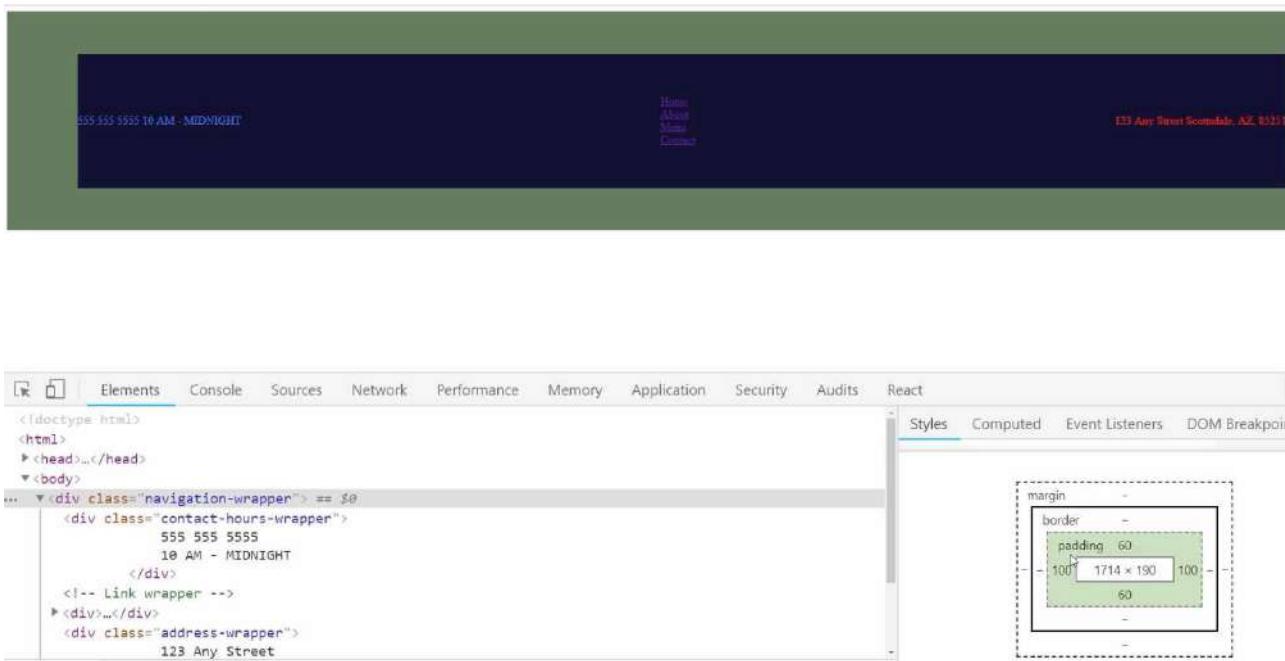
If I hit save, hit refresh, you can see we have the exact same values. Now, this is going to be the best solution for this particular use case. Just because we are in a situation where we want to have top and bottom the same and then left and right the same.

That is what I'm going to go with, but I did want to show you all three options just so you're always familiar with that. Hit Refresh. Make sure everything is working.



Now, before we take a break and go on to the next video, I want to show you what padding represents. Thankfully, there is this very helpful little diagram here that's provided with the Google developer tools that shows what padding is.

Right here, you can see that if you hover over padding on the top there, it actually shows in that green box what the padding values are. It even says that right there we have 60 on the top and bottom and then 100 on the left and right.



Now, make sure, if you're not seeing this exact value, make sure that you either go in here in the elements and click on `navigation-wrapper` or just go and select the `navigation-wrapper` using the `inspect` tool.

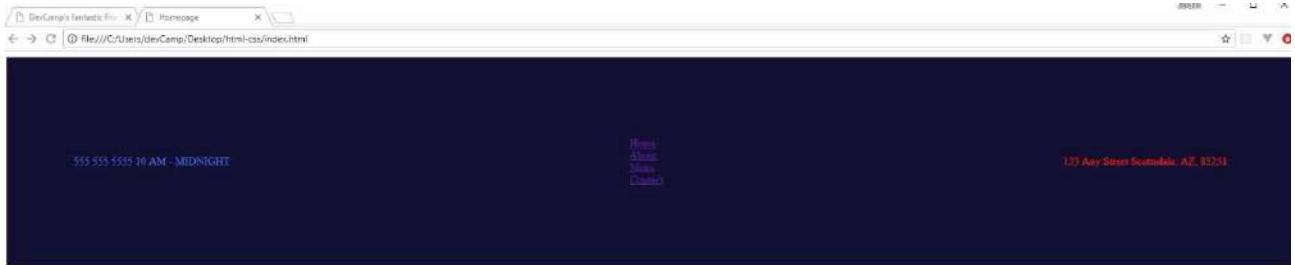
What this does is this gives you what's called the `box model in CSS`. We haven't talked about all of these yet, so I'm not going to get into them yet, but I wanted to show you padding because this is very helpful.

Whenever you're building out some kind of interface and maybe the items are too far to the right or to the left, or they're just not lining up the way that you want, one of the best ways to debug that is to come down here to this box diagram and see what the values are.

There are plenty of times where I've done that, and then I've seen that "Oh, there's some padding value that needs to be set," or, "The border's throwing it off, or maybe the margin," which we'll talk about these later on.

That gives you the full box model. You have the core content, which, if you hover over it, this shows you all of the flex items, and then outside of that is where the padding lives. Then the

**border**, which right now, there is no border, and then the **margin**. We haven't provided a margin or border, so that's why they don't have any values.



For right now, I think it really helps to visualize exactly what the padding represents and what it represents for every stage and around each side of the div. If you went through that, great job. You now know how to work with padding in CSS.



## Coding Exercise

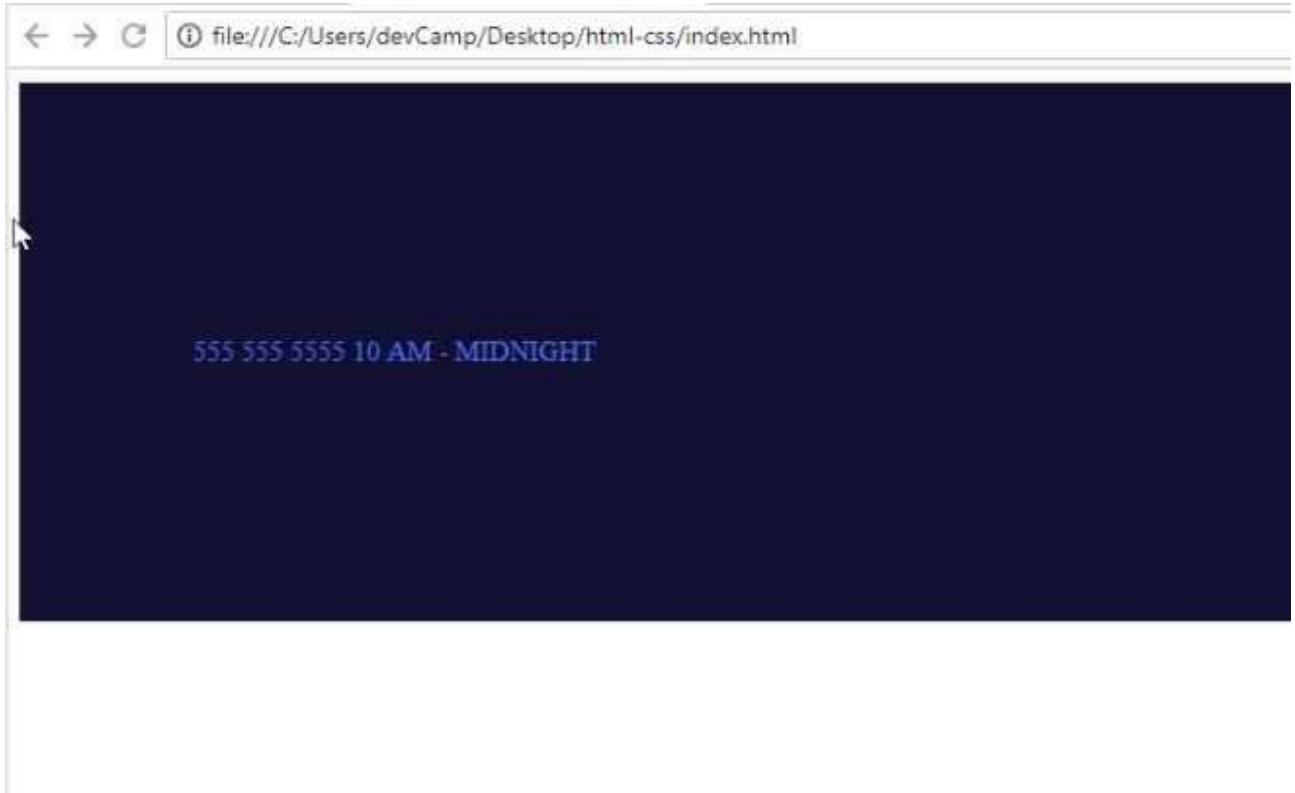
On the div tag below, add padding of **20px** to the top, **43px** to the right, and **50px** to the left.

```
<div class="style-this-div">
  <div>Item 1</div>
  <div>Item 2</div>
</div>
```

# 1.15 CSS Margin

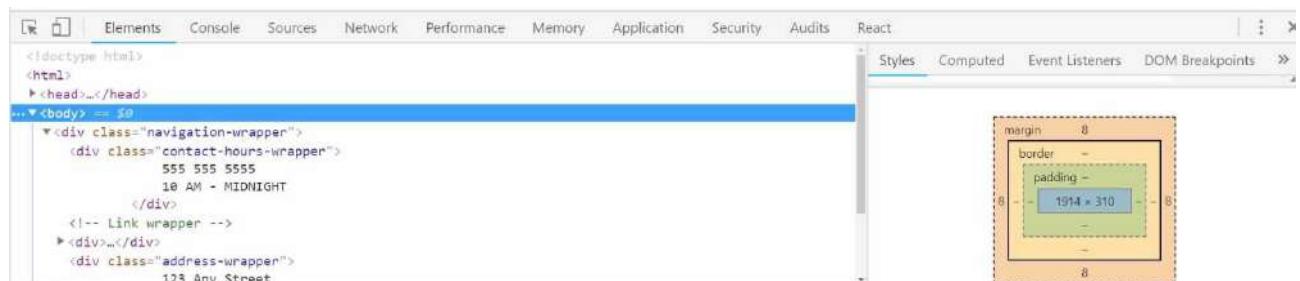
In the last guide, we talked about padding. I think it is a natural progression to now talk about margin.

We also have a little bug here that I want to fix, and it's a perfect case study for how we can use margin. Now, do you see right here? Might be a little bit hard to see on your screen, but you see how there is a white border going around the entire div that we have here?



That's not what we want. We want our entire website to be flush. The reason for that border is because, by default, HTML wraps a **margin** around the **body** tag.

You can see this for yourself if you click on **body** here, and then you come down to where you see our little box model diagram. Do you notice how it has a margin of 8 on the top, right, bottom and left parts of the body?



That means that if you put all over kinds of elements here, you had all kinds of images and text and everything, you'd always have these 8px of margin right here. Well, we can get rid of that by overriding the value.

Let's switch back to the code here, and I'm going to come up to the very top. I'm just going to add a comment, and we'll call these "Common Styles" because these are styles that are going to be used for every page of the application.

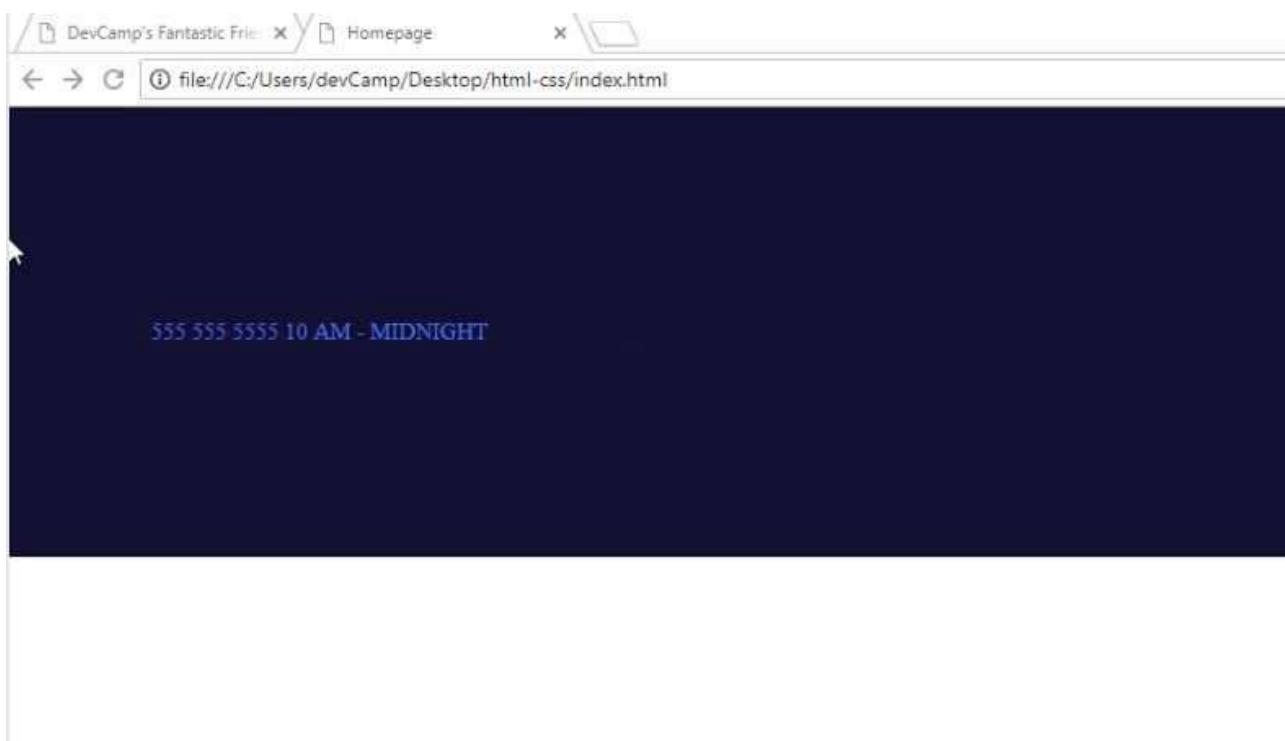
The way that you can select the body because it's a tag, you can just say `body`, you don't have to have a class or anything like that for it. You can say:

### styles.css

```
/* Common Styles */
body {
    margin: 0px;
}

/* Common nav Styles */
.navigation-wrapper {
    height: 190px;
    background-color: #11122b;
    color: #cfcfcf;
    /* Flexbox container */
    display: flex;
    justify-content: space-between;
    align-items: center;
    padding: 60px 100px;
}
```

Now if you hit save and come back, you can see that that has now overridden the value and now we don't have any margin here. That is the way it works.



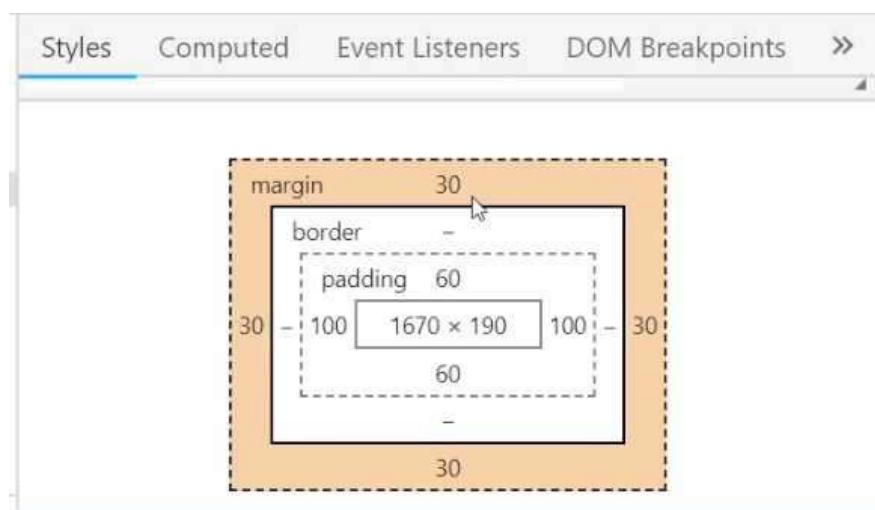
Now, if part of that is still a little confusing or fuzzy, what I recommend is playing around with both values and seeing how they change. After this video's done, just don't go on to the next one.

If it's not clear, update the padding values, and watch how everything changes right here in the diagram. You can look at the body and see that now we have no margin. There's no margin going around it, but if you click on navigation-wrapper, you can see how we have padding and no margin.

You can also even edit that, so you can click on `navigation-wrapper`. See what happens if you come down here and say, "okay, what happens if I have a margin of 30px". You can see how it is now wrapping 30px around every edge of that div.

Now that's not what we want, but the whole goal of this project and this course is for you to understand the way that HTML and CSS work. So, I definitely recommend for you to explore those.

This diagram right here is incredibly helpful. It gives you a really nice visual for being about to tell the difference between margin and padding.



The key one, being I think pretty clear right here, and that is, the margin is the space. It's the white space that goes around the outside of whatever element that you're working with. The padding is the white space inside of it. That is the main key difference.

Later on, we're going to talk about the border. The border is the little component that separates the margin and the padding, but we're not going to use a border on this element, so it's not needed.

After you're done, you can always either just hit refresh or click this little checkbox and it'll go away, and you'll be back to where we need to be.



## Coding Exercise

Apply a margin of 15px to all sides of the below div.

```
<div class="card">  
  Some content  
</div>
```

# 1.16 Custom Icons, logos, ...

In this lesson, we're going to walk through how we can import Font Awesome to start integrating icons into our website.

As you can see here from the finished version, you can see that we have this little phone icon here and we have a map icon. Those are, even though they look like images, they're actually just fonts. They're just characters that we can use.

That makes them very helpful because you're able to treat them like fonts, which means they scale up as big as you want, they don't get pixelated or distorted and they worked quite nicely for that. Now to bring Font Awesome in, just start typing in `font awesome` into Google, and then it'll take you right to [FontAwesome.com](https://fontawesome.com).

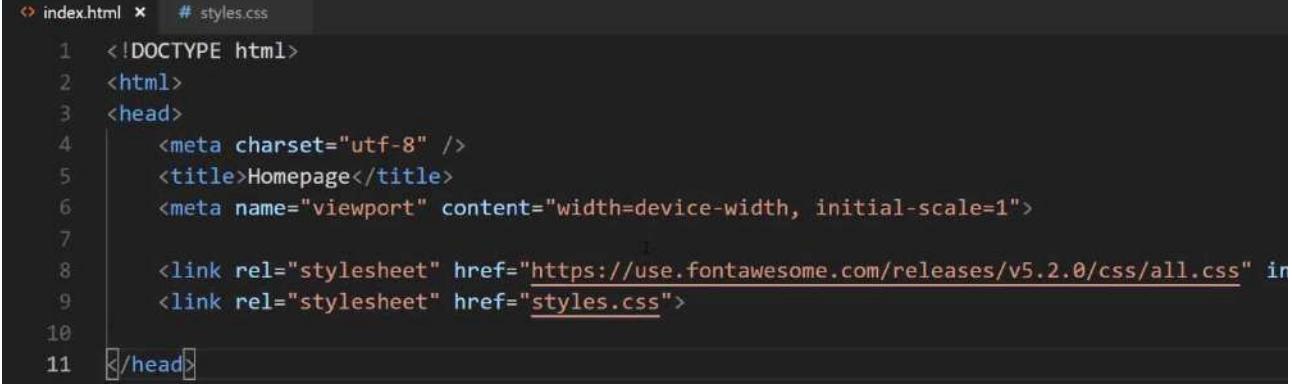
Depending on the version that you are using, then you may have some slight differences, some slight variations, but for the most part, you should be able to use exactly the same workflow that I'm showing right now.

If you click on `start using` here, this is going to give you whatever you need in order to import font awesome into the website. What we're going to be doing is we're actually going to be pulling in a CSS file provided by Font Awesome.

When we do that, we'll have access to their full free library. They also have a paid version, but we're just going to use the free one, so you can click on the little copy code icon here.

The screenshot shows the Font Awesome website at <https://fontawesome.com/how-to-use/on-the-web/setup/getting-started/using-web-fonts-with-css>. The page title is "Getting Started on the Web". A prominent blue button labeled "Web Fonts & CSS" is highlighted. Below it, a "Copied..." message with a clipboard icon appears over a "Copy" button. The URL `link rel="stylesheet" href="https://use.fontawesome.com/releases/v5.2.0/css/all.css" integrity="sha384-hWJfIwFxL6sNzntlh27bfXkr27Pmb` is shown in a text input field. At the bottom, there are two sections: "Call Font Awesome in Your Files" and "Add Icons Into Your UI".

Then if you switch back, and go to the **index.html**. Now, this is very important. You need to make sure that you are putting this above the styles. I'm going to paste that in and hit save.

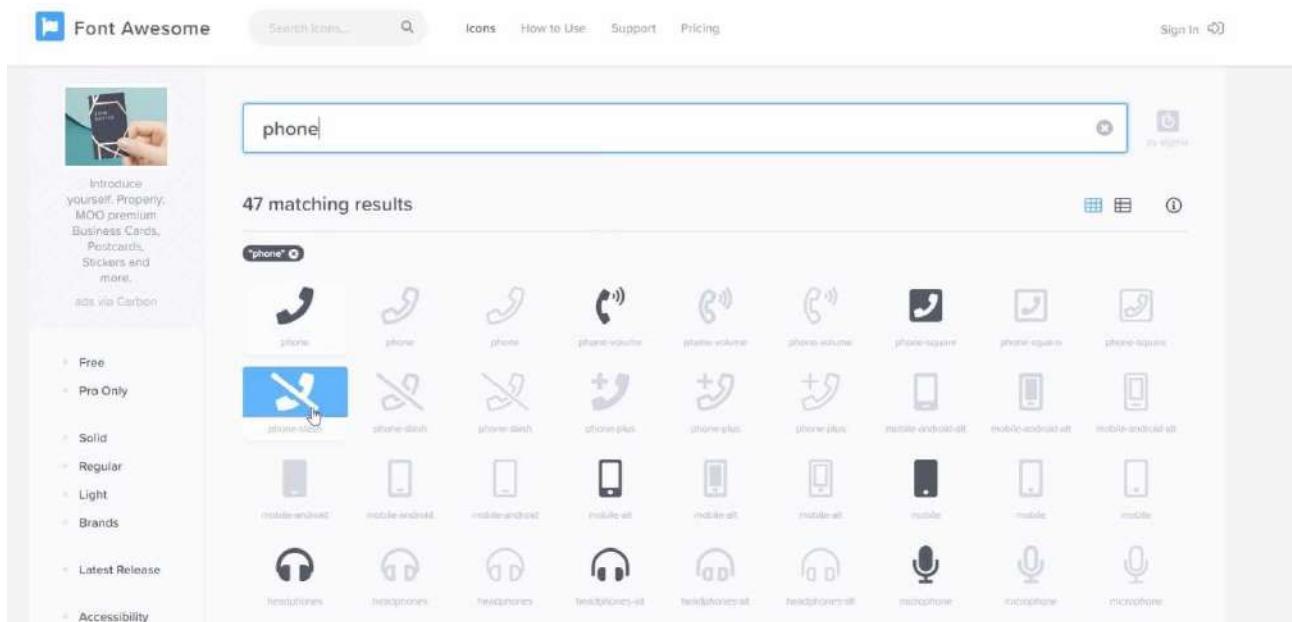


```
1 <!DOCTYPE html>
2 <html>
3 <head>
4     <meta charset="utf-8" />
5     <title>Homepage</title>
6     <meta name="viewport" content="width=device-width, initial-scale=1">
7
8     <link rel="stylesheet" href="https://use.fontawesome.com/releases/v5.2.0/css/all.css" in
9     <link rel="stylesheet" href="styles.css">
10
11 </head>
```

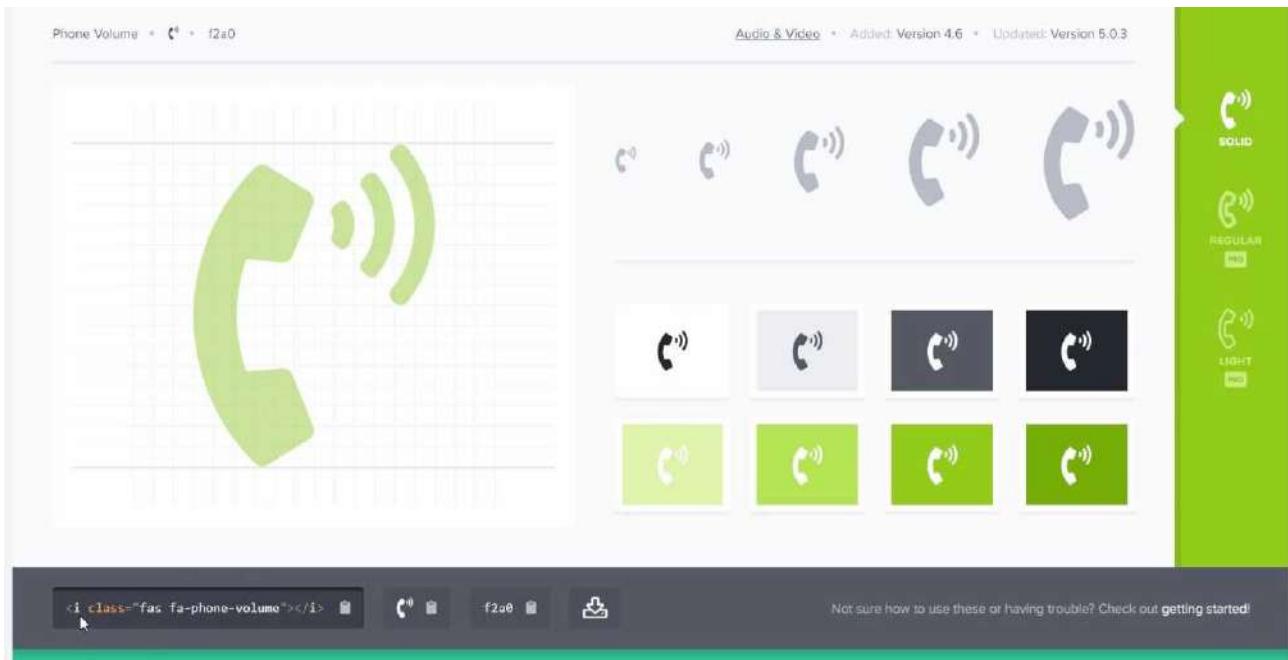
The reason for that is because we're not going to do it in this course but I don't want you to be confused in a different project. If you ever call any of the icons from the CSS file, because of the cascading nature of CSS, you need to make sure that you're importing at above whatever file you're going to use.

We're going to be calling it directly in the HTML though, so you could technically put it anywhere. Just as a best practice, you always want to make sure your fonts are above your style sheet calls. Now that we have that, let's see if we can actually get this working.

Now I'm going to go to Font Awesome and you can go search for icons. I'm just going to type in phone and you can see that we have all kinds of different icons. If they are darkened, that means that they're in the free version. If they're lightened and they're kind of grayed out, that means they are in the paid pro version.



Just click on this phone volume icon. This is going to show you all kinds of different variations and different things like that, which is helpful. The way that we're going to import it is it actually gives us the HTML code right here.



It gives us an icon tag which is represented with the `i` and then it has a specific class. The way that this works is pretty cool. It is connected directly into that CSS file that we imported.

That CSS file from font awesome that we just put in the `index.html` file, what this is going to do is when the webpage finds this icon with the class of `fas fa-phone-volume`, it's going to look to see if there is any spot where these classes are defined and it's going to find them.

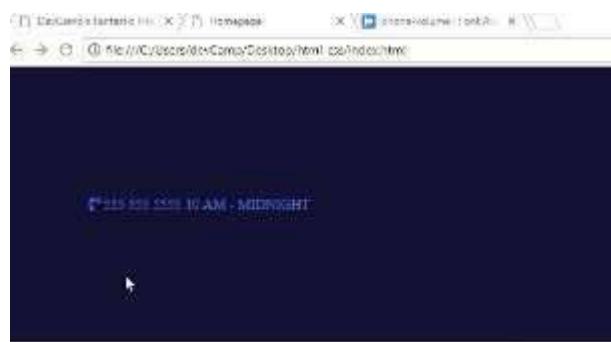
Then it's going to render this icon on the page. To get this working, just click on `copy code`.



Then go back to the homepage and open up the text editor. Now inside of `contact-hours-wrapper` here, let's go and let's place in that code. We're still going to do some work and we're going to add some more divs to organize this properly, but for right now, let's just put it right next to each other.

```
index.html # styles.css
5 <title>Homepage</title>
6 <meta name="viewport" content="width=device-width, initial-scale=1">
7
8 <link rel="stylesheet" href="https://use.fontawesome.com/releases/v5.2.0/css/all.css" in-
9 <link rel="stylesheet" href="styles.css">
10
11 </head>
12 <body>
13 <div class="navigation-wrapper">
14 <div class="contact-hours-wrapper">
15 <i class="fas fa-phone-volume"></i>
16 555 555 5555
17 10 AM - MIDNIGHT
18 </div>
```

Hit save and hit refresh over here. You can see we have our little icon there. Now we'll style it and we'll make it look good, but that gives us exactly what we want.

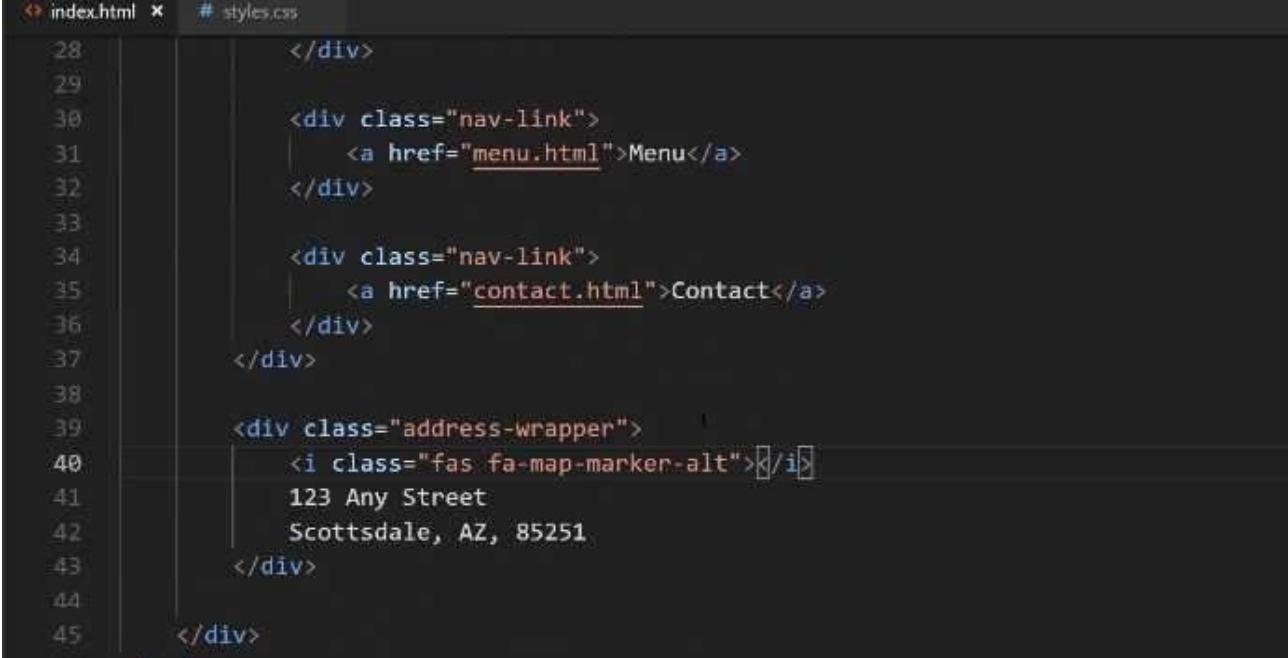


That means that we have one, we've imported Font Awesome properly and two, we have called it correctly. Now that we have that, there's one other icon, this little map icon, so let's come back over here and search for a map.

There you go, you can see it right here. It's called `map marker alt` or short for alternative. Let's copy this.

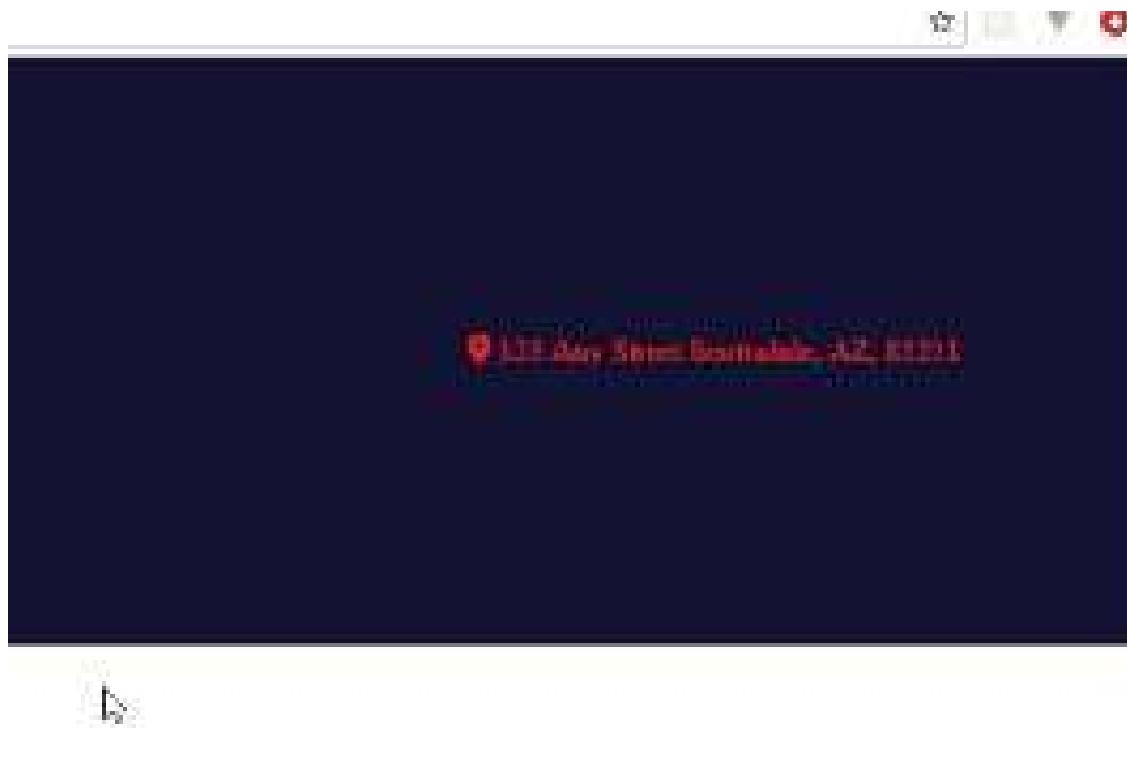
A screenshot of the Font Awesome website. The search bar at the top contains the word 'map'. Below the search bar, a message says '75 matching results'. A grid of icons is displayed, with one icon in the top row highlighted in blue. To the left of the grid, there are filters for 'Free', 'Pro Only', 'Solid', 'Regular', 'Light', and 'Brands'. At the bottom left, there is a link to 'Latest Release'.

We're going to do the exact same thing down here just to make sure that we can call it inside of our address-wrapper.



```
index.html # styles.css
28      </div>
29
30      <div class="nav-link">
31          <a href="menu.html">Menu</a>
32      </div>
33
34      <div class="nav-link">
35          <a href="contact.html">Contact</a>
36      </div>
37  </div>
38
39  <div class="address-wrapper">
40      <i class="fas fa-map-marker-alt"></i>
41      123 Any Street
42      Scottsdale, AZ, 85251
43  </div>
44
45 </div>
```

Hit save and hit refresh and there you go. We have our little font awesome icon for the map.



Now we're going to learn how we can move it and arrange it properly. We're going to use some other divs and CSS classes to do that, but for right now, we have now successfully brought in Font Awesome and now we can use their full library of icons.

# Course Update: New Import for Font Awesome

## Note

You will need to use the FontAwesome CDN for the latest version of icons. Place this script tag in your project instead of the link tag used in the ###video

```
<script src="https://kit.fontawesome.com/0f3793b933.js"
crossorigin="anonymous"></script>
```

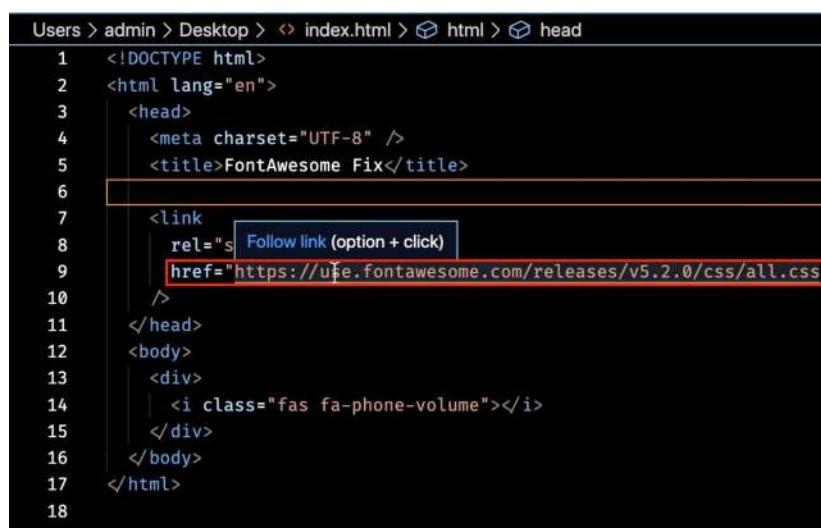
Also note you may need to apply styles for FA icons to show up in your project, as FA sometimes removes or changes icons so check their free icon ###names.

Alternatively, if you continue to have issues with using FA, it is recommended you create a free account with an individual provider link:

<https://docs.fontawesome.com/web/setup/get-started>

Whenever I hear that students are running into a potential issue, then I wanna push up and update just so that you're not spending a lot of time confused or having to go through a complex debugging type session. So in the very next episode, I'm gonna show you how to integrate Font Awesome into the application. But Font Awesome has changed slightly since I originally recorded this, and so there's a slightly different process that you need to implement in order to get it working, and so I'm gonna show that to you right now, and then you'll be able to implement it in the next video.

So right here, I'm not working on the project itself. I have just a dedicated html file, and this is gonna give you a little preview of how to import Font Awesome. So right here, this is what we're gonna do. We're gonna make a link call, and we're going to import the style sheet for Font Awesome, and that's gonna give us the ability to have all the icons, and this url right here had a few issues for certain students on certain systems,



The screenshot shows a code editor displaying an HTML file named index.html. The file contains the following code:

```
Users > admin > Desktop > index.html > html > head
1  <!DOCTYPE html>
2  <html lang="en">
3    <head>
4      <meta charset="UTF-8" />
5      <title>FontAwesome Fix</title>
6
7      <link
8        rel="stylesheet" Follow link (option + click)
9        href="https://use.fontawesome.com/releases/v5.2.0/css/all.css" />
10
11    </head>
12    <body>
13      <div>
14        <i class="fas fa-phone-volume"></i>
15      </div>
16    </body>
17  </html>
18
```

The line 9, which contains the href attribute of the link tag, is highlighted with a red box. A tooltip "Follow link (option + click)" is visible over the href attribute.

and this is what I, this is the exact url that I show in the next video, so I don't want you to use that one, or if you do use it and your icons don't show up, then you can implement this fix.

So if I switch to the browser here, you'll see that this is working for me, and it might still work on your machine. It's only on machines that have certain firewall settings and certain browser rules that this breaks for. And so this, the fix I'm gonna show you, is something that may not even be necessary, but I wanted to include it just in case. So in order to get this working, we simply need to change this url here.

So I'm gonna get rid of the entire link here, and in another window, I brought over the link that you need to use, so I'm just gonna paste this in. And this is a pretty long link, so do not worry about actually having to type all of these characters in, because that would not be fun.

```
<link  
    rel="s Follow link (option + click)  
    href="https://use.fontawesome.com/releases/v5.7.1/css/all.css"  
    integrity="sha384-fnmOCqbTlWILj8LyTjo7mOUStjSJKC4p0PQbqyi7RrhN7ud19RwhKkMHpvLbHG9Sr"  
    crossorigin="anonymous"  
/>
```

In the show notes, at the bottom of this guide, you can go and just copy this link and that's what you can use in the next video. You can see, I'm using a slightly different version of Font Awesome, but it's gonna be exactly the same in terms of how you make the calls and how the icons show up, and then it has a few flags here, and this is the reason why it's going to work, it has a cross origin flag, and then it has an integrity flag, and so certain browsers that have settings where the security's pretty high, the old url call was not working for those.

So this is one that will work, and so if you run into any issues, you can just use this code, and if you switch right back to the browser, you'll see that still working. So good luck in the next lesson.

## New Import Code

```
<script src="https://kit.fontawesome.com/0f3793b933.js"  
crossorigin="anonymous"></script>
```

## Updates

```
<i class="fa-solid fa-phone-volume"></i>  
<i class="fa-solid fa-location-dot"></i>
```



## Coding Exercise

Go to [FontAwesome.com](https://FontAwesome.com) and search for "Instagram". Then place the `i` tag with the Font Awesome class name for Instagram, below.

## 1.17 Child tag elements

In this guide, we're going to work more on our icon and hours of operation. The way that we'll do this is utilizing something called "Parent-Child relationships", which will give us greater control in our styles.

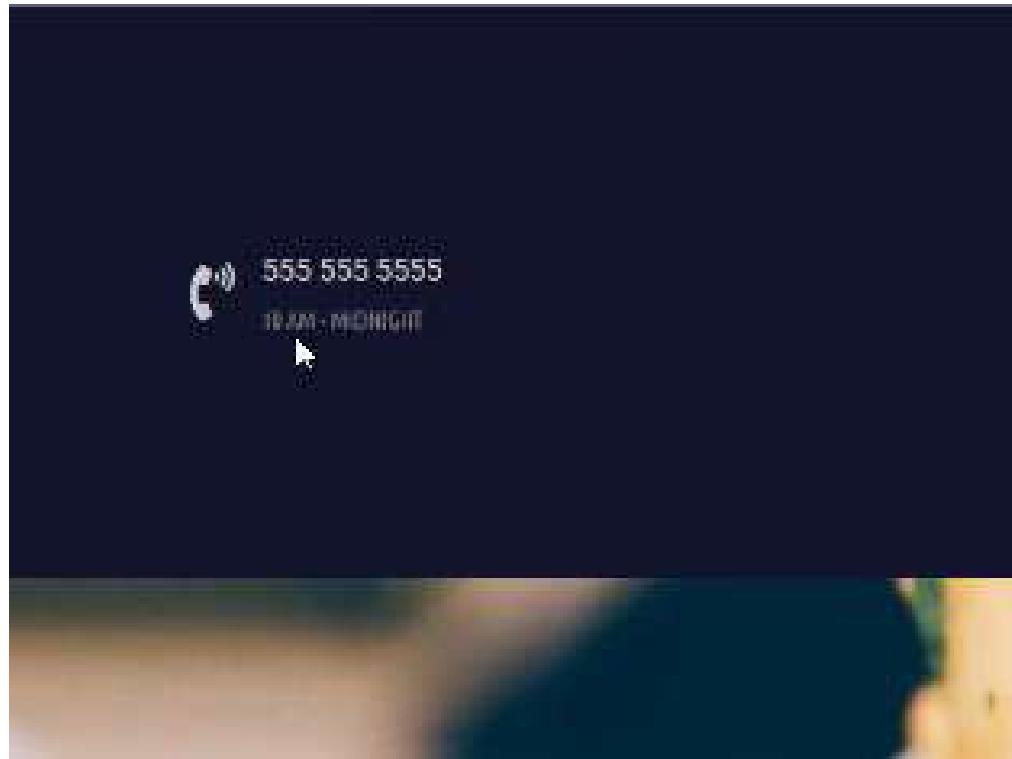
We're making great progress as we're building out this project. Now, if you are curious, and you're wondering if it takes this long to build out every single website, the answer is no.

Part of the reason why this is taking us awhile is that we're not just building out the website, we're also learning all of the key HTML and CSS fundamentals as we go. So, listening and going through those explanations definitely takes longer.

The more times you do this you're going to be able to eventually simply run through and build these types of projects very quickly, especially when you get a lot of practice with using tools like Flexbox.

Those are gonna be some of the bread and butter items that are gonna help you understand exactly how to place items on the page, how to align them, and how to style them however you need. So, with all that being said let's keep on building this out.

As you can see here on the left-hand side we have a little bit different set up than we're working with right now.



Right now we have all of our items right next to each other.



That is the default HTML behavior, but what I'd like to do now is I'd like to start customizing this.

The very first thing we're gonna do is we're going to create a wrapper div for all of this content. So, just like we have a wrapper Flexbox container for the entire nav bar, we're now going to create a component that is going to wrap up our little phone icon, the phone number, and then the hours.

So, let's get started with that. Open up `index.html` and in this contact hours wrapper, let's create a wrapper for that, and I'm going to call it the left column because that's what it is. It's gonna be that left column.

### **index.html**

```
<div class="navigation-wrapper">  
  <div class="left-column">  
    <i class="fas fa-phone-volume"></i>  
  
    <div class="contact-hours-wrapper">  
      555 555 5555  
      10 AM - MIDNIGHT  
    </div>  
  </div>
```

let's just see what this looks like now.



Now you can see that it's changed the color because this is no longer highlighted and it's no longer in that contact hours wrappers, so it's not blue anymore. This is a little bit better except right now it's using the default HTML behavior, and instead, I'd like to use Flexbox.

This gets to a very important topic, which is that you can have flex items that also are flex containers because remember, this whole nav bar is a flex container.

This contact element here is a flex item, but we can make it a flex container, and then all of the elements inside can be aligned however we need to. Let's go and let's do that. I'm also gonna add another wrapper class for this icon.

### index.html

```
<div class="navigation-wrapper">

  <div class="left-column">
    <div class="icon">
      <i class="fas fa-phone-volume"></i>
    </div>

    <div class="contact-hours-wrapper">
      555 555 5555
      10 AM - MIDNIGHT
    </div>
  </div>
```

I had one of my early mentors who helped me understand some best practices with HTML and CSS. What he told me was that he has never had a situation where he had too many divs, but he had plenty of situations where he didn't have enough.

Remember that when you add divs you're giving yourself the ability to have more granular control over the items. You can align them better. You can style them with more specificity. That's gonna give you just a lot more control as you build out your interfaces.

So, now that we have this left column let's actually turn this into a flex container. Let's go into `styles.css`. This is still under common nav styles, and also just to make sure that we are not confusing ourselves I'm gonna get rid of these placeholder color items.

### styles.css

```
/* Common nav styles */
.left-column {
  display: flex;
  align-items: center;
}
```



Now if we hit save and hit refresh that is getting us a little bit closer.

That looks really good. The next step is going to be to give some styles to this icon.

## styles.css

```
/* Common nav styles */  
.icon {  
    margin-right: 15px;  
}
```

Now, this is going to take us into a very important topic which is Parent-Child Relationship Selectors. Now if that sounds very foreign and confusing do not worry. We're gonna go over a lot throughout this course.

We want to apply styles not only to the icon, we also want to apply styles to that little 'i' tag inside of it. What we can do with CSS is grab specific tags, Like so.

## styles.css

```
/* Common nav styles */  
.icon {  
    margin-right: 15px;  
}  
  
.icon i {  
    font-size: 2em;  
}
```

Now, do not worry. We have not talked about this yet, so that's what we're gonna discuss this little EM. So far everything we have talked about has been in pixels. Pixels are what you use when you want to have a hard coded unit of measure.

We talked about this before, and we said pixels are kind of like inches or centimeters. They are a hard coded, definite type unit of measure. A EM and its sibling REM, what these are is they are more like percentages. So, if I said, `1em` nothing would change here.

That `1em` is going to give you just essentially 100%. What `2em` is gonna do is it's gonna give you whatever double the normal size is there. So, we're gonna be able to control this font size, and this is gonna increase the size of the icon.

Now that we have that, our phone number and the hours are sitting right on top of each other, and that's not what we really want.

In the next guide what we're gonna do is we are going to learn how to use a very powerful tool called CSS Grid to be able to build a little grid here that is gonna help us align these items.



## Coding Exercise

Using CSS Parent-Child Relationship Selectors, make the nested `p` tag's text red.

```
<div class="elephant">  
    <p>Make this text red!</p>  
</div>
```

# 1.18 CSS grid

Right now, we have our nav bar that's using Flexbox, and we have a little component for a font awesome icon, a phone number, and then our hours, and this is another Flex container.

Now, CSS Grid is a tool that has some similarities to Flexbox, but it gives us some different utility, and that's what we are going to utilize here. Specifically, we're going to use it so that we can have the phone number on one line, and then our different hours right below that.

Let's go and let's see what that syntax looks like. In our `styles.css` you can see we have our icon, we have this left column, and we're going to use the left column, and we're going to learn, also, about how we can give some more specific styles.

You can see our contact hours wrapper is a child of the left column div. Here's what we can do

`styles.css`

```
.left-column > .contact-hours-wrapper {  
}
```

What this is telling CSS is "I want you to find the left-column, and then from here, I want you to find the child class of contact-hours-wrapper. We're going to refactor and dive a little bit more into this in one of the next guides.

You're going to see how we can do this for all of our classes. We'll also talk about the importance of it. But for right now, just know that this is very similar to what we did on line 27, where we used the icon class and then we grabbed all of the I-tags inside of it.

```
27   .icon i {  
28     font-size: 2em;  
29 }
```

Let's add some styles.

`styles.css`

```
.left-column > .contact-hours-wrapper {  
  display: grid;  
  grid-template-columns: 1fr;  
}
```

Now, this might be a little mind-blowing if you've never seen it before, so we'll walk through it. What we're saying here is, Grid, as you may have guessed, it works on a grid-type system.

If you were to look at the website, what that means is you have the ability to set up a grid, however you want, here on the page. You could have 12 columns, you could have one column, and you could have it separated and organized however you want.

Now, I'm using Flexbox for these other elements, and so we're going to simply be using Grid right now for this element, but what I'd like to do is I would like to stack the phone number on top of the hours.

Now we're just going to have two rows, and it's only going to have one column.

Right now, we're saying we want to have `1fr grid-template-columns`. What does `fr` mean? We haven't seen that before. What `fr` means is it stands for one fractional unit. This is pretty much the same as saying, "I want you to take up all of the space available, and I want you to just go with the default."

Now, we're going to talk about a few options later on how you could change that, and we'll also have some other examples.

But for right now, know that if you use `grid-template-columns`, and you just say, "1fr," this means that the first child element is going to take up all of the first row, and then any elements underneath that are going to take up all of their rows.



Let's save this and see what this does.

If we hit refresh nothing is changing, so let's debug this. Let's go to our left column here, and then we have our icon. We have our contact hours wrapper, and we have `grid-template-columns`. Oh, and I know what our issue is.

Earlier, I said how you can never have too many divs? Well, I was right, and you could see what the bug is here.

```
<!DOCTYPE HTML>
<html>
  <head>...</head>
  <body>
    <div class="navigation-wrapper">
      ...
      <div class="left-column"> == $0
        <div class="icon">...</div>
        <div class="contact-hours-wrapper">
          555 555 5555
          10 AM - MIDNIGHT
        </div>
      </div>
    </div>
```

This works the same way the Flexbox works. Remember how Flexbox works where it does not have deep nesting. It simply looks at all of the elements inside. So, in the case of our contact hours wrapper, even though we have two lines, it still only sees one element.

What we need to do is to create some wrapper divs here.

## index.html

```
<div class="contact-hours-wrapper">
  <div class="phone">
    555 555 5555
  </div>

  <div class="hours">
    10 AM - MIDNIGHT
  </div>
</div>
```

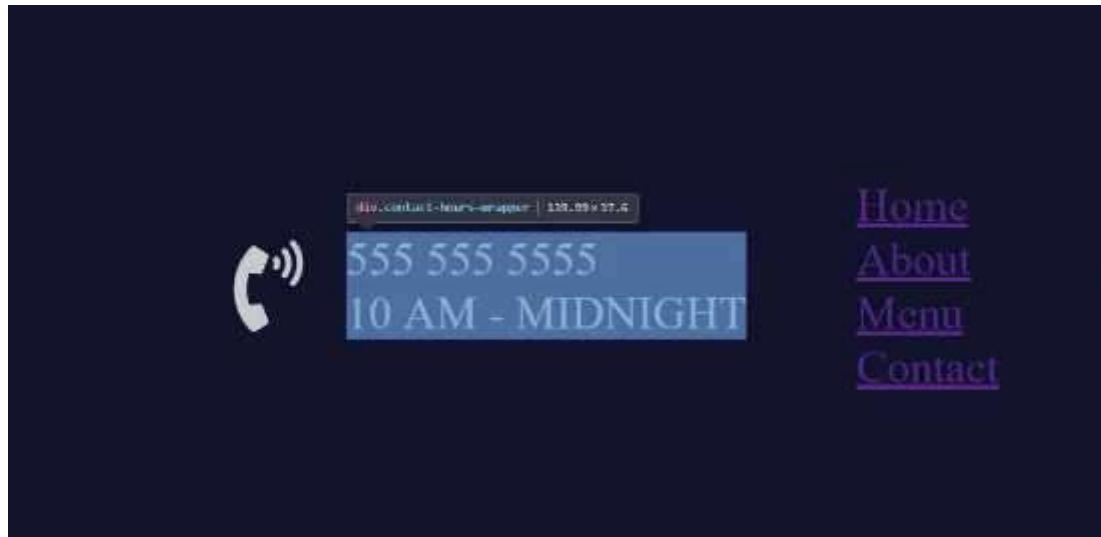
Now, I did not do that on purpose, but I am kind of glad that it happened, because you're able to see the issue on why it's so important to have these wrapper divs and to wrap your elements up, because it gives you the ability to treat them as separate components on the page.



Now if we hit Refresh, there we go.

Now this is doing exactly what we're wanting.

Now, I promised that we were going to have a visual, so let's take a look at this. If you are using Chrome or Firefox and you click the Inspect tool, and then come over here and click on contact hours wrapper, do you see how those little-dotted lines there that are separating our two rows?



That is a very helpful little tool that the browsers give us. This is invisible to regular users unless they open up the dev tools, but what this does is it is only shown whenever you're using CSS Grid, and it also gives you the ability to visualize where those columns are.

Let's play around with this a little bit.

```
<html>
  <head>...</head>
  <body>
    <div class="navigation-wrapper">
      <div class="left-column">
        <div class="icon">...</div>
        <div class="contact-hours-wrapper"> == $0
          <div class="phone">555 555 5555</div>
          <div class="hours">10 AM - MIDNIGHT</div>
        </div>
      </div>
      <div class="right-column">
        <a href="#">Home
        <a href="#">About
        <a href="#">Menu
        <a href="#">Contact
      </div>
    </div>
  </body>
</html>
```

Styles tab content:

```
.left-column > .contact-hours-wrapper {  
  display: grid;  
  grid-template-columns: 1fr 2fr;  
}
```

When you say `2fr` then that tells the grid that we don't want one column, we want two columns.

The first column we want to take up just the normal amount of space. The second column we want to take up double whatever this one is, which is the reason why the hours here are much wider than the phone number.

Now, if you reverse this, you can see how this is now taking up double the amount of space.

```
<html>
  <head>...</head>
  <body>
    <div class="navigation-wrapper">
      <div class="left-column">
        <div class="icon">...</div>
        <div class="contact-hours-wrapper"> == $0
          <div class="phone">555 555 5555</div>
          <div class="hours">10 AM - MIDNIGHT</div>
        </div>
      </div>
      <div class="right-column">
        <a href="#">Home
        <a href="#">About
        <a href="#">Menu
        <a href="#">Contact
      </div>
    </div>
  </body>
</html>
```

Styles tab content:

```
.left-column > .contact-hours-wrapper {  
  display: grid;  
  grid-template-columns: 2fr 1fr;  
}
```

This is something that's really helpful. We're going to be talking quite a bit more about Grid as we go throughout the course. This is just the introduction.

But you can see how powerful this is. This gives you the ability to create these kinds of grids anywhere in the application that you want.

It's technically possible, to lay out your entire application on a grid, so everything from the nav bar down to all the elements underneath it, all of that would be one master grid. Then you could integrate Flexbox and Grid inside of it. That's definitely a possibility.

Let's go back to where we were before, and there's one more element I want to show you because it's quite helpful, and that is called Grid Gap.

We've implemented Flexbox, and now we've implemented Grid. They're very similar, your first question may be, "Why would I want to use Flexbox versus Grid?"

My answer to that is usually if you know how many elements that you are going to have for just a guaranteed number, then I like to use Grid, where there is a little bit more hard coding into it. So, if you know exactly how many columns you're going to have and that kind of thing, then Grid works really well.

If you want to have more of a variable number of items, kind of like you saw how we were able, earlier on, how we were able to add as many addresses here as we want, and we were able to make the navigation bar very flexible simply by using Flexbox, then that's what I like to use.

But what my top advice really is to work with both of them a lot, and then use them when you feel like it's the most appropriate. There's not really a right or wrong answer, so don't feel afraid thinking that you need to know when it's right to use one versus the other.

I have seen some fantastic applications where the developer might have done the exact opposite. They would have used Flex when I used Grid, and vice versa, and it still works. So, I'd say, use whatever gives you the best results, and whatever you understand the best.

As we go through the rest of this application, we're going to be using both quite a bit. That is what we have there.

Now, the other thing I really like about Grid that isn't really available in Flexbox is, do you notice how these two items are kind of close together? If you look the end goal, they have some space between them.



Well, Grid has this really cool little tool here, and it is called grid-gap. I can say, "grid-gap," right here, and then I can pass it a value, like 10 pixels.

Now do you see how we have this space here?



We didn't have to mess with padding, we didn't have to mess with margin, anything like that because those can throw your items out of alignment. You notice how we had to do that for this icon, and that was perfectly fine because we were just moving this and giving a little space in between.

But here, whenever I'm working with vertical alignment, grid-gap works very nicely, because it just does all that for you. It manages the margin, the padding, and it looks really nice.

Now, if you click on here, now you can see that our columns actually have some space in between them.



They're both exactly the same height and width as they were before, but now what it does is it just automatically puts some space between those, and it's still aligned perfectly.

If you were to try to say, "Add some margin-top here," or "some margin-bottom here," you might run into a situation where they get out of alignment, but grid-gap allows you to have that space automatically.

I'm just going to copy this, and this is a pattern I follow quite a bit, is whenever I don't know the exact styles I want to implement, I will use the developer tools here in Chrome, and build those styles, get them perfect to exactly where I want them.

Then from there, I can just go and I can add them directly into the code. So, I can just paste that in,

### styles.css

```
.left-column > .contact-hours-wrapper {  
  display: grid;  
  grid-template-columns: 1fr;  
  grid-gap: 10px;  
}
```

This is starting to look a lot more like what we have in our design. In the next guide, we'll go and we'll style this up, and we'll also start importing our custom fonts.



## Coding Exercise

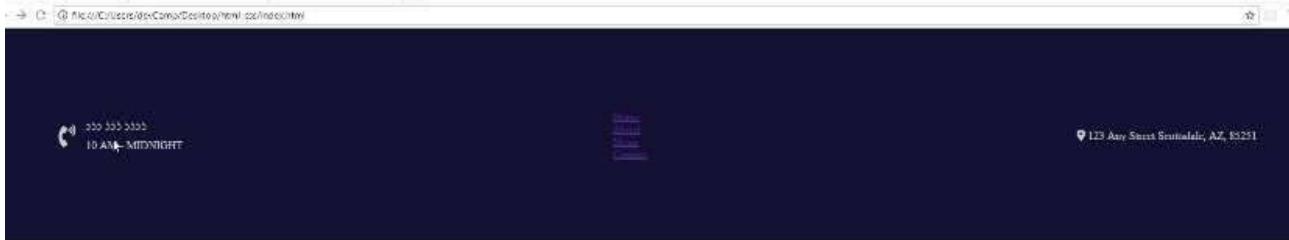
Apply grid to the parent container. Then apply 3 template columns of the same size, each being 1fr, and a grid gap of 23px.

```
<div class="grid-container">  
  <div>Use</div>  
  <div>Some</div>  
  <div>Grid</div>  
</div>
```

# 1.19 Importing Custom Fonts

Our navbar is coming along nicely. Along the way, you are learning a lot about HTML and CSS. The next thing that we're going to learn is how to integrate custom fonts.

Fonts are absolutely critical to giving a really nice look and feel to your application. You can even see the difference, look at right here. We have pretty much all of the same components here, from a content perspective.



Look at the difference between this application, and what we have right here just on the navbar. The fonts make a huge difference, and so now we're going to learn how we can install custom fonts.



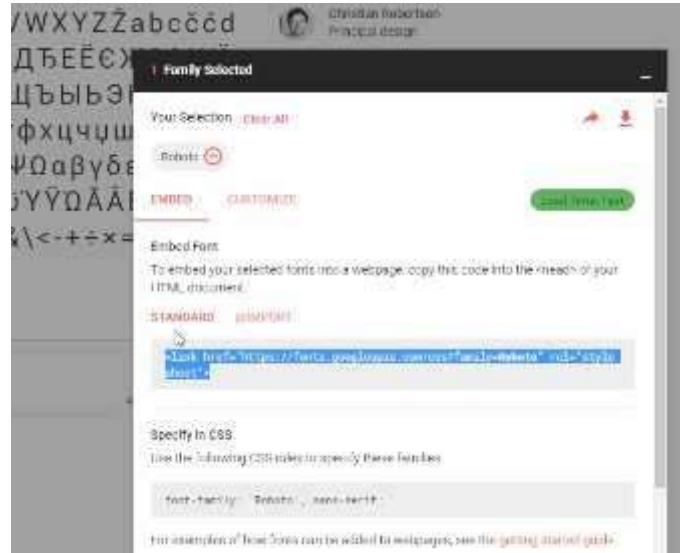
The first thing you're going to do is go to [fonts.google.com](https://fonts.google.com). This is a great source for free custom fonts that'll look really good. They also have some really nice ways of showing how you can integrate it directly into your project.

There are going to be two fonts that we're going to pull in right now. The first is called **Roboto**, and you can either search for it right here, in the top right-hand side. You also have access to see it right here.

I'm going to search for it, just because I'm assuming that most of the time it's not going to be something that's readily available. You can see it is right here, and there's a couple things you can do.

When you're researching fonts, you can click on the name, and then it has all kinds of helpful details. It shows all the different versions, it shows usage, it shows some nice combinations, different things like that.

Now, you can also click on `select this font`, and it says you have one family selected. It gives you instructions on how to integrate it into your application. I'm just going to grab this standard link here.



Let's switch back to the code, and let's put this at the very top. You can just put it right above where it says `font awesome`. That's going to be the first one. Now, we also want to install a second font.

```
index.html • #.styles.css
1  <!DOCTYPE html>
2  <html>
3  <head>
4      <meta charset="utf-8" />
5      <title>Homepage</title>
6      <meta name="viewport" content="width=device-width, initial-scale=1">
7
8      <link href="https://fonts.googleapis.com/css?family=Roboto" rel="stylesheet">
9
10     <link rel="stylesheet" href="https://use.fontawesome.com/releases/v5.2.0/css/all.css" in
11     <link rel="stylesheet" href="styles.css">
```

Oh, one other quick note, we need to bring in this `font-family` for the entire project. This is going to be the `master font-family`.

If you want to use something else, that's perfectly fine. Have fun, and search through the Google Font library and use whatever you want, because the process is going to be identical with what we're doing right here.

In addition to importing it, we also have to use it. So, I'm going to switch back, hit save, and come to `styles.css`. Right in the body, because this is going to be the master font for the entire application. Hit save, and this is going to be imported.

## styles.css

```
body {  
    margin: 0px;  
    font-family: 'Roboto', sans-serif;  
}
```

Now, the way that this works is we have a normal style declaration here, where it says `font-family`, and then we give the name that we want to use. The other items and you can have more than one extra one, but what this does is like you can see, if you hover over it. It says it specifies a prioritized list.



That means that the project is going to try to load `Roboto`, and this is the imported version. Imagine that you are, say, in an airplane and you're working on this site, and you do not have access to pull this down. Because in order to import this, this has to call a website and ask to bring it down.

This is not going to be an issue doing that for regular users, because the only way they can access your site is if they're on the Internet. That's not going to be an issue, assuming Google doesn't go down, which it is pretty reliable.

What this does, though, is it gives you the ability to have backups. Every system in the world has these fonts that are built directly into the system.

What we're saying here is if for some reason `Roboto` is not available, such as you working on the site when you're on the airplane, and you don't have `Roboto` on your system, then just use the default `sans-serif` font.

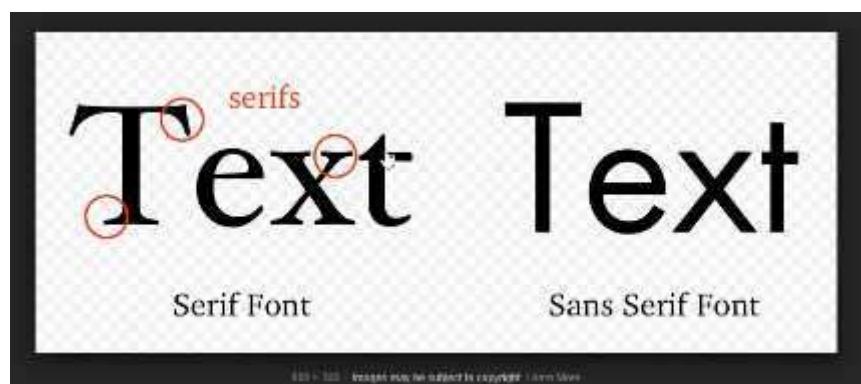
What `sans-serif` means is there are a couple different kinds of fonts out there. You're going to, as far as generic fonts, you're either going to have `sans-serif` fonts, or you're going to have `serif` fonts.

The difference is something I can show you. Let's see. I'll actually just Google it because it'll be a little bit easier for you to see. I'll say `serif font`. You can see right here, just by clicking on some of these images, what the key differences are.

If it's a serif font, it means it has these little edges on the end. This is just kind of a generic serif font, here.



Whereas a sans-serif font, if you click right here, doesn't. This is helpful, you can see them right next to each other. Serif has all these little curves and edges and it's a little bit more kind of old-school. Old-school meaning like a couple hundred years ago in regards to how the original typewriters used to be.



Right here, this is just more of a machine-based kind of font. This doesn't have any kind of curves or edges around the beginning or end. This is a sans-serif font. This is what we're going to use for the majority of the application.

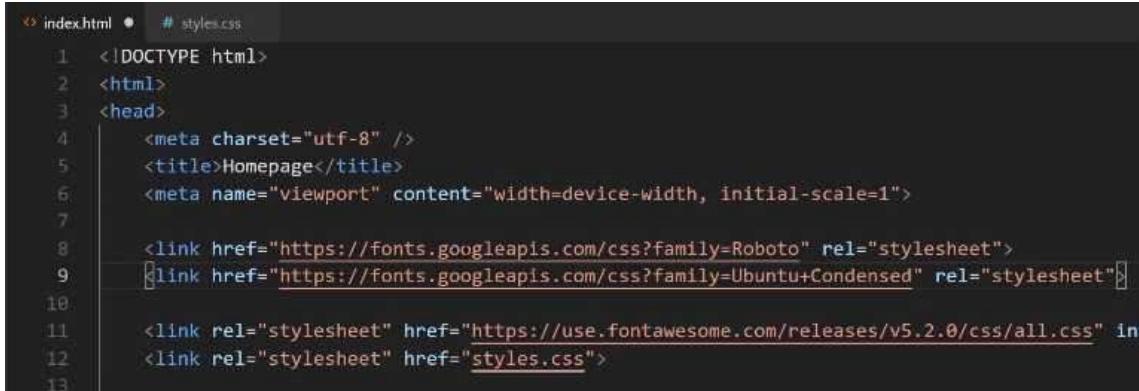
That is all that that means. Let's put it back, so we're using sans-serif, hit save, and now we can actually go and look at the website, even before we go and import the other one. If I hit refresh now, you can see we have a different font. This is now using Roboto.

The screenshot shows a browser developer tools window with the 'Elements' tab selected. The page's HTML structure is visible, with the body tag selected. In the 'Computed' styles panel, the 'font-family' rule is shown as 'font-family: 'Roboto', sans-serif;'. The browser window shows a dark blue header with white text, including a phone number and address.

You can see right there in the body we now have `font-family`: Roboto, and this is looking better. It's not perfect yet, but it is looking better, so let's bring in the second font here. I am going to say clear all, and then let's search for it.

I'm going to use one called Ubuntu. I'm going to use Ubuntu Condensed. Right from here, you can either click on it, or you can just click the plus sign right from here, and it's going to have the exact same instructions.

I'm going to import this into the HTML page, so let's do it right here.



The screenshot shows a code editor with two tabs: 'index.html' and '# styles.css'. The 'index.html' tab contains the following code:`1 <!DOCTYPE html>
2 <html>
3 <head>
4 <meta charset="utf-8" />
5 <title>Homepage</title>
6 <meta name="viewport" content="width=device-width, initial-scale=1">
7
8 <link href="https://fonts.googleapis.com/css?family=Roboto" rel="stylesheet">
9 <link href="https://fonts.googleapis.com/css?family=Ubuntu+Condensed" rel="stylesheet">
10
11 <link rel="stylesheet" href="https://use.fontawesome.com/releases/v5.2.0/css/all.css" int...
12 <link rel="stylesheet" href="styles.css">`The '# styles.css' tab is currently active but contains no visible code.

Then we want to call it, but we only want to use it on a few elements. Roboto's going to be the main font we use throughout the entire project, but we want to use this on a few custom elements.

If you come back and look at the final version, a few spots that you can have right in the nav that are going to use it are going to be the hours, the little nav links, and then the address right here. Let's go and apply those right now.

For our phone, let's come down and see where we put the phone number. It looks like we haven't added a custom one yet, so let's do that now. I'm going to say, inside of the `contact-hours-wrapper`, we are going to have a phone class. There we want the `font-family` to be the Ubuntu Condensed.

### styles.css

```
.contact-hours-wrapper > .phone {
  font-family: 'Ubuntu Condensed', sans-serif;
}
```



Let's just make sure that our selector is working. If I come back here, hit refresh, you can see that that phone number's working. Except, it looks like I actually reversed it. We want the phone number to be the Roboto, I want the hours to be different.

Let's see what that hours selector is, I'm pretty sure I just used `hours`, and yes, that's correct. Instead of `phone`, just change this right here to say `hours`.

### styles.css

```
.contact-hours-wrapper > .hours {  
    font-family: 'Ubuntu Condensed', sans-serif;  
}
```

Hit save, hit refresh, and there you go. Later on, we'll add some polishing touches so that we can have some of that different coloring and also the smaller font size, but we're not going to worry about it now.

Now, let's go and let's apply this to each one of these links. We're going to have a whole section dedicated just to styling these links, but for right now, let's just apply the font to them.

We know what we want to apply if you just click on inspect. We have this nav-link, so let's go and let's go and style the nav-link. We can say `nav-link a` because we want the `a` tag inside of that nav-link, and there we want to use that font-family.

### styles.css

```
.nav-link a {  
    font-family: 'Ubuntu Condensed', sans-serif;  
}
```

Now, if we come and hit refresh, it's a little bit hard to see, so let's add a different color here, as well. Just because I know, especially if you're watching on video, it's going to be hard to see.

### styles.css

```
.nav-link a {  
    font-family: 'Ubuntu Condensed', sans-serif;  
    color: white;  
}
```

We'll change it temporarily right now, and we'll most likely update it so we're not using a pure white for the links. Let's just do it so we can see it, and there you go. You can see that it is using that different font.



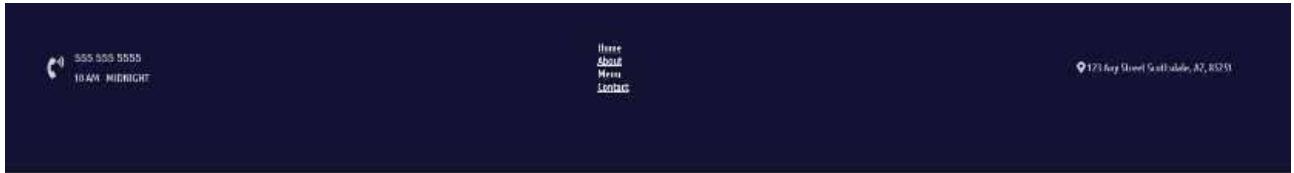
Lastly, let's go and do the same thing for that address. We have that `address-wrapper`, and let's come down here. We have the `address-wrapper`, we're going to have to apply `CSS grid` just like we did before.

For right now we'll just have the address-wrapper use this font. So here, we can just copy this whole line and paste that in.

### styles.css

```
.address-wrapper {  
    font-family: 'Ubuntu Condensed', sans-serif;  
}
```

Hit refresh, and there you go. We now have all of our custom fonts that we're going to want to use in the navbar, and we're going to be using that throughout the rest of the project, as well.



Great job if you went through that. You now know how to import and then use custom fonts in a website.

GoogleFonts: <https://fonts.googleapis.com>



## Coding Exercise

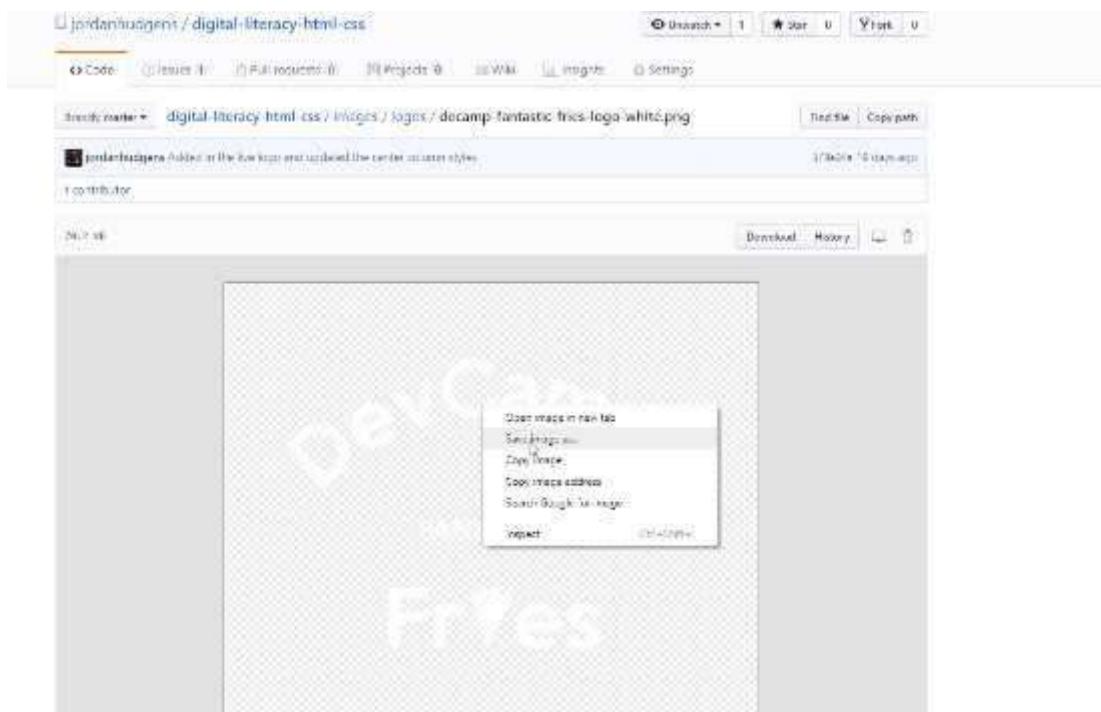
Go to GoogleFonts go to the categories dropdown selector and select monospace. Apply the CSS font family to the below span tag.

```
<span class="greeting">Hello World!</span>
```

# 1.20 HTML images

We're going to see how we can import the image, how we can select it using CSS, and then how we can style it and customize its size. If you want to, you can use your own logo or you can use one of the demo ones that I have here. I'll include a link in the show notes for you.

The very first thing you can do is open this up and you can see we have a dark version and then a white version. For this specific background, we really need to use the white version. So, right-click on this and then click **Save Image As**.



You can keep the name exactly the same and then place it inside of your project. Now, you can just hit **Save** right now, but we'll customize exactly where it's at so that we can see the path and so we can organize it properly.

If we switch back here and then go to the code, I'm going to place this above all the other nav-links here and let's just call it `.banner-image`. This is going to be a wrapper div. Then I want to use an image tag.

If you're using Visual Studio Code, you can type `img`, hit tab, and this is going to give you the values that you want. Now because we place this at the root of our project, you can see that we have this `decamp-fantastic-fries-logo-white` right here.

We could just grab this value. Let me actually copy that name, so that I'm not typing it out verbatim. Then you can place that inside of this `src` tag.

Now this `src` tag. It stands for **Source**, which means that we're telling the `image` tag that the source of this image is this path. Then we'll play around with how we can place this image inside a different part of the project.

### index.html

```
<div class="banner-image">
  
</div>
```

Now we also have this `alt` tag. Here I'm going to say the **Logo**. Now technically, you could say anything that you wanted right here.

### index.html

```
<div class="banner-image">
  
</div>
```

The rationale for the `alt` tag is, there are a couple of reasons. One is if someone has disabled images in their browser, then the alt tag will show up instead of the image. The most common reason for using the alt tag is for accessibility reasons.

So if someone, say, someone who's blind, is going through your site, the way it works is there are systems out there that will read things like the alt tags in your content. If you leave the alt tag blank, then they're not going to know that a logo was there. That's part of the reason.

Google also reads through this for **search engine optimization** reasons. It's always good to put some kind of value here that describes what the logo is or that a logo is there. That is going to give us our basic import.

Let's go back and hit refresh, and wow, you can see, that it is imported, but that's definitely not what we're looking for. That's fine, that's what we expected.



By default, the way that HTML works is it brings in the image and it does not do anything to it. It doesn't try to make it fit or anything like that. That is our job. Now let's see how we can select this image.

We know that we have a class here called `banner-image`, and there are a couple of different ways that we could customize it. I could come in the tag itself and just say that I want to provide a hard-coded width inline.

Here I could say `width`, and for this kind of size, the value that I saw that looked the best was something like `216px`. Then let's give it a height, and the height here, you can combine pixels and percentages. So, for the height here I could say just `100%`.

### index.html

```
<div class="banner-image">
  
</div>
```

What this means is that we are controlling the width, we're saying how wide it should be. Then we're telling the height to just be automatic. I'll hit save, and now if I hit refresh, you can see that works. That looks really good.



That's perfectly fine. I usually don't do it this exact way. The approach I usually take is to apply an actual CSS class to it. So, if I have my `banner-image` here, what I can do is come down to the bottom of the `styles.css` file, and say `.banner-image`.

Then I want to select the image tag inside of that. Now I can apply the exact same rules. So here, I can say:

### styles.css

```
.banner-image img {
  width: 216px;
  height: 100%;
}
```

Hit save, go back, and hit refresh. You can see that is working properly. That's just my own personal preference. Part of the reason is because I don't really like my images to get really messy.

The more code that you put inside of this tag, the longer it's going to be for you to find it when you need to fix it, and it just starts to look a little bit cluttered. I'd much rather put it inside of a style tag. That is how you can import and then customize an image using HTML.

There's only one more thing that I want to do before we end this guide and that is I want to place this inside of a little bit more logical place. Right now, you can see that I have the image at the root of the project, but this doesn't make a lot of sense because.

If I start bringing in dozens or hundreds of images, this is going to start to get really messy. Instead, what I'm going to do is I'm going to create a new tag or a new folder here. I'm just going to call it **images**, and then inside of images, I want to pass in another folder.

I can right-click on it and click **new folder**. Here I'm going to say **logos**, and then what I can do is I can just click and then drag this in, and it says, "are you sure you want to move this?" Yes, I say I want to move it.

Now this will break the site. If I hit refresh here you can see ... oh, one thing that is cool. You see that little alt tag is showing up. If the image is not available, it will show the alt tag.



Now we have a broken image and it's because we need to update the path. So here, I will say that I want it to find it in the **images directory**, in **logos**, and then it will go and find the image file.

### **index.html**

```
<div class="banner-image">
  
</div>
```

Switching back to the browser, hitting refresh. You can see that the image is back and it is working properly.



## Coding Exercise

Create a image tag with the src path set to **my-great-logo.jpg** and the alt set as **My Logo**.

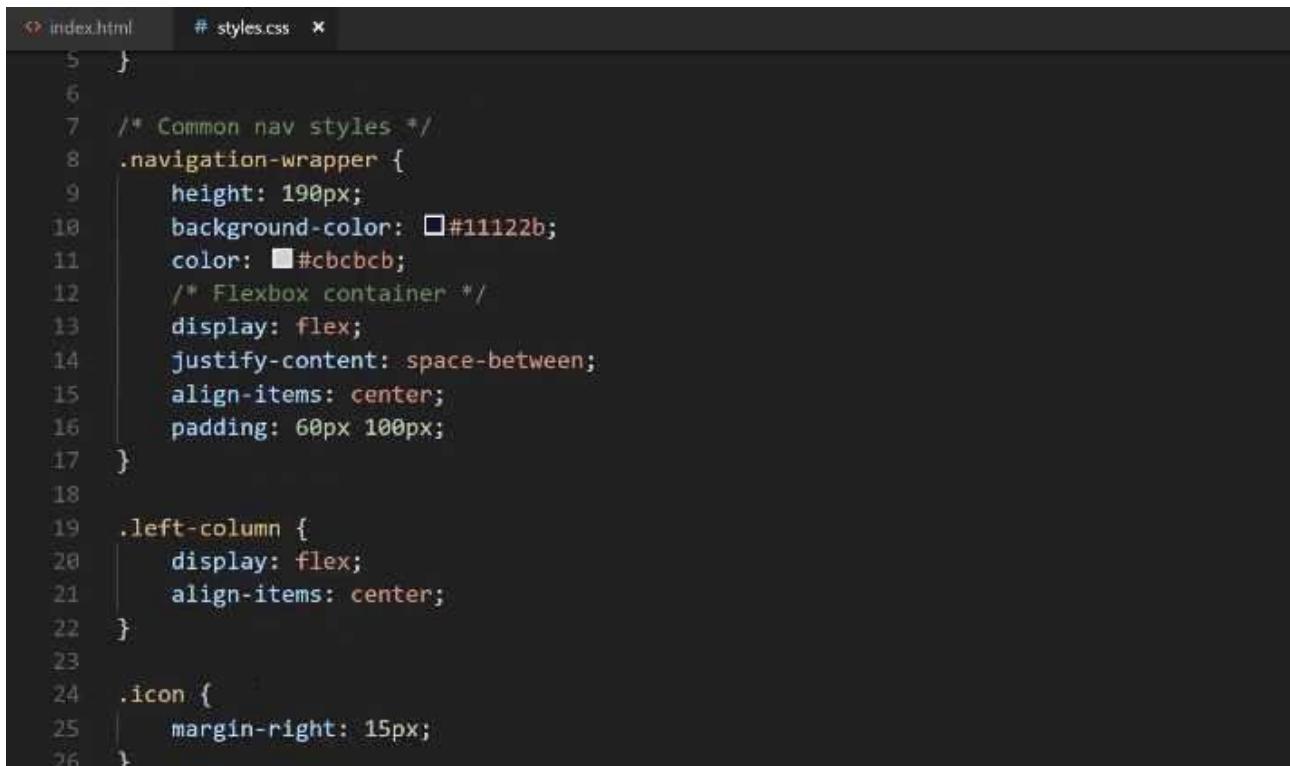
# 1.21 CSS code Refactoring

In this lesson, we're going to perform some refactoring. If you've never heard of what refactoring is, what it essentially means is when you have code that can be improved.

Many times it revolves around making sure that you conform to best practices, and so that's what we're going to do here. We, for the most part, have been following along with all the best practices that you'd usually want to use, but there are a few little items that I purposefully left off.

The main reason for that is because it's really important in understanding how CSS works. I wanted to start off with more of a simple implementation, and then show you why you will want to become a little bit more specific.

Let's switch back to the code, and I'm going to close this off to the side. Now, do you see right here how we have some generic styles here? So, `navigation-wrapper`, that's pretty specific but `left-column`.



A screenshot of a code editor showing two files: index.html and styles.css. The styles.css file contains the following CSS code:

```
index.html # styles.css x
5 }
6
7 /* Common nav styles */
8 .navigation-wrapper {
9   height: 190px;
10  background-color: #11122b;
11  color: #cbcfcf;
12  /* Flexbox container */
13  display: flex;
14  justify-content: space-between;
15  align-items: center;
16  padding: 60px 100px;
17 }
18
19 .left-column {
20   display: flex;
21   align-items: center;
22 }
23
24 .icon {
25   margin-right: 15px;
26 }
```

Let's imagine that you have a scenario where you want to use the name `left-column` later on, but you may want it on a different page. You may want to use it on the About page and you want to use this `left-column` and you may not want to use Flexbox. You may want to use CSS grid for it.

Well, with what we have right here, that would be a problem. We would have `left-column` and then either this set of styles would be applied for that other page or we would implement some other styles later on that would override these. Then that would cause an issue with the navbar.

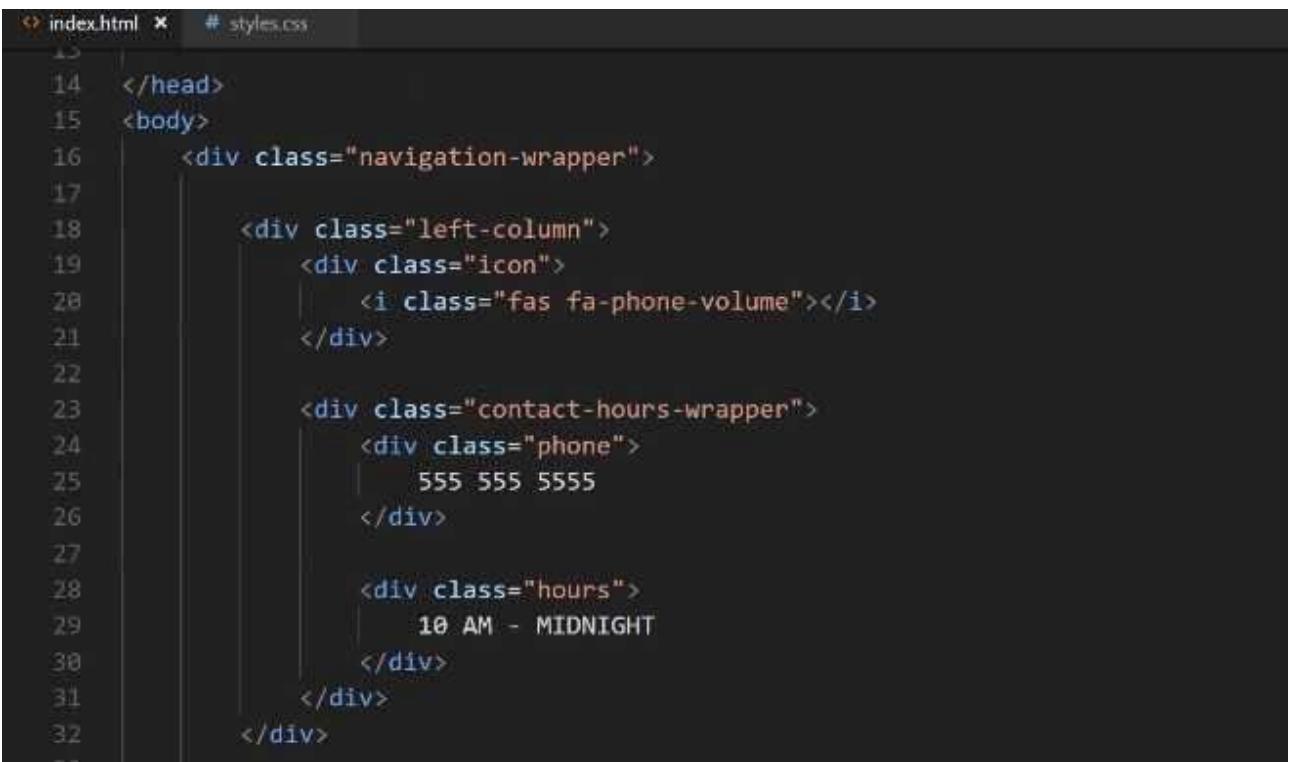
What we're going to do is we're going to make our system a little bit more specific. We know that our left-column needs to have these styles when it's inside of the navigation-wrapper, so what we're going to do is we're going to use inheritance. We've shown how to do this right here on line 32.

We're essentially going to be doing the same thing. So here, I can say `.navigation-wrapper` and then we want to say `.navigation-wrapper > .left-column`. So inside of here, whenever you find a left-column we want you to apply these styles.

What this means is that if we ever want to use a `left-column` in some other part of the project, it's perfectly fine because it's not going to be nested inside of the `navigation-wrapper`. That's what we're doing there.

Same thing for this icon. We don't want to use a `margin-right: 15px` for every icon on the entire site. We simultaneously don't want to use a `font-size`, a `2em`, across the board either.

We can look at the HTML here and so we can see that that icon here specifically is the icon that is inside of the left-column. That's where we want to apply those styles. We also have our little navigation-wrapper here.



```
14  </head>
15  <body>
16      <div class="navigation-wrapper">
17
18          <div class="left-column">
19              <div class="icon">
20                  <i class="fas fa-phone-volume"></i>
21              </div>
22
23              <div class="contact-hours-wrapper">
24                  <div class="phone">
25                      555 555 5555
26                  </div>
27
28                  <div class="hours">
29                      10 AM - MIDNIGHT
30                  </div>
31              </div>
32          </div>
33
```

Let's play around with this and see exactly how the inheritance works. If I say `.navigation-wrapper > .icon`, let's see if these same styles get applied, and we'll do the same thing here. So, say: navigation-wrapper, hit save, and let's see if that works.

### styles.css

```
.navigation-wrapper > .icon {
```

```
        margin-right: 15px;
    }

.navigation-wrapper > .icon i {
    font-size: 2em;
}
```

If I come here and hit refresh, you can see that those styles disappear.



So, why is that exactly? Well, it's because of how this little `>` symbol works. This does not work on a deeply nested level. Instead, what it says is you first need to become more specific. For the navigation-wrapper, if you want to grab this icon, it needs to know that chain of events.

It needs to know for the navigation-wrapper what other classes connect to icon. In this specific case, if you look at HTML, we have the navigation-wrapper, then we have the left-column, then we have the icon.

We have a couple different options. We could just do this. We could say `.navigation-wrapper > .left-column > .icon`, and then do the same thing here.

### styles.css

```
.navigation-wrapper > .left-column > .icon {
    margin-right: 15px;
}

.navigation-wrapper > .left-column > .icon i {
    font-size: 2em;
}
```

Hit save, hit refresh, and now you can see that that is back to working.



In this specific scenario, it's up to you. If you're making a very large application, you do want to be specific like this. Imagine a scenario where you leave off the navigation-wrapper, so you just do it like this `.left-column > .icon`. This will still technically work for this one-use-case.

You can see nothing changes here, but the one issue is if you create a new left-column class and it has an icon class inside of it, then you might be overriding those values or these values might be overridden.

All of this goes back to the nature of CSS and its cascading kind of behavior. When in doubt, the best option is to be as specific as possible. Now when you start to get into more advanced techniques, so there is a skill set out there, it's a very cool tool called SCSS and it's usually, even though it sounds like SASS, it actually is usually spelled out like SCSS.

When you get into that you'll see that this process becomes much more intuitive, but we can't do it in this introductory course just because you need to learn the fundamentals before you start getting into the more advanced techniques.

It's very important to understand that CSS operates like this. This is an important thing to understand, especially if you're just learning how to build websites for the first time. So I'm happy with these. Let's keep on moving down.

Now this left-column right here, we could really apply the same exact technique here. I don't see any reason why not to. I could say `.navigation-wrapper`, `.left-column`, and then that's all going to work.

#### styles.css

```
.navigation-wrapper > .left-column > .contact-hours-wrapper {  
    display: grid;  
    grid-template-columns: 1fr;  
    grid-gap: 10px;  
}
```

`.contact-hours-wrapper` is going to be the same thing, except here, what I'm thinking, let's say that if I want to do `navigation-wrapper` and `left-column` just like that.

#### styles.css

```
.navigation-wrapper > .left-column > .contact-hours-wrapper {  
    display: grid;  
    grid-template-columns: 1fr;  
    grid-gap: 10px;  
}  
  
.navigation-wrapper > .left-column > .contact-hours-wrapper > .hours {  
    font-family: 'Ubuntu Condensed', sans-serif;  
}
```

Let's just verify that that is correct. Okay, yes, so `contact-hours-wrapper`. All that should be working fine. So `nav-link`, this is one that we're going to have to fix. If we had the `nav-link` anywhere else then this might start to get a little bit messy.

Now I'm not going to use the navigation-wrapper here. Instead, what I'm going to do is we're going to create a link class, another link class. The reason for this is because I want to show you how later on you can reuse some of this functionality.

That way you can use these links and have a lot of these styles built-in in the footer or in some other spot that doesn't have a navigation-wrapper. In this case, I'm going to create, just like I have that comment up there, I'm going to create a class called `links-wrapper`.

```
index.html # styles.css
30 |     </div>
31 |     </div>
32 | </div>
33 |
34 |     <!-- Link wrapper -->
35 |     <div class="links-wrapper">
36 |         <div class="banner-image">
37 |             
38 |         </div>
39 |         <div class="nav-link">
40 |             <a href="index.html">Home</a>
41 |         </div>
42 |
43 |         <div class="nav-link">
44 |             <a href="about.html">About</a>
45 |         </div>
46 |
47 |         <div class="nav-link">
48 |             <a href="menu.html">Menu</a>
49 |         </div>
```

Now what I can do, because I'm pretty confident that `links-wrapper` is not going to be placed anywhere that I don't want these navigation links placed. What I can do is, for this nav-link, I can say:

### styles.css

```
.links-wrapper > .nav-link a {
    font-family: 'Ubuntu Condensed', sans-serif;
    color: white;
}
```

Then we want to say everything inside of that is going to take in those same styles. For the `address-wrapper`, if we come here you can see that right now we have this entire thing as its own div, I think it would make sense to actually update that.

```
index.html
54 |     </div>
55 |
56 |     <div class="address-wrapper">
57 |         <i class="fas fa-map-marker-alt"></i>
58 |         123 Any Street
59 |         Scottsdale, AZ, 85251
60 |     </div>
61 |
62 | </div>
63 | </body>
```

I think we should have some wrapper divs. This `links-wrapper` div should also be the `center-column`, and then we should have a `right-column` here. I think that way is best because, if you remember, we have a left-column on this side so let's perform the same task.

Let's say that we want this to be the `center-column`.

```
index.html # styles.css
22
23     <div class="contact-hours-wrapper">
24         <div class="phone">
25             555 555 5555
26         </div>
27
28         <div class="hours">
29             10 AM - MIDNIGHT
30         </div>
31     </div>
32
33
34     <div class="center-column"></div>
35     <div class="links-wrapper">
36         <div class="banner-image">
37             
38         </div>
```

Then if we move down here to the closing div, and we want to close that off. Now let's also make sure we keep our indentation just so it's nice and readable.

```
44             <a href="about.html">About</a>
45         </div>
46
47         <div class="nav-link">
48             <a href="menu.html">Menu</a>
49         </div>
50
51         <div class="nav-link">
52             <a href="contact.html">Contact</a>
53         </div>
54     </div>
55 </div>
```

Now our center-column has our links-wrapper inside of it. Now, for the address-wrapper we're going to do the same thing. So here, we'll say this is going to be the `right-column`, let's indent this, and then close off that div.

```
55     </div>
56
57     <div class="right-column">
58         <div class="address-wrapper">
59             <i class="fas fa-map-marker-alt"></i>
60             123 Any Street
61             Scottsdale, AZ, 85251
62         </div>
63     </div>
64
```

Now if you hit save and go back. Now you can see that everything is still working. We have our right-column here, our center-column, and then our left-column. Now we can get a little bit more specific.

Right here we have our `navigation-wrapper` which has the font, then we have our `links-wrapper` here. For this one, I am going to just keep that the same because, like I said, I may want to use that in the footer or something when it's not in the center column.

For `address-wrapper` we definitely want this to become a little bit more specific. For this one we could say something like:

#### styles.css

```
.navigation-wrapper > .right-column > .address-wrapper {  
    font-family: 'Ubuntu Condensed', sans-serif;  
}
```

Hit save, come back, hit refresh, everything there's looking good. Lastly, here, we have our `banner-image`. Now the banner-image is going to be inside of the navigation-wrapper because I want these styles really only when I'm on the home page on that navbar.

So it's going to be `.navigation-wrapper > .links-wrapper > .banner-image img`. As I'm saying this, I think there's a better way of doing this. I think this should be the center-column, I'll explain why here in a second, so I'm going to say:

#### styles.css

```
.navigation-wrapper > .center-column > .banner-image img {  
    font-family: 'Ubuntu Condensed', sans-serif;  
}
```

If I come over here, you see how I have `links-wrapper` and I have that `banner-image` inside of it? I think that's actually a mistake because I like my code to be very descriptive. You should as well.

If I come to a class called `links-wrapper` my expectation is that it's only going to have links inside of it. So instead, let's take this `banner-image` and place it outside of that wrapper. Now we have a center-column that contains a logo and then all of the links just like that.

```
index.html • # styles.css  
30         </div>  
31     </div>  
32 </div>  
33  
34     <div class="center-column">  
35         <div class="banner-image">  
36               
37         </div>  
38  
39         <div class="links-wrapper">  
40             <div class="nav-link">  
41                 <a href="#">index.html>Home</a>  
42             </div>  
43  
44             <div class="nav-link">  
45                 <a href="#">about.html>About</a>  
46             </div>
```

Let's hit save, and let's see if we broke anything. Nope, everything is still looking good.



What we've done here is, and sorry if this is a little tedious, but it is important to understand how CSS works. I am assuming that's the whole reason why you're taking the course and spending all the time going through here is being able to understand the exact process.

You're not always going to have a tutorial to follow. You're going to get to the point where you're going to have to build this kind of project out by yourself, and understanding the fundamentals behind it is absolutely critical.

What we've done here is we have refactored all of our styles so that we're very specific. If you're ever curious on a style not being applied. Say you think that you have built out your style, so they should be working, and then you see that it's not working. Well, that is where the `devtools` are really helpful and that's where they come into play.

Here you can see we have our `div`, our `navigation-wrapper`, and look how much easier this is to read. It's very clear that everything here is in the left-column, everything here's in the center, and everything here's on the right.

If I click on the center column and then click on one of these links. If I scroll down here, you can see that you can actually trace the entire lineage. All the way from the `body` to the `navigation-wrapper` to that `div`. Just like we're doing right here.



```
body>
  <div class="navigation-wrapper">
    <div class="left-column">...</div>
    <div class="center-column">
      <div class="banner-image">...</div>
      <div class="links-wrapper">
        <div class="nav-link">...</div>
        <div class="nav-link">...</div>
        <div class="nav-link">...</div>
      </div>
    </div>
  </div>
```

You can become even more specific. If you say that you apply a style or you add in a style and it's not working, well, the browser thankfully does not lie. You can see right here that you have a links-wrapper and it is grabbing using the > symbol.

It's grabbing the next item, which is the nav-link. Then it's grabbing the actual `a` tag inside of there. That's where we're getting the `color` and the `font-family`. You can also trace which values have been overridden through that.

That is the way that, whenever I have a style that isn't working right, I always go straight here to the browser tools to see exactly what is happening and what values are being set. Now we have code that's been refactored, it's now conforming to best practices, and hopefully, you have an even better understanding of how CSS works.



## Coding Exercise

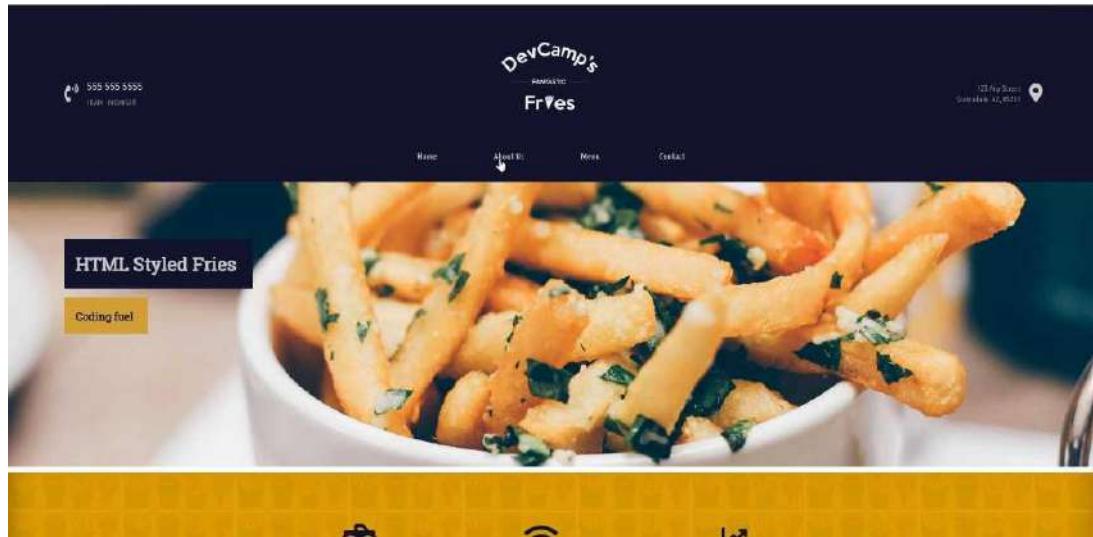
Using CSS inheritance selectors, make the below paragraph tag's font size `30px`

```
<div class="item-wrapper">
  <span class="item-description">
    <p>Make this font size 30px!</p>
  </span>
</div>
```

# 1.22 Flexbox inside Grid: Align

In this guide, we're going to work on centering the column element on the page here and we're going to see how we can use CSS grid combined with Flexbox in order to get the type of behavior that we're looking for.

Remember this is what we're actually looking for.



We want to have some nice space and styles here for this nav bar and then the logo right on top of it. So let's start building that out. We already have all of the structure that we're going to need from an HTML perspective.

We're not going to do any work here but we do need to start adding some custom styles, and as you know we have our navigation wrapper and then we have all of our left columns styles and now what we're going to do is we're going start working on the center column.

So far we've only used the center column banner image selector but right above here I'm going to create some styles just for the center column.

**styles.css**

```
.navigation-wrapper > .center-column {  
    display: grid;  
    grid-template-columns: 1fr;  
    grid-gap: 42px;  
    width: 500px;  
}
```

Hit save, now come back and hit refresh and you can see that that's working.



Now it's not centered but that's perfectly fine. We're not done yet. Let's right click and hit inspect and let's see exactly what this is looking like if we inspect the element.

You can see we have the image, if we click on center-column you may notice that what it's doing, it may look like it's off centered but the element itself is actually perfectly centered.



A screenshot of the Chrome DevTools Elements tab. The left panel shows the DOM tree with the "center-column" element selected. The right panel shows the "Styles" tab with the CSS rules applied to that element. One rule is ".navigation-wrapper &gt; .center-column { display: grid; grid-template-columns: 1fr; grid-gap: 42px; width: 100%; }" from "styles.css:51".

It just goes from left to right, so in order to center the items, we'll see what we need to do in order to do that. But as far as the column itself, it is taking up the exact amount of space that we're wanting.

Now what I want to do and this is part of the reason why I added a banner-image-wrapper instead of just having the image, I want to take just the banner-image-wrapper, so I'm going to remove the image selector here at the very end.

### styles.css

```
.navigation-wrapper > .center-column > .banner-image {
  display: flex;
  justify-content: center;
}
```

Let's hit save. Notice how we have a grid container that has two grid elements, the banner-image itself and the nav-links, and inside of that banner-image, we are turning that into a flex container so we can manage the image positioning.

So now if I hit refresh you can see that our image is now centered perfectly.



So that is looking good.

Now what we have to do is grab those links and do the same thing with them.

### styles.css

```
.links-wrapper {  
    display: flex;  
    justify-content: space-between;  
    align-items: center;  
}
```

I'm just going to call the links-wrapper by itself so that we can use it in other parts of the application. I think this is going to be a better use of that links-wrapper class so that it's a little bit more scalable and reusable.

let's come and hit refresh and look at that.



We actually have something that is getting much closer to the final version. This is coming along quite nicely. And notice how we have been able to use two Flexbox containers and we have those nested inside of a CSS grid container.

One of the common questions that I get, especially from new students who have just been introduced to tools like CSS grid and Flexbox, is which one I prefer.

In my personal opinion, it's been my experience in the projects I've built out that it's not a choice between the two but the two actually complement each other perfectly.

I know I've said that before but I do want to just reiterate it because this is a question that I've been asked multiple times so I want to make my own feelings on it clear.

I think that Flexbox and the CSS grid tools should be used in conjunction with each other because like you've seen right here we were able to take both of those and we're able to create a really nice looking interface by combining them and utilizing their strengths.

We're going to take a break right now. That is a great job, you have learned how to combine CSS grid and Flexbox and how you can switch between them in order to align items perfectly on the page.



## Coding Exercise

Utilize flexbox on the below links to align them vertically and horizontally in the same way as demonstrated in the lesson video.

```
<div class="link-box">
  <span class="link">
    <a href="home.html">HOME</a>
  </span>
  <span class="link">
    <a href="contact.html">CONTACT</a>
  </span>
  <span class="link">
    <a href="portfolio.html">PORTFOLIO</a>
  </span>
</div>
```

# 1.23 CSS animations

Now, let's look at what we're going to be implementing first. Right here in this little navbar, if I hover over any of these links, do you see how the letters kind of spread out and the color changes? Also, there's a smooth kind of easing in and easing out as I hover over?



Each one of these is something in CSS that we have control over. First, we have control over what is called a **pseudo-element**, which is also referred to as a **hover state**. When I hover over one of these items, that gives a little trigger to CSS that we can use.

We can listen for that kind of an event and then what we can do is make some changes to the styles. We can do things like change the letter spacing, change the color, and then we can add an animation so that it all happens really nice and smooth.

Right now, we just have a basic set of links here. What we're going to walk through is how we can animate these and implement that type of behavior. I'm going to switch back to the text editor here.

We have our **links-wrapper**, and if you look at the HTML structure, we have the **links-wrapper**, the inside of the **links-wrapper**, and we have the four **child-nav-links**. So, we want to work with this nav-link class right here.

```
34 <div class="center-column">
35   <div class="banner-image">
36     
37   </div>
38
39   <div class="links-wrapper">
40     <div class="nav-link">
41       <a href="index.html">Home</a>
42     </div>
43
44     <div class="nav-link">
45       <a href="about.html">About</a>
46     </div>
47
48     <div class="nav-link">
49       <a href="menu.html">Menu</a>
50     </div>
51
52     <div class="nav-link">
53       <a href="contact.html">Contact</a>
54     </div>
```

The very first thing we want to do is, let's just test to make sure we have access to that value. I'm going to say `links-wrapper`, and then I'm going to grab the `nav-link`. Then, I just want to grab the `a`, that is the `a-tag`. So I'm grabbing the link itself right here.

Let's make sure that we have access to it by being able to remove that underline, to customize the font, and to do some things like that.

I'm going to say, `font-family` here, and then I want to use the `Ubuntu Condensed`, and then have a backup of `sans-serif`. Let's go with a custom color here. We'll say `#CBCBCB` for the hexadecimal value, giving us that nice little light gray.

### styles.css

```
.links-wrapper > .nav-link a {  
    font-family: "Ubuntu Condensed", sans-serif;  
    color: #CBCBCB;  
}
```

The way we can remove the underline is by saying `text-decoration: none`. Then I'm going to put something here that is going to be required in order to have that nice smooth animation.

What we're going to say is, `not transform`, but `transition`. Then transition takes a few arguments, but for right now we can just pass in the amount of time we want the transition to take place over. I want it to take `0.5s`. That is going to work well.

### styles.css

```
.links-wrapper > .nav-link a {  
    font-family: "Ubuntu Condensed", sans-serif;  
    color: #CBCBCB;  
    text-decoration: none;  
    transition: 0.5s;  
}
```

Now, whenever one of those hover effects is triggered, what this transition does is it's a rule that says, "If any kind of action takes place, then we want to have a smooth transition that should take half a second to get to that next state."

Let's save this, and if we hit refresh, you can see that we now have the new font. We don't have the animations yet, but it removed the underline. So, that is working well.



Now, let's come up here and I'm going to just copy this selector. Then, we're going to update it a little bit. Instead of just grabbing the `nav-link` and the `a`-tag, we're going to say, `a:hover`.

### styles.css

```
.links-wrapper > .nav-link a {  
    font-family: "Ubuntu Condensed", sans-serif;  
    color: #CFCFCB;  
    text-decoration: none;  
    transition: 0.5s;  
}  
  
.links-wrapper > .nav-link a:hover {  
}
```

What `hover` is, it is a pseudo-state. This is a pseudo-selector, meaning that we're not technically grabbing an element. We're saying that "Whenever a hover event takes place, I want you to change the value to whatever we pass inside of these curly braces."

Right here, I want to change the color. We're going to use that gold color. I have that at, `#CEA135`, and then let's also add that space between the letters. The way we can do that is with a rule called `letter-spacing`, and I want to add `2px` of spacing between those letters.

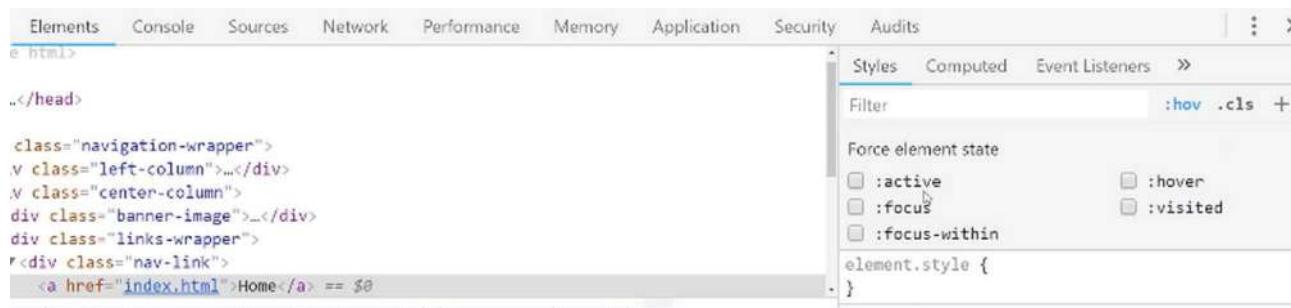
### styles.css

```
.links-wrapper > .nav-link a {  
    font-family: "Ubuntu Condensed", sans-serif;  
    color: #CFCFCB;  
    text-decoration: none;  
    transition: 0.5s;  
}  
  
.links-wrapper > .nav-link a:hover {  
    color: #cea135;  
    letter-spacing: 2px;  
}
```

Let's switch back. Hit refresh, and now, if we hover over, you can see we're getting that nice hover effect. That is working really well.



If you're ever curious about some of the states, let me go and select one of these items. If you're ever curious about, say, the hover state. If you come here to the Chrome developer tools and click on **hover**, you can see that you have access to view what they look like in one of these states.



The screenshot shows the Chrome Developer Tools interface. The top navigation bar includes tabs for Elements, Console, Sources, Network, Performance, Memory, Application, Security, and Audits. The Elements tab is active, displaying a portion of the HTML code. On the right side, the Styles panel is open, showing a list of pseudo-selectors under the heading 'Force element state'. The ':hover' selector is highlighted with a blue background. Below the list is a preview area showing a small snippet of CSS: 'element.style { }'. A tooltip or hint above the ':hover' entry says ':hover .cls +'.

It also gives you a nice hint of telling you some of the other **pseudo-state selector** names. Right here, if you click on hover, you can see what happens when you hover over, even without having to go and perform the action by yourself.

That's a nice little helpful tool whenever you're trying to test out some of your pseudo-selectors. Now, we're not quite done. If you notice when I hover over, do you see how this actually moves the other elements slightly?

That's because each one of these items is dynamic. We're using **Flexbox** for the spacing. Whenever I hover over one of these items, it is actually shrinking the space for each one of the other elements. That's the reason why that is occurring.

What we can do is we can actually give a hard-coded defined width for each one of these items. I'm going to open up the code again and I'm going to work with this **nav-link wrapper** because each one of those a-tags is wrapped up inside of that nav-link div.

I'm going to remove the **a-tag**, and now what I can do is, I'll just say **width** and let's set it at **70px**. That's not going to give us exactly what we want, and I'll show you why here in a second.

### styles.css

```
.links-wrapper > .nav-link {
    width: 70px;
}

.links-wrapper > .nav-link a {
    font-family: "Ubuntu Condensed", sans-serif;
    color: #CFCFCF;
    text-decoration: none;
    transition: 0.5s;
}
```

If I hit refresh, you see how they move slightly to the left? We fixed our issue with how it's not pushing the items anymore. The reason why is if you click on one of these nav-links, do you see where it shows, right there, how the div is now a little bit wider than it was before?

Before we had this width, if you'd hover over and inspect that item, the full width just was whatever the width of the content was. When you set this 70px, you're creating a container so that you can make sure that it doesn't

```
<body>
  <div class="navigation-wrapper">
    <div class="left-column">...</div>
    <div class="center-column">
      <div class="banner-image">...</div>
      <div class="links-wrapper">
        <div class="nav-link" style="width: 70px;">HomeAboutMenuContact
```

spill over. So, the letter spacing is not going to spread that item past that 70px.

Now, if you are working with a navigation bar or you're building this feature and you have some elements that maybe have wider names or something, then 70px won't work. You might have to do 100px or something like that.

The reason why all of these items are to the left is because now, with these 70px, all of the text elements are `text-aligned` to the left. We can fix that if you just say `text-align` inside of this wrapper, and say `text-align: center`.

Now, do you see how those are all perfectly centered? Now when I hover over, they spread out to the left and right because they are aligned to the center. So, let's go change that in the code. So I'll say:

### styles.css

```
.links-wrapper > .nav-link {
  width: 70px;
  text-align: center;
}

.links-wrapper > .nav-link a {
  font-family: "Ubuntu Condensed", sans-serif;
  color: #CBCBCB;
  text-decoration: none;
  transition: 0.5s;
}
```

Hit save, hit refresh, and now we should have the final version of this navbar. As I hover over, each one of these works. The animations work perfectly. We've also wrapped them up in the right kind of format so that they don't get distorted whenever you're creating that hover effect.



Great job if you went through that tutorial. You now know how to work with animations, transitions, how to work with pseudo-classes, and also how to align text inside of wrapper divs.



## Coding Exercise

Find and fix the spelling error below (also known as a syntax error). Also add a **transition** property with a duration of **0.7s** to the first CSS block.

```
.parent > .first-chld-element > span {  
    width: 100px;  
    background-color: green;  
}  
  
.parent > .first-child-element > span:hover {  
    color: white;  
}  
  
<div class="parent">  
    <div class="first-child-element">  
        <span>Item 1</span>  
        <span>Item 2</span>  
        <span>Item 3</span>  
    </div>  
</div>
```

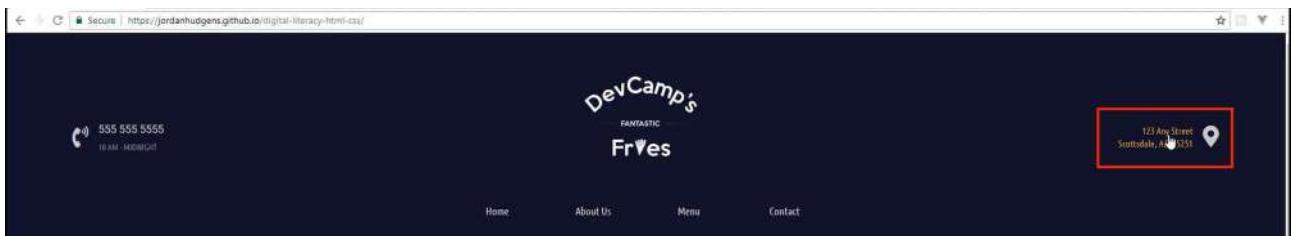
# 1.24 NavBar: Styling as a column

So you're making great progress and you're learning a lot along the way. You now have about 75% of this nav bar done, and it's just gonna take a couple more tutorials to get us 100% done, and then we can finish off the home page, and I promise you the rest of the site will not take as long as what we've done right here.

I know that we've spent a lot of time in building out this nav bar but that's because as you may have seen, we're learning a lot of the HTML and CSS fundamentals in order to do this, and you're building a professional looking page element. After you have these fundamentals down, then you can simply populate that down throughout the rest of the site.

So most of the time these kinds of things don't take this long but if you're learning from scratch, then it's important to not only build it, it'd be very quick and easy for you to just follow along and copy and paste the code that I write but it's more important that you actually understand the underlying concepts.

So with all that being said, let's build out this right column set of styles. If you look at the finish design and what we want to have, you'll see that we have this address on two lines and then we have this little icon bar here that's for the map, and you may have also noticed, this has a nice little hover effect, both of these do and they look a lot like what we already implemented here in the navigation bar



So we're gonna be able to use some of those same styles. So let's switch back now to the code and let's see what kind of structure we have. So we have this right column and then we have the address wrapper and then we have this little icon here.

Now we want to reverse what we did on the left hand column, so I'm going to take this icon and move it down a couple lines below and let's create another div. So here I'm gonna call this the contact icon and then let's move the closing div tag right here, and so now what we have is our address wrapper and then a separate div for the contact icon.

## index.html

```
<div class="right-column">
  <div class="address-wrapper">
    123 Any Street
    Scottsdale, AZ, 85251
  </div>

  <div class="contact-icon">
    <i class="fas fa-map-marker-alt"></i>
  </div>
</div>
```

So as you may have guessed, we have a couple different options, we can use Flexbox or we can use CSS grid to align these, and you could use either option. I personally think I'm going to go with Flexbox for this one. So we want to select the right column styles right here, so let's make sure, let me see if we've already done anything for the right column, I don't believe that we have.

So we do have some right column styles. I personally like to keep these organized, so I don't like to have things like the right column out in spots where it's gonna be harder to find or might get lost, so I'm going to cut this and come down, and this is a really good spot to have some comments here. So right above this I'm gonna add a comment and I'll say, right column styles, and where our center column is, let's add something for that.

So let's go up top to where we have the start of the center column, and I'm just going to say, center column styles and then up top where we start with the left, now we're going to have our left column styles. This just makes it a lot easier so that as we are navigating through if we ever want to make a change, we will know right where all of the left column styles are, where all the center ones and then where all the right ones are.

So that is the reason why I like to do that. The one thing you'll notice, the larger the application, the more cognizant you have to be with how you organize your code. There's entire fields of study based on how important it is to organize code because a messy code base is one that you're not gonna want to spend a lot of time with, and it's gonna be hard to maintain. So it's always best if you start with using your best practices right from the beginning.

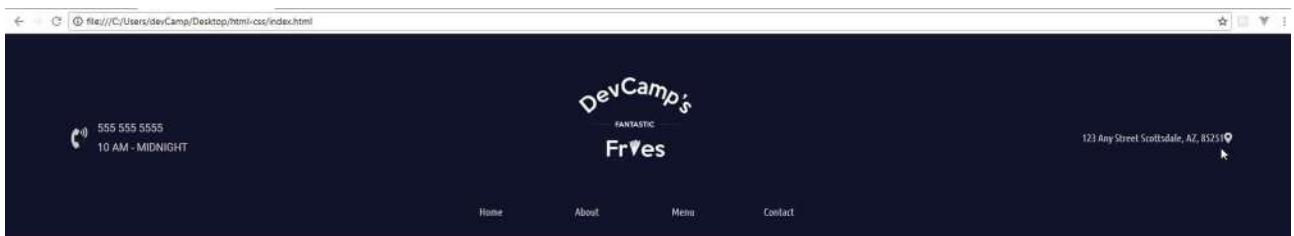
So I'm gonna give a navigation wrapper selector and then grab the right column here and then inside of here, as you may have guessed, we want to display Flex and then I just want to use, align-items and say center, and I believe that's all we're going to need here.

### styles.css

```
/* Right column styles */
.navigation-wrapper > .right-column {
  display: flex;
  align-items: center;
}

.navigation-wrapper > .right-column > .address-wrapper {
  font-family: 'Ubuntu Condensed', sans-serif;
}
```

So if I switch back and hit refresh, you can see that, that has been moved over but we still have a little bit of work to do on how these are gonna be separated.



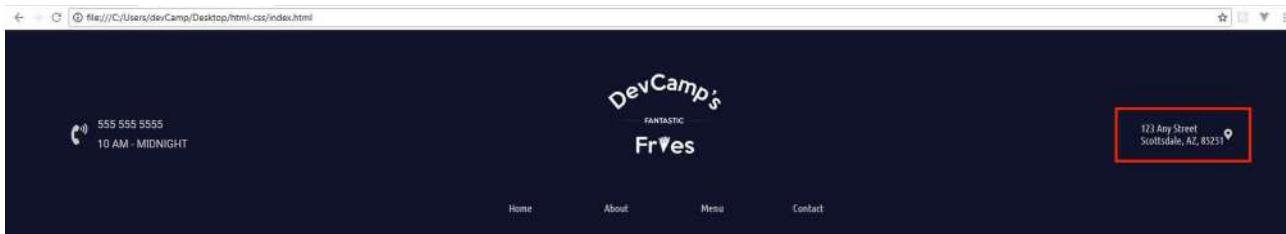
So first and foremost, we want to have a line break right here. Now we had a line break over on this left hand side however, on the right hand side we want to have one that is actually for the same type of content. So here we have a phone number and we have the hours. It makes sense for those to separate divs, but here this is really all the same thing, this is just an address. And so we have another really helpful tool here called the BR tag and this gives us a line break.

So if I come here to where it says, any street and I say BR and pass in this tag, if I come back and hit refresh, you'll see that, that is now on two lines and we didn't have to do any other CSS whatsoever.

### index.html

```
<div class="right-column">
  <div class="address-wrapper">
    123 Any Street<br>
    Scottsdale, AZ, 85251
  </div>

  <div class="contact-icon">
    <i class="fas fa-map-marker-alt"></i>
  </div>
</div>
```



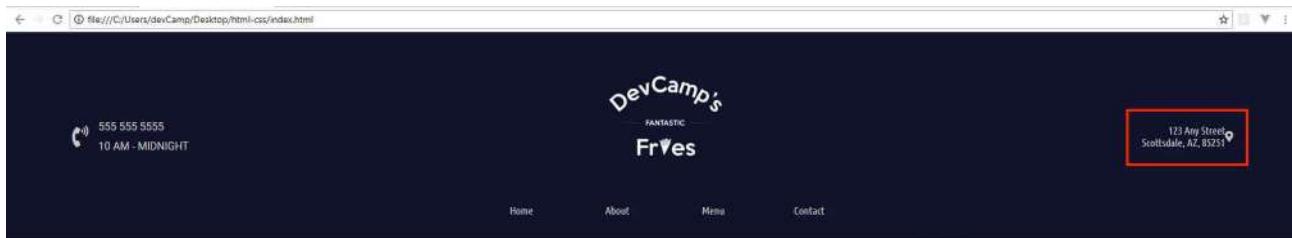
So that just gives us a nice little line break. So we also need to have this address wrapper styled a little bit differently. So right now we have the font family there, but let's also align the text. So I'm gonna say, text align to the right.

Before this, we'd only seen text align to the center but if we say to the right, this should give us what we need and there you go. Now you can see the address is lined up to the right hand side.

### style.css

```
/* Right column styles */
.navigation-wrapper > .right-column {
    display: flex;
    align-items: center;
}

.navigation-wrapper > .right-column > .address-wrapper {
    font-family: 'Ubuntu Condensed', sans-serif;
    text-align: right;
}
```



And then we also have the issue where this is supposed to be a link, actually the icon and the address are both supposed to be links. So let's go to our code here, and let's convert all of this into a link. So there is a little bit of a tricky way to do this because we have this all on one line but actually the BR tag can fit right inside.

So I'm going to give an, a tag and then an href and here we want this pointed to the contact HTML document and then at the very end, we're just going to close off that, a tag.

## index.html

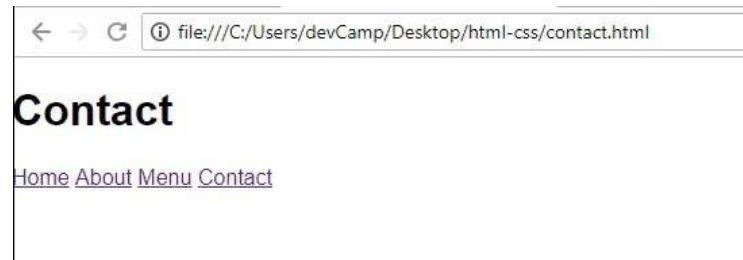
```
<div class="right-column">
  <div class="address-wrapper">
    <a href="contact.html">123 Any Street<br>
      Scottsdale, AZ, 85251</a>
  </div>

  <div class="contact-icon">
    <i class="fas fa-map-marker-alt"></i>
  </div>
</div>
```

So now if you hit refresh, this is actually a link.



So if I click on the link, you can see it goes to the contact page.



So that is working well. We obviously need to style this better but at least that is a link and now we're gonna do the same thing here for our icon.

So these icons really can be treated like any other HTML code, which means we can wrap them up in a link as well. So I'm going to say, a and then href, same thing, contact HTML and whenever I have a lot of code, just like we have right here, and really like we have here, I usually like to wrap it up and then indent it.

## index.html

```
<div class="right-column">
  <div class="address-wrapper">
    <a href="contact.html">123 Any Street<br>
      Scottsdale, AZ, 85251</a>
  </div>

  <div class="contact-icon">
    <a href="contact.html">
      <i class="fas fa-map-marker-alt"></i>
    </a>
  </div>
</div>
```

It just is a little easier to read and thankfully white space in HTML is something that's completely ignored, and when I mean white space I mean that you can break this up into different lines and

HTML will not render it as a different line. So this just allows to organize our code in a way that's a little bit easier to read.

So now you can see if you click on that little icon we're taken to the contact page. So everything's working well but we still need to work on some of our animations and also our styles. So we have this contact icon, I think it's probably time to style this up a little bit.

So let's come down here and let's add a selector, so I'm gonna have my navigation wrapper, I want a right column and then what was the name of that class, one more time, it was the contact icon class, contact icon and here I'm gonna set the font size and I'm also gonna set some margin to the left.

So I'll say margin left here, and let's go with 15 pixels and then I want to do a font size of 2em, which remember that will give me double the size of the font.

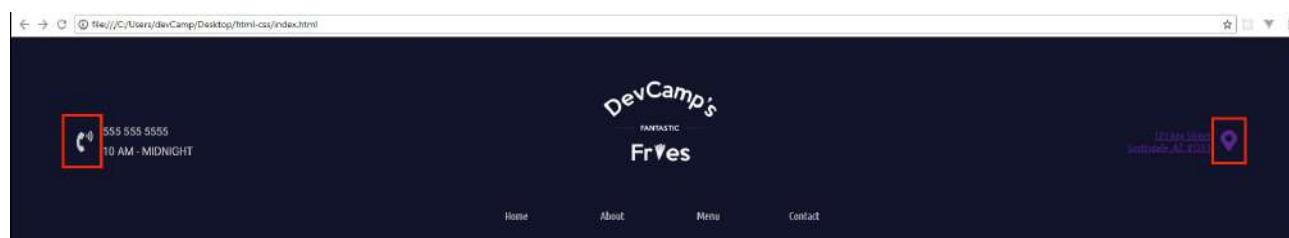
## styles.css

```
/* Right column styles */
.navigation-wrapper > .right-column {
    display: flex;
    align-items: center;
}

.navigation-wrapper > .right-column > .address-wrapper {
    font-family: 'Ubuntu Condensed', sans-serif;
    text-align: right;
}

.navigation-wrapper > .right-column > .contact-icon {
    margin-left: 15px;
    font-size: 2em;
}
```

Hit refresh and there you go, our icon is looking much better, looks a lot more like what we have over here.



So now the last few items we have to do just revolve around styles and then some of those hover effects. So let's look at the final version. You can see that we want to have a start state here where this link, where this address, starts with kind of this darker gray and the icon starts with that off white but they both switch to that yellowish, gold-ish type color.

So let's start with the, I think with the address. We'll go with that one first just so we can move from left to right, and your first inclination may be to simply add the content inside of this address wrapper but actually what we need to do is to get a little bit more specific, we need to add it to the, a tag.

So I'm going to copy this, and then at the very end just add a, a tag. We can get rid of the font family and the text align. Those are fine in the wrapper class but now what we need to do is add the styles that are specific just to the link.

So a few of those are going to be things like the color, and so we want to have that dark gray which the hexadecinal value for that is gonna be #858585 and then let's go with text decoration of none, and then from there we want to have a font size that is a little bit smaller. So I want to have a font size of 0.9em which is just around 90% or so, and then for animation purposes, which is really just gonna be a nice hover effect where we change the color, then I want to do a transition of 0.5s, so just half a second.

## styles.css

```
/* Right column styles */
.navigation-wrapper > .right-column {
    display: flex;
    align-items: center;
}

.navigation-wrapper > .right-column > .address-wrapper {
    font-family: 'Ubuntu Condensed', sans-serif;
    text-align: right;
}

.navigation-wrapper > .right-column > .address-wrapper a {
    color: #858585;
    text-decoration: none;
    font-size: 0.9em;
    transition: 0.5s;
}
```

So if I hit refresh now, you can see that, that is working and that color looks a lot better.



So now what we want to do is let's add our hover effect. So I'm going to just copy that and then I'm gonna add a sudo selector here at the very end. So instead of just A it's gonna be, a:hover and then we're gonna remove each one of those items because all we really need here is going to be the color, and for that we'll reuse that #cea135.

### styles.css

```
.navigation-wrapper > .right-column > .address-wrapper a:hover {  
    color: #cea135;  
}
```

Now if I come back and hit refresh, come over and hover, you can see we have that nice fade in and fade out on the link and the link is still functional, so that's working perfectly.

So now let's switch back and let's just add the styles to the card icon and we're gonna be all done. So here we'll get rid of the values inside. First we're just gonna grab the, a tag and then we'll change the color. So the color that we're gonna want is kind of that off white color, which is that #cbcfc color and then for the hover effect, we're just going to switch it up, add that hover element and now we'll go with that same #cea135.

### styles.css

```
.navigation-wrapper > .right-column > .contact-icon a {  
    color: #cbcfc;  
}  
  
.navigation-wrapper > .right-column > .contact-icon a:hover {  
    color: #cea135;  
}
```

Hit save, hit refresh and now you can see the start of the color change, just like we have our nice icon for the phone over here and now I hover over this, it works, I hover over that and it's working perfectly.

So as you may have noticed, this transition doesn't work and it's probably because I forgot to put a transition in here. So if you ever have some type of hover effect that doesn't work, always make sure that you have your transition. So I'm gonna say half a second on that transition.

```
.navigation-wrapper > .right-column > .contact-icon a {  
    color: #cbcfc;  
    transition: 0.5s;  
}
```

Hit refresh and there we go, now we have that nice little fade in and fade out. When it comes to animations you typically don't want to overdo it, you don't want to go over the top and have things flying all over the page, it's better to just kind of have nice little subtle types of animation that will give the site a more professional look and feel, while also giving some nice dynamic behavior.



## Coding Exercise

Align all text in the wrapper to the right, change the subtext's color to #0597e6, make the header's font-size 18px, and make the subtext's font-size 14px.

```
<div class="sidebar-wrapper">
  <div class="header">
    Yay its a header!
  </div>
  <div class="subtext">
    Cool, heres some more text
  </div>
</div>
```

# 1.25 NavBar: Styles meeting Best Practices

Nice job in making it this far. We are just about done with the navbar.



We're going to make a few final little changes. We're also going to refactor the styles and place them in a file just so the code is nice and organized.

The last element that we need to style is just the `hours` right here. As you can see, we have a slightly different set of styles that we need to implement. Let's switch over to the code and go up and see if we have our `hours`.

As you can see from our comments, this is already a little bit easier to find. Right here we have `contact-hours-wrapper` and then we actually have the `hours` themselves. We have the correct font but now let's go and let's change the size and the color.

I'll say `color` and we want to use that same dark gray, so it's at `#858585`. Then I want to use a font size that is `0.8` so about `0.8em`.

## styles.css

```
.navigation-wrapper > .left-column > .contact-hours-wrapper > .hours {  
    font-family: "Ubuntu Condensed", sans-serif;  
    color: #858585;  
    font-size: 0.8em;  
}
```

Hit save, hit refresh. Oh, it looks like we did not select it right, so let's see what's going on.

Let's go into the `index.html`, and go up here. You can see we have `hours` inside of `contact-hours-wrapper`. Theoretically, that should work. I have `navigation-wrapper` to `contact-hours-wrapper`, and then to the `hours` class.

We have the color correct and then we have the font-size. Whenever we have an issue like that then the next thing I do is I'm going to come here and look and see what could be causing that by inspecting the element.

Here you can see we have the `hours` and there are no styles here for that class. That is the reason why that's happening, so that could either be an issue related to the `selector` or something else. I think I already have it figured out.

```

<div class="contact-hours-wrapper">
  <div class="phone">
    555 555 5555
  </div>
  ...
  <div class="hours">
    10 AM - MIDNIGHT
  </div> == $0
</div>
</div>
<div class="center-column">
  <div class="banner-image">...</div>

```

You see where we have a `.navigation-wrapper > .left-column > .contact-hours-wrapper`? That's the issue. We need to add a left-column selector to `.navigation-wrapper > .left-column > .contact-hours-wrapper > .hours`. Hit save and now hit refresh. There we go, now that's working.



```

<div class="contact-hours-wrapper">
  <div class="phone">
    555 555 5555
  </div>
  ...
  <div class="hours">
    10 AM - MIDNIGHT
  </div> == $0
</div>
</div>
<div class="center-column">...</div>
<div class="right-column">...</div>

```

Now part of the reason why I like to leave these kinds of bugs inside of the tutorials is because if you run into those issues, and I promise you that you will. I've been doing this for a long time, I still run into those issues as I'm building websites out.

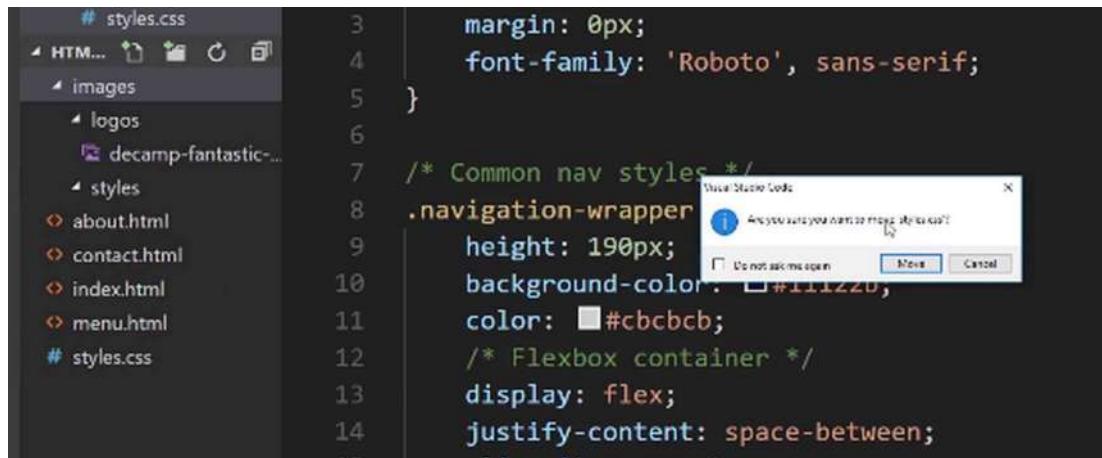
When you do, I want you to, not just being able to follow along, because when you're doing this on your own projects, you're going to have to figure it out and debug this yourself.

I want to walk you through the exact process that I go through whenever something doesn't work. The very first rule is: do not panic. I know if you're new to development that's kind of the first kind of default reaction.

Don't panic, just know that there is a process for being able to work through and then figure out exactly why something is not working. Now we have all of that set up properly. Let's go back to the code here and make sure we don't have any other issues. I think this all looks good.

Right now we just have this single style sheet, and we're going to start adding a lot more styles as we go through the rest of all of the styles we're going to build out. So what I'd like to do is break out our navigation styles into their own file.

I'm going to close this off and let's actually create a new directory here called **styles**. Then inside of this **styles** directory, I'm going to move our **styles.css** file. If you get this little pop up here that says, "Are you sure you want to move it?", just say "yes."

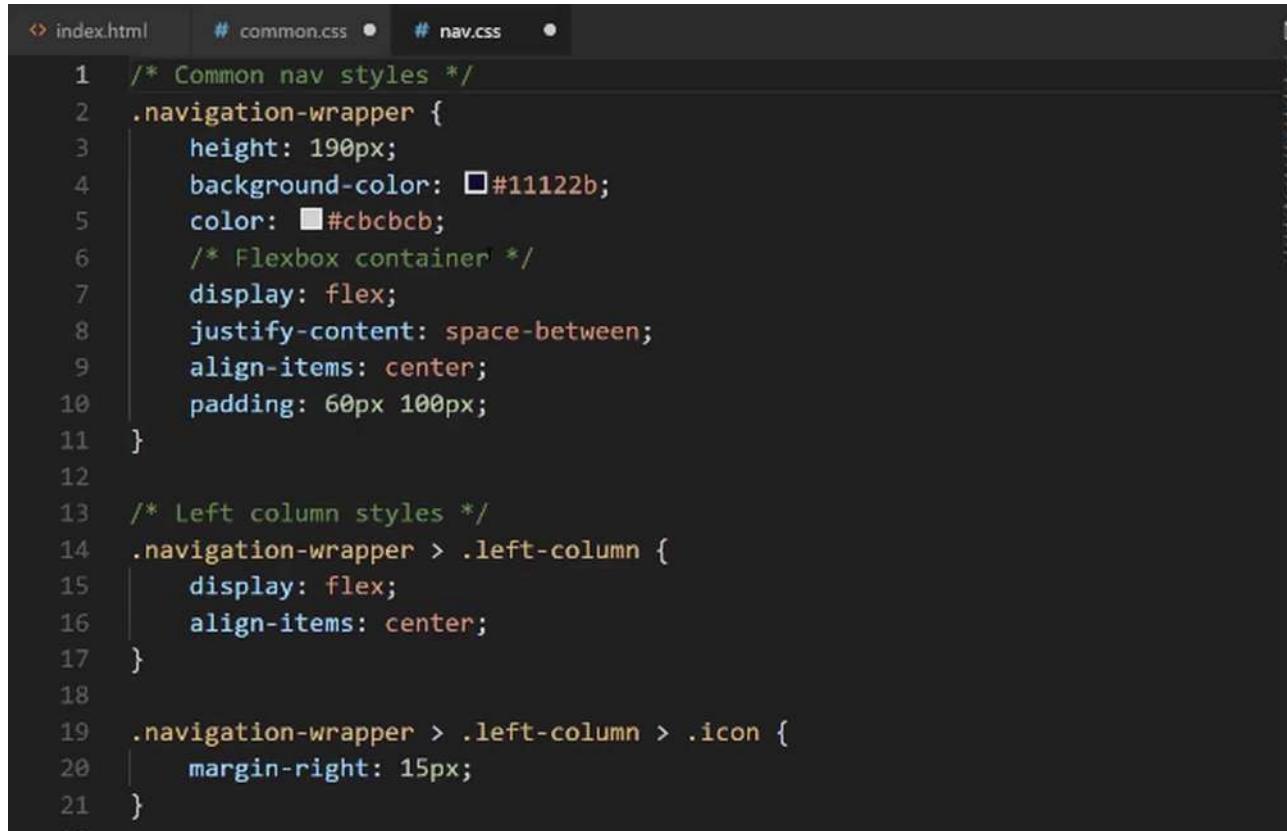


The screenshot shows the Visual Studio Code interface. On the left is the file tree, which includes files like `index.html`, `menu.html`, and `styles.css`. The `styles.css` file is currently selected. The main editor area contains CSS code for a navigation wrapper. A small modal dialog box from Visual Studio Code is overlaid on the screen, asking "Are you sure you want to move styles.css?". There are two buttons: "Move" and "Cancel".

```
# styles.css
# styles.css
  3 |   margin: 0px;
  4 |   font-family: 'Roboto', sans-serif;
  5 |
  6 |
  7 | /* Common nav styles */
  8 | .navigation-wrapper {
  9 |   height: 190px;
 10 |   background-color: #11122b;
 11 |   color: #cbcfcf;
 12 |   /* Flexbox container */
 13 |   display: flex;
 14 |   justify-content: space-between;
```

Then I'm going to rename this one. I'm going to call it **common.css**, and what that means is that any of the styles that go in this file are going to be styles that every page in the entire application needs.

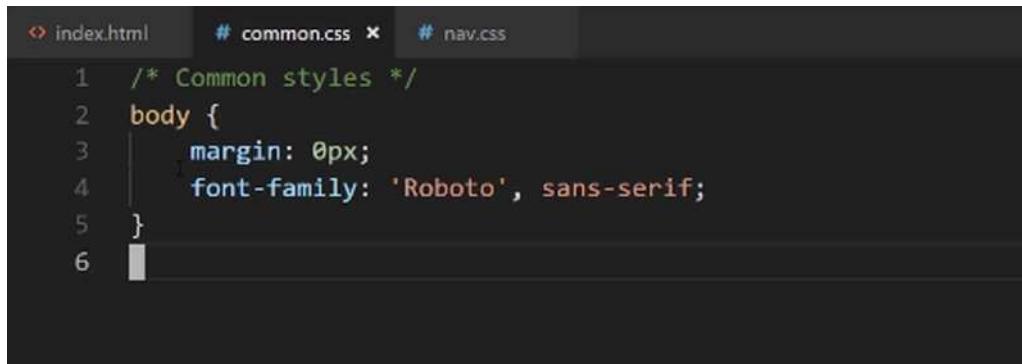
Then I'm going to create a new file here called **nav.css**. Inside of this file, this is where I'm going to grab all of those values and then paste those in.



The screenshot shows the Visual Studio Code interface with two tabs open: `common.css` and `nav.css`. The `common.css` tab contains the CSS code for the navigation wrapper, which has been moved from the original `styles.css` file. The `nav.css` tab is currently active and is empty, indicating it is where the copied styles will be pasted.

```
# common.css
# common.css
  1 | /* Common nav styles */
  2 | .navigation-wrapper {
  3 |   height: 190px;
  4 |   background-color: #11122b;
  5 |   color: #cbcfcf;
  6 |   /* Flexbox container */
  7 |   display: flex;
  8 |   justify-content: space-between;
  9 |   align-items: center;
 10 |   padding: 60px 100px;
 11 }
 12
 13 /* Left column styles */
 14 .navigation-wrapper > .left-column {
 15 |   display: flex;
 16 |   align-items: center;
 17 }
```

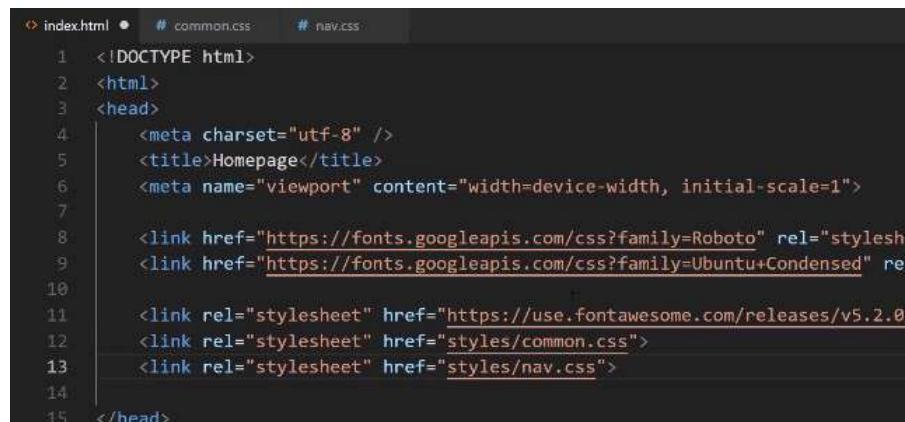
These are going to be the nav styles, and then we have our common ones, which now just has that body tag or that body tag selector in those styles.



```
index.html # common.css # nav.css
1 /* Common styles */
2 body {
3     margin: 0px;
4     font-family: 'Roboto', sans-serif;
5 }
6
```

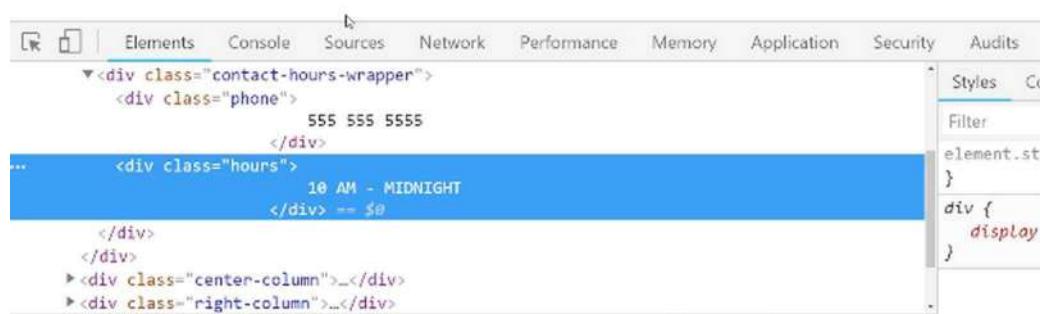
Now everything on the site's going to break because we've changed the path and we've also renamed the files. Let's go and let's fix that. Right here for our **style.css** sheet, this is no longer going to be a reference to a styles file.

Now we're going to say **styles/common.css** and then we want to have **styles/nav.css**.



```
index.html # common.css # nav.css
1 <!DOCTYPE html>
2 <html>
3 <head>
4     <meta charset="utf-8" />
5     <title>Homepage</title>
6     <meta name="viewport" content="width=device-width, initial-scale=1">
7
8     <link href="https://fonts.googleapis.com/css?family=Roboto" rel="stylesheet">
9     <link href="https://fonts.googleapis.com/css?family=Ubuntu+Condensed" rel="stylesheet">
10
11    <link rel="stylesheet" href="https://use.fontawesome.com/releases/v5.2.0/">
12    <link rel="stylesheet" href="styles/common.css">
13    <link rel="stylesheet" href="styles/nav.css">
14
15 </head>
```

Assuming I don't have any typos, then everything should be working. Nope. Looks like I did have a typo so let's scroll up and let's see the issue.



There's actually a couple of spots where you can do this. First, you can go and look for this reference. You can see we have styles, we have **common.css**, and then nav. You can also see where it has these little errors.

```
<link href="https://fonts.googleapis.com/css?family=Roboto" rel="stylesheet">
<link href="https://fonts.googleapis.com/css?family=Ubuntu+Condensed" rel="stylesheet">
<link rel="stylesheet" href="https://use.fontawesome.com/releases/v5.2.0/css/all.css"
integrity="sha384-hWVjflwFxL6sNzntih27bfxr27PmbbK/iSvJ+a4+0owXq79v+lsFkw54b0Gb1DQ"
crossorigin="anonymous">
<link rel="stylesheet" href="styles/common.css">
<link rel="stylesheet" href="styles/nav.css">

```

If you click on that, it'll say there was an error trying to access these files, so it tried to go and do **styles/common** and it couldn't find it. Let's go and let's see the issue. The issue is because it put when I clicked on new directory, it actually put it inside of the images directory.

Let's move that entire folder. Now you can see it's lined up perfectly. That should be the issue. Now if I hit refresh, there you go, we're back to working, and everything is looking really good.



```
...<div class="hours">
  10 AM - MIDNIGHT
</div> == $0

```

Now the really nice thing about our file structure is if you ever need to make a change to the navbar styles, you'll know exactly where to go do that. Then once you get into the **nav.css** file, you can see where the common nav styles are, where the **left-column** is, where the **center**, and then where the **right** is.

The better organized you can keep your code, the more enjoyable working on the projects are going to be. I can promise you, if you come back to a project that you haven't worked on for even just a few weeks or a few months, then it's going to take you a while to become acclimated with where you put all of the files, especially for a larger project.

The better you can get at naming your directories and your files properly, the easier it's going to be whenever you need to go and make changes. Also, when you're working on a team, so if you're working to build a project out with more developers than just yourself, they are really going to appreciate when you wrap up your code and you organize it properly.

We now have completed the navbar and you've learned all kinds of things along the way. You've learned about **animation**, you've learned about **Flex Box**, **CSS grid**, images and how to implement all kinds of styles.

In the next guide, we are going to start moving down the line and we're going to see how to build out this really cool little **parallax** feature. I will see you in that guide.



## Coding Exercise

The second description class needs to have slightly different styles than the first one. Use CSS inheritance selectors to change the second description's font size to **14px** and the color to **lightblue**.

*/\*These are the default styles for the description. Leave these styles\*/*

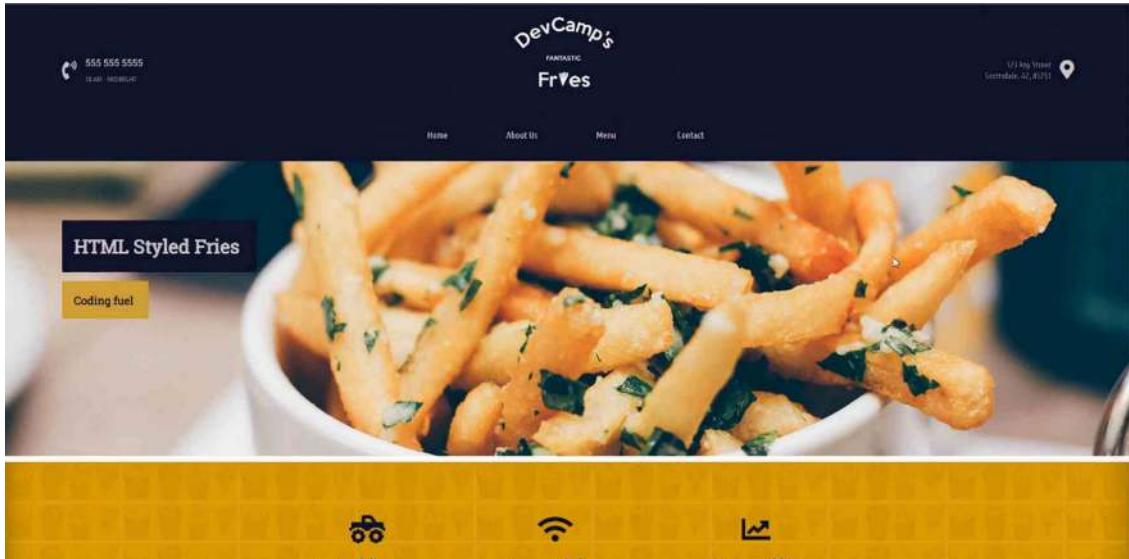
```
.description {  
    color: lightgrey;  
    font-size: 12px;  
}
```

*/\*Write your code here to overwrite the above styles\*/*

```
<div class="item-1">  
    <p class="description">A great description</p>  
</div>  
  
<div class="item-2">  
    <p class="description">Another great description</p>  
</div>  
  
<div class="item-3">  
    <p class="description">The greatest description</p>  
</div>
```

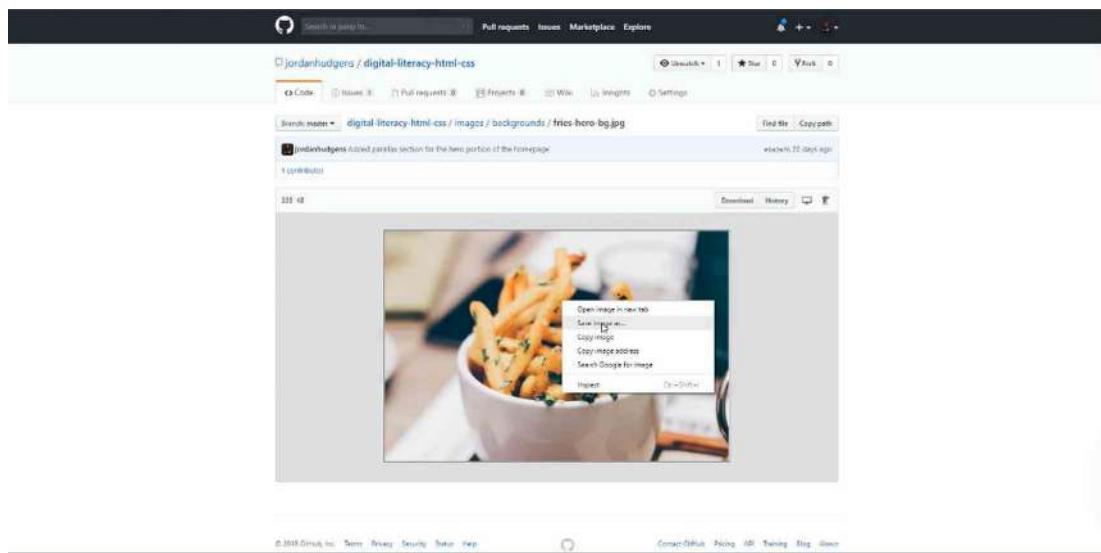
# 1.26 Parallax Scrolling

If you do not know what a parallax feature is, it's our background image. And whenever the user scrolls, you can see how it looks like the image actually stays in the background and looks like the rest of the site is just scrolling on top of it. That is what a parallax feature is.



That's what we're going to build. We have a starter where we have the nav bar and then nothing underneath it. So we're going to add the image into this blank space.

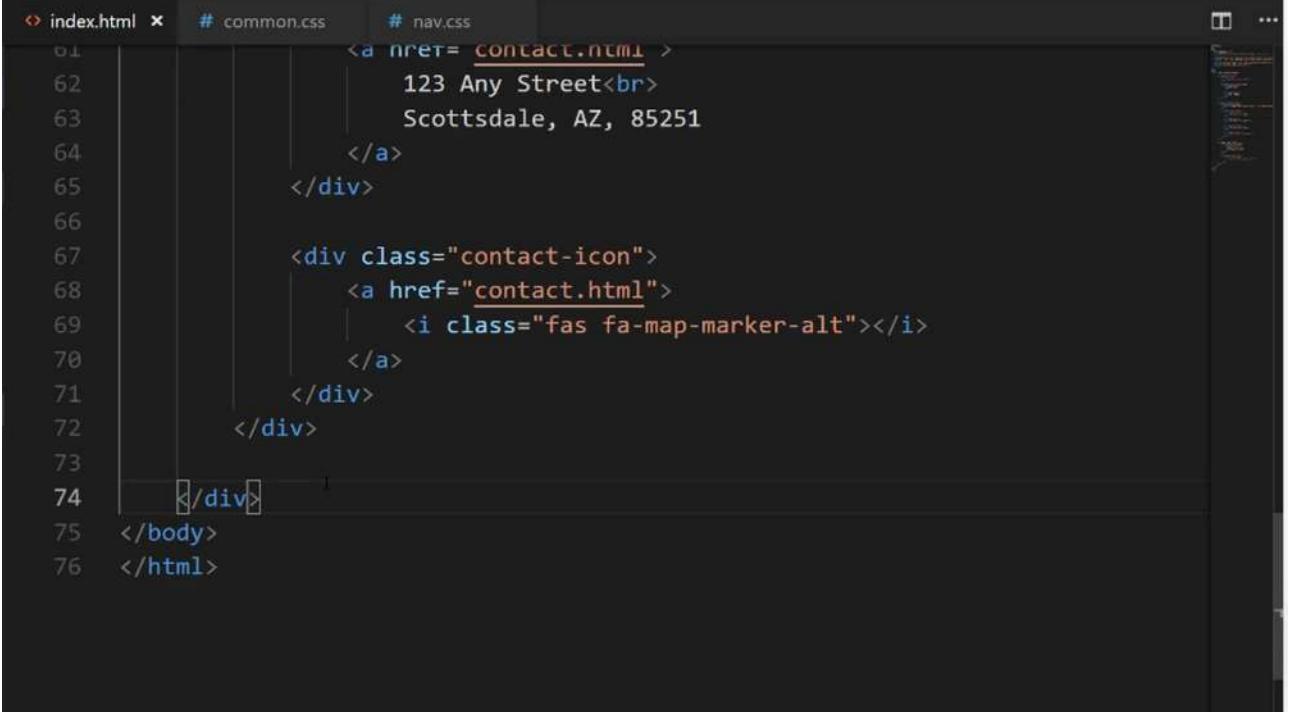
Now you can use any image that you personally want to have, or you can get it from the show notes. I'm going to go to a link and I'll include this link in the show notes as well. Inside of the backgrounds directory, I have a file called `fries-hero-bg.jpg`.



Go to this link and you can just right click, Save Image as, and then save it into your project. So I'm going to add it to images. I'm not going to put it right here, I'm going to create a new directory and let's call this Backgrounds just so we're organizing all of our code properly.

So we have `fries-hero-bg.jpg`, and this is what we're going to use for our demo. Once again you can use any image you want. I will say you probably want to have something that is around this size just because this is going to be similar to the actual size on the screen.

Now that we have that image let's add our HTML structure. I'm gonna switch over into the code here and below where that nav bar is and you can see where the closing tags are.



```
index.html # common.css # nav.css
61
62     <a href="contact.html">
63         123 Any Street<br>
64         Scottsdale, AZ, 85251
65     </a>
66 </div>
67
68     <div class="contact-icon">
69         <a href="contact.html">
70             <i class="fas fa-map-marker-alt"></i>
71         </a>
72     </div>
73 </div>
74 </div>
75 </body>
76 </html>
```

You can see that this is where the navigation wrapper ends. Here I'm going to create a new div.

## index.html

```
<div class="hero-section">
    <div class="top-heading">
        <h1>HTML Styled Fries</h1>
    </div>

    <div class="bottom-heading">
        <h3>Coding fuel.</h3>
    </div>
</div>
```

Believe it or not, that is all of the HTML code that we are going to have to write. Everything else is going to be set using CSS. So if you hit refresh you can see that we have the content there.



Now we're going to switch over to CSS and we're going to be able to add that parallax feature. So I'm going to create a dedicated file here for homepage styles. So in the styles directory, we'll create a file called `homepage.css` and then I will make sure to import it here at the very top of the file.

The screenshot shows the VS Code interface with the following details:

- OPEN EDITORS**: 1 UNSAVED
- index.html** is the active editor.
- HTML-CSS** is selected in the sidebar.
- Outline** view is visible at the bottom.

The code in the editor is:

```
<meta charset="utf-8" />
<title>Homepage</title>
<meta name="viewport" content="width=device-width, initial-scale=1">

<link href="https://fonts.googleapis.com/css?family=Roboto" rel="stylesheet">
<link href="https://fonts.googleapis.com/css?family=Ubuntu+Condensed" rel="stylesheet">
<link rel="stylesheet" href="https://use.fontawesome.com/releases/v5.2.0/css/all.css" integrity="sha384-hWVjflwFxLzJGZB1n5J3Pd9EtOOGUkqD9HhEg7QF0XtqPfC9yZ" crossorigin="anonymous">
<link rel="stylesheet" href="styles/common.css">
<link rel="stylesheet" href="styles/nav.css">
<link rel="stylesheet" href="styles/homepage.css">



</head>
<body>
    <div class="navigation-wrapper">
        <div class="left-column">
            <div class="icon">
                <i class="fas fa-phone-volume"></i>
            </div>
        </div>
        <div class="right-column">
            <h1>Fries & More</h1>
            <h2>Your Local Fast Food Restaurant</h2>
            <p>We offer a variety of delicious meals, including burgers, sandwiches, salads, and more! Come visit us today!</p>
            <div class="contact-hours-wrapper">
                <div>
                    <h3>Contact Information</h3>
                    <ul>
                        <li>Address: 123 Main Street, Anytown USA</li>
                        <li>Phone: (555) 123-4567</li>
                        <li>Email: info@friesmore.com</li>
                    </ul>
                </div>
                <div>
                    <h3>Opening Hours</h3>
                    <ul>
                        <li>Monday - Friday: 11:00 AM - 9:00 PM</li>
                        <li>Saturday: 10:00 AM - 10:00 PM</li>
                        <li>Sunday: 11:00 AM - 9:00 PM</li>
                    </ul>
                </div>
            </div>
        </div>
    </div>
</body>
```

Let's go into `homepage.css` I'm going to add a comment here and say this is the styles for the hero section. Then let's just grab that class. The first thing we want to do is to grab that image.

### homepage.css

```
/* Hero section styles */
.hero-section {
    background-image: url(../images/backgrounds/fries-hero-bg.jpg);
}
```

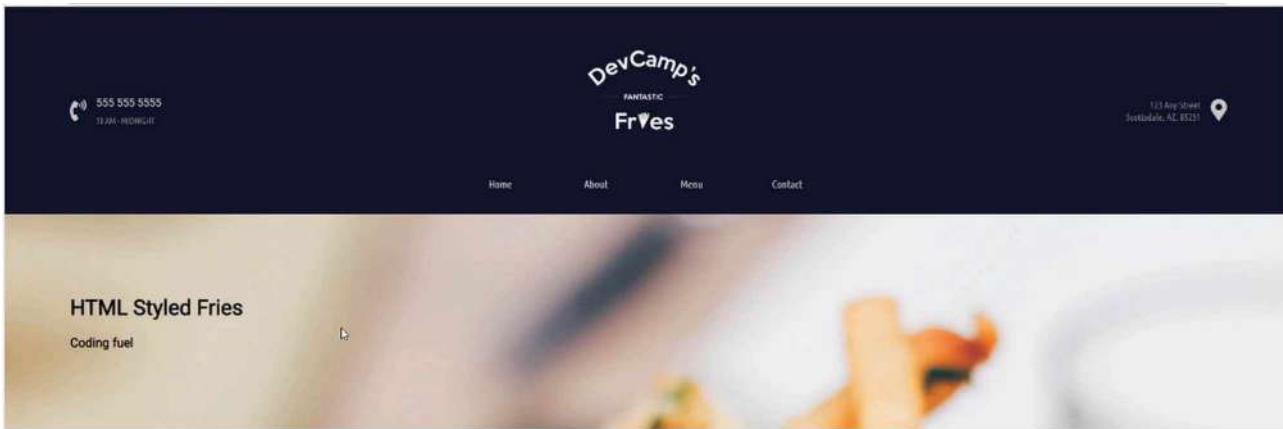
If you're using VS code, VS code kind of cheats for you a little bit. I remember back when I first started coding it was not anywhere near this easy but as you can see it pulls up some set of valid options.

Now that we have that, now we want to add some padding because if you look here you can see that these two headings here are flush up against the left and the top so I want to add a little bit of padding here.

### homepage.css

```
/* Hero section styles */
.hero-section {
    background-image: url(../images/backgrounds/fries-hero-bg.jpg);
    padding: 100px;
}
```

When I hit save you can see we don't really have what we're looking for, it just gets the top of those fries and that's not exactly ideal.



What I want to do is I want to hard code a height. One other item of note, the only reason why we even have this height already is because we have the padding and these headings, so it's simply going off of the content. That's not what we want. We actually want to dictate the height.

I'm going to add a height value here.

### homepage.css

```
/* Hero section styles */
.hero-section {
    background-image: url(../images/backgrounds/fries-hero-bg.jpg);
    padding: 100px;
    height: 300px;
}
```

So now that we have that let's actually build the parallax feature.

### homepage.css

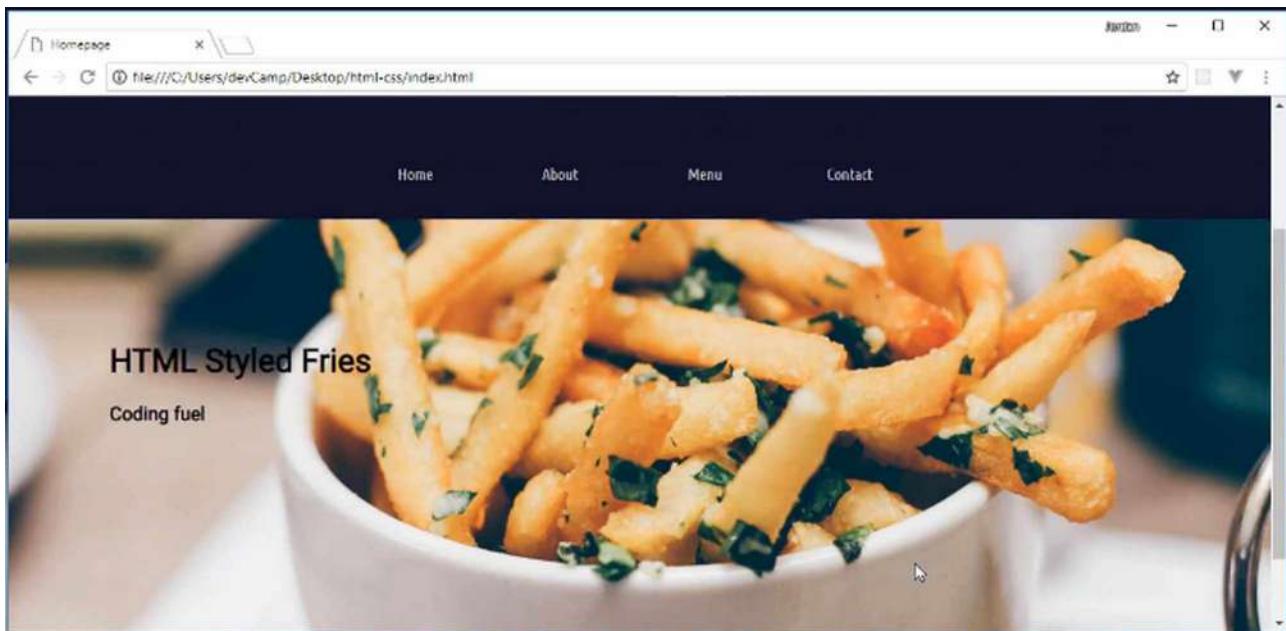
```
/* Hero section styles */
.hero-section {
    background-image: url(../images/backgrounds/fries-hero-bg.jpg);
    padding: 100px;
    height: 300px;
    background-attachment: fixed;
    background-position: center;
    background-repeat: no-repeat;
    background-size: cover;
}
```

We'll walk through what each one of these represents as well, but for now, hit refresh, you can see that our image is centered.



Now it's kind of hard to tell that the parallax feature is working because we don't actually have any scrolling. So let me pop this open here really quick in a new browser window where we have a little bit more control over the size.

So let's shrink this down just a little bit. There you go. OK. So now watch what happens. Do you see how the image there is now held in place even when we're scrolling?



So that is the full parallax feature. This is all working really nicely and that's all of the code that you needed to implement to build a parallax.

Now let's switch back and walk through exactly what this is doing. We've talked about background image. I think that's kind of self-explanatory, that's just grabbing a path to the URL. Then we added some padding for the content and then we declared a hard-coded value for the height.

Now the background-attachment. What this is saying is that we want the image to be fixed. We do not want it to slide up and down the way, an image normally would. Now the background position, this takes the image and it just centers it, so we could position it however we want.

I want the majority of the middle of the image to be shown, so I had it centered and then by default images actually repeat. So a background image if you just want it to be shown one time you have no repeat.

Now, if you had someone who has a widescreen monitor, and you didn't set it to no repeat, They would actually see multiple versions of the image.

Lastly, we have this really cool little feature called `background-size: cover;`. What this does is this will take the image and it shrinks it down and also crops it so that it's able to be shown as we see in our browser.

That is something that is really cool. I use the background cover quite a bit because what it does is it maintains the aspect ratio but it crops the image. So it is centered and so it's just grabbing the very middle of the content and it's doing kind of like a digital crop of that image.

Now if you're working with an image and it performs the wrong type of crop then you may need to edit the image itself or something like that. But for most cases, I've found that that works quite nicely. So in review you now know how to implement a full parallax feature in HTML and CSS.



## Coding Exercise

For the below div, add a background image with the url set to `assets/images/puppies.png` and note that the assets folder is two folders behind. Set the padding to `430px`, and the height to `375px`. Position the image to the center and also set the no-repeat value.

```
<div id="product-header"></div>
```

# 1.27 Parallax: Text overlays

In this lesson, we are going to finish out the styles for this hero unit. We're going to style up these two heading elements.

Now you may notice, if you compare what we have with the design we need to implement, you can see that we're using a custom font. Now that font is called **Roboto Slab**. We can go to [fonts.google.com](https://fonts.google.com) and we can pull that up.

I'm going to type in **Roboto Slab** and you can see that this has a few nice little designs that really make the font stick out a little bit more than if we just use **Roboto** by itself. I'm going to click on the + sign. Let's add this.

The screenshot shows the Google Fonts interface. At the top, a dark header bar displays '1 Family Selected'. Below it, a sub-header says 'Your Selection' with a 'Clear All' link. To the right are two red navigation arrows. The main content area shows the 'Roboto Slab' font family selected, indicated by a red minus sign icon in a grey button. Below this are two tabs: 'EMBED' (in red) and 'CUSTOMIZE' (in pink). A green button labeled 'Load Time: Fast' is also present. The 'EMBED' section contains instructions for embedding the font into an HTML document, showing the following code snippet:

```
link href="https://fonts.googleapis.com/css?family=Roboto+Slab" rel="stylesheet">
```

The 'CUSTOMIZE' section contains CSS rules for specifying the font family:

```
font-family: 'Roboto Slab', serif;
```

At the bottom, a note states: "For examples of how fonts can be added to webpages, see the [getting started guide](#)".

This is good practice, just because it allows us to repeat the same type of task. You'll really get in the habit of knowing what you have to do whenever you want to bring in a custom font. So I'm going to bring that in and what I want to do now is I actually want to have all the headings on the entire site to use this font.

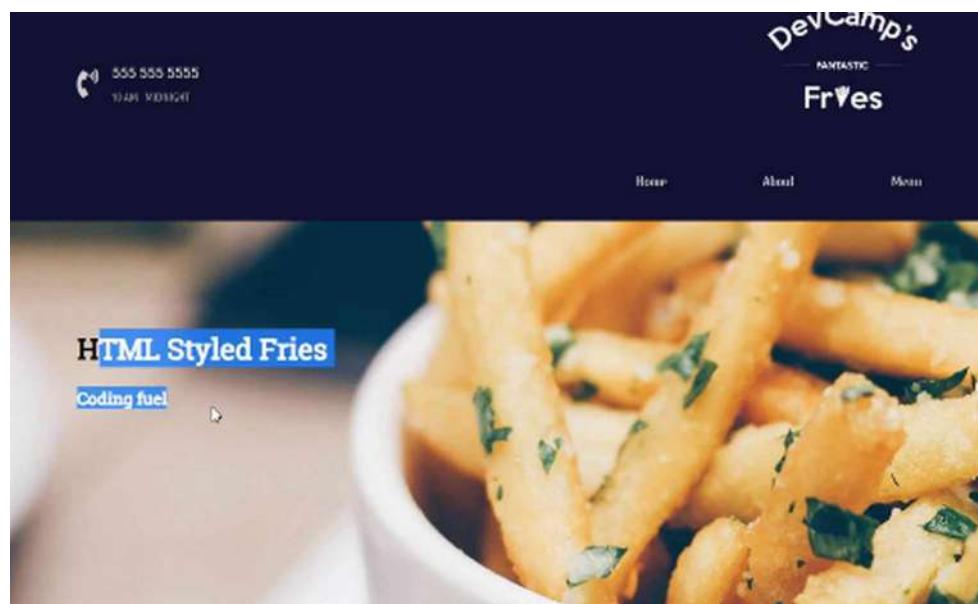
There's a cool way of doing that. I'm going to go into **common.css** here and let's grab this **font-family** call. The way that you can have multiple elements all using the same style is just by listing them all out.

I want all of the headings that I might use to all use the same font. I can say **H1**, **then H2, H3, H4, H5, and H6**. I have never gone any further than an **H6** and it's pretty rare that I even do anything beyond an **H4**.



```
index.html      # homepage.css      # common.css ✘      # nav.css
1  /* Common styles */
2  body {
3      margin: 0px;
4      font-family: 'Roboto', sans-serif;
5  }
6
7  h1, h2, h3, h4, h5, h6 {
8      font-family: 'Roboto Slab', serif;
9  }
```

This just makes sure that all of our headings are going to use the correct font. Now if I save this and switch back, if we hit refresh, we should get the correct font. and there you go. Now you can see that our headings have that cool font.



Now let's go back to the code and inside of our homepage, let's add the other styles. What we're going to do is we're going to add that blue and then that gold type of wrapper around. Just to make these pop out a little bit more. Let's see what we need to do in order to get that done.

We have our `.hero-section` and then the next element we want to select is the `.top-heading`.

The top heading, we want to add a `background-color`. I'm going to use the hex color of `#11122B`. You'll see that gives us that dark blue. For the color, I want to go with that off-white once again, which is `#CBCBCB`. Then I want to have a `width` of `290px`.

### homepage.css

```
.hero-section > .top-heading {  
    background-color: #11122B;  
    color: #cbcfcf;  
    width: 290px;  
}
```

We're going to talk about a new attribute that we've not used before called the `border-radius`. If you've ever seen some kind of element on a page that has nice, little-rounded corners, a `border-radius` is how you can do that.

So I'm going to say `'border-radius'`, and we're just going to go with a small one. The only reason for this is because I just don't want to have a sharp edge around these boxes.

### homepage.css

```
.hero-section > .top-heading {  
    background-color: #11122B;  
    color: #cbcfcf;  
    width: 290px;  
    border-radius: 2px;  
}
```

Now if we come back here and hit refresh, you can see that that's working. Now it's not perfect yet, and in order to get this exactly to the final version, we just need to add a little bit of padding.



HTML headings, by default, have a few padding elements and margin elements that they are going to come with. What we need to do is just to add onto those. So I'm just going to say `padding` here. I want to add `1px` to the top and bottom, and then I want `20px` on the left and right.

### homepage.css

```
.hero-section > .top-heading {  
  padding: 1px 20px;  
  background-color: #11122B;  
  color: #cbcbcb;  
  width: 290px;  
  border-radius: 2px;  
}
```

Now if I come back, hit refresh, you can see that that is working really nicely. That's exactly what we wanted to implement. Now that we have that, now let's come and let's grab the next one. This is going to be the `hero-section` once again, and then it is going to be the `bottom-heading`.

I'm going to use a `background-color` with that kind of goldish color, which is `#CEA135`. From there, I want another color, so for the actual font color, let's go with `#11122B`.

### homepage.css

```
.hero-section > .top-heading {  
  padding: 1px 20px;  
  background-color: #11122B;  
  color: #cbcbcb;  
  width: 290px;  
  border-radius: 2px;  
}  
  
.hero-section > .bottom-heading {  
  background-color: #CEA135;  
  color: #11122B;  
}
```

Now if you're curious about how I seem to know those, I have these written down on a different screen so that I can always reference them. Do not feel like you have to have hexadecimal numbers memorized.

The process that I personally follow is before I build out a project, I usually will pick out kind of a style guide. I'll pick out the colors that I want to use throughout the project, and then I just keep those handy. I just have those as some notes as we're going through and we're building this out.

This gives you as you can see, pretty much the exact same background color that we have here, which is also our navigation color. That is going to be the color for the font and then from there, we just need to add some padding.

Once again, I'm going to go with the same `1px` top and bottom, `20px` left and right. I need to have some margin to separate the bottom heading from the top headings. So I'm going to say, `margin-top` and for this let's add just a little bit, so `15px`.

## homepage.css

```
.hero-section > .bottom-heading {  
    background-color: #CEA135;  
    color: #11122B;  
    padding: 1px 20px;  
    margin-top: 15px;  
}
```

The width here is going to **107**, which is something that I played around with when I was building the original version of this and **107** was the perfect one. We're also going to use a border-radius of **2**.

## homepage.css

```
.hero-section > .bottom-heading {  
    background-color: #CEA135;  
    color: #11122B;  
    padding: 1px 20px;  
    margin-top: 15px;  
    width: 107px;  
    border-radius: 2px;  
}
```

Once again, that is just to give us a little bit of a rounded edge. If I hit refresh, you can see that that is working really nicely. This is going to be all we need to do for the entire **hero-section**.



In review, what we covered in this guide is how we can add custom fonts or really any kind of styles to all of the different elements, so multiple elements. We did it for all the headings on the entire website, but you could do it to any other kinds of tags or classes just by separating them with commas.

Then we styled those headings with some custom colors and background. In the next guide, we are going to keep on moving down the line. We're going to now build out this cool little grid right here.



## Coding Exercise

**Apply these styles to main-header:** Using only the padding property, apply 7px to the top and bottom and 32px to the right and left. A background color of #fa42c3 and a width of 125px.

**Apply these styles to the sub-header:** Using the padding property, apply 4px to the top and bottom, and 15px to the right and left. Add a background color of #7c75c7 and make the text color white.

```
<div class="product-info">
  <div class="main-header">I'm the main header</div>
    <div class="sub-header">You guessed it! I'm the sub header</div>
</div>
```

# 1.28 HTML char./ASCII “&” icons

In this lesson, we are going to add the HTML content and structure for these three elements.

We're going to set it up so that we can create a grid, and then from there, we can set up these columns however we want. We can use FlexBox. We could use CSS Grid. The behavior is going to be what we have right here. Where we essentially have one big wrapper div and then we have three columns inside of that.

Let's get started on that. We're not going to worry about the background image or anything like that. My own personal process, whenever I have a feature like that to build, is first to just get the HTML code out there, get a basic structure, and then from that point, I can start actually building in the styles and making it look good.

Let's come down, and this is going to be after the **hero-section**. What I'm going to do is create a new div here. I'm just going to call it the **features-section** because these are the features for the website. I'm also going to do something that may seem a little bit weird here.

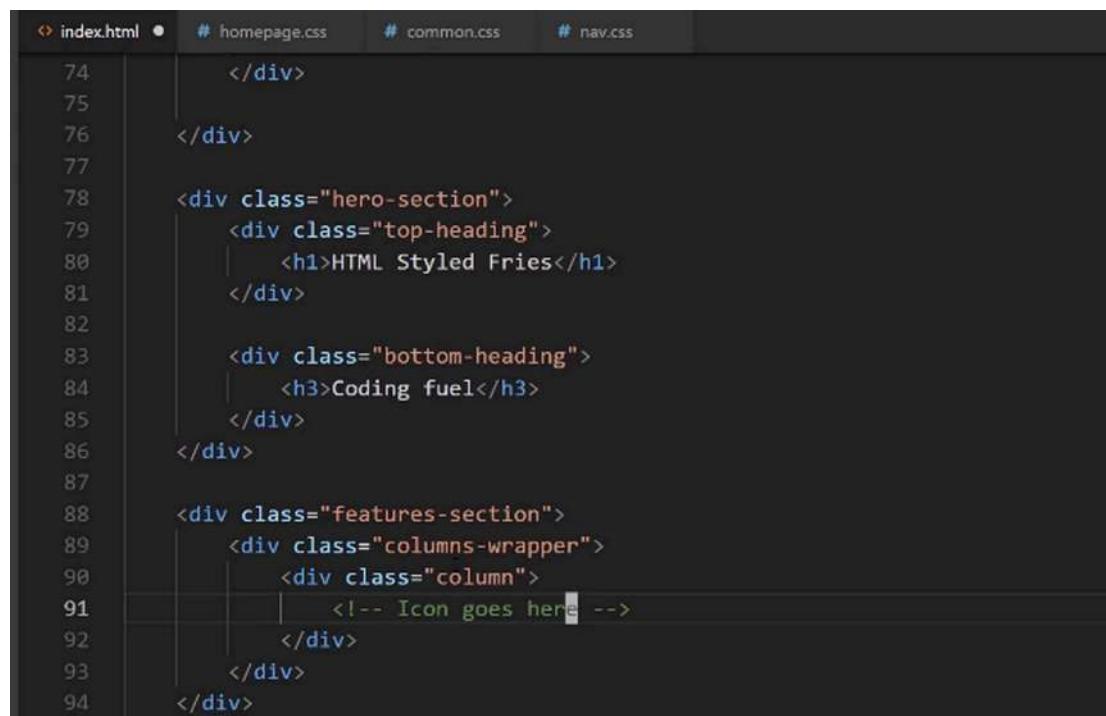
I'm going to explain why I'm going to do it. I'm going to create a **columns-wrapper**. The features section is kind of going to be like the wrapper for the entire div. So, you can see all the way from left to right and then top to bottom. That's going to be the features section.



Then from there, I want to create a columns wrapper that's going to be about right from here. It's only going to contain the content. Then inside of that, that's where we're going to add each one of these columns.

The reason why I want to do that is because, if you remember, whenever we're working with tools like Grid or FlexBox, all that those tools care about is the immediately nested div element. If we were to only have one wrapper div, we're not going to have as much control over where we want to align these items.

I'm going to right here create another wrapper div, and we'll just call this the `columns-wrapper`. This is where we're going to put in our content. We'll have one column, and then inside of here, I'm just going to put a little comment and say, `icon goes here` just so we know about it and we don't forget.



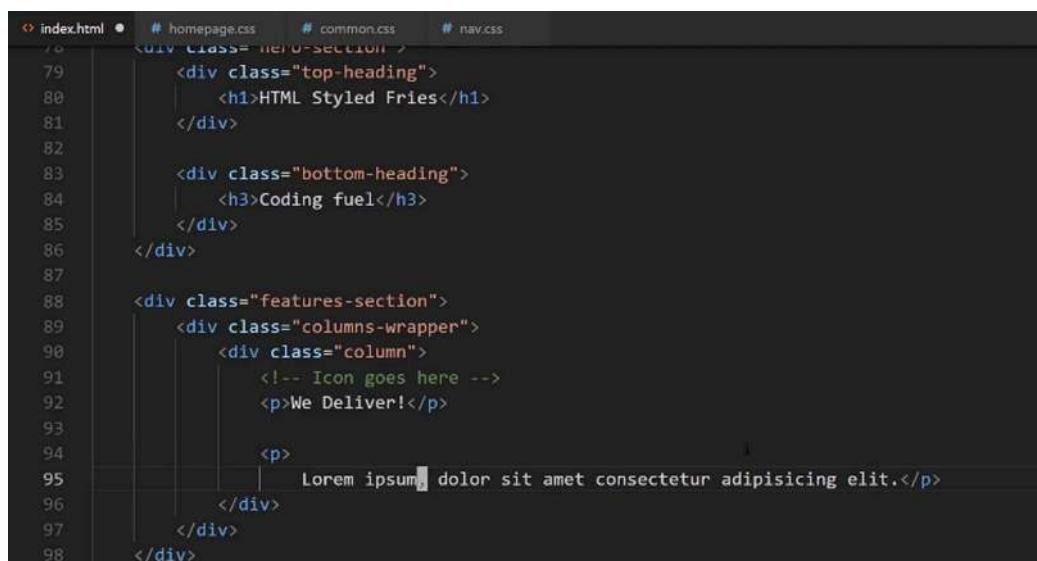
```
index.html • # homepage.css # common.css # nav.css
74      </div>
75
76    </div>
77
78    <div class="hero-section">
79      <div class="top-heading">
80        <h1>HTML Styled Fries</h1>
81      </div>
82
83      <div class="bottom-heading">
84        <h3>Coding fuel</h3>
85      </div>
86    </div>
87
88    <div class="features-section">
89      <div class="columns-wrapper">
90        <div class="column">
91          
92        </div>
93      </div>
94    </div>
```

Then let's have a `p`-tag, and say `p` for paragraph, and we'll just say `We Deliver!`. I believe that's what we have if we switch back. Yes. It says, "We Deliver." Then that's in the paragraph tag.

I'm using the paragraph tag because I want to teach you another trick about selectors later on. That's going to be in a paragraph tag and the next one is also going to be in a paragraph tag.

If you ever wonder, if you've ever seen a template site, and you see this `Lorem Ipsum` text, well that is just Latin, and it's a way that developers are able to implement their own placeholder content.

If you're using a tool like Visual Studio Code, there's actually this really neat little feature. I can say, `p>` and then here I'll say something like, I believe it is ... Let me see. Is that `Lorem`? I think it's something like `Lorem 8`, and if I hit `tab` ... Yes. That worked.



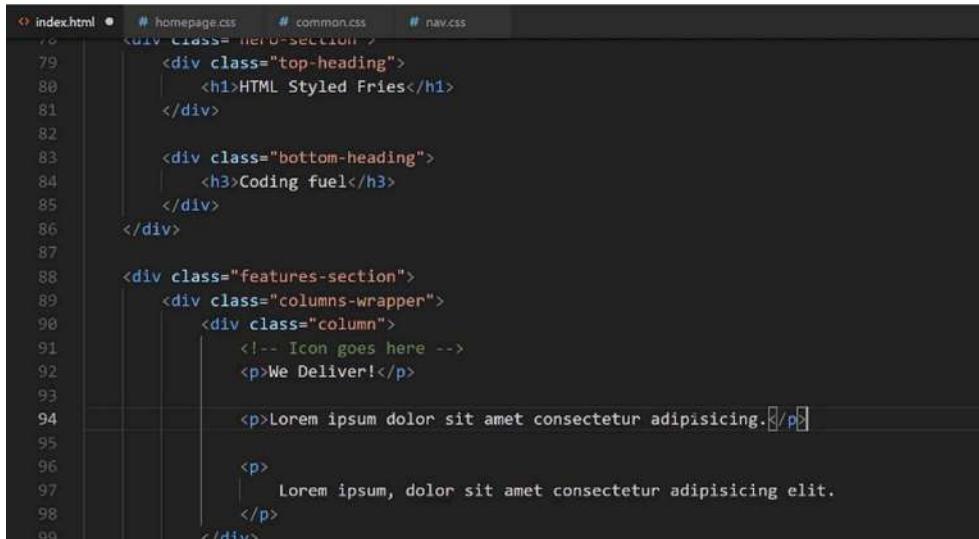
```
index.html • # homepage.css # common.css # nav.css
79      </div>
80
81    <div class="top-heading">
82      <h1>HTML Styled Fries</h1>
83    </div>
84
85    <div class="bottom-heading">
86      <h3>Coding fuel</h3>
87    </div>
88
89    <div class="features-section">
90      <div class="columns-wrapper">
91        <div class="column">
92          <!-- Icon goes here -->
93          <p>We Deliver!</p>
94
95          <p> Lorem ipsum dolor sit amet consectetur adipisicing elit.</p>
96
97        </div>
98      </div>
```

It's been a while since I've had to do that. I usually use Vim for a lot of my other developments. I don't use the Lorem shortcut, but that worked perfectly. What I did there, and I'll do it again just in case, because it can be a really, really handy little tool.

I gave the name of the tag just like if I did a `p`-tag and then did a tab just so I would be able to have that P-tag created. So, I said, `p` and then greater than, and what emmet does is, this is something specific to emmet. If you're using the emmet Plugin with any other development environment, this should also work.

We are saying that inside of the `p`-tag that I want you to create I want you to put this  text. If I did it just like this, this would still work, but it would give me too much content. So, you can see of you just go with Lorem by itself, it gives you all of this content. I don't want that much.

As you can see, if you also, you also have the ability to say how much content you want. So, I can say, `p>Lorem` and then if I say just 7. That'll give me seven words, just like that. I did 8 because I thought 8 would give me the amount I wanted.



```
<!-- index.html -->
<div class="hero-section">
  <div class="top-heading">
    <h1>HTML Styled Fries</h1>
  </div>

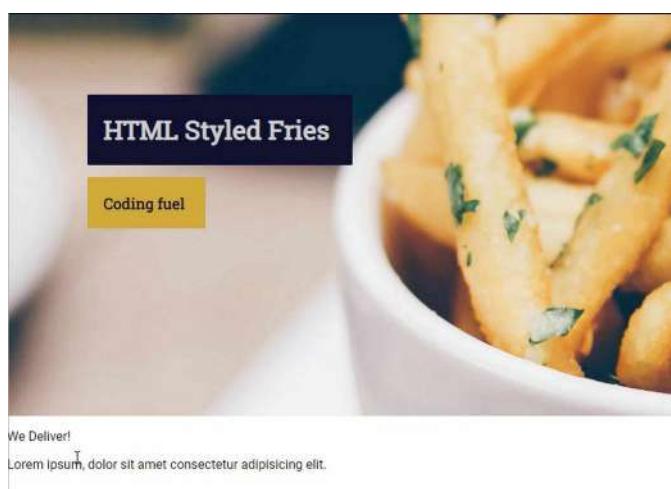
  <div class="bottom-heading">
    <h3>Coding fuel</h3>
  </div>
</div>

<div class="features-section">
  <div class="columns-wrapper">
    <div class="column">
      <!-- Icon goes here -->
      <p>We Deliver!</p>

      <p>Lorem ipsum dolor sit amet consectetur adipisicing elit.</p>
      <p>Lorem ipsum, dolor sit amet consectetur adipisicing elit.</p>
    </div>
  </div>
</div>
```

If I hit save there, and oh I'm on the page and it looks like it's all done. We did a great job. It all went by so fast. Anyway, so here we have the first column and we have our demo content. That's

working perfectly.



The icon will go there. Now, let's just go, and we're just going to clone this and then once again. Now, let's just add in the additional content. For this one, I believe it said something like, You can code from here and then on the other one, I believe it was like the fry count or something like that. Let's just verify.

You can see that right here we don't have anything else. I'm not sure if I saved it yet. Yeah, it's You can code from here and then 100+ types of fries. Not that this matters. You can put anything you want, but I like to match the design whenever I can.

```
index.html # homepage.css # common.css # nav.css
93
94      <p>
95      |   Lorem ipsum, dolor sit amet consectetur adipisicing elit.
96      |</p>
97      </div>
98
99      <div class="column">
100     |   <!-- Icon goes here -->
101     |   <p>You can code from here!</p>
102
103     <p>
104     |   Lorem ipsum, dolor sit amet consectetur adipisicing elit.
105     |</p>
106     </div>
107
108     <div class="column">
109     |   <!-- Icon goes here -->
110     |   <p>100+ types of friens!</p>
111
112     <p>
113     |   Lorem ipsum, dolor sit amet consectetur adipisicing elit.
114     |</p>
```

Hit save and now if we go back to our version we should be able to hit refresh and have all of that content there. Yes, it is ugly, but this is exactly what we need. Technically, we actually have, except for the icons, we have all of the content and all the HTML structure that we're going to need in order to have that section look like this.



We Deliver!

Lorem ipsum, dolor sit amet consectetur adipisicing elit.

You can code from here!

↓  
Lorem ipsum, dolor sit amet consectetur adipisicing elit.

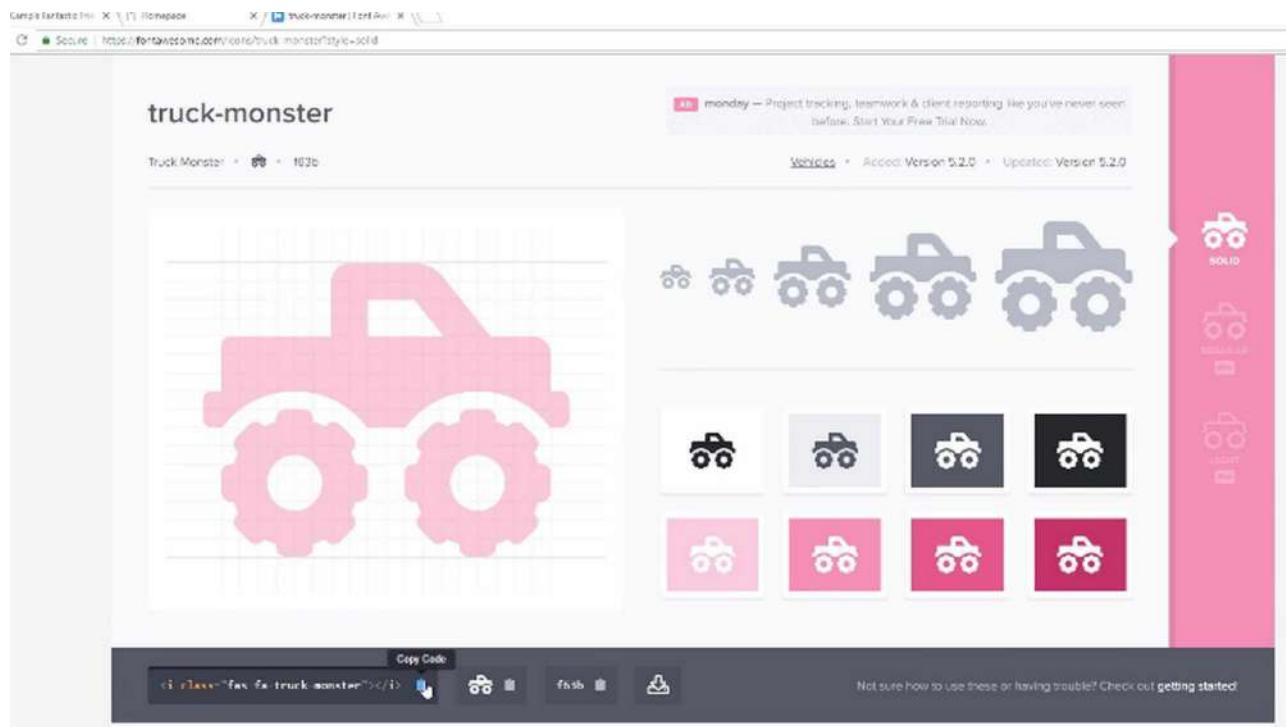
100+ types of friens!

↓  
Lorem ipsum, dolor sit amet consectetur adipisicing elit.

That is one of the very amazing things about CSS is that think about how much different this looks compared to what we have here. And the only difference really is just the CSS. That shows how important it is.

We've been going for a few minutes now, but before we take a break I say let's just go and let's go grab those icons. Let's go into [Font Awesome](#), and the reason is just because this way we'll be able to have 100% of our HTML code done.

The very first one is that monster truck for delivering. Let's search for **truck**. We have a few different types of truck. I like this little **monster-truck**. I thought he looked pretty cool. You can use anything you'd like though.



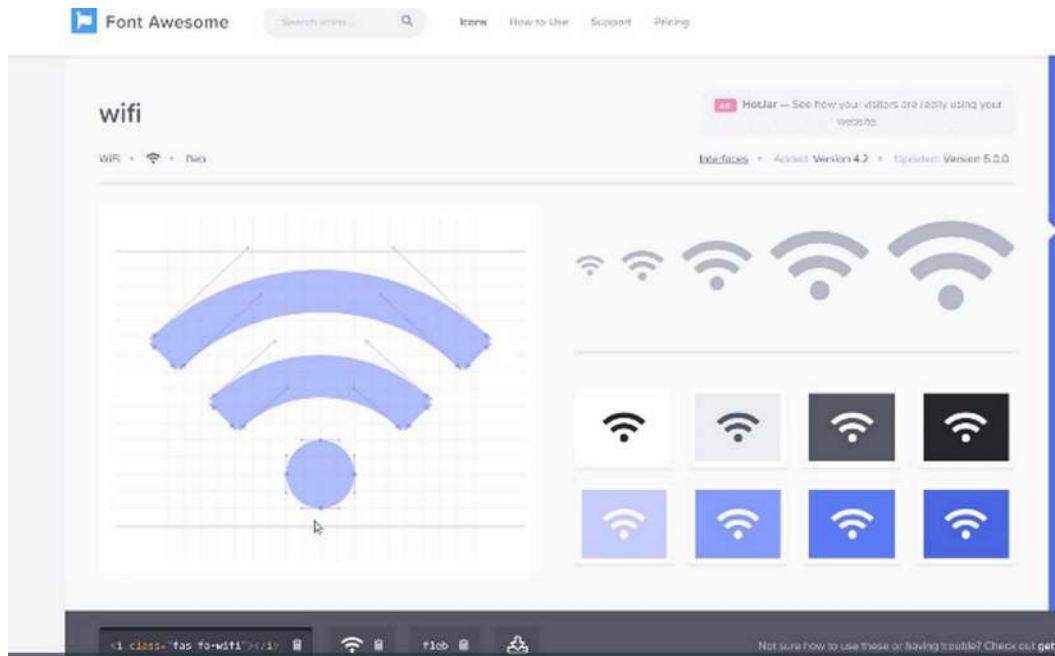
I'm going to copy that code here. We've already done this before, but it never hurts to practice it. So, I'm going to get rid of that comment, and paste it in.

A screenshot of a code editor showing a file named 'index.html'. The code is as follows:

```
87
88     <div class="features-section">
89         <div class="columns-wrapper">
90             <div class="column">
91                 <i class="fas fa-truck-monster"></i>
92                 <p>We Deliver!</p>
93
94                 <p>
95                     Lorem ipsum, dolor sit amet consectetur adipisicing elit.
96                 </p>
97             </div>
98         </div>
```

The line '91' is highlighted with a blue selection bar.

Then for this next one, this is a WiFi icon. We'll say, WiFi. There are a few different options. This is kind of the most universal one. I'll click and copy that in, and then we just have one more.



We have 100, oh, I misspelled that, 100 types of fries. Now the fry one, I believe that was some type of chart. Yes, that is a line graph. Let me see, if we type chart, I believe it comes up. I could cheat and I could just right-click on here and it would tell me, but I feel like that's cheating.

I want to show you the exact process I followed. When I was building this out was I originally created a design and I had some wire frames. Then I created some more high-res designs to have exactly what we have here. So, I want to walk through the exact process that I followed, and that's what I did right here.

We have chart, and that one is here. Yeah, chart - line. If I copy this and now I come down here. We now have that line chart.

```
99      <div class="column">
100        <i class="fas fa-wifi"></i>
101        <p>You can code from here!</p>
102
103        <p>
104          Lorem ipsum, dolor sit amet consectetur adipisicing elit.
105        </p>
106      </div>
107
108      <div class="column">
109        <i class="fas fa-chart-line"></i>
110
111        <p>100+ types of fries!</p>
```

If I hit save, then we should now see our icons. Click refresh, and there you go. You have icons right here.



We Deliver!

Lore ipsum, dolor sit amet consectetur adipisicing elit.



You can code from here!

Lore ipsum, dolor sit amet consectetur adipisicing elit.



100+ types of fries!

Lore ipsum, dolor sit amet consectetur adipisicing elit.

We have all of the HTML structure that we need, and now it's time to start styling it.



## Coding Exercise

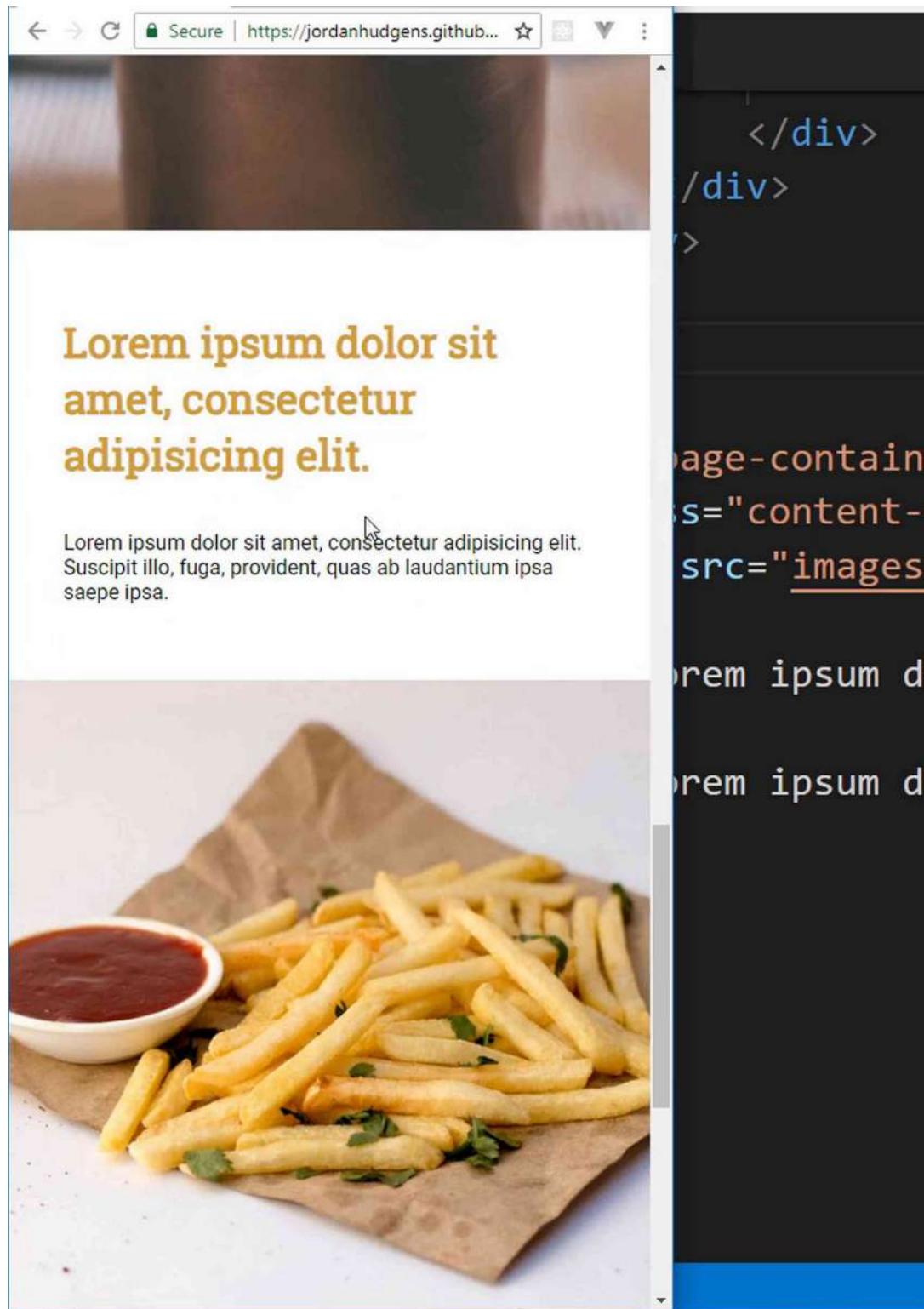
Create 3 divs each with a class name of `block`. Inside of each of those div's, place an image tag.

Set the first img tag's source path to `assets/dog.jpg`. The second to `assets/parrot.jpg`.

And the third to `assets/kiwi.jpg`.

# 1.29 :hoover Animations

Just as a quick refresher, we want them to look something like this.



Now, I'm not going to implement the shadow because I'm going to do that in a dedicated guide. Because working with box shadows is not as easy as you may think, especially if it's the first time you've done it, so I want to spend an entire section just dedicated to that. So we're just going to

implement the styles, we're going to use Flexbox, and we're going to see how we can align these all and style them properly.

Now, the background image here is an image that you have access to in the show notes. So, if you go to this [link](#) and then images, backgrounds, and then this one is called fries-multiply-bg.jpg, if you go to that and just click Save Image As, I'm going to put it inside the project, inside images, and then backgrounds. Hit Save, and that is all we need to do. It's just a very small jpeg image, and we want it to be the background.

Now, if you remember back to the parallax section, do you remember how I told you we did not want this image to repeat but that the default nature of background images was actually to repeat? That is what we're going to do here, we're going to use that default and have this background image multiplied throughout the page.

Now that we have access to that, let's start going and building these styles. So I'm in the homepage, this was the hero section styles. Let's come down here, and I want to now add a new comment, this is going to be the features section styles.

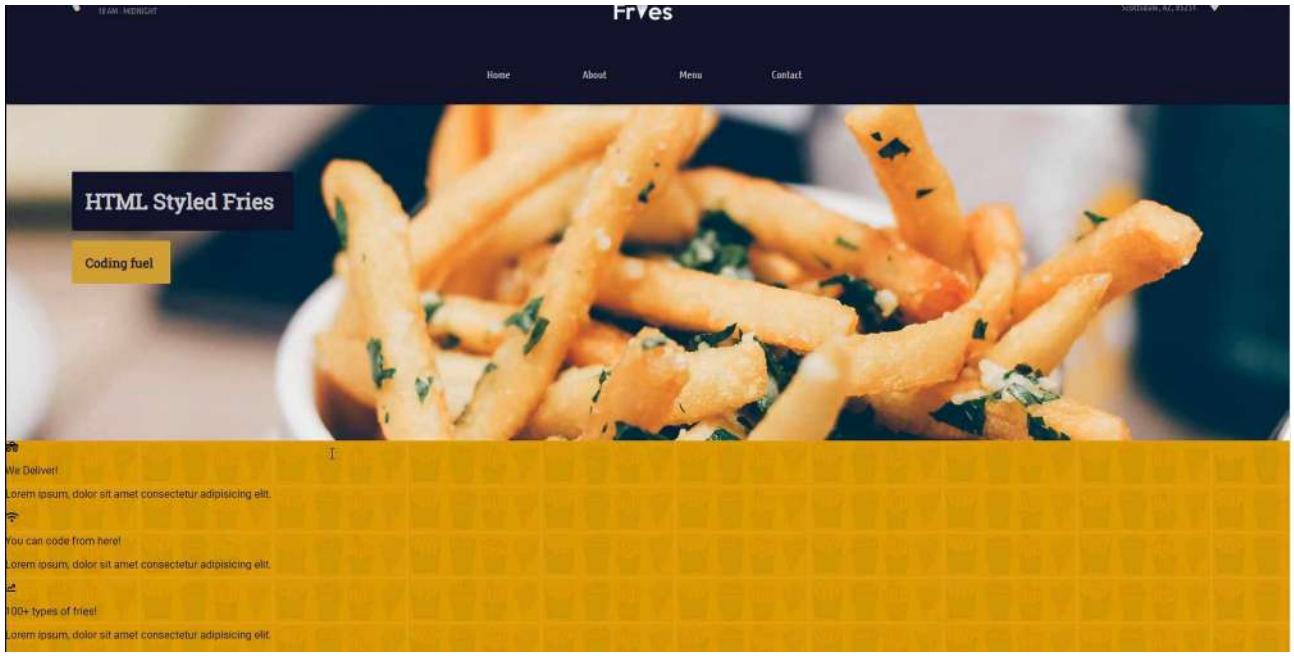
Now let's start with that class of features-section. We want this to have a hard-coated height, kind of like we did with the parallax. So I want this to be 400 pixels, and then we want to get access to that background image, so I'll say background image URL, and then we want to go, if you remember, we need to jump back into the previous directory, then go to images, backgrounds, and we want the fries-multiply.bg, and then from here we also want to have a background color.

Part of the reason for this is we want to have something around those borders, and we want to have something that is there in case the image does not load. So for that, I want to go with that kind of goldish color we've been using up to this point.

## homepage.css

```
/* Features section styles */
.features-section {
    height: 400px;
    background-image: url('../images/backgrounds/fries-multiply-bg.jpg');
    background-color: #cea135;
}
```

Then let's go see what this does for us. Let's switch back here, hit refresh, and there you go, that was pretty easy.



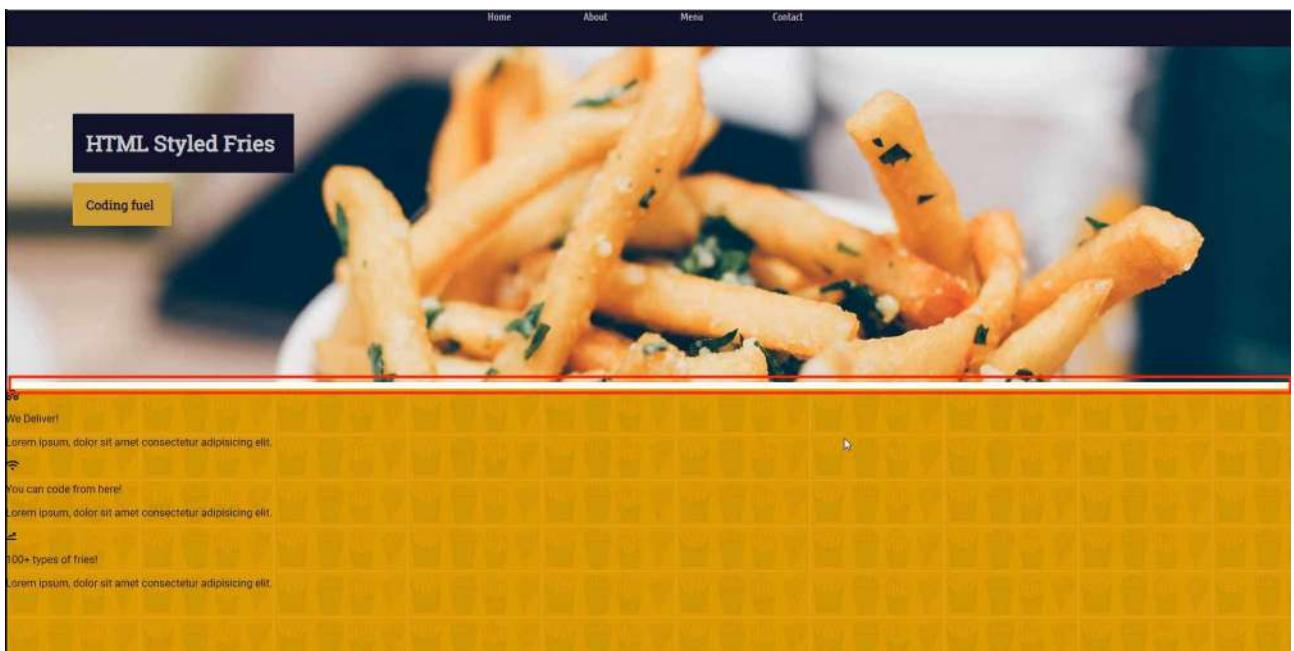
You can see that that background image is now multiplying, and it's looking really nice.

Now, I also want to have a border here. Do you notice how this parallax works, but I kind of like the idea of having some way of separating the two besides just going and jumping right into this background image, so I want to add a border. Now, we've not used borders up until this point so this will be a nice little introduction to them, but we will get into them more later on as well.

So here, I want to give a border-top property, and here you can pass in a number of values to the border. I want it to be pretty thick, so I'm going to say 10 pixels, and then from there I want it to be solid. You could do things like dash lines or anything like that, but I want this one to be solid, and I want it to be pure white. I'm going to have it just like that,

```
border-top: 10px solid white;
```

and now if you hit refresh, you can see we have this nice little separator,



and that also shows how well that parallax feature is working, so that is looking nice.

Now, moving down the line, let's give ourselves a little more room, I want to have that dark blue for the font color, not columns, color. For that, if you remember, we have it up there, it's that hex color of #11122b. That's going to give all of our fonts and all the text that nice dark blue color.

It's kind of hard to tell the difference between that and black, especially on a screen, but I think that especially when we make the images or the icons larger, that's going to make a difference. So we have that color, and now let's turn this into a flex container.

So I'm going to say display flex and then from there, I want to justify content. Now, remember how we added a single div inside of this flex container, so these styles are only going to be applying to that one wrapper div. I'm going to say justify content, and this is going to be center, because I want it to be centered horizontally, and then align items because I want it to be centered vertically.



So that's going to be centered, just like that, and then let's go with our font family. This one's going to have the Ubuntu Condensed. I wish we had code completion for this one because I do spell condensed wrong quite a bit. I believe that's right, condensed, and then this is going to be sans-serif.

## homepage.css

```
/* Features section styles */
.features-section {
    height: 400px;
    background-image: url(../images/backgrounds/fries-multiply-bg.jpg);
    background-color: #cea135;
    border-top: 10px solid white;
    color: #11122b;
    display: flex;
    justify-content: center;
    align-items: center;
    font-family: "Ubuntu Condensed", sans-serif;
}
```

We'll find out in a moment if I have a typo there. Hit Refresh, and there we go.



The font is getting pulled in properly. That is looking good, and that's all of the styles that we're going to add to this feature

section. Later on, we'll come back, and we'll add a box shadow to it, but for right now, that's all that we need.

Now that we have that, now let's go and let's grab the columns wrapper. If you reference back to the HTML, you remember that we have the feature section. That's going to be a Flexbox container, then we have columns wrapper, which is the only element inside of the feature's container. This is what's going to do all the magic for us and get us those columns.

You could use grid for it, but I personally like Flexbox for this because it has the space between and space around properties, and I like using those. So let's come here, and let's grab the features section, and then we want to go with the columns wrapper.

Now, I want to display a hard-coded width here, so I want to say that this is always going to be a 1000 pixels wide, and then it's going to be a flex container by itself, and then we'll use justify-content here, and this one is going to be space-between.

### homepage.css

```
.features-section > .columns-wrapper {  
  width: 1000px;  
  display: flex;  
  justify-content: space-between;  
}
```

Let's hit save, and let's see if that did what we're asking it to do. Hit refresh and there you go, that is looking much better.



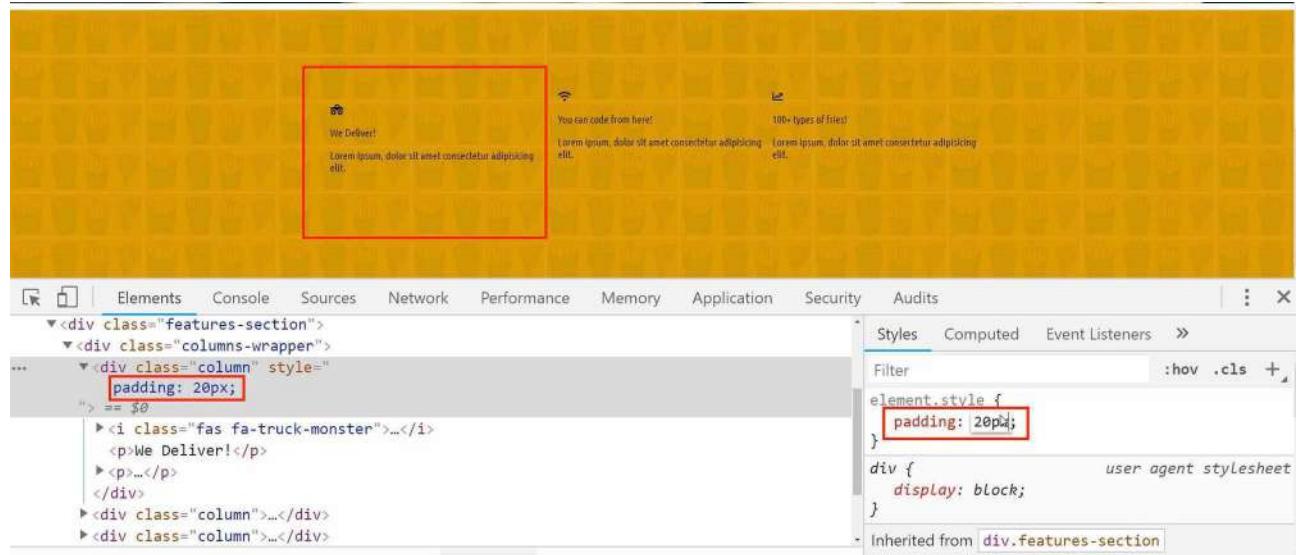
Notice how much easier it was that we've done this now a few times. We've used Flexbox for the nav bar, for nav elements, and we've done it so many times that now you can just go through it, and it starts to become a habit. And I've been doing this, and I've been using Flexbox and CSS grid for a while now.

And I can tell you, I had to constantly look up all the property names when I was first learning it. So don't feel intimidated if you don't have everything memorized, that's perfectly normal. The more times you do it, it's just going to become second nature, it's just like any other skill, so let's now move on. Those are all the styles that we need for the columns wrapper.

Now let's move to the elements themselves, and we have that column class, and let's do something a little bit different than we've done before. I want to show you how you can add all of these values

here and how you can do it right in the inspector so that you can test out to see how these are going to look.

Right here, we have this column class, and inside the column class, if we wanted to add, say, padding of 20 pixels, you can see that that affects all of them just like that, that's working.



One thing to note, this is only getting applied to this first column specifically, so it's going to throw off these other two, but that's perfectly fine.

Now we're going to say we want the margin, just the margin itself to be at 42 pixels. So far so good. Then I want the text align property to be centered. I want all the content to be centered, and then I want the border bottom, and this is going to sound really weird to you, but there is a method to the madness, I promise. It's going to be solid and transparent. You're like, "Why in the world did we just put a transparent border there?" Well, I will explain in a second, so don't worry.

Now, a border radius of 10 pixels, and then let's have a transition, so this is going to have a nice little animation of one second. That is everything that we need.



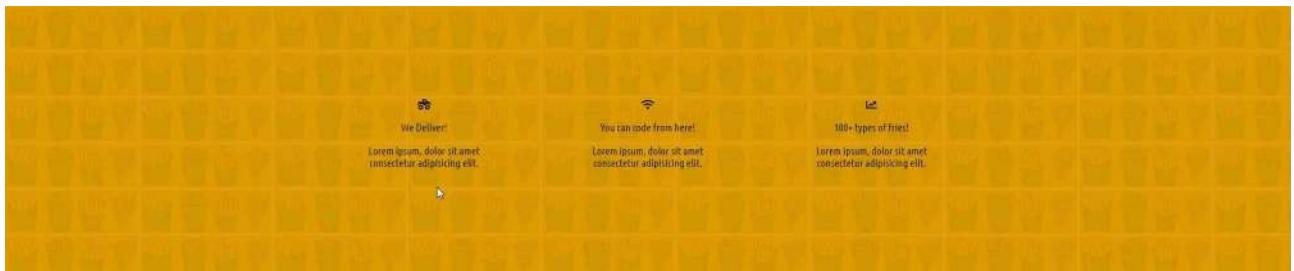
Now, obviously, there's still a lot other things we need to do there, but for right now, those are all the values. And this is something that I do quite a bit when I'm building out interfaces where if I want to see what the code is doing in real time, I can type it all right here into the little element style dialogue box, and I can then afterwards just copy it and then come over and paste it directly into the code.

So I can come here, and say I want my column class, and then from there, just paste it all in and hit save.

### homepage.css

```
.column {  
  padding: 20px;  
  margin: 42px;  
  text-align: center;  
  border-bottom: 5px solid transparent;  
  border-radius: 10px;  
  transition: 1s;  
}
```

Now, if I hit refresh, you can see that that has now been applied to all of the values, so that is looking really good.



Now, before we take a break and we finish out some of these different styles in the feature section, I want to show you why I added that weird, transparent border. So that may look really weird, but if you remove it, you may notice it makes it jump a little bit, and here, you can see a hint of what we're looking to do, but you can't actually see anything on the screen, but I'm going to show you exactly why it's needed here.

If we switch back, and I'm also going to be a little bit more specific with the column because if I ever want to use column for anything else, I do not want these values overwriting that. So I'm going to go with the features section, and then the columns wrapper, and then the column.

Then below this, what I want to do is I'm going to add a new pseudo listener, I'm going to add column : and then hover. Up to this point, the only thing we've used a pseudo selector for has been for the link tags, but you can use these types of hover listeners and these pseudo selectors for anything on the page.

So right here, I'm saying that if you ever hover over one of these columns, I want you to perform this task, and so the task that I want to do is going to be to add a bottom border, except now it's not going to be transparent. Now we're going to grab that dark blue color here.

## homepage.css

```
.features-section > .columns-wrapper > .column:hover {  
    border-bottom: 5px solid #11122b;  
}
```

Now I'm going to hit save and hit refresh. Now, if I hover over here, do you see how we have that cool little black border?



What I was going for with the design was I wanted it to look like a restaurant tray. Since this is a fry restaurant, they have trays, and I thought it'd be cool if the features looked like they're being held up on trays. You can see that's working perfectly.

Now, if you're curious on why we needed the transparent border, watch what happens if I remove the transparent border rule right here. If I hover over it, do you see how all of the code gets lifted up? That is really ... That does not look good. That is ... You do not want to go and give this to a client or to a boss or anything, that is really is bad.

What we did by having a transparent border, what we did was we told the page to always have the border there, but for it to be transparent, so that means that that 5 pixels on the bottom, that's already there in place right now. It's reserved, and then all that happens when we hover is it changes the color. So that is a really nice way of removing those annoying types of jumps, so that is how you can add a dynamic and animated border to an element in CSS.



## Coding Exercise

We've used Flexbox to move the below elements to the center, but now we need to spread them apart evenly. In the below box, place the missing properties that would spread out the elements with an even amount of space between them.

```
.card-container {  
    width: 1100px; background-color: #000000;  
    display: flex;  
    align-items: center;  
    font-size: 13px;  
    color: white;  
  
    /*add your code here*/  
}  
  


Some stuff



some more stuff



The last stuff


```

# 1.30 ::nthChild selectors

One of the more confusing aspects of working with CSS is the nth-child selector, and if you have never even heard of that, that's perfectly fine. It looks like this. `:nth-child()` I'm going to teach it a little bit different than I've seen before.

Many of the tutorials that I've seen that try to explain it in a sense almost confuse it even more, and I think that that's a mistake, because at the end of the day the nth-child selector just gives you very granular control on a numerical basis on which items you want to select.

If hearing that definition just makes you even more confused, let's just kind of walk through a visual way of doing it.

What we're going to do is we're going to style these three elements.



If you look at the HTML, we have this column class. Inside of the column class we have the icon, then we have a paragraph tag, and then we have another paragraph tag.

If you're curious on why I didn't wrap our icon in a class or the paragraph or the other paragraph, either of these in classes, I did that on purpose because I wanted to teach you how nth-child selectors worked.

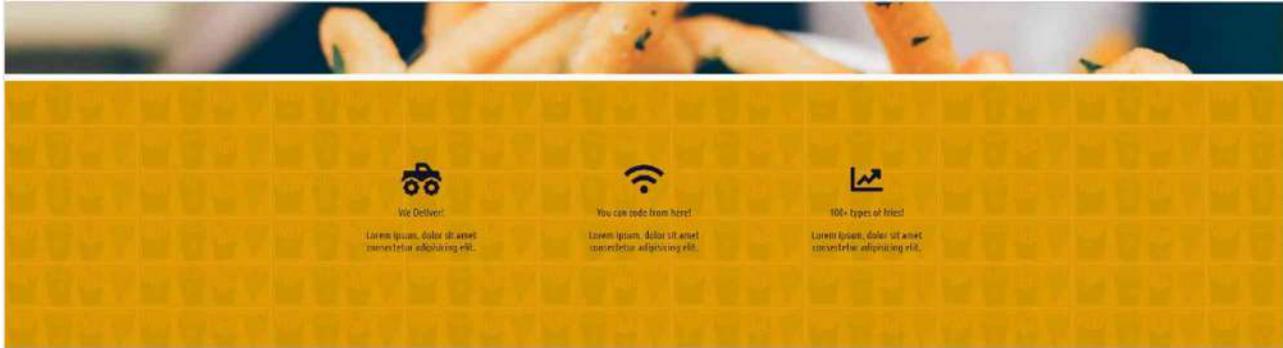
The way it works is: this column has three children. It has an icon, and then it has two paragraph tags. What the nth-child selector does is it allows you to grab each one of these elements and select them without actually having a class, an ID, or even a tag name.

With the very first one, we're going to say nth-child one. We're saying, I want you to grab me your first child. Then we're going to say nth-child two. We want to grab the second one, and then nth-child three, we'll grab the last one. That's really all there is to it. Let's now go and let's build that out.

## homepage.css

```
.features-section > .columns-wrapper > .column :nth-child(1) {  
    font-size: 3em;  
}
```

Let's go back and hit refresh.



And there we go. Now we have it working.

Now you can see we have effectively grabbed the first element here. The way that it works, imagine that if we went to the index here and one of these items, the paragraph tag, was on the top, it is the one that's going to be affected.

When I said that the way nth-child works is it gives you the ability to select by a numerical value. That's all I meant, was that you have the ability to ignore the type of tag or classes or ids, because you may not always know those, but you may know that every element is going to have three children.

Then you know how to select those automatically.

Now, technically we could have written all of this code by writing a div wrapper with the class and then adding the class to this paragraph tag and another class to this paragraph tag. It would've worked exactly the same way, but I thought this was the best way for you to understand how the nth-child works.

Now with that, let's come back to the homepage and let's grab the second child. Just grab all of the last styles we wrote. The only difference is going to be the number.

I'm going to show you an attribute we've not seen before called font weight.

Now, font weight gives you the ability to have bold and light type of text, so that is just a pretty much the same. If you've ever worked with Excel or in Word and you highlighted a word and you said, "I want this to be bold." That's really all we're doing.

## homepage.css

```
.features-section > .columns-wrapper > .column :nth-child(2) {  
    font-size: 1.5em;  
    font-weight: 900;  
    height: 50px;  
    transition: 1s;  
}
```

Let's hit save here. Hit refresh and you can see that's giving us exactly the styles that we're looking for.



The animation I want to have applied is for when we hover over the items, I want the letters to spread out a little bit, exactly like what we did up for the nav. The main reason isn't even for stylistic kind of purposes in a real-life application.

I'm not sure if I want these to be animated, but I really just wanted to show you that you can use pseudo selectors on nth children.

## homepage.css

```
.features-section > .columns-wrapper > .column :nth-child(2):hover {  
    letter-spacing: 1px;  
}
```

Hit save, hit refresh. Now if you hover over one of those words, it spreads out just a little bit, just to give a little bit of an animation, then we still keep our animation here.

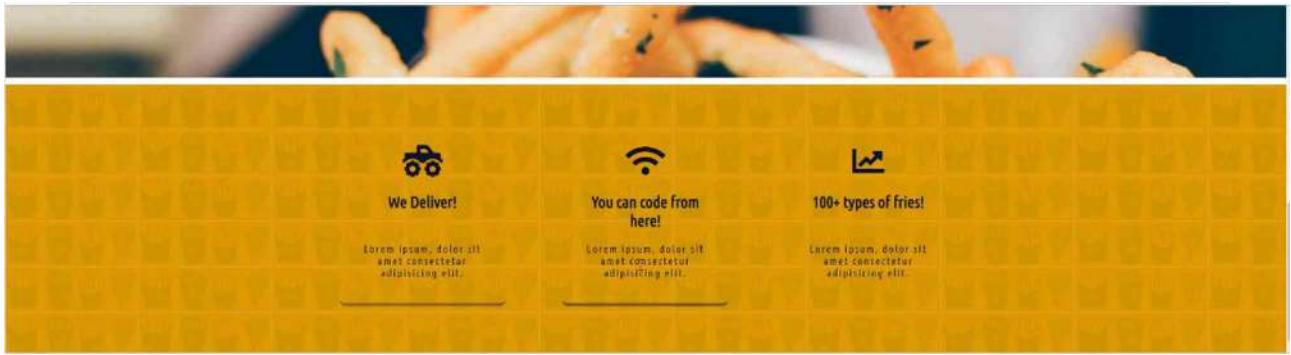
Now, one thing you may be curious about, the hover effect is only applied to the element we're hovering over, so when we hover over the Div, that little border gets added, but the text in the title doesn't actually start to get animated unless we hover over that.

That's something that is important to understand with how CSS works.

Now let's finish this out, and let's grab that last one. So I'm going to just copy all of this and now we want the third child here.

## homepage.css

```
.features-section > .columns-wrapper > .column :nth-child(3) {  
    letter-spacing: 2px;  
}
```



As you can see, that gives us the exact effect that we're looking for. Just a little subtitle and the letter spacing can make some of these letters a little bit easier to read.

Great job if you went through that, you now know how to work with nth children and hopefully it's a little bit more straightforward than some of the other approaches that are out there.

In the next guide, we are going to walk through how we can add a box shadow to the bottom of this element.



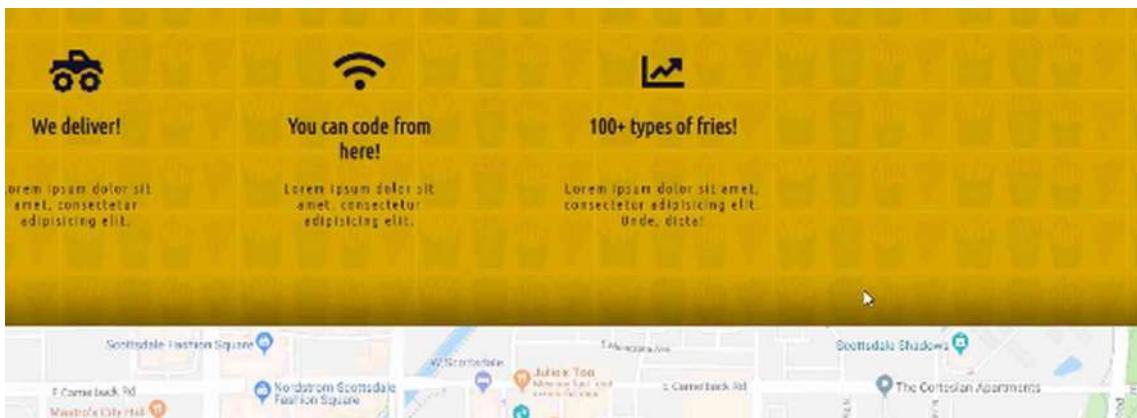
## Coding Exercise

Using Parent-Child Relationship Selectors, (don't use nth-child), give all the span tags a text color of green, a background color of #42cffa and a letter spacing of 3px.

```
<div class="parent">
  <div class="social-link-ideas">
    <span class="idea">Heres an idea</span>
    <span class="idea">Oh look heres another idea</span>
    <span class="idea">The last idea</span>
  </div>
</div>
```

# 1.31 CSS Box Shadows

Specifically, we're going to add a shadow that looks like this. Where we have a box shadow. It's called an `inset box-shadow`. What it does is it can give a little bit of depth to elements on the page.



Right here, you can see how we have the map, and because it has the box-shadow inset, inside of this section, it almost looks like we have a real-world map that is sitting on top of this. This is a feature I've been asked to build multiple times, so I wanted to include it in this guide.

I'm going to switch back to what we're going to be building it on, and the goal will be to place it right here at the bottom. Let me switch back to the code, and let's find in `index.html` where this is going to go.

We have this `features-section`. That is the CSS class that we're going to have, and we want to have the box-shadow at the very bottom here, but only on the bottom. We don't want it on the sides or the top, and so that is the element that we're going to select.

If I come to `homepage.css` here, I already have this in the code. I already have a featured section with some styles. What we're going to do is we are going to add a `box-shadow` here. We can do it with a single line of code.

I'm going to write out the entire line. I've played around with this a few times, to get it where I wanted it to be. I'm going to write it all out. Don't worry if you do not understand it. Then, we're going to dissect it afterward. I'm going to say ....:

```
index.html # homepage.css • # common.css # nav.css
27 }
28
29 /* Features section styles */
30 .features-section {
31     height: 400px;
32     background-image: url(..../images/backgrounds/fries-multiply-bg.jpg);
33     background-color: ##cea135;
34     border-top: 10px solid white;
35     color: #11122b;
36     display: flex;
37     justify-content: center;
38     align-items: center;
39     font-family: "Ubuntu Condensed", sans-serif;
40     box-shadow: inset 0px -26px 79px -8px |
41 }
42
```

Then I'm going to use `rgba`. What that allows me to do is instead of giving a hexadecimal value, it allows me to pass in the percent of red I want to use, of green, of blue. The main reason is because it gives me the ability to have an `alpha` color, which means that I can have the shadow be slightly transparent.

```
index.html # homepage.css • # common.css # nav.css
27 }
28
29 /* Features section styles */
30 .features-section {
31     height: 400px;
32     background-image: url(..../images/backgrounds/fries-multiply-bg.jpg);
33     background-color: ##cea135;
34     border-top: 10px solid white;
35     color: #11122b;
36     display: flex;
37     justify-content: center;
38     align-items: center;
39     font-family: "Ubuntu Condensed", sans-serif;
40     box-shadow: inset 0px -26px 79px -8px rgba(red, green, blue, alpha)
41 }
42
```

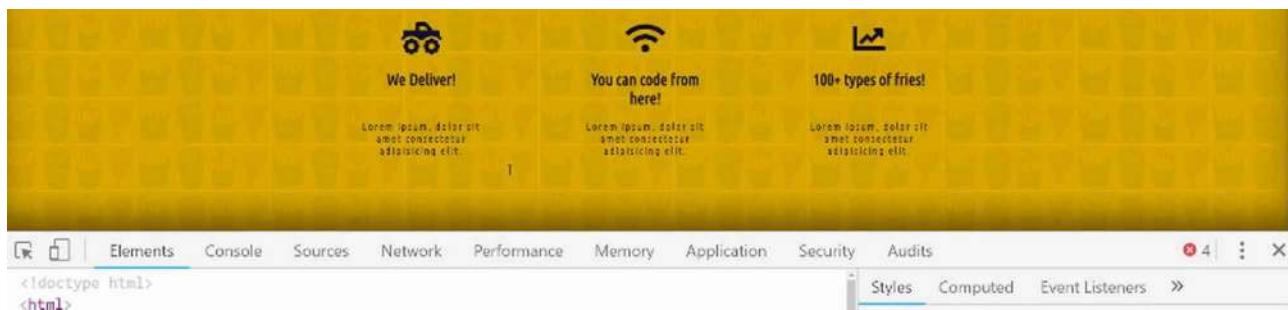
Usually, I like to use hexadecimal colors, just because I find that it's a little bit easier to type, once I have picked out the ones I want, but if I ever want to work with transparency, using `rgba` is a good option.

Here, what I'm going to do for these values is I'm going to say `0`, and then for the next one, for this green value, I'm going to go with `0` once again, and then here I'm going to say `0`, and then for the alpha, I'm going to say `0.58`.

## homepage.css

```
box-shadow: inset 0px -26px 79px -8px rgba(0,0,0,0.58);
```

Let's close that off, and let's see if that gives us what we're looking for. If I hit refresh, you can see our border's right there.



Now, if you've never worked with box shadows, then that might seem like a little bit of magic because we have all of those weird values. Your question should be, how in the world will I know what to type in? Well, that's what we're going to dive into, seeing exactly what each one of those values represents.

If I select the features section and come over here, you can see that we have each one of those values that I typed in. If I want to change one of these values, so the very first one, if I want to change this to **10**, do you see how on the left-hand side let me change it to **100**. Do you see on the left-hand side? that's what got added.



What we're doing here is we're saying, I do not want any values on that left-hand side. I want the **0px** there on the left. Now, what does this represent? Well, we have **-26px**. If I remove that, you can see that that would give us a shadow there on the top.

The screenshot shows a web browser's developer tools with the 'Elements' tab selected. The page has three columns. The rightmost column contains text and icons. The 'Styles' tab in the developer tools shows the following CSS rule for the right column:

```
div {
  align-items: center;
  font-family: "Ubuntu Condensed", sans-serif;
  box-shadow: inset 0px 26px 79px -8px rgba(0, 0, 0, 0.58);
}
```

If I say `0`, it still gives me a little bit of shadow, so that's not what I'm wanting, and so that has to deal with the up and down value and where the shadow is sitting.

The screenshot shows the developer tools again. The rightmost column now has a box shadow of `inset 0px 0px 79px -8px rgba(0, 0, 0, 0.58)`. The 'Styles' tab shows the updated CSS rule:

```
div {
  align-items: center;
  font-family: "Ubuntu Condensed", sans-serif;
  box-shadow: inset 0px 0px 79px -8px rgba(0, 0, 0, 0.58);
}
```

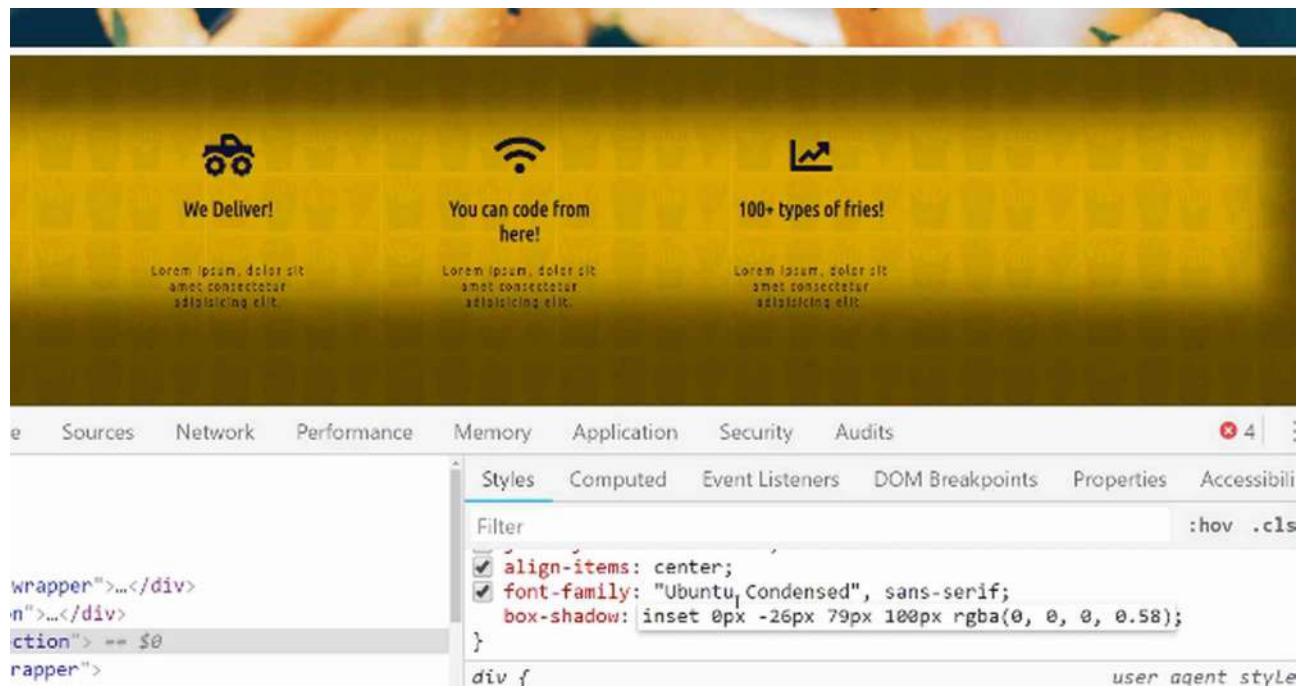
Let's move down to the next one. That's at `79`. If I change this to `0`, you can see that that deals with the level of blurring that takes place. Right now, at `0`, it pretty much is no blur. It's just like a regular border. With `79px`, that gives a nice little blur right there.

The screenshot shows the developer tools again. The rightmost column now has a box shadow of `inset 0px 0px 0px -8px rgba(0, 0, 0, 0.58)`. The 'Styles' tab shows the final CSS rule:

```
div {
  align-items: center;
  font-family: "Ubuntu Condensed", sans-serif;
  box-shadow: inset 0px 0px 0px -8px rgba(0, 0, 0, 0.58);
}
```

Okay, we're on our very last one. This was at `-8px`. If I just remove that and say it's `0`, you can see that that has to deal with the right-hand side. If I put it at `10` or even `100`, you can see that that is

bringing it in even further, but we do not want that. I think that just going with `-8px` gives exactly what we're looking for.

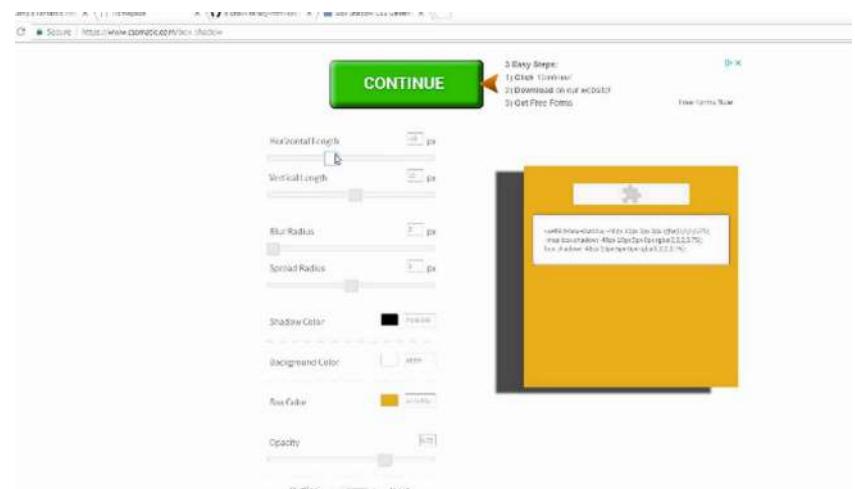


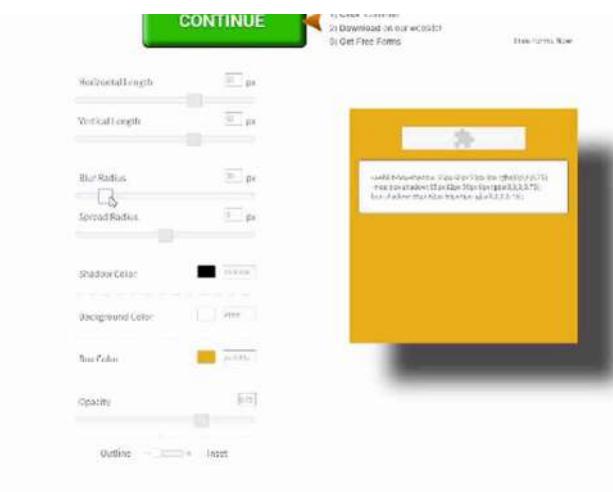
Then, once again, this color is whatever color that you want to use. Here, we're using this transparent kind of white, in a sense, or I'm sorry, transparent kind of dark, in a sense. That is giving us exactly what we're wanting. Now, that is a full walkthrough of how the box-shadow property works in CSS.

Another tool that is very helpful is there are plenty of box-shadow generators. Whenever I'm trying to come up with a new one, I'll usually use one of those. If you type in `CSS box shadow generator`, you'll see there are all kinds of these available. If you click on one of these, [CSSmatic](#) is one of the ones I use the most.

If you go, you can see it gives you a starter, and it gives you all of the code. You can see, it has each one of these values, and so I can change the horizontal value, and notice how that left side is changing, so that left property. Right now, it's at `11`. Ours was at `0`, so you can see that has to deal with the horizontal length.

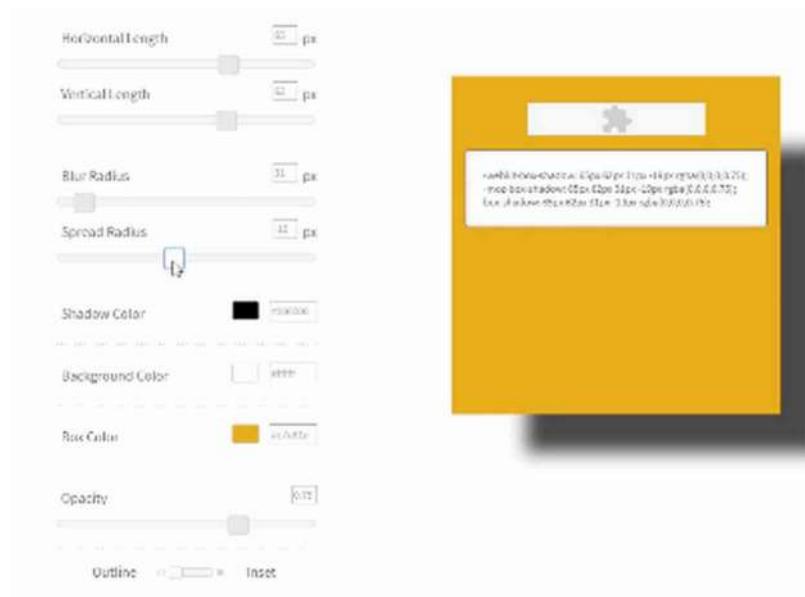
Then, the vertical length, right here, is that second property, so do you see how this is mapped directly to what we just walked through. The first item is for the horizontal length, the second one is for vertical length.





Now, let's look at that `blur radius`. Right now, it's at 5px, but if we want to blur it more, you just drag this more. You can see that you can also just change the value to whatever that you're wanting.

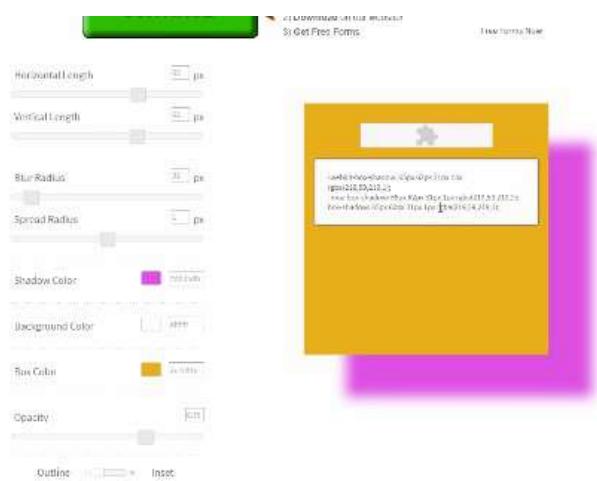
Now, the `spread radius`. This is that very last one. This is how tall or wide, or I should say, it's just the overall size, the overall area that it takes up.



You can see, with each one of those, you have the ability to control the box shadow and give it any kind of coloring that you want.

Now, we also have the colors here. You have the shadow color, so at black, this is exactly what we have, where each one of those values is 0.

If you wanted a purple one, then you can drag it there, and you can see it automatically rendered the rgba code for you.



With the opacity, you could make this even more transparent, just like this, and then you have the ability to use the last property, which is outline or inset. We used inset, but you could use outline, as well.



3 Easy Steps:  
1) Click Continue  
2) Download on our website  
3) Get Free Forms

Free Forms Now

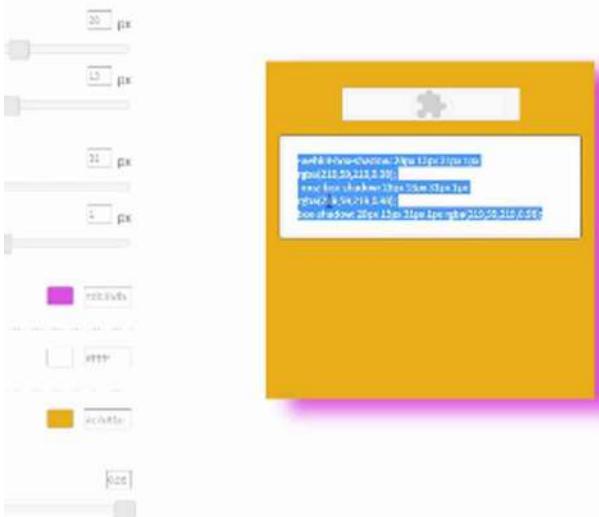
Now, I'll include a link to the [CSSmatic](#) in the show notes, but, and as helpful as this is, I don't want you to feel like you just have to rely on this. I don't want you to think that there's magic going on here.

All that is happening is this gives you a way of having a tool where you can preview what it looks like before you put it into your code.

Hopefully, you can see that this isn't magic. Each one of these properties is simply mapped to a style here. It's a way of controlling how that box-shadow looks.

Then, from there, you can copy this and paste all that code directly in. It includes the web kit and the Mozilla, which is Firefox, different styles, but from what I've seen, the box shadow for me has worked, just by typing out the box shadow.

Horizontal Length: 20 px  
Vertical Length: 15 px  
Blur Radius: 2px  
Spread Radius: 1px  
Shadow Color: #00FFFF (cyan)  
Background Color: #FFFF00 (yellow)  
Base Color: #FFFF00 (yellow)  
Opacity: 100%  
Outline: Inset



Definitely, feel free to copy all of these, because it's not going to hurt you to bring all of them into your program. Congratulations. If you went through that, you now know how to work with box shadows in HTML and CSS.



## Coding Exercise

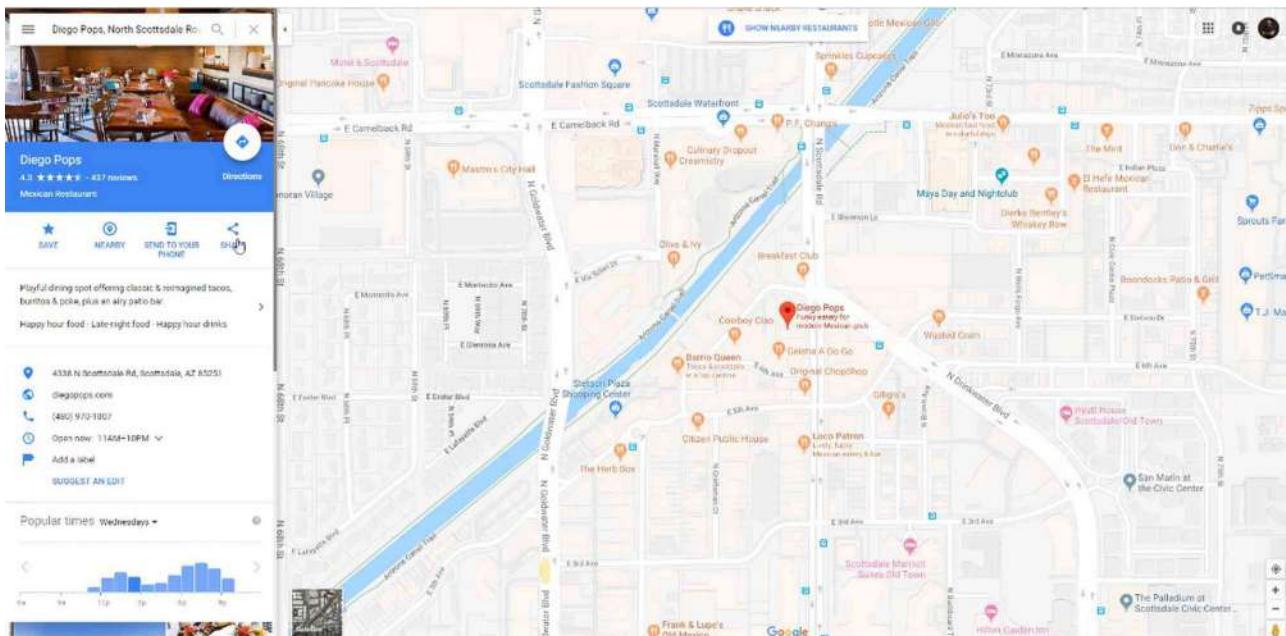
Go to the Box Shadow Generator and create a box shadow that is applied to all sides with a value of 0px. Include a blur radius of 33px and a spread radius of -4px. The shadow color should be #8f8f8f

```
.box-header{  
}  
  
<div class="box-header">  
    Hey give me an awesome box shadow!  
</div>
```

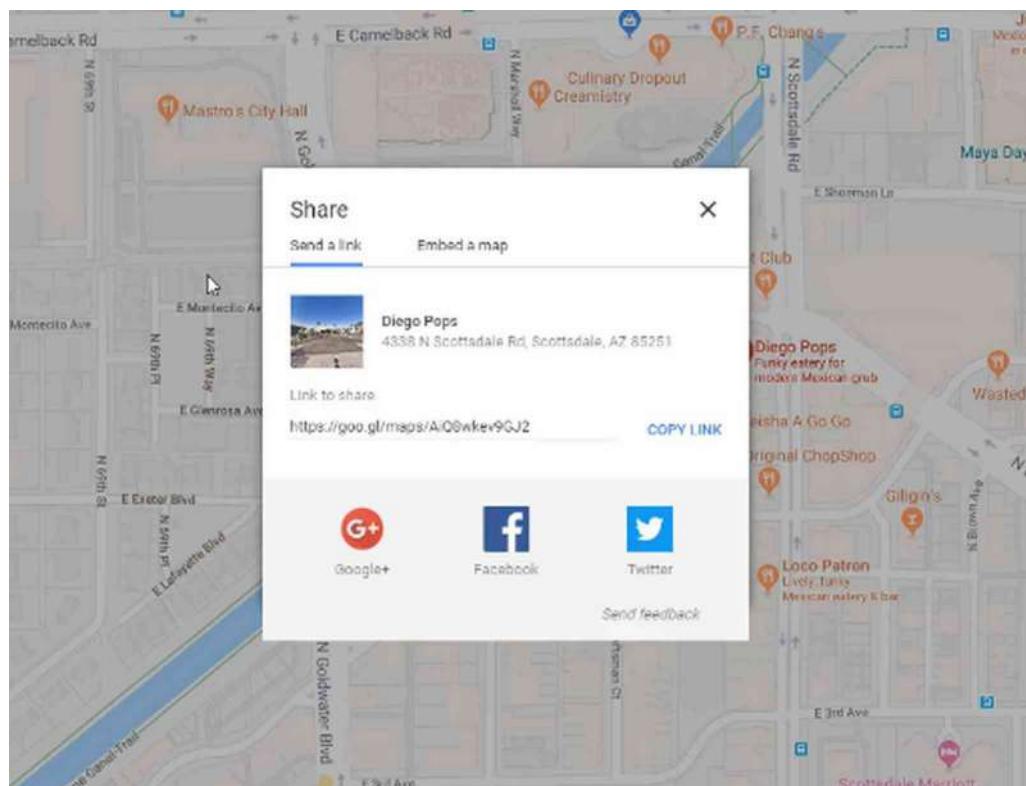
# 1.32 Embedding content: Gmaps

I'm going to go back to our project. If you go to maps.google.com, this is going to be a relatively straightforward process, even if you've never done it before. If you go and search for an address or some business, I'll pick out one of my favorites, Diego Pops in Scottsdale, Arizona.

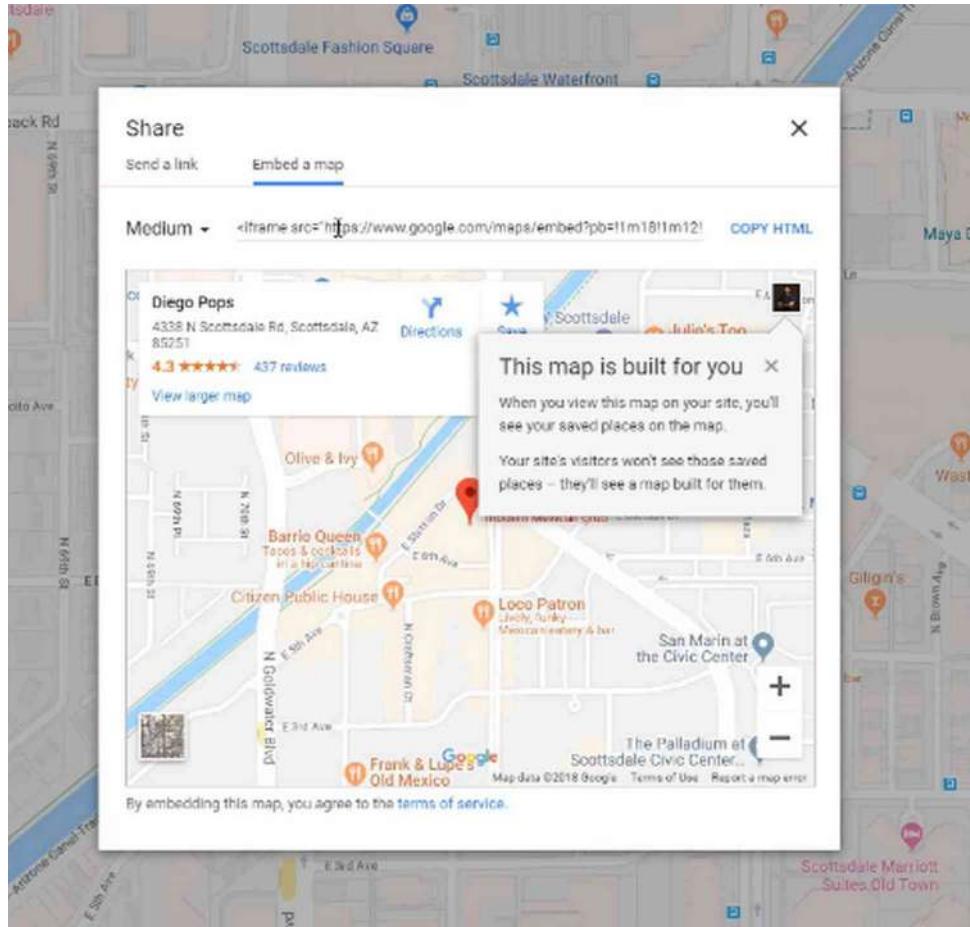
If you search for that, and then go and click the Share icon here.



This is going to bring up a little dialog box.



It gives you the ability to grab a link, but that's not what we're wanting. We actually want to embed the map. Now, the size that this gives you is not exactly ideal. It's not what we're going to want to use, but I'll show you how we can customize it.



They also have the ability to use a custom size but we don't need to do that because I want something that is going to have the ability to fit into our site and it's actually going to go bleeding edge, so it's going to go from side to side.

You can just hit Copy HTML at any of the sizes, and then we will figure it out after that.

If you switch over to the code, we can just add this in, and then we'll be able to see what needs to happen from there. We have this div of features section, so let's come here now, and I'm just going to add a wrapper div. We'll add a class later.

## index.html

```
<div>
  <iframe src="https://www.google.com/maps/embed?pb=!1m18!1m12!1m3!
  1d3327.0765176913264!2d-111.92875828479936!3d33.49938628076005!2m3!1f0!2f0!
  3f0!3m2!1i1024!2i768!4f13.1!3m3!1m2!
  1s0x872b0bbdbed13a73%3A0xfab78c0f014377d8!2sDiego+Pops!5e0!3m2!1sen!2sus!
  4v1534788183429" width="600" height="450" frameborder="0" style="border:0"
  allowfullscreen></iframe>
</div>
```

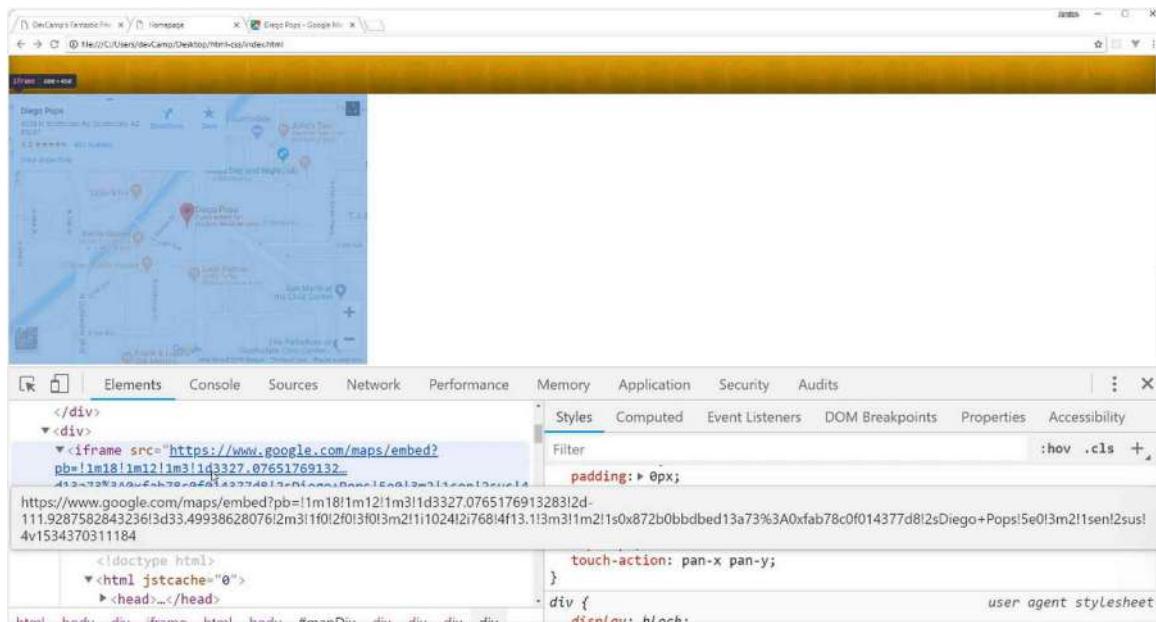
For right now, I just want to show you how it can be embedded. Hit save, and now if you switch back and hit refresh, you'll see that we do have a map right here.



Now, there are a few ways that we can customize this. For right now, let's just play inside of the browser. So, I'm going to select this, and we actually want to select the entire element. Now, what we're doing here is we are working with what is called an IFrame.

If you've never worked with an IFrame before, what that is in HTML is an IFrame gives you the ability to embed an entire website, so if you look at the code at the very top, even though this isn't what we pasted in, this is what renders on the screen.

It is an IFrame, and it is going to go and grab this URL, so it's actually pulling this entire website inside of our website. This has its own document. It has its own HTML, its head, its body, so it's like a mini-website pulled directly into ours. So, that's pretty cool.



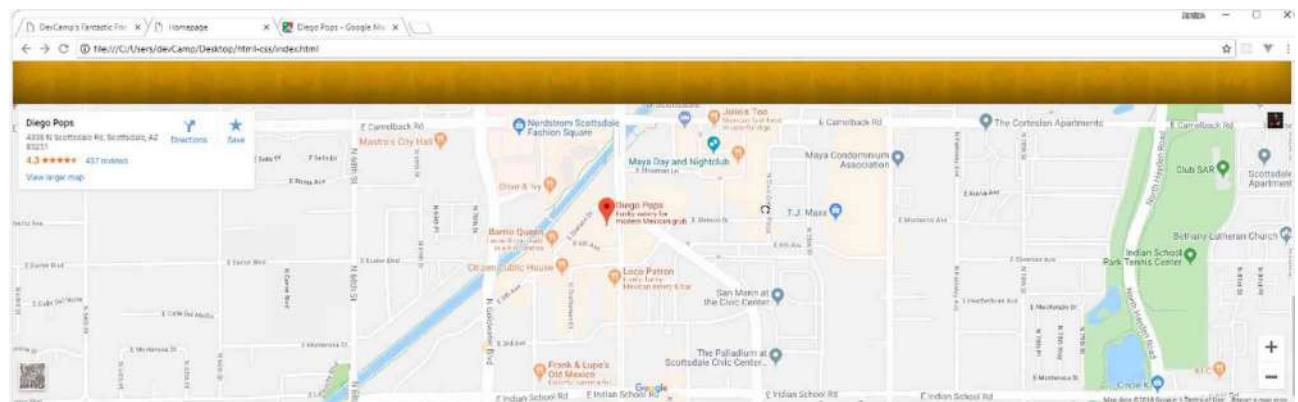
If you click on IFrame right here you can actually edit this value. So, if you want the width to be 100%, then you can see that that is all that you need to do. Now we have the nice little shadow effect right here, and if we come back, you can see that's exactly what we're looking for.

All we need to do in order to edit this is to be able to wrap it up and to edit that width and say we want that to be 100%.

## index.html

```
<div>
  <iframe src="https://www.google.com/maps/embed?pb=!1m18!1m12!1m3!1d3327.0765176913264!2d-111.92875828479936!3d33.49938628076005!2m3!1f0!2f0!3f0!3m2!1i1024!2i768!4f13.1!3m3!1m2!
  1s0x872b0bbdbed13a73%3A0xfab78c0f014377d8!2sDiego+Pops!5e0!3m2!1sen!2sus!4v1534370311184" width="100%" height="450" frameborder="0" style="border:0" allowfullscreen></iframe>
</div>
```

Hit save, and now if you come back and hit refresh, you can see our map is working perfectly.



This is giving us exactly the look and feel that we want on the home page and now we have a dynamic map that users can drag, they can click on, they can zoom in, and they can use just like they would if they went to Google Map. That's how easy it is to embed a Google map directly into your own website.

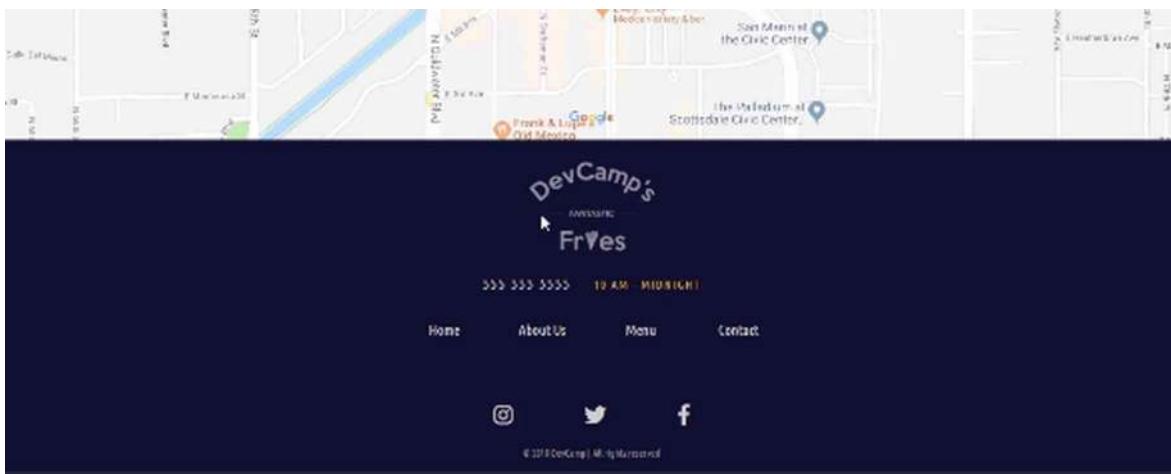
## Coding Exercise

Go to Google Maps and get the tag for an embedded map. It can be any address you'd like.

# 1.33 HTML Footer

Now, as you may have guessed, we're going to follow the same pattern we've been following for this entire course, where we add all of the HTML code that we need and build the structure. After that, we will go and we'll add the styles.

Right here, we can see that we have an image, and then we have a phone number, a set of hours, our nav-links, some social media sharing icons, and then a little copyright symbol down at the bottom.



Now, I'm going to ask you to do something, and it's completely up to you if you want to do it or not. By now, we have covered all of the skills that you will need in order to implement what I'm showing you right here.

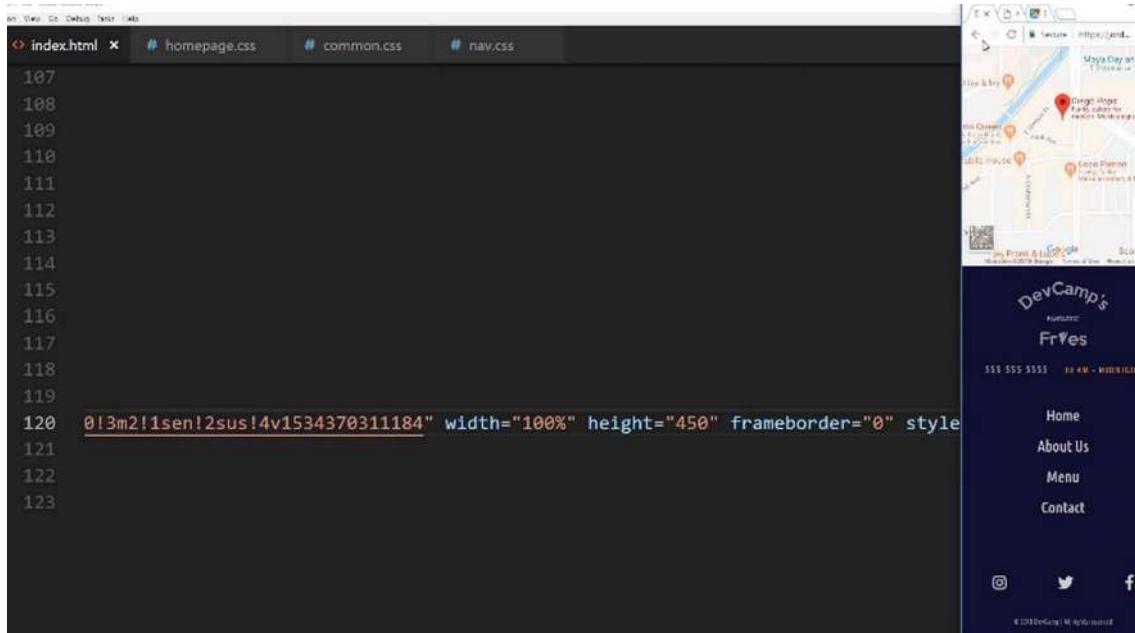
So, what my recommendation would be, if you're taking this course because you want to learn HTML and CSS, and it's really your goal to be able to build any kind of website that you're ever asked to build, then I would recommend that you pause the video right now.

Maybe take a screenshot of what we have here, and then go and try and build it yourself. Now, if you do not feel ready for it, that's perfectly fine, but even if you feel like you're about halfway ready, then I'd recommend trying to do it.

It's been my experience that I really learned the most not by watching someone else implementing it, but I learn by following a model and then going and implementing that myself. I discover that there are many concepts that I maybe thought I understood until I started writing the code. That is what I'd recommend for you to do.

Now, I'm going to go, and I am going to implement this. I'm going to start in this video, I'm just going to do the HTML, and then, later on, we'll add the CSS. We know we need a logo, a phone. Okay, let me actually, just to make life a little bit easier on us.

This is what I would do in a real-life project. I would come down here. I would shrink this so that I don't have to keep switching back and forth, so I'm going to move this all the way to the side, and then I'll shrink the code here so that we can be looking at this.



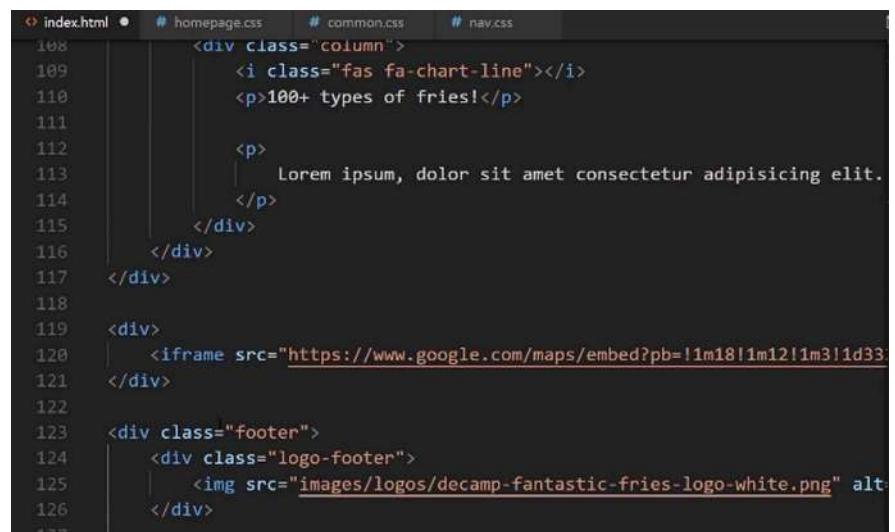
A screenshot of a browser window showing the file structure on the left and the rendered page on the right. The file structure includes index.html, homepage.css, common.css, and nav.css. The rendered page shows a map of a location with a red marker, followed by a sidebar with the text "DevCamp's Fries" and a phone number "555 555 5555". Below the sidebar is a footer with links for Home, About Us, Menu, and Contact, along with social media icons for Instagram, Twitter, and Facebook.

That'll make it a little bit easier for us. Let's go to **index.html**. We have our map here. This is going to go below the map, and I'm going to create a class called **footer**. Now, all of these elements are going to go inside of it.

We know that we have a logo for the foot, and inside of here, we're going to need that image tag. So, I'll say **img** and the source is going to be **images/logos/decamp-fantastic-fries-logo-white.png**. Let's also add an **alt-tag**, so here we'll just say **Logo**, and that's all we need for the Logo footer.

Below that, you can see that we need to have the phone number and then the hours. I'm going to create a new class here, and I'm just going to call that **footer-phone-hours**, and inside of here, we're going to place both of those elements.

We'll place the phone number **555-555-5555**. Just in case you are wondering, no, that is not my real phone number, so don't try calling it. Now, if I come here, I'm going to wrap this up in a class called **phone** and close it off.



A screenshot of a code editor showing the **index.html** file. The code is as follows:

```
108 <div class="column">
109   <i class="fas fa-chart-line"></i>
110   <p>100+ types of fries!</p>
111
112   <p>
113     Lorem ipsum, dolor sit amet consectetur adipisicing elit.
114   </p>
115 </div>
116 </div>
117 </div>
118
119 <div>
120   <iframe src="https://www.google.com/maps/embed?pb=!1m18!1m12!1m3!1d33...
121 </div>
122
123 <div class="footer">
124   <div class="logo-footer">
125     
126   </div>
127 </div>
```

```
123  <div class="footer">
124      <div class="logo-footer">
125          
126      </div>
127
128      <div class="footer-phone-hours">
129          <div class="phone">
130              555 555 5555
131          </div>
132      </div>
133  </div>
134  </body>
```

Now, I want to show you something that I'm going to do here that's a little bit different, and it's a topic that we've not talked about yet. I am going to, instead of using a div here, I'm going to use what is called a **span**.

Now, a span is very similar to a div. The only key difference here is that a span if you put it on the same line as other text, so if we put two spans together like we're about to, it's not going to give the same type of line break that we usually would get with a div.

Because we want to have these two elements sitting right next to each other, a span is a good option. So, for the hours, I'm going to say **span**. Now just say **hours**, and now, inside of here, we can put those hours. I'm going to say **10 a.m.**, give us space in between here. 10 a.m and then to midnight, just like that.

```
123  <div class="footer">
124      <div class="logo-footer">
125          
126      </div>
127
128      <div class="footer-phone-hours">
129          <span class="phone">
130              555 555 5555
131          </span>
132
133          <span class="hours">
134              10 am - midnight
135          </span>
136      </div>
137  </div>
138  </body>
```

Now, moving on to the navigation links, you may have noticed that I have the identical links here as we had in the top. Well, there is no reason to re-type all of that or to create custom classes when you want the identical behavior.

If you also remember back to when I was talking about how we wanted to make sure our **links-wrapper** was scalable so we could use it in different parts of the application, this is part of why I wanted to do that.

What we can actually do is come here and highlight all of the content for **links- wrapper**, and we can just copy all of it.

```
index.html # homepage.css # common.css # nav.css
39         
40     </div>
41
42     <div class="links-wrapper">
43         <div class="nav-link">
44             <a href="index.html">Home</a>
45         </div>
46
47         <div class="nav-link">
48             <a href="about.html">About</a>
49         </div>
50
51         <div class="nav-link">
52             <a href="menu.html">Menu</a>
53         </div>
54
55         <div class="nav-link">
56             <a href="contact.html">Contact</a>
57         </div>
58     </div>
59
60 
```

Then I'm going to put it inside. I'm going to have our `links-wrapper`, and then we need to fix the indentation just like that. That's all we need to do, and now we have all of our `links-wrapper`.

```
index.html # homepage.css # common.css # nav.css
134         10 am - midnight
135             </span>
136         </div>
137
138     <div class="links-wrapper">
139         <div class="nav-link">
140             <a href="index.html">Home</a>
141         </div>
142
143         <div class="nav-link">
144             <a href="about.html">About</a>
145         </div>
146
147         <div class="nav-link">
148             <a href="menu.html">Menu</a>
149         </div>
150
151         <div class="nav-link">
152             <a href="contact.html">Contact</a>
153         </div>
154     </div>
155 
```

Now, moving down, we have our social media icons. Let's create a div. We'll call it `social-media-icons-wrapper`.

Inside of here, we're going to have a few links. We'll have one link, and this first one's going to go to Instagram.

I'm not going to actually put the links in here because this restaurant doesn't have an

Instagram page, but we've already walked through how to do that. You could wrap that up and put all of your Instagram code in there.

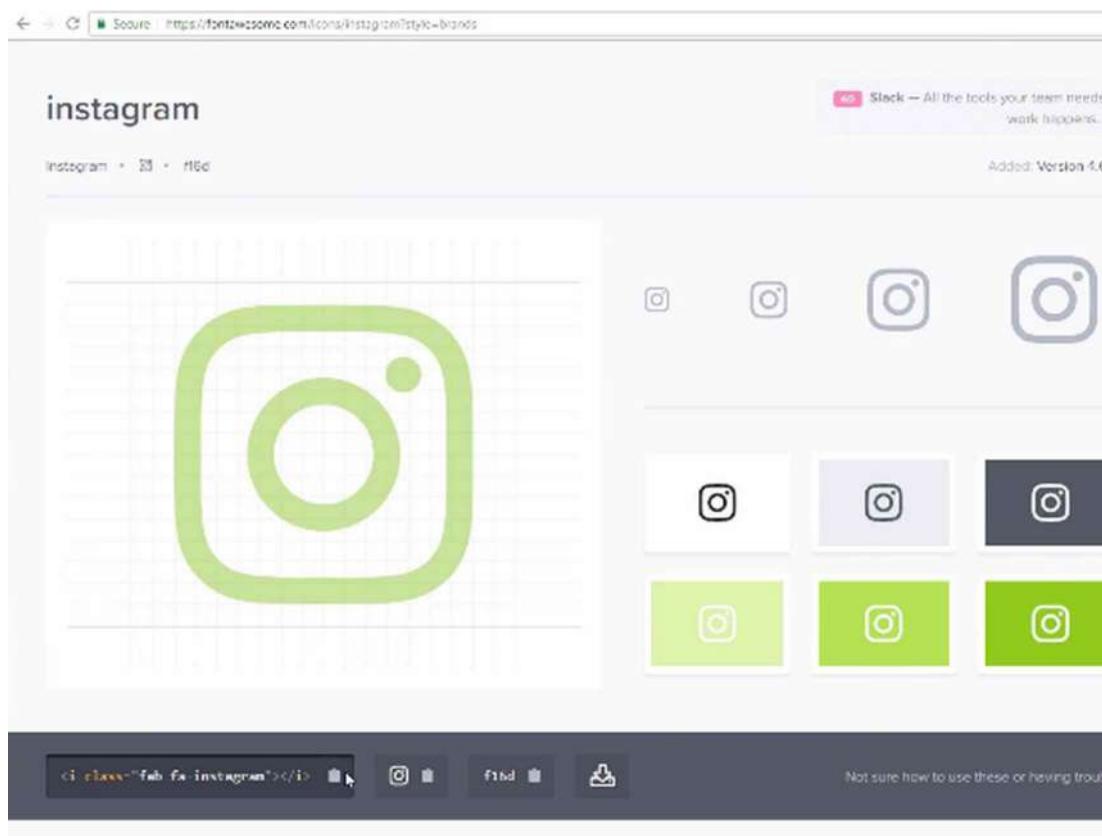
I'm just going to put a `#`. This is kind of the convention that you will see whenever you want to create a link but you don't want to point it anywhere. Just put a `#` in there, that way, you're not going

to get an error if a user clicks on it. It's just going to put a hash up in the URL bar and not do anything else.

```
151      <div class="nav-link">
152          <a href="contact.html">Contact</a>
153      </div>
154  </div>
155
156      <div class="social-media-icons-wrapper">
157          <a href="#"></a>
158      </div>
```

Now that we have that, we're going to want an icon here, so let's switch over into the browser. Let's open up a new window, just so we don't have to move our browser from where it's at. Then we'll go to [Font Awesome](#) and will grab the icons that we need.

We want to grab Instagram, Twitter, and Facebook. Let's say [Instagram](#). There we go, and now you can just copy this link right here and then paste it in. Here, we're going to do the same thing for the other links, and one more.



Now we have all three links, but they're all pointing to Instagram. Let's grab the Twitter one next, and then we want this first one. This one is just `fa-twitter`. Technically, you can just come here and edit that name.

I'm guessing this one's going to be `facebook`, but it may be `fb`. Let's just verify, just to make sure. So, type in `Facebook`. You have a few different options, and they went with `Facebook`, and then just a `- f`. I'm glad that we looked at that, so `facebook - f`. Just like that.

```
156     <div class="social-media-icons-wrapper">
157         <a href="#">
158             |     <i class="fab fa-instagram"></i>
159         </a>
160
161         <a href="#">
162             |     <i class="fab fa-twitter"></i>
163         </a>
164
165         <a href="#">
166             |     <i class="fab fa-facebook-f"></i>
167         </a>
168     </div>
169 </div>
```

Then we want, at the very bottom, below `social-media-icons-wrapper`, we want one more. This is going to be our `copyright-wrapper`. Inside of here, I'm going to show you how to implement something this is called an HTML entity.

If you ever wonder how you could have special characters, like a copyright symbol, what you can do is use the `HTML entity property`. The one for a copyright is kind of, I want to say it's easier to remember, but it's probably only because I've been doing it for a while.

It is going to start with an `ampersand(&)` followed by the word `copy`. Then you do get some nice autocomplete here. Finish that off with a semicolon, and that's going to give you the copyright symbol. From there, you can say whatever year you want and then the name, and then, I want to go with a pipe character.

Now, the pipe character, I did have to put in my notes because I do not remember this one. It is going to be the `ampersand(&)` followed by `#124`, and then from there, we can say, `All rights reserved`.

```
170     <div class="copyright-wrapper">
171         |     &copy; 2018 DevCamp &#124; All rights reserved
172     </div>
173 </div>
174 </body>
```

What an HTML entity is is whenever you have a special character, these are the types of characters that maybe aren't directly on a keyboard. HTML needs to know how to process those properly, and the best approach is usually to use an HTML entity.

You can look those up, just in case you're wondering how you will know how to put them in your own application, if you just type **HTML entities** into Google, you will have a full list of all of them. It has some of the more popular ones here, and you can see here is the copyright one.

The screenshot shows a table titled "Some Other Useful HTML Character Entities". The table has four columns: Result, Description, Entity Name, and Entity Number. The rows include:

Result	Description	Entity Name	Entity Number
	non-breaking space	&nbsp;	&#160;
<	less than	&lt;	&#60;
>	greater than	&gt;	&#62;
&	ampersand	&amp;	&#38;
"	double quotation mark	&quot;	&#34;
'	single quotation mark (apostrophe)	&apos;	&#39;
¢	cent	&cent;	&#162;
¤	pound	&pound;	&#163;
¥	yen	&yen;	&#165;
€	euro	&euro;	&#8203;
©	copyright	&copy;	&#169;
®	registered trademark	&reg;	&#174;

**Note:** Entity names are case sensitive.

You have some of the financial symbols, you have ampersands, greater than symbols. They have the entity name, and the entity number, and you can use either one of these. Then, they have some other options. That's what you can do to look that up.

With all of that in-place, I believe that is everything we are going to need, so let's close off Font Awesome. Let's spread this

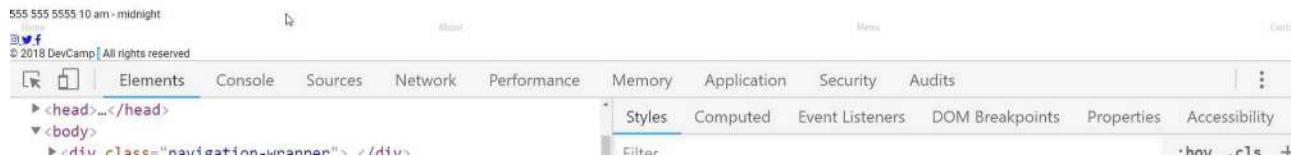
back out. Now, if we come to our own site, hit refresh, and come down, you'll see, right now, if you remember our image, our logo is gigantic, so it's actually here.

The screenshot shows a browser window with a large blue background image. Below the image, the developer tools are open, specifically the Elements tab. The DOM tree shows a `<div>` element with a `src` attribute pointing to a Google Maps embed URL. The styles panel on the right shows the following CSS rule:

```
body {
  margin: 0px;
  font-family: 'Roboto', sans-serif;
}
```

It's taking up all of that space, so I'm not going to worry about that. We do have our little nav-icons here or nav-links and those are working. We're going to have to create a wrapper with our styles to do that.

You can see that we have each one of our social media icons. Our HTML entities are working, so we have the copyright symbol, the little pipe in the middle.



So, we have all of the content that we need for our footer. In the next guide, we're going to start styling it.

## Updates

```
<i class="fa-brands fa-facebook-f"></i>
<i class="fa-brands fa-twitter"></i>
<i class="fa-brands fa-instagram"></i>
```



## Coding Exercise

Fill in the `right-footer-column` with, one link tag to <https://www.linkedin.com/>. One `div` with the phone number `555-555-5555`. Then go to [https://www.w3schools.com/html/html\\_symbols.asp](https://www.w3schools.com/html/html_symbols.asp) to find the HTML entity for the Spade symbol (`♠`) and place that in one last `div`. Please note that you must use the **HTML entity** for the Spade symbol, not the icon or character itself.

```
<div class="right-footer-column">
  <!--Fill in this div-->
</div>
```

# 1.34 Flex Direction: Grid vs Column

Now that we have the HTML structure all in place, now let's start adding the styles.

Right here you can see that we have a footer class wrapper right here, so this is going to be I think the best place to start.

```
123     <div class="footer">
124         <div class="logo-footer">
125             
126         </div>
127
128         <div class="footer-phone-hours">
129             <span class="phone">
130                 555 555 5555
131             </span>
132
133             <span class="hours">
134                 10 am - midnight
135             </span>
136         </div>
```

I'm going to come into the `common.css` styles because these footer styles are going to be common and then we'll refactor it out into its own little module later. I'm going to add in footer styles and we'll start off with our main footer class.

## common.css

```
/* Footer styles */
.footer {
    background-color: #11122b;
    color: #cbcbcb;
    font-family: "Ubuntu Condensed", sans-serif;
}
```

Let's just see where we're at right now, so I'm going to switch back and hit refresh.



You can see we have our correct color, obviously, we need to style and format this logo because it's massive, it's just bringing it in with its full regular size but we have our correct fonts, we have our good background color so we are making progress.

Now that we have that, I think it's good to start aligning it. So we're going to use Flexbox to align this.

## common.css

```
/* Footer styles */
.footer {
    background-color: #11122b;
    color: #cbcbcb;
    font-family: "Ubuntu Condensed", sans-serif;
    display: flex;
    justify-content: center;
    align-items: center;
}
```

If I hit save here and switch back and hit refresh, you're going to see that we aren't quite getting the behavior that you might expect.



Do you see how we have the logo here, then we have the hours and the phone number then the nav bar and everything is lined up horizontally? Well, this is something we have not discussed yet and this is what we're going to focus on in this guide.

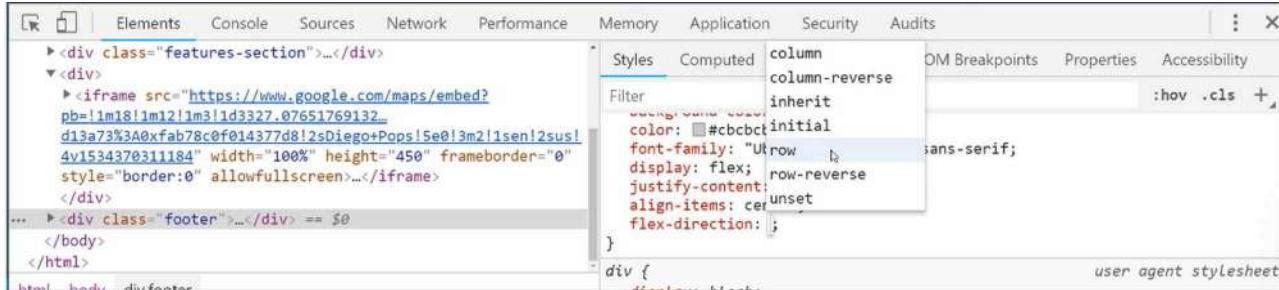
By default, Flexbox uses what is called a row direction. So what that means is that if you do not tell Flexbox any different, it will take all of the divs that are nested inside and then it will just line them up side by side.

It assumes that you want to set them up with this column kind of layout. Now what I'm going to say may sound a little bit weird and I personally when I was learning Flexbox I thought the naming was a little bit different, it felt like it was opposite and it took a little while to memorize it.

The default direction that we use here is called the row direction. So Flexbox uses this and it calls this a row and in their minds, in the developers of Flexbox, what they believe is that they're saying that they're placing all the content on a single row.

I personally think they look like columns, but they say this is one single row. What we wanna do is to use a custom value for the direction and so we wanna say that we want to use a column and that column direction is going to allow us to stack the elements on top of each other.

Let's do it right here I'm going to click on footer with our inspector and come into our browser tools. I'm going to say flex direction and you can see the options, we have column, we have column reverse, we have inherit initial, row, row reverse and unset.



Well if we click on column, you'll see that that actually gets us what we're wanting. So this is the correct way to use flex direction. Whenever you have a set of divs, so all the child elements are items that you want to stack on top of each other, use flex direction column.

That's what we're going to do here, I'll just copy this, switch back and then we'll paste that in and fix the indentation. So now that we have that, let's also declare a height, I wanna use 400 pixels for the height and let's just see what that gives us.

### common.css

```
/* Footer styles */
.footer {
    background-color: #11122b;
    color: #cbcfcf;
    font-family: "Ubuntu Condensed", sans-serif;
    display: flex;
    justify-content: center;
    align-items: center;
    flex-direction: column;
    height: 400px;
}
```

Okay, so now if we hit refresh, that is kind of working, this image is throwing it off though.

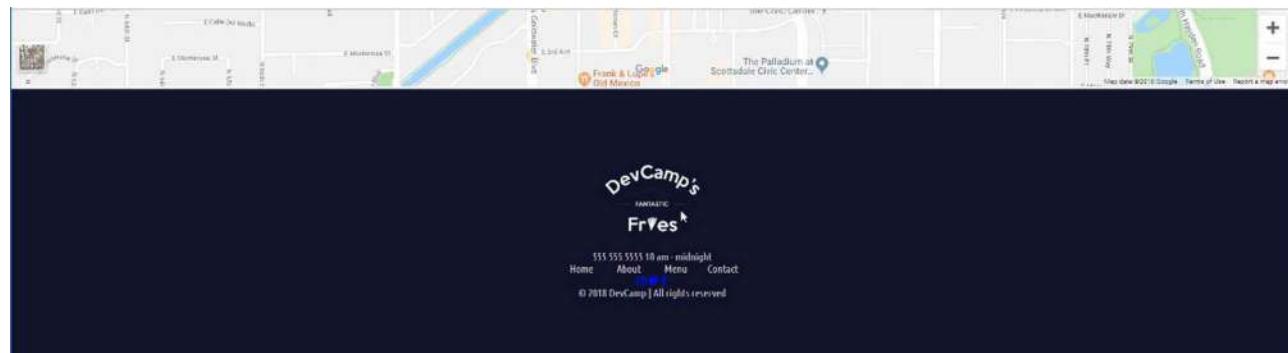


Let's go and let's fix this image before we do anything else. So now let's grab the footer and then I believe I called the class logo-footer.

## common.css

```
/* Footer styles */
.footer > .logo-footer img {
    width: 180px;
    height: 100%;
}
```

That should fix this little distortion issue. and as you can see that is now working nicely. We obviously still have some work to do on the styling, but this is giving us the size we want.



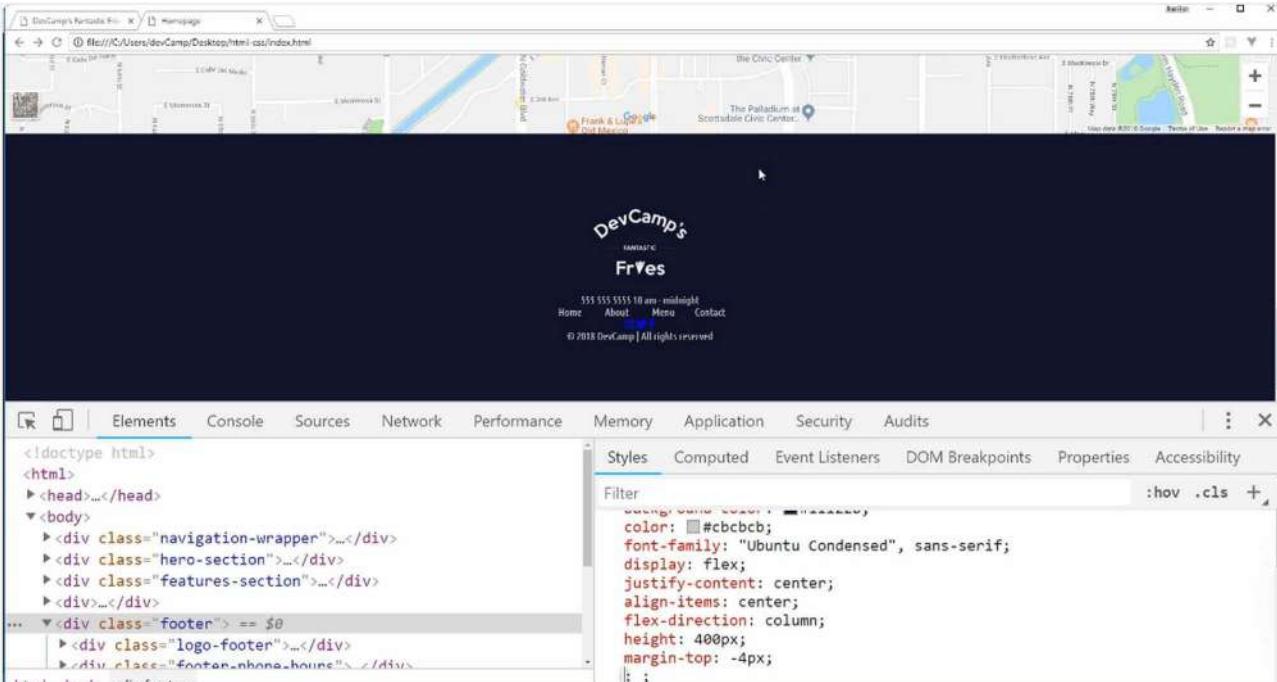
Now there's one other little thing here that's bugging me a little bit and so let's zoom in. So do you see how we have, and it's kinda hard to see on the screen but you notice how there's a little bit of a white border around the map? I do not like that, I want that to go away and the map to be flush.

There are a couple ways we can do that but I think this is a good example of where we can use a negative margin.

If I select the footer here and what I'm going to tell you to do is something that you don't want to do a lot but there definitely are times to do it and you will run into plenty of situations where you see this so I wanted to teach it to you.

And that is that usually when you're using margin, you're using it to add some space, to add some margin. But you also have the ability to use a negative margin.

If I say here with the footer class we want the margin-top to be negative four pixels, do you see how that gets rid of that white border right there and that gives us more of what we're looking for so it's nice and flush?



Let me copy this.

## common.css

```
/* Footer styles */
.footer {
    background-color: #11122b;
    color: #cbcfcf;
    font-family: "Ubuntu Condensed", sans-serif;
    display: flex;
    justify-content: center;
    align-items: center;
    flex-direction: column;
    height: 400px;
    margin-top: -4px;
}
```

Now go back to the browser. This should give us all of the footer styles that we're going to need, at least for the wrapper. So if I hit refresh, everything here is working properly.

In this guide, we walked through a number of new skills, we talked about things like negative margin, but one of the most important things I want you to take away from this specific video is that you have the ability with Flexbox to change how all of the elements are rendered.

Let me just show you a few of the other options. So we have column but you remember how we had column reverse? Well if you ever have a situation where you're getting some divs in and you wanna reverse the value, you do have the ability to do that.

If you don't list your flex direction, then row works automatically. But if you want to reverse those items you can reverse them just like that, which is pretty cool, and there are a few times where I have had to do that, so it's good to understand it.

But the two values I use the most for flex direction are going to be the default one, which is just going to be your basic row, and then column. This is what you wanna do whenever you wanna have the elements stacked on top where every one of the divs is going to be its own row.

Once again, I think that their naming is a little unfortunate because when I was learning it I like to think of columns as items that are stacked right next to each other, but that is the way they named it. The best way to work through it and understand it is really just to practice it a few times.

So I definitely recommend not only to implement this but also to see how you can play with a few of those other values just so it can become more ingrained.

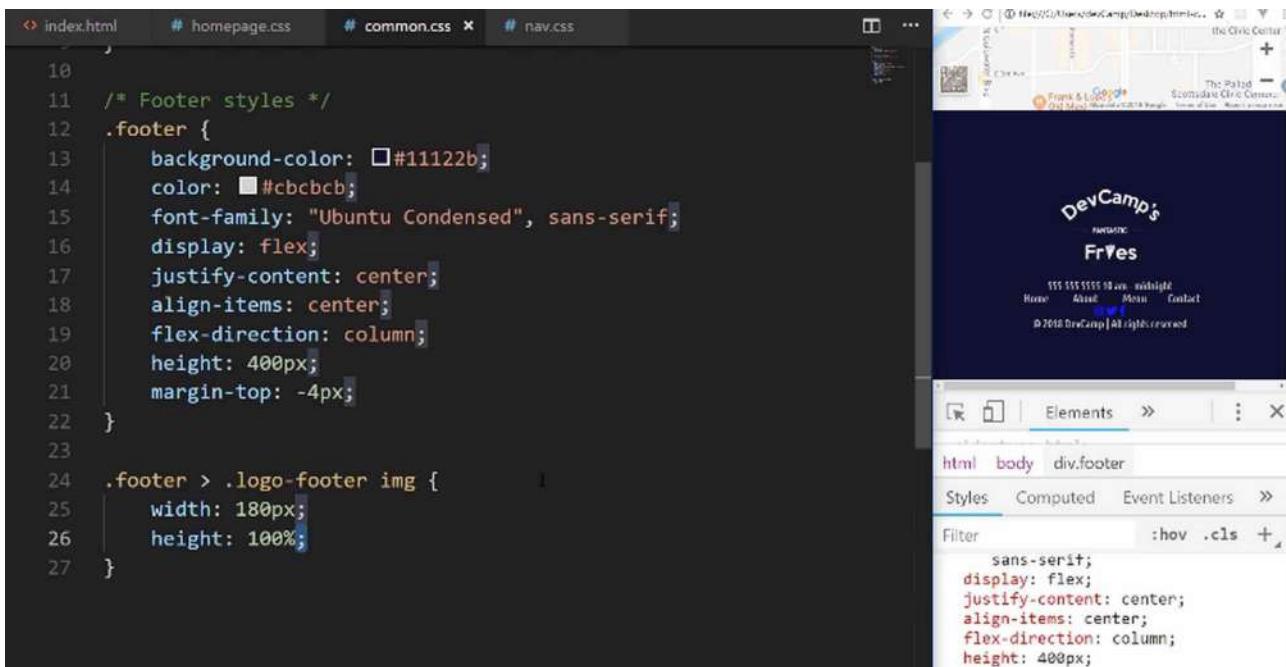


## Coding Exercise

Using Flexbox, the below div needs its content to be centered and a flex direction property set to column.

# 1.35 CSS image filters

As you can see right there for this logo, right now it is just the basic `white` that came in from the image.



What we want to do is if you come here and see the finished version, we want to have a little bit of a `lighter gray` color.

You may think that you need to just get another image, redo it in Photoshop or something like that, but using CSS we actually have the ability to manage this. That's what we're gonna go through and I'm gonna show you two different ways that you can manage the brightness or kind of the coloring for images in HTML.

What I'm gonna do here is I've selected the image and I am going to start off using one process. I'm gonna show you how you can use a `filter`.

With this filter what you can do, you can see all of the different options, you can add a blur, you can control the brightness, the contrast, you can add a drop shadow, you can make it gray scale, change the hue and then a few other options.

```
index.html    # homepage.css    # common.css    # nav.css
10
11  /* Footer styles */
12  .footer {
13      background-color: #11122b;
14      color: #cfcfcf;
15      font-family: blur()
16      display: brightness()
17      justify-content: contrast()
18      align-items: drop-shadow()
19      flex-direction: grayscale()
20      height: hue-rotate()
21      margin-top: inherit
22  }                                initial
23  }                                invert()
24  .footer > .logo-footer img {
25      width: 100% opacity()
26      height: saturate()
27      filter:
28  }
```

What I want to show you though, and I definitely recommend that you experiment with all of those so you can become familiar with it, the filter process is very popular. For what we want to do, I'm gonna start off with just controlling the brightness.

I'm going to switch this and say I want the brightness to be 50%. So if I save that and come here and hit refresh, you can see that the brightness of this image is now a lot darker. It looks like we actually changed the color. That's one option.

The screenshot shows a browser window with a dark blue footer. On the left is a code editor with the following CSS:

```
/* Footer styles */
.footer {
  background-color: #11122b;
  color: #cfcfcf;
  font-family: "Ubuntu Condensed", sans-serif;
  display: flex;
  justify-content: center;
  align-items: center;
  flex-direction: column;
  height: 400px;
  margin-top: -4px;
}

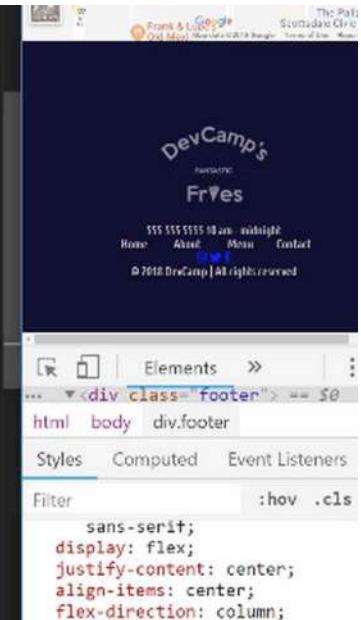
.footer > .logo-footer img {
  width: 180px;
  height: 100%;
  filter: brightness(50%);
}
```

The footer contains a logo with the text "DevCamp's Fr̄es". The browser's developer tools are open, showing the element inspector with the footer selected. The styles panel shows:

- sans-serif;
- display: flex;
- justify-content: center;

Now the other option is to change the opacity. The opacity, if you've never seen it before, it's essentially allowing us to change the transparency percentage. I can say `opacity: 0.5` for this,

and it will give me a 50% opacity. Now if I hit save and come back right here and it refresh, you can see that it has changed.



```
1  /* Footer styles */
2  .footer {
3      background-color: #11122b;
4      color: #cbcfcf;
5      font-family: "Ubuntu Condensed", sans-serif;
6      display: flex;
7      justify-content: center;
8      align-items: center;
9      flex-direction: column;
10     height: 400px;
11     margin-top: -4px;
12 }
13
14 .footer > .logo-footer img {
15     width: 180px;
16     height: 100%;
17     /* filter: brightness(50%); */
18     opacity: 0.5;
19 }
```

What we did with the first option with filter, we actually changed the color itself. We did that by altering the brightness. Here, what we did is we actually made the image itself slightly transparent.

So, depending on your own personal use case, then both of these options will work in their own situation. That is how you can change the color using just pure CSS without actually having to alter the image at all.



## Coding Exercise

Use a filter to reduce the below image's brightness to 30%.

```
<div class="cute-picture">
    
</div>
```

# 1.36 Footer styles

This is actually the second time I've filmed this. I filmed the entire episode and finished out the rest of the homepage only to discover that the microphone was not plugged in, so that wasn't a fun discovery. Anyway, we are back here.

Now that we have everything that we need, let's go and let's start adding the next section, or the styles for the next section. That's going to be for the `footer-phone-hours`, and if you look at the HTML here, you'll see we have a class of `footer-phone-hours` and then we have two spans inside of it.

```
119      <div>
120        |   <iframe src="https://www.google.com/maps/embed?pb=!1m18!1m12!1m3!1d3327.0765176913
121        |   </div>
122
123      <div class="footer">
124        |   <div class="logo-footer">
125        |     |   
126        |   </div>
127
128      <div class="footer-phone-hours">
129        |   <span class="phone">
130        |     |   555 555 5555
131        |   </span>
132
133        |   <span class="hours">
134        |     |   10 am - midnight
135        |   </span>
136      </div>
137
```

I want to create some styles that are going to apply to both of these elements, and then we're going to add some more specific styles. Hopefully, they'll also illustrate, once again, how the specificity works with CSS.

Remember, the more specific the selector, that is going to override any less specific selector. If we select `footer-phone-hours-span` class and apply anything, if we then go and say, `footer-phone-hours` and then grab the `hours` class, that is going to be more specific. It's going to override any of the other styles that we may want to apply.

I'm going to come down here, and grab the footer. From there, I want to grab the `footer-phone-hours`, and then the `span` tag. Inside of here, I want to add some space on the right and left-hand side. I'll say, `margin-left`, and we'll go with `10px`, duplicate that and say, `margin-right` at `10px`.

## common.css

```
.footer > .footer-phone-hours span {
  margin-left: 10px;
```

```
        margin-right: 10px;  
    }
```

Then I want to use a `font-size` of `1em`. This is going to give me a baseline font size and then we're going to override this for the hours. I'm going to do that and then I also want to spread out the letters a little bit. I can say, `letter-spacing: 2px`.

## common.css

```
.footer > .footer-phone-hours span {  
    margin-left: 10px;  
    margin-right: 10px;  
    font-size: 1em;  
    letter-spacing: 2px;  
}
```

That'll just kind of add a little bit of a unique look and feel for those words. Now, if you hit refresh, you can see that that worked nicely. We have letter spacing, and now, these two items are spread out.



Let's keep moving down. Now what we want to do is add a selector that's even a little bit more specific. Instead of just footer, and then the class name and then the span tag, we're going to go one more level deep, and we want to grab the hours.

What we want to override here, in the hours, is going to be the `font-size`. I want to make this a little bit smaller than the phone number. Let's say, `0.8em`, and then from there let's add a color of that kind of goldish color, so `#cea135`.

## common.css

```
.footer > .footer-phone-hours > .hours {  
    font-size: 0.8em;  
    color: #cea135;  
}
```

Now if I hit refresh, you can see that that's working. Now, you may notice, if you compare this with the finished site, you may notice here that, where it says "MIDNIGHT" and also "AM," you may notice that those are all in caps. Well, we do not have that here.



Your first thought may be to just go and change the HTML, and that's perfectly fine. You could go into the HTML and make this in all uppercase. With CSS we actually have the ability to control this.

If you inspect this and grab that element. You can go and use the `text-transform` property. Now, this is one that we've not used before, and so you can use `uppercase`. You see how that has translated it into uppercase.

A screenshot of the Chrome DevTools Styles tab. The element selected is the footer hours section. The CSS rule shown is: 

```
.footer > .footer-phone-hours > .hours { font-size: 0.8em; color: #cea135; text-transform: uppercase; }
```

You also have a few other options. You could do `capitalize`. What `capitalize` does is, it will take each word and it will capitalize the first word there. You could also do `lowercase`, and they have a few other options, but I typically use it mainly just for uppercase.

It's a really nice way of being able to format this using CSS without even having to touch the HTML. I'm going to copy this and add it as one more rule, and that's all we need to do for the hours.

#### common.css

```
.footer > .footer-phone-hours > .hours { font-size: 0.8em; color: #cea135; text-transform: uppercase; }
```

Now let's come to our `links-wrapper`. Now, right here we have these links. If you're curious on how this is still working, you may have noticed that even though we haven't applied any styles, we still have the same type of behavior that we have up here in the navbar.

It's because we used a generic links wrapper class, and we did that on purpose. We also have the ability to override and to add to that class. So, let's see how we can do that here. I can say, `footer` and then `links-wrapper`, and now inside here, any new styles that we want to add on top of the `links-wrapper` will only get applied when this is in the footer.

#### common.css

```
.footer > .links-wrapper {  
}
```

This is nice because we can apply different styles to the links-wrapper without affecting the one that is in the top nav or on any of the other pages. What I can do here is, I want to add some margin to the top and bottom, so I'll say, `margin-top` and we'll go with `40px`, and `margin-bottom: 40px`.

#### common.css

```
.footer > .links-wrapper {  
    margin-top: 40px;  
    margin-bottom: 40px;  
}
```

Let's also add a `width` here, so that we can control how wide the wrapper is going to be. For this, we'll go with, let's just say, `400px`. Have it nice and wide.

#### common.css

```
.footer > .links-wrapper {  
    margin-top: 40px;  
    margin-bottom: 40px;  
    width: 400px;  
}
```

Let's just see what that gives us, so hit refresh. You can see that spaced it out nicely. This is all looking really good, and we still have all of our cool other effects.

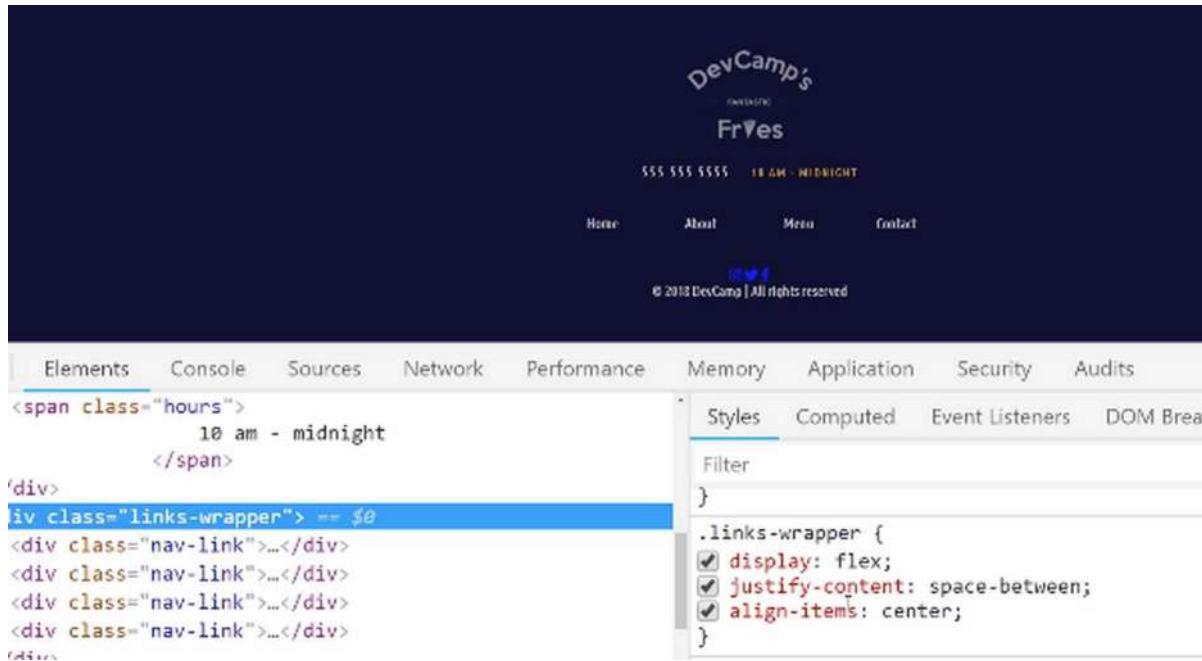
The screenshot shows the footer of the DevCamp's website. The footer contains the text "10 AM - MIDNIGHT" and "10 AM - MIDNIGHT". Below this, there is a "hours" span containing "10 am - midnight". The "links-wrapper" div is highlighted with a blue selection bar. The developer tools' styles tab is active, showing the CSS rule ".footer > .footer-phone-hours > .hours { font-size: 0.8em; color: #ce4135; text-transform: uppercase; }".

This is fantastic, I love the way this is coming along, and notice how much faster this is because we created a lot of general, more abstract styles up here, and we're able to reuse these.

If you're also curious, and it still isn't making perfect sense to you, let's highlight this, and I'm going to grab the links wrapper here. Now notice that we can trace the lineage of all of these styles.

The screenshot shows the footer of the DevCamp's website. The "links-wrapper" div is highlighted with a blue selection bar. The developer tools' elements tab shows the DOM structure: <span class="hours"> 10 am - midnight </span> <div class="links-wrapper"> ... </div> <div class="nav-link">...</div> <div class="nav-link">...</div> <div class="nav-link">...</div> <div class="nav-link">...</div> <div class="social-media-icons-wrapper">...</div>. The developer tools' styles tab is active, showing the CSS rule ".footer > .links-wrapper { margin-top: 40px; margin-bottom: 40px; width: 400px; }".

So when a links wrapper is inside of the footer, we have the margin-top, bottom, and then the width. These things are only going to be applied when the links wrapper is living inside of the footer, but if you scroll down, you see where we also have the generic links wrapper. This has FlexBox and justify-content, and the align-items.



What this means is that you can create a class with some baseline styles and then reuse that throughout the application, which is very helpful. You can see that even though we have quite a few components here, we're going to be able to apply all of the same types of styles that we spent hours here building in this navbar at the top.

Because we're able to reuse a lot of the things that we've already done, then it's much faster down here. We're going to be able to do the same task in minutes.

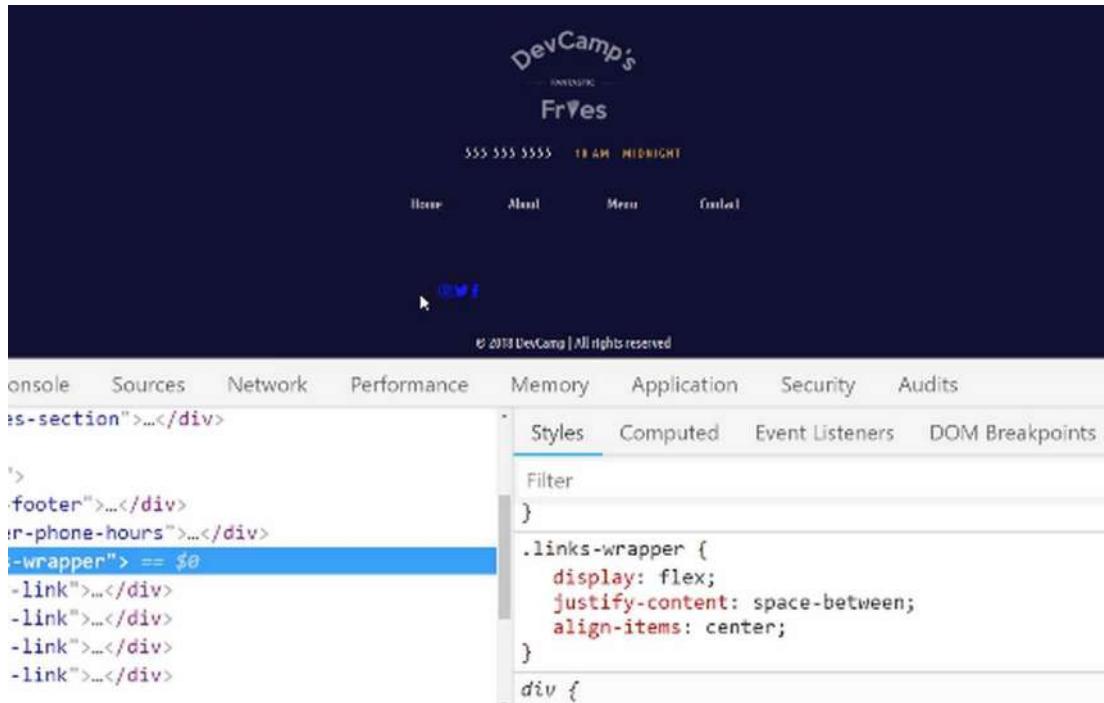
Now that we have that, let's move down to the social media icons. We have the footer, and now with the `footer`, we can go grab the inside of there, the `social-media-icons-wrapper`, and let's add the same kind of margins.

I'm going to add the margin and, for the top and bottom, and we're also going to add a width, but this one I want to be a little bit more narrow. So I'm going to say here, `300px`, and we also want to make this a Flexbox container and I'm going to show you why here.

### common.css

```
.footer > .social-media-icons-wrapper {  
  margin-top: 40px;  
  margin-bottom: 40px;  
  width: 300px;  
}
```

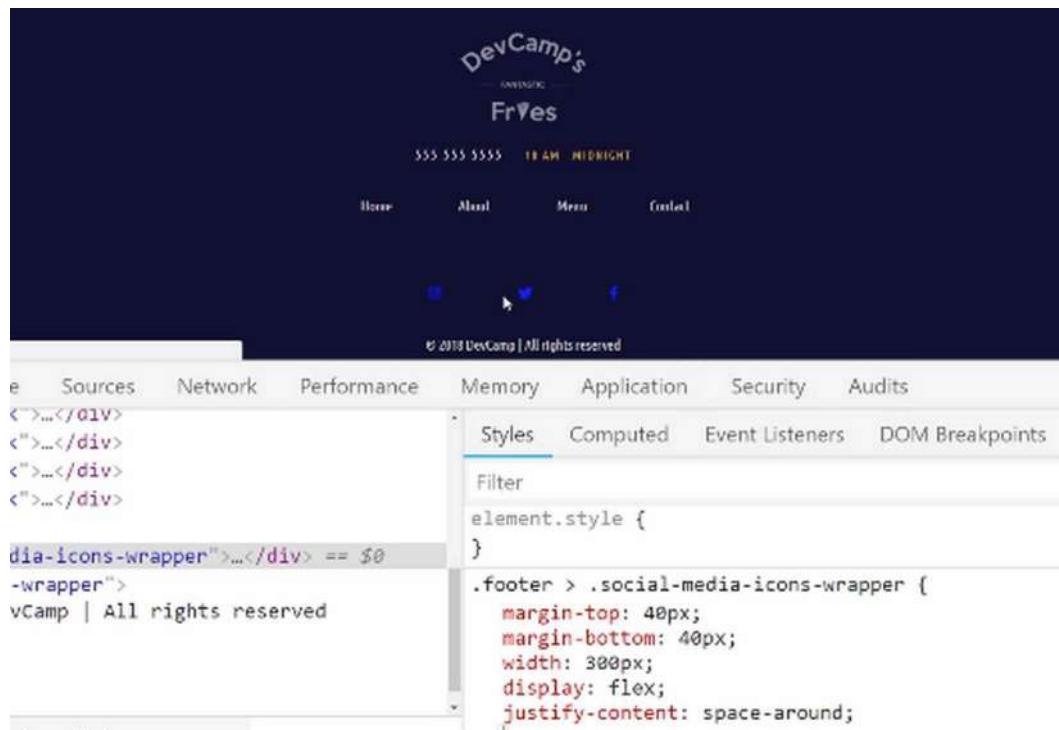
I'm going to hit refresh, and do you see how it moves all of the links all the way to the left. Well, that is the natural HTML behavior.



```
<div>
  <div>
    <div>
      <div>
        <div>
          <div>
            <div>
              <div>
                <div>
                  <div>
                    <div>
                      <div>
                        <div>
                          <div>
                            <div>
                              <div>
                                <div>
                                  <div>
                                    <div>
                                      <div>
                                        <div>
                                          <div>
                                            <div>
                                              <div>
                                                <div>
                                                  <div>
                                                    <div>
                                                      <div>
                                                        <div>
                                                          <div>
                                                            <div>
                                                              <div>
                                                                <div>
                                                                  <div>
                                                                    <div>
                                                                      <div>
                                                                        <div>
                                                                          <div>
                                                                            <div>
                                                                              <div>
                                                                                <div>
                                                                                  <div>
                                                                                    <div>
                                                                                      <div>
                                                                                        <div>
              </div>
            </div>
          </div>
        </div>
      </div>
    </div>
  </div>
</div>
```

The screenshot shows the DevTools Elements tab with the DOM tree expanded. A specific element, the social media wrapper, is selected and highlighted with a blue background. The right panel displays the CSS styles for this element, which includes a flex container setup with `display: flex`, `justify-content: space-between`, and `align-items: center`.

If you select the social media wrapper here, and if you want to make this a flex container. If I say `display: flex`, and then `justify-content: space-around`. Do you see how that places them in the exact spot we're looking for? That's just another example of how powerful Flexbox is. Without using a tool like that, it would take a lot of time.



```
<div>
  <div>
    <div>
      <div>
        <div>
          <div>
            <div>
              <div>
                <div>
                  <div>
                    <div>
                      <div>
                        <div>
                          <div>
                            <div>
                              <div>
                                <div>
                                  <div>
                                    <div>
                                      <div>
                                        <div>
                                          <div>
                                            <div>
                                              <div>
                                                <div>
                                                  <div>
                                                    <div>
                                                      <div>
                                                        <div>
                                                          <div>
                                                            <div>
                                                              <div>
                                                                <div>
                                                                  <div>
                                                                    <div>
                                                                      <div>
                                                                        <div>
              </div>
            </div>
          </div>
        </div>
      </div>
    </div>
  </div>
</div>
```

The screenshot shows the DevTools Elements tab with the DOM tree expanded. A specific element, the social media wrapper, is selected and highlighted with a blue background. The right panel displays the CSS styles for this element, which includes a flex container setup with `margin-top: 40px`, `margin-bottom: 40px`, `width: 300px`, `display: flex`, and `justify-content: space-around`.

You'd have to tweak all of the margins very specifically in order to get this type of behavior that we're able to get in just two lines of code. So, let's add this in. Hit save and refresh. Everything there is working and looks quite nice.

## common.css

```
.footer > .social-media-icons-wrapper {  
    margin-top: 40px;  
    margin-bottom: 40px;  
    width: 300px;  
    display: flex;  
    justify-content: space-around;  
}
```

Now let's update these styles, so that they have the correct links and also the link color, and then we add our hover effect. I'm going to grab everything here, copy it, and now we want to select just the link. So just that `a`-tag.

Then here, we'll get rid of the content, and now we can add our custom styles. I want these to be a little bit larger, so I'm going to say, `font-size` of `1.5em` and then we want to have a color of, kind of that off-white. We're going to go with that CB hexadecimal color.

## common.css

```
.footer > .social-media-icons-wrapper a {  
    font-size: 1.5em;  
    color: #cbcfc;
```

```
}
```

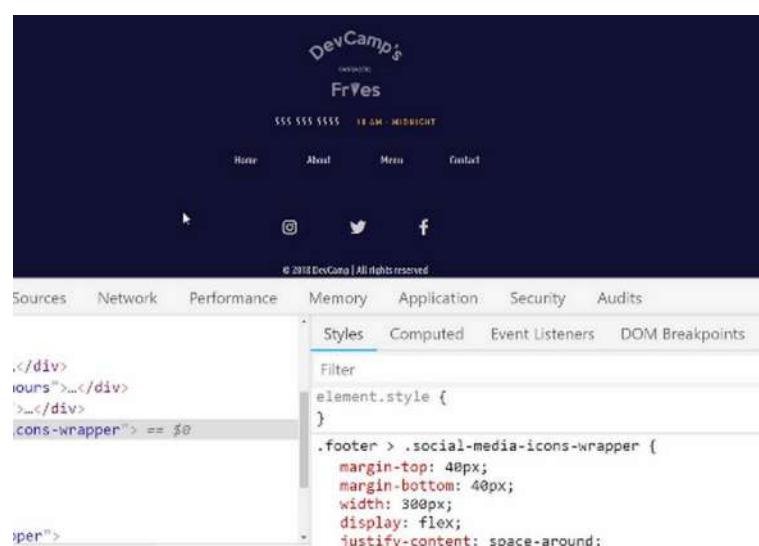
I don't believe that they have a `text-decoration`. I don't believe that they have an underline, it doesn't look like it. So, we shouldn't even have to override that.

Now the transition, because we want to have our hover effects. This should be `0.5s`.

## common.css

```
.footer > .social-media-icons-wrapper a {  
    font-size: 1.5em;  
    color: #cbcfc;  
    transition: 0.5s;  
}
```

Hit save. Hit refresh, and there we go. Now, this is looking much better. Now we just need to add our hover effects, with our pseudo-state.

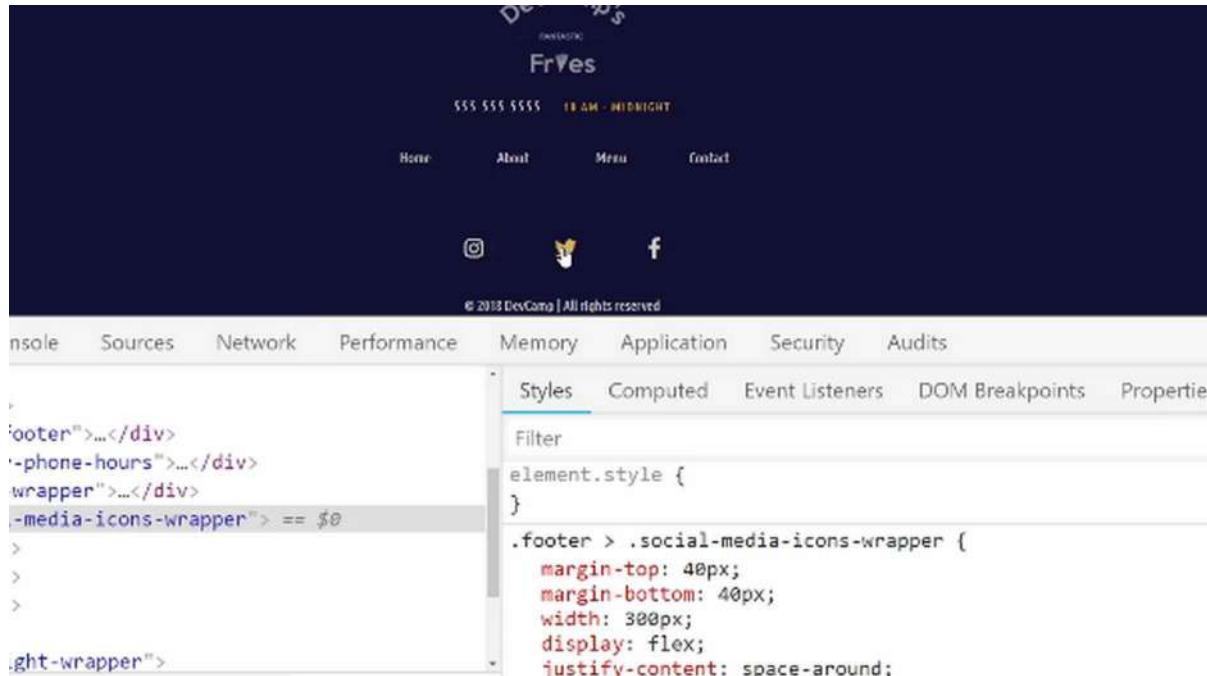


Copy that, and now if you come back to the very end, we don't want to grab just the a. We want to work with that hover state. Then update the color. We want to go with that, our gold color of #cea135.

## common.css

```
.footer > .social-media-icons-wrapper a:hover {  
    color: #cea135;  
}
```

Hit save and refresh. Now if you hover over, you can see we have that nice gold color. This is working perfectly, and we're just about done.



The last element is going to be this copyright here, and let's take a look and see. It's supposed to look like this. Now this may be a little bit small for your taste, it is for mine, but the designer sent it over like this.

I like to try to be pixel-perfect whenever I'm working with a designer because I assume that they know exactly what they're talking about. If you want yours to be a little larger, that's perfectly fine. It's up to you.

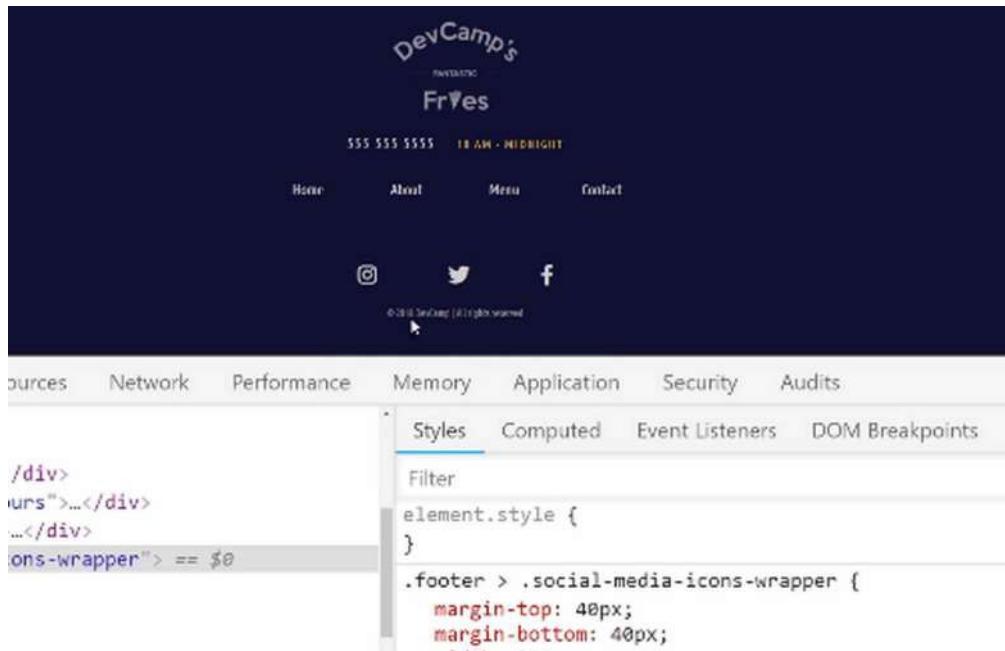
Let's get the copyright footer, and let me make sure the class name, this is going to be ... Scrolling down. Okay. It's just called `copyright-wrapper`. The way we can select that is by saying `.footer` and then `copyright-wrapper`.

What we're going to have to do here is, I don't want it flush against the very bottom of the page, so I'm going to say I want to add a margin to the bottom of 43px. I want the `font-size` to be quite small, `0.7em`, so about 70% of its normal size. I want the color to be that kind of darker, muted gray, which is #858585.

## common.css

```
.footer > .copyright-wrapper {  
    margin-bottom: 42px;  
    font-size: 0.7em;  
    color: #858585;  
}
```

Hit save, and if you hit refresh, you can see that that is matched perfectly to the design.



If you have any desire to, feel free to change any of these elements. If you want to have maybe some more margin or less margin, anything like that, I definitely want you to be able to explore it. That is how you're going to learn when you make changes, how they are rendered on the page. That is what we need to do to finish out the footer.

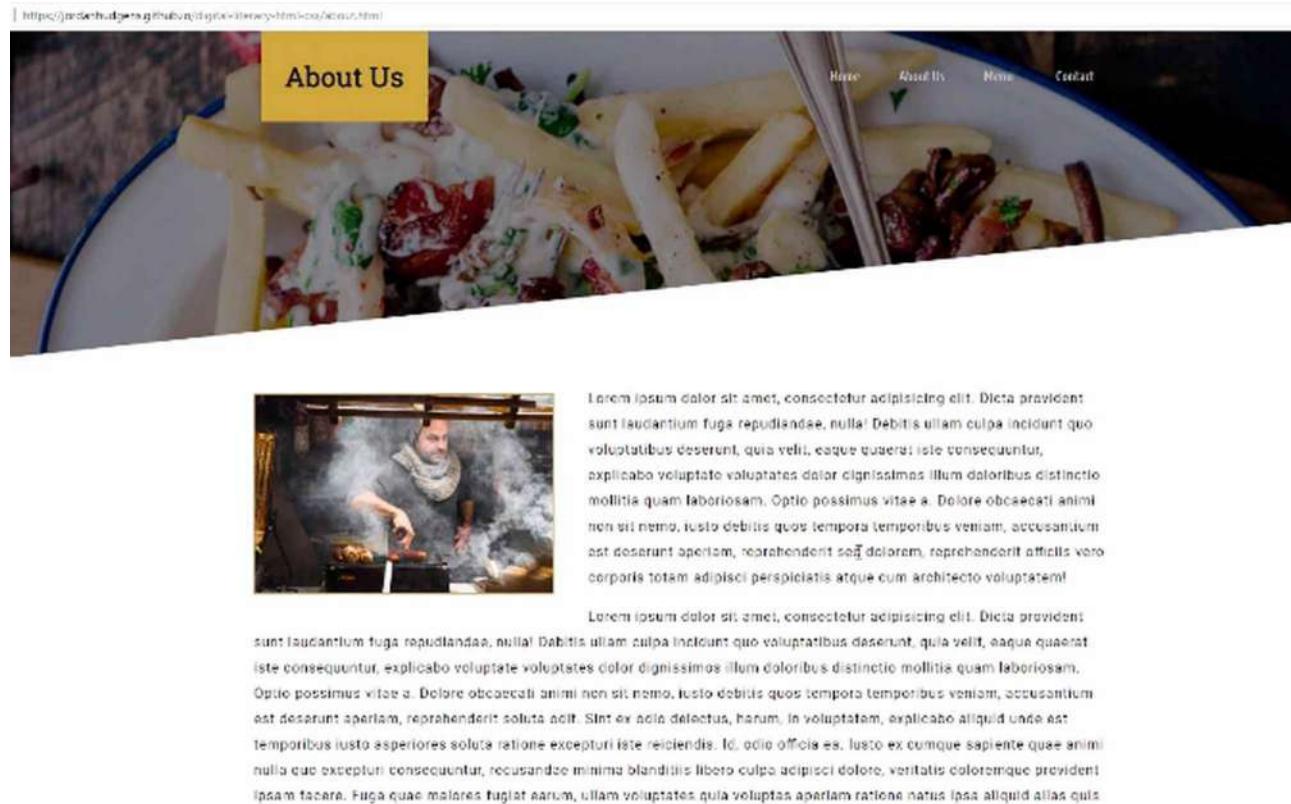
Now, not only did we just finish out the footer, we finished off the entire homepage. Fantastic job, and I know that that took a while, but I can tell you from experience that when I was originally starting to build websites, years and years ago, even just building out a homepage like this, would have taken a much longer time than it just took us.

It may have seemed like it took a while to build all this out, but in all reality, it really only took a few hours. When I was originally building out websites like this, before there were tools like Flexbox and CSS Grid, this would've taken hours and hours and hours, and even days.

I just want you to understand that using these modern types of technologies really helps to make the entire development process much more efficient. As you saw here, when we got down to the footer, we're using the same type of tools and skills that we used here in the navbar, but we're able to do it much faster.

Because you've already learned a lot of these skill sets you need, in order to align items on the page, and so, the more you do this, it's going to really go quite quickly for you.

Now that we have all of this done, it is time to move on to the about page. Right now, there's nothing on the about page. If you want to see what it's going to look like, in the final version, we are going to have a cool interface here.



Where we're going to be able to work with these kinds of boxes, layout items, and we're going to be able to see how we can also implement a navbar that looks a little bit different but still has the same behavior.

One of my favorite new little design tools is working with the `clip-path`, so you're going to see how you can slice images, and do all of that in CSS. I will see you in the next section when we start building this out.

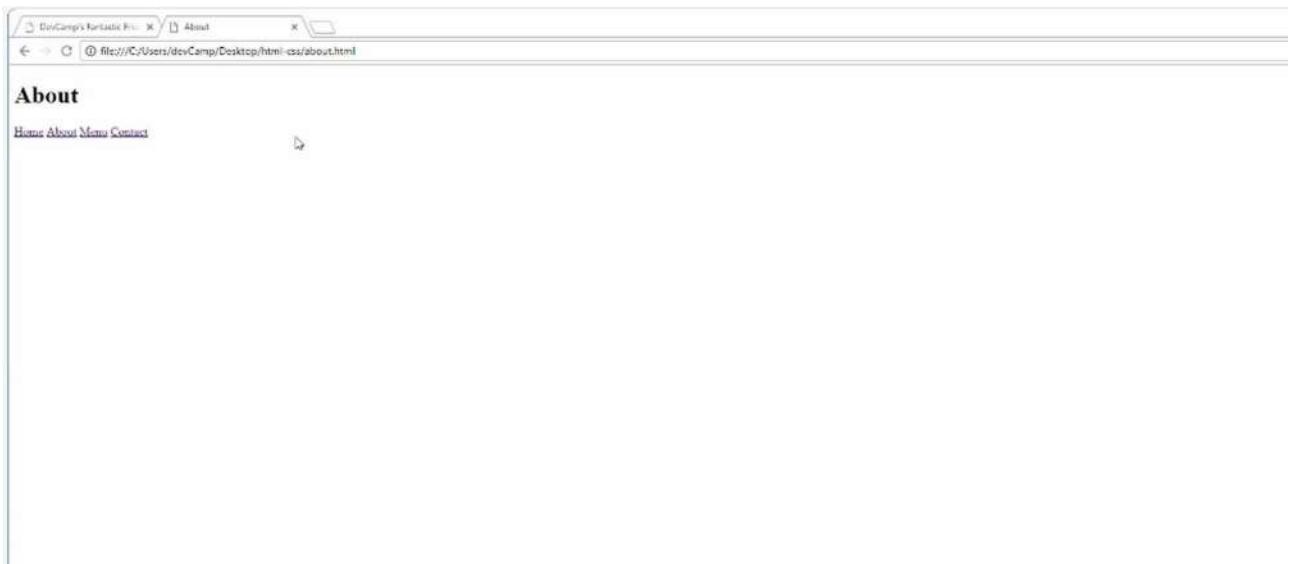


## Coding Exercise

The link text needs to be uppercase. The brand name needs a font size of `17px` and letter spacing of `4px`

# 1.37 Creating an About.html

We're going to start with this header element, and we're also going to bring in all of the common styles, fonts, and all of those types of dependencies into the project. So, I have the "About" page here, which right now is pretty boring but that is about to change.



Let's open up our file system. And moving to the "About" page, we're going to be able to reuse quite a bit of what we have in the index, so, let's actually get rid of everything here, and I'm going to grab the first content here at the top of the index page. So, let's just copy everything up to the body.

## about.html

```
<!DOCTYPE html>
<html>
<head>
    <meta charset='utf-8'>
    <title>About</title>

    <link href="https://fonts.googleapis.com/css?family=Roboto"
rel="stylesheet">
    <link href="https://fonts.googleapis.com/css?family=Roboto+Slab"
rel="stylesheet">
    <link href="https://fonts.googleapis.com/css?family=Ubuntu+Condensed"
rel="stylesheet">

    <link rel="stylesheet"
href="https://use.fontawesome.com/releases/v5.2.0/css/all.css"
integrity="sha384-hWVjflwFxL6sNzntih27bfxkr27PmbbK/iSvJ+a4+0owXq79v+lsFkW54
bOGbiDQ">
        crossorigin="anonymous">
    <link rel="stylesheet" href="styles/common.css">
    <link rel="stylesheet" href="styles/navigation.css">
</head>
<body>
</body>
</html>
```

Now, we can put a heading just to test it out, and say, "Hey" just to make sure we didn't break anything.

## about.html

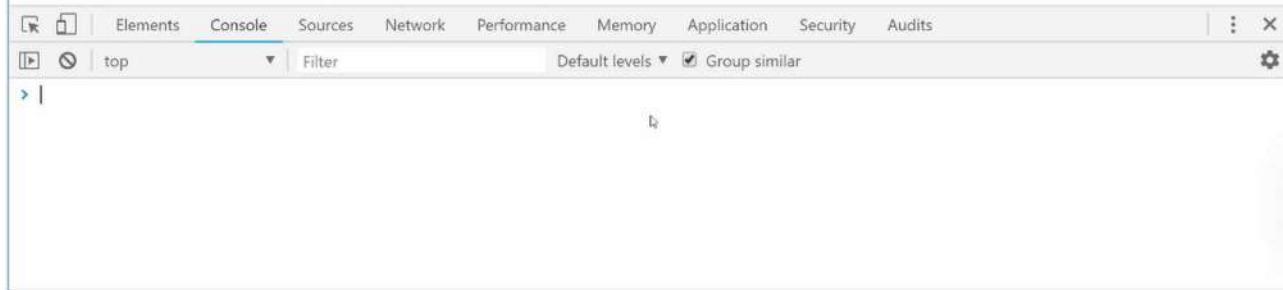
```
<body>
  <h1>Heyyy</h1>
</body>
```

Hit refresh. And you can see that that's all working.

Heyyy

Now, also another good practice just to make sure is to click inspect, and make sure that in the console here, you're not getting any errors because if you have any kind of issue, such as one of the CSS files not being brought in properly, that would give you an error, but it looks like all of that is working.

Heyyy



Now that we have that in place, now we can start bringing in our structure. So, let's take a look at what we're going to need. We definitely are going to need a spot to put this cool skewed item, which is going to be kind of the focal point.



Lorem ipsum dolor sit amet, consectetur adipisicing elit. Dicta provident sunt laudantium fuga repudiandae, nullat. Debitis ullam culpa incidentum quo voluptatibus deserunt, quia velit, eaque querat iste consequuntur, explicabo voluptate voluptates dolor dignissimos illum doloribus distinctio mollitia quam laboriosam. Optio possimus vitae a. Dolore obcaecati animi non sit nemo, iusto debitis quos tempora temporibus veniam, accusantium est deserunt aperiam, reprehenderit sed dolorem, reprehenderit officiis vero corporis totam adipisci perspicillatis atque cum architecto voluptatem!

Lorem ipsum dolor sit amet, consectetur adipisicing elit. Dicta provident sunt laudantium fuga repudiandae, nullat. Debitis ullam culpa incidentum quo voluptatibus deserunt, quia velit, eaque querat iste consequuntur, explicabo voluptate voluptates dolor dignissimos illum doloribus distinctio mollitia quam laboriosam. Optio possimus vitae a. Dolore obcaecati animi non sit nemo, iusto debitis quos tempora temporibus veniam, accusantium est deserunt aperiam, reprehenderit sed dolorem, reprehenderit officiis vero corporis totam adipisci perspicillatis atque cum architecto voluptatem!

We'll take care of that in one of the next guides but for right now we're just going to add the HTML, we have this heading up here and then we have the nav component. So, let's see what we need to do in order to bring all of this in.

I'm going to start off by creating a wrapper div here, and it's going to wrap up that image, the nav bar, everything like that.

## about.html

```
<body>
  <div class="skewed-header">
    <div class="header-bg"></div>

    <div class="skewed-header-wrapper">
      <div class="skewed-header-content">
        <div class="heading-wrapper">
          <h1>About Us</h1>
        </div>
      </div>
    </div>
  </div>
</body>
```

If you're wondering why I'm setting up the structure like this, it's because we're going to be using tools like Flexbox and CSS Grid, and remember, the more divs we have, the easier it's going to be to control the elements and select them.

Let's go and let's add our links wrapper. Now, we are going to be able to not just reuse the styles, we're actually going to be able to reuse the homepage elements, so, all of the links themselves, just like we did with the footer.

## about.html

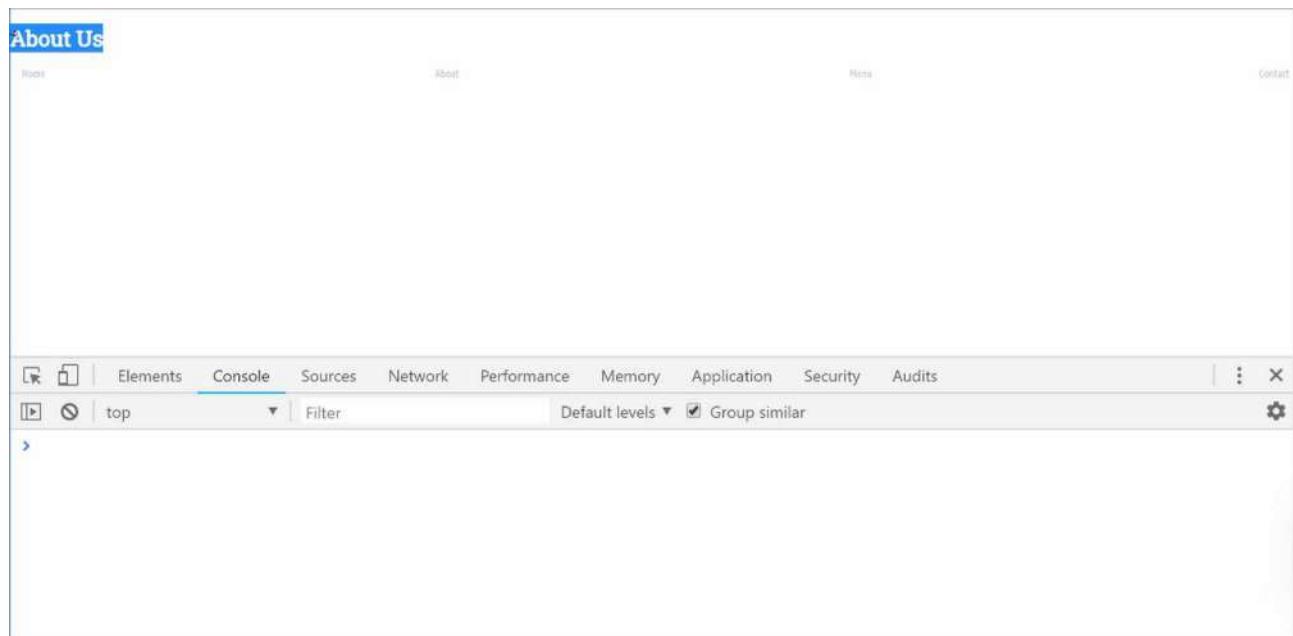
```
<div class="skewed-header-content">
  <div class="heading-wrapper">
    <h1>About Us</h1>
  </div>

  <div class="links-wrapper">
    <div class="nav-link">
      <a href="index.html">Home</a>
    </div>
    <div class="nav-link">
      <a href="about.html">About Us</a>
    </div>
    <div class="nav-link">
      <a href="menu.html">Menu</a>
    </div>
    <div class="nav-link">
      <a href="contact.html">Contact</a>
    </div>
  </div>
</div>
```

Any time that you find yourself building out a system like this, it's really a good idea to be able to figure out how you can manage it so that you can share styles as much as possible. The reason for that is just that it makes the entire layout much more scalable.

You're able to make a change in one part of the application, and then it'll populate to the rest of that. It's definitely a best practice when it comes to building out websites.

Okay, so now that we have our links wrapper there, this should be all of the content that we're going to need. So, if I hit refresh, you can see that it has our links wrapper, it has the "About Us" section, and everything that we're going to need is actually there.



It may look very different than what we have at the moment here, but actually, this is all the HTML structure we're going to have to have. And so, in the next guide, we're going to dive into how we can work with the clip-path property so that we can create those cool skewed images.



## Coding Exercise

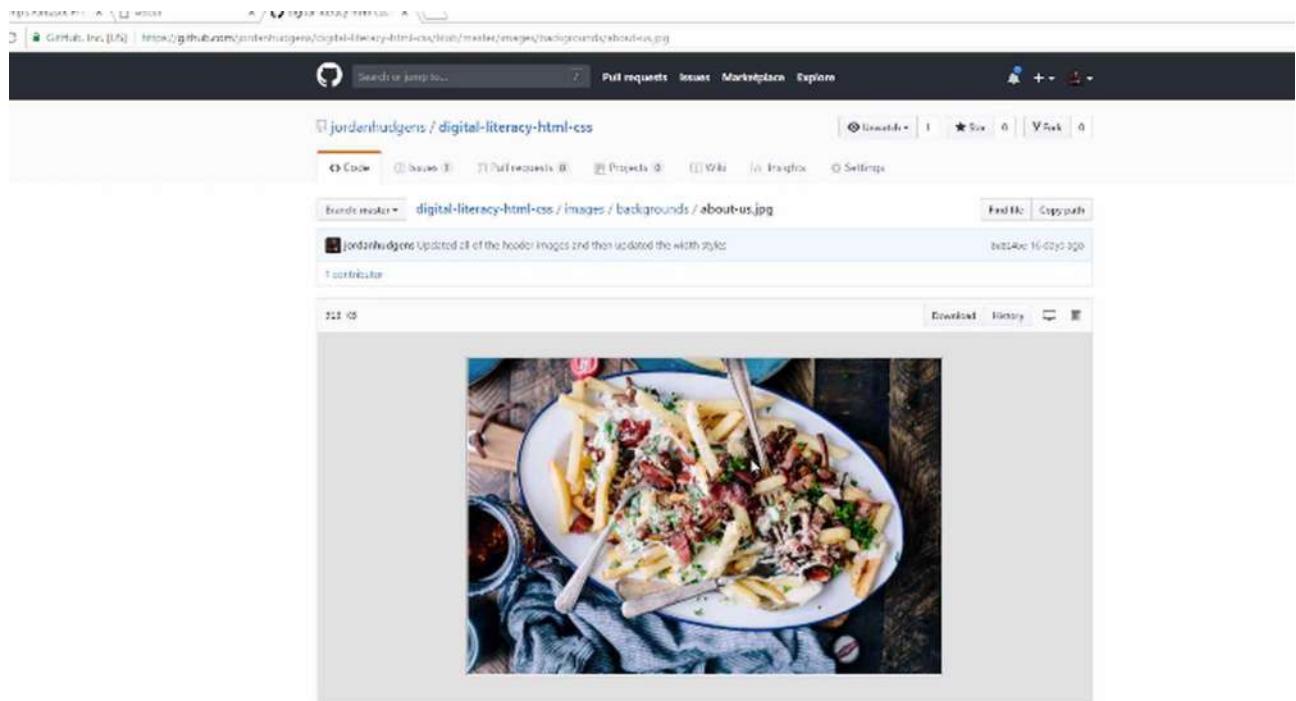
There is a syntax error on two of the links and they need to be fixed! It could be a character missing or one that's not supposed be there!

```
<div class="header-links">  
  <div class="link-background">  
    <a class="links" href="home.html">HOME</a>  
  </div>  
  <div class="link-background">  
    <a class="links" href="menu-html">MENU</a>  
  </div>  
  <div class="link-background">  
    <a class="links" href="order.html">ORDER HERE</a>  
  </div>  
</div>
```

# 1.38 Skew Images in CSS

We're also going to walk through how we can work with the `absolute` and `relative` positioning, so that we can have page elements, such as this heading and this navbar, to sit on top of this image.

Now, I will provide an image for you, though you definitely can feel free to use your own images. If you don't have one that you'd like to use, then I have one here, and I'll include this in the show notes. This is one that's going to be for an about page.



We can click on `save image` here, and I'm just going to save it into the **backgrounds** directory in the project. Click save, and that brings it down, so we have access to it and can work with it. Switching over into the code, what I'm going to do is I'm going to use a combination of `inline` and `external` styles.

The reason why I'm going to do that is because if you want to use a background image and you want to use different images for each page, so for example, we have this about page, and say we have a menu page and a contact page. Do you notice how the background image, or the skewed image, changes for each one of them?

Well, if you try to add a background to the CSS file, you'd have to create separate classes for each one of those backgrounds. I don't really want to do that, so the way we can get around that is by actually adding in an inline style.

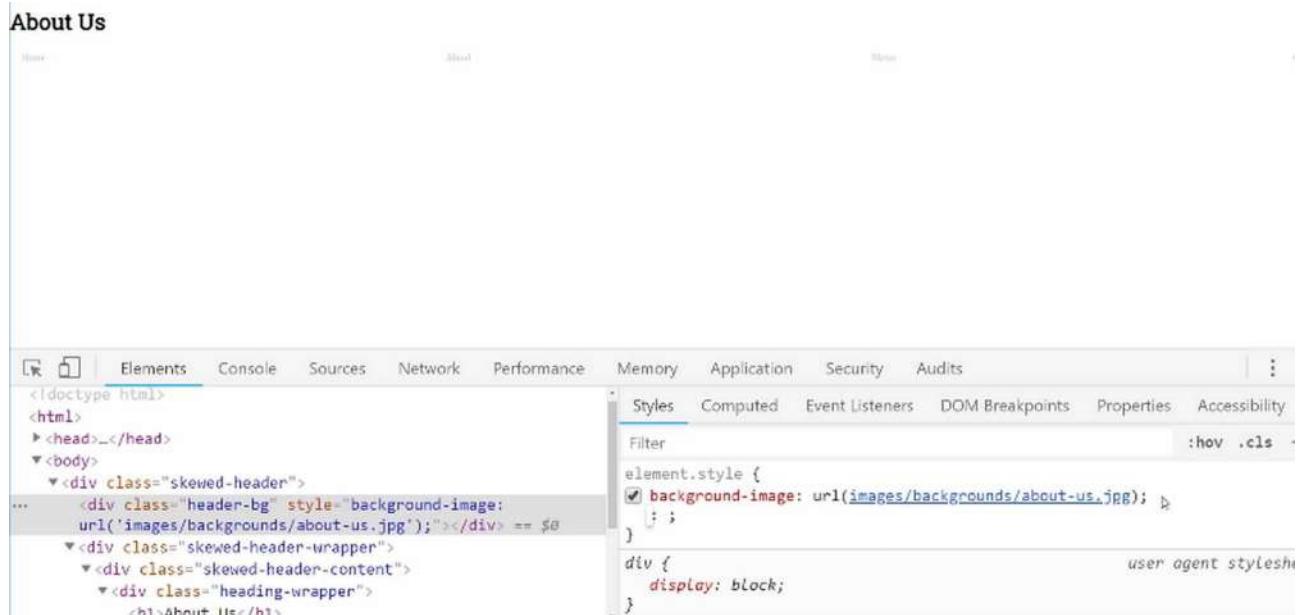
This is a great example of when an inline style could work quite nicely. What I'm going to do here is I'm going to start off by adding inside of this `header-bg` div class the inline style. I'm going to say `style` and then we'll say `background-image`, and then I want to use a URL.

Inside of the URL and use single backticks here, I'm going to grab `images/backgrounds/about-us.jpg`. I believe that should give us access to it, and then at the very end put a semi-colon there.

## about.html

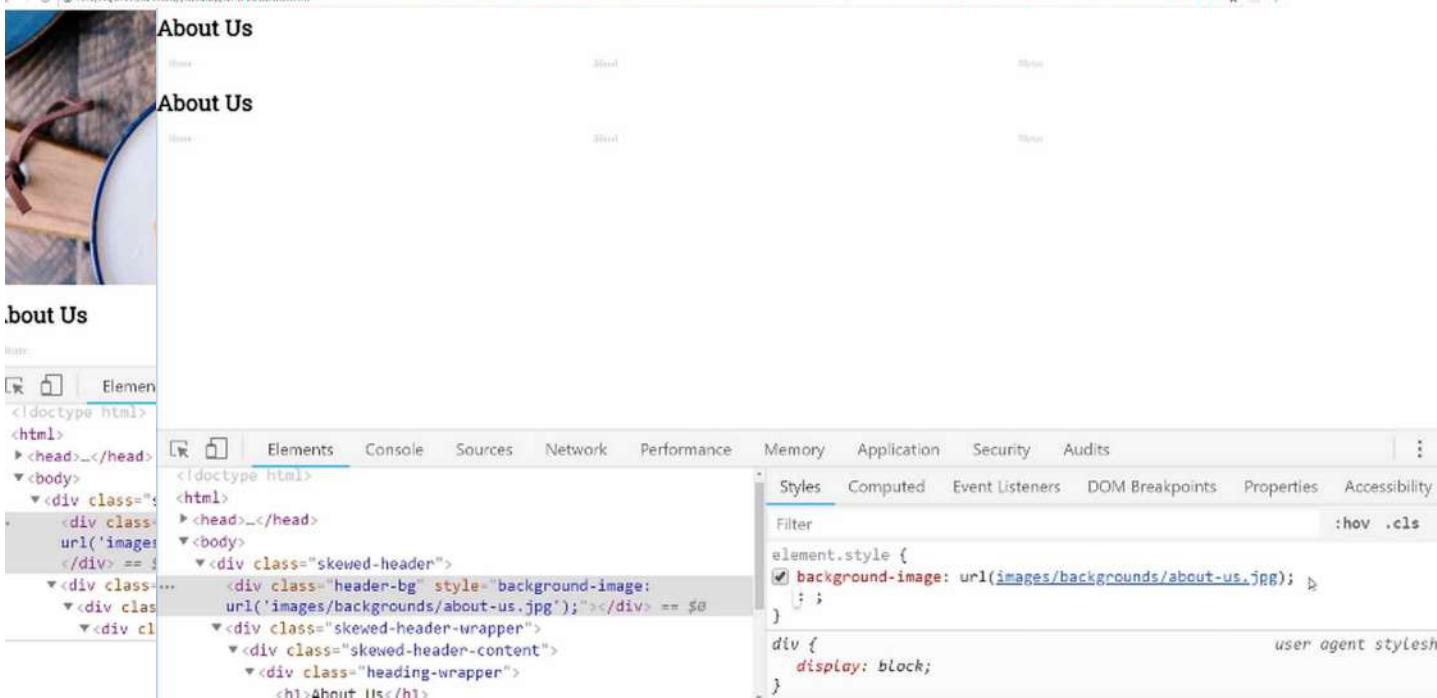
```
<div class="header-bg" style="background-image:  
url('images/backgrounds/about-us.jpg');"></div>
```

Now, this should give us the background image. Now if you hit refresh, let's see if we have maybe a typo, let's select it. Actually, I know what the issue is, so this is a good one, we're going to click on it and notice how it is finding it and you can click there. It'd give us an error if it was an issue.



The screenshot shows the browser's developer tools open to the 'Elements' tab. The DOM tree on the left shows the structure of the 'about.html' page. A specific element, a `div` with the class `header-bg`, is selected. In the inline styles of this element, the `background-image` property is set to `url('images/backgrounds/about-us.jpg')`. On the right side, the 'Styles' panel displays the CSS rule that corresponds to this inline style: `element.style { background-image: url(images/backgrounds/about-us.jpg); }`. The browser window above the tools shows a blank white page with the title "About Us".

We need to give it a height, so if I say a height of say, `400px`, you can see that it's right there. We're going to do that but we're going to do it separately. We now know that the background image is being pulled in properly.



Now if we go and I'm going to add a full set of custom styles here. I'm going to say these are the styles for the **Skewed Header**. The very first thing I'm going to do is I'm going to grab the `skewed-header-wrapper`, just that kind of generic class, and I'm going to use `position: relative`. I'll talk about why we need to do that here in a moment.

Then I'm going to give the height, just like we talked about, of `400px`. I'm going to say `overflow is hidden`. All of these items are necessary for implementing the `clip-path` that we're going to have.

### common.css

```
/* Skewed header */.skewed-header {position: relative; height: 400px; overflow: hidden;}
```

The reason why we need to use this `position: relative` is because anytime that you want to have this type of behavior where you have elements that are sitting on top of an image like this, then you need to use what is called a `position absolute` property.

In order to use `position: absolute`, the parent element needs to use `position: relative`. Now that may not make any sense to you if you've never done that before, but it is a rule in CSS.

It's something that I have run into a number of bugs through the years where I was trying to add absolute positioning. I was trying to create something like this and I had forgotten to put `position relative` in the parent class and so it didn't work.



Lorem ipsum dolor sit amet, consectetur adipiscing elit. Dicta provident sunt laudantium fuga repudiandae, nulla! Dabitis ullam culpa incident quo voluptatibus deserunt, quia velit, eaque quaerat iste consequuntur, explicabo voluptate voluptates dolor dignissimos illum doloribus distinctio mollitia quam laboriosam. Optio possimus vitae a. Dolore obcaecati animi non sit nemo, iusto debitis quos tempora temporibus veniam, accusantium est deserunt apertam, reprehendit sed dolorum, reprehendit officiis vero corpora totam adipisci perspiciat is atque cum architecto voluptatem!

Lorem ipsum dolor sit amet, consectetur adipisciing elit. Dicta provident sunt laudantium fuga repudiandae, nulla! Dabitis ullam culpa incident quo voluptatibus deserunt, quia velit, eaque quaerat iste consequuntur, explicabo voluptate voluptates dolor dignissimos illum doloribus distinctio mollitia quam laboriosam. Optio possimus vitae a. Dolore obcaecati animi non sit nemo, iusto debitis quos tempora temporibus veniam, accusantium est deserunt apertam, reprehendit sed dolorum, reprehendit officiis vero corpora totam adipisci perspiciat is atque cum architecto voluptatem!

That's something you just always have to know, if you want to have this overlay effect like we're going to build out and you want to use position absolute, the parent element does have to have that value. Now if I hit refresh, you can see we still need to update some values here in the header background.



```

<!DOCTYPE html>
<html>
  <head>...</head>
  <body>
    <div class="skewed-header">
      <div class="header-bg" style="background-image: url('images/backgrounds/about-us.jpg');"></div> == $0
      <div class="skewed-header-wrapper">...</div>
    </div>
  </body>
</html>

```

Styles Computed Event Listeners DOM Breakpoints Properties Accessibility

Filter

```

element.style {
  background-image: url(images/backgrounds/about-us.jpg);
}
div {
  display: block;
}

```

Inherited from body

We have the skewed header, now let's come and select the background. Let's go `skewed-header` and then we want the element inside called `header-bg`. Inside of `header-bg`, this is where we're going to start off by saying we want the `position: absolute` and then we want it at the very top.

## common.css

```
/* Skewed header */
.skewed-header {
  position: relative;
  height: 400px;
  overflow: hidden;
}

.skewed-header > .header-bg {
  position: absolute;
}
```

We're going to say, I want to use the `top` property of `0`, I want the `bottom` to be `0`, I want the `right` to be `0` and as you may have guessed, I want the `left` to also be `0`. That's going to give us the ability to have the image sitting flush on all sides.

## common.css

```
.skewed-header > .header-bg {
  position: absolute;
  top: 0;
  bottom: 0;
  right: 0;
  left: 0;
}
```

Then I want the `height` to be `100%` and I want the same thing with the `width`. Now let's add some of our styles for using background images. We already are bringing in the background image inline, but we can still add background image properties here.

## common.css

```
.skewed-header > .header-bg {
  position: absolute;
  top: 0;
  bottom: 0;
  right: 0;
  left: 0;
  height: 100%;
  width: 100%;
  background-repeat: no-repeat;
  background-size: 100%;
}
```

So I'll say `background-repeat`, we want `no-repeat`. Then we want the `background-size` to be `100%`. We want it to take up the full width. Now let's hit save here, hit refresh, and you can see that is working.



Screenshot of the Chrome DevTools Elements tab showing the HTML structure and the Styles panel. The background image is defined in the CSS.

```
<!DOCTYPE html>
<html>
  <head>...</head>
  <body>
    <div class="skewed-header">
      <div class="header-bg" style="background-image: url('images/backgrounds/about-us.jpg');"></div> == $0
    <div class="skewed-header-wrapper">...</div>
  </div>
</body>
```

Styles panel:

```
element.style {
  background-image: url(images/backgrounds/about-us.jpg);
}
.skewed-header > .header-bg {
  position: absolute;
  top: 0;
  bottom: 0;
```

common.css:

Now we have our image and everything there looks nice. Now what we need to do now is to implement our skew. The way we're going to do that is pretty cool and let's test it out here first. What we're going to use is the `transform` property.

We're going to say `transform` and I want to use a `skewY`. This is going to use the `y radius` and you also have to pass in the value. So `skewY` and then `skewY` is going to take in a value. I'm going to say `-6deg`, just like this. You can see that that gives us a cool little skew element, but we're not done yet.



Screenshot of the Chrome DevTools Elements tab showing the HTML structure and the Styles panel. The transform properties are added to the background element.

```
<!DOCTYPE html>
<html>
  <head>...</head>
  <body>
    <div class="skewed-header">
      <div class="header-bg" style="background-image: url('images/backgrounds/about-us.jpg');"></div> == $0
    <div class="skewed-header-wrapper">...</div>
  </div>
</body>
```

Styles panel:

```
top: 0;
bottom: 0;
right: 0;
left: 0;
height: 100%;
width: 100%;
background-repeat: no-repeat;
background-size: 100%;
```

transform: skewY(-6deg);

We need to say where we want the transform to happen. So I'm going to say `transform-origin` and we're going to say `top-left`. You can see that's working perfectly.



Screenshot of the Chrome DevTools Elements tab showing the HTML structure and the Styles panel.

HTML Structure:

```
<!doctype html>
<html>
  <head>...</head>
  <body>
    <div class="skewed-header">
      <div class="header-bg" style="background-image: url('images/backgrounds/about-us.jpg');"></div> == $0
    <div class="skewed-header-wrapper">...</div>
  </div>
</body>
</html>
```

Styles Panel:

Filter: :hover .cls +

Style	Computed	Event Listeners	DOM Breakpoints	Properties	Accessibility
bottom: 0; right: 0; left: 0; height: 100%; width: 100%; background-repeat: no-repeat; background-size: 100%; transform: skewY(-6deg); transform-origin: top left;					

That is how you can use the `transform` property in order to get this to work. Earlier I kind of misspoke, I said `clip-path`, but we're actually using the `skew` and the `transform` property in order to make that possible.

I'm also going to add a filter. So I'm going to say `filter: brightness()` and the autocorrect

Screenshot of the browser showing the "About Us" page and the DevTools Elements tab.

Page Title: About Us

DevTools Elements Tab:

HTML Structure:

```
<!doctype html>
<html>
  <head>...</head>
  <body>
    <div class="skewed-header">
      <div class="header-bg" style="background-image: url('images/backgrounds/about-us.jpg');"></div> == $0
    <div class="skewed-header-wrapper">
      <div class="skewed-header-content">
        <div class="heading-wrapper">
          <h1>About Us</h1>
        </div>
      </div>
    </div>
  </body>
</html>
```

Styles Panel:

Filter: :hover .cls

Style	Computed	Event Listeners	DOM Breakpoints	Properties	Accessibility
element.style { background-image: url(images/backgrounds/about-us.jpg); }; div { display: block; };					user agent stylesheets

here is not helping me out. So I just want to darken this a little bit. I'm going to say `brightness` and we'll say `50%`. That darkened it.



The screenshot shows the Chrome DevTools interface with the 'Elements' tab selected. The left pane displays the DOM tree:

```
<!doctype html>
<html>
  <head>...</head>
  <body>
    <div class="skewed-header">
      <div class="header-bg" style="background-image: url('images/backgrounds/about-us.jpg');"></div> == $0
      <div class="skewed-header-wrapper">...</div>
    </div>
  </body>
</html>
```

The right pane is expanded to show the 'Styles' tab, which lists the following CSS properties:

- right: 0;
- left: 0;
- height: 100%;
- width: 100%;
- background-repeat: no-repeat;
- background-size: 100%;
- transform: skewY(-6deg);
- transform-origin: top left;
- filter: brightness(50%);

Now when we want to have elements sitting on top of it, that will work nicely. I'm just going to grab these transform properties here and put them in the actual code. Now they'll be permanent. I'll hit save, hit refresh and that is looking really good.

Now that we have all of that in place, let's go and let's actually have the content. We'll update the content so that it allows for being able to lay on top of this image. If you look at the **about.html** right here, we have the **skewed-header-content**, we have the **heading-wrapper**, and then a **links-wrapper**.

```
# homepage.css # common.css < about.html x # nav.css

17 y>
18 <div class="skewed-header">
19     <div class="header-bg" style="background-image: url('images/backgrounds/about-us.jpg');">
20
21     <div class="skewed-header-wrapper">
22         <div class="skewed-header-content">
23             <div class="heading-wrapper">
24                 <h1>About Us</h1>
25             </div>
26
27             <div class="links-wrapper">
28                 <div class="nav-link">
29                     <a href="#">index.html>Home</a>
30                 </div>
31
32                 <div class="nav-link">
33                     <a href="#">about.html>About</a>
34                 </div>
35             </div>
36         </div>
37     </div>
38 
```

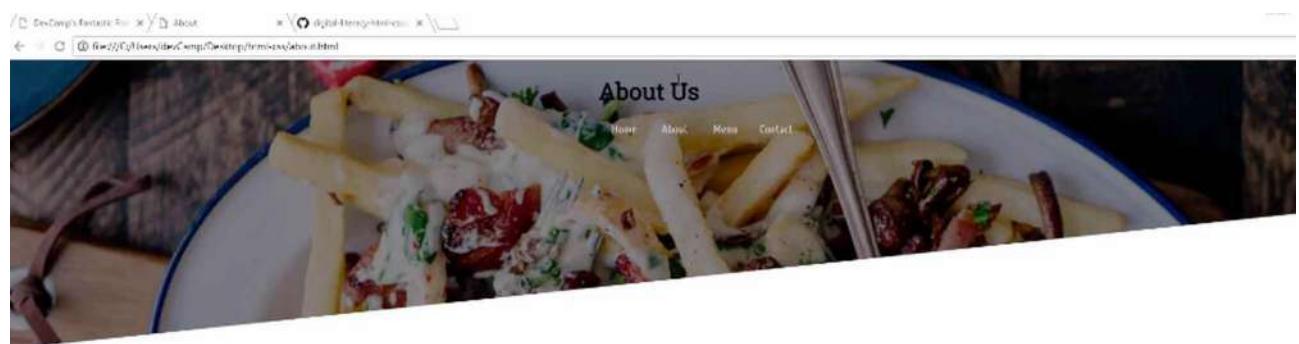
Those are the class names that we have to work with. Next, we can say that we want to have the `skewed-header` and then from there we want the `skewed-header-wrapper`. We want this to sit on top of the image, so here we'll say `position: absolute`.

There you go, and then we want this to utilize Flexbox, so I'm going to say `display: flex`, `justify-content: center`, and then I want the width to take up the full view width. So I'm going to say `width: 100vw`.

## common.css

```
.skewed-header > .skewed-header-wrapper {  
  position: absolute;  
  display: flex;  
  justify-content: center;  
  width: 100vw;  
}
```

What this does is it looks at the viewport, meaning it looks at the window and it says, "I want to take up 100% of that window space." Let's hit save, if you hit refresh, now you can see that that content is now laying on top of the image like we're wanting.

A screenshot of the developer tools in a browser. The left pane shows the DOM tree with the following structure:

```
<!doctype html>  
<html>  
  <head>...</head>  
  <body>  
    <div class="skewed-header">  
      <div class="header-bg" style="background-image: url('images/backgrounds/about-us.jpg');"></div> == $0  
      <div class="skewed-header-wrapper">...</div>  
    </div>  
  </body>  
</html>
```

The right pane shows the "Styles" tab with the following CSS code:

```
right: 0;  
left: 0;  
height: 100%;  
width: 100%;  
background-repeat: no-repeat;  
background-size: 100%;  
transform: skewY(-6deg);  
transform-origin: top left;  
filter: brightness(50%);
```

Now it's not done yet, we still have a few more styles to implement, but we are getting a lot closer. You can see that now we have the ability to have these nav elements here and the heading is also working as well.

Now that we have this `skewed-header-wrapper`, what's next? If we look at the about page you can see this is what we just worked on. Now we have `skewed-header-content`, this is the actual Flexbox container that will hold these elements.

Let's come down here and we'll copy that selector because we're going to be using part of it. Then if you come and grab the content, we'll say that we want the `skewed-header-content`. We want this to be only `1000px` wide.

The top one we want going from edge to edge, but here we're actually having the content, we want to wrap it in 1000px container. This is going to also use Flexbox. We're going to say `justify-content: space-between` because we want to have a space between the heading and then the nav elements.

### common.css

```
.skewed-header > .skewed-header-wrapper > .skewed-header-content {  
    width: 1000px;  
    display: flex;  
    justify-content: space-between;  
}
```

This is going to be one div, this is another div, and we want the space between to be what fits there. That's going to be how we separate that out. Then we also want to say `align-items: center` to center them vertically.

### common.css

```
.skewed-header > .skewed-header-wrapper > .skewed-header-content {  
    width: 1000px;  
    display: flex;  
    justify-content: space-between;  
    align-items: center;  
}
```

Now coming back, you can see that that is looking gorgeous, I really like the way that that is coming along. We are almost done with this entire heading, so moving down.



A screenshot of the Chrome Developer Tools Elements tab. The left sidebar shows the DOM structure: &lt;html&gt; -&gt; &lt;head&gt; -&gt; &lt;body&gt; -&gt; &lt;div class="skewed-header"&gt;. The right panel shows the "Styles" tab with a list of applied styles: "right: 0;" and "left: 0;". The status bar at the bottom indicates the file path: "file:///C:/Users/dew/Cmpsp2024/headers/index.html".

The next item that we're going to grab is going to be the `skewed-header`. Let's go the skewed-header.

We have the `skewed-header-heading-wrapper`, and then the `heading-wrapper`. If you notice that's really all we have left to do is to style this, so we put that nice little kind of gold block around here.

Let's come back here now that we know the class and we'll grab the `skewed-header` and then inside of that we're going to go with the `skewed-header-wrapper`. Inside of that, we're going to go with the `skewed-header-content`, and lastly and finally we get to the `heading-wrapper`.

#### common.css

```
.skewed-header > .skewed-header-wrapper > .skewed-header-content >
.heading-wrapper {  
}  
}
```

Now, technically, you could have just grabbed the `skewed-header-content` and then the `heading-wrapper`, and that would have been perfectly fine. I just wanted to make it really clear what element that we're grabbing here.

Now I want to have a `background-color` of that `#CEA135`. From there, I want to use dark blue for the coloring, so that's that `#11122B`. We've talked about `border-radius`, now I want to show you how we can be even more specific.

#### common.css

```
.skewed-header > .skewed-header-wrapper > .skewed-header-content >
.heading-wrapper {  
    background-color: #CEA135;  
    color: #11122B;  
}
```

You notice how in the final design, you notice how this box right here has a little bit of a rounded edge on the bottom-right and bottom-left-hand side, but it's not up here at the top. Well, we can be really specific with our border-radius.

We can say that I want to do a border-radius here but instead of just the regular border-radius, we can actually dictate that we only want it to be on the left and right bottom sides. The property for doing that is going to be `border-bottom-left-radius` and for that, I'm going to say `2px`.

#### common.css

```
.skewed-header > .skewed-header-wrapper > .skewed-header-content >
.heading-wrapper {  
    background-color: #CEA135;  
    color: #11122B;  
    border-bottom-left-radius: 2px;  
}
```

Then we want to do the same thing for the right, and then let's also give it a width. So we'll say a width of `175px`. We'll have `padding` all around it or all inside of it of `10px`. Lastly, let's align the text right in the center.

#### common.css

```

.skewed-header > .skewed-header-wrapper > .skewed-header-content >
.heading-wrapper {
  background-color: #CEA135;
  color: #11122B;
  width: 175px;
  border-bottom-left-radius: 2px;
  border-bottom-right-radius: 2px;
  padding: 10px;
  text-align: center;
}

```

If I hit save here and then hit refresh, you can see that that is working gorgeously. Right here we still have our nice nav component, it has all of its animations, we have our clipped kind of image here and we're able to use the transform.



A screenshot of the Chrome DevTools Elements tab. It shows the HTML structure of the page, including the skewed header component. The Styles panel on the right displays the CSS rules for the 'header-bg' class, which includes 'background-image: url('images/backgrounds/about-us.jpg')', 'background-repeat: no-repeat;', and 'background-size: 100%;'.

You learned how to use the transform both with the origin and just creating the skew, and you're able to put it all together. Great job if you went through that, this is definitely not a trivial thing to build out.

The first time that a designer handed me a design that had to have this, I'd never built it before and it took me a while to figure out. I'm really happy that I learned it because this is a skill that is really handy, especially right now.

A lot of designers that I've been working with have been putting together some skews like this. Great job if you went through that, you now know how to use `transform` and `skew`s in CSS.



## Coding Exercise

The below div tag needs a skew on the Y axis of -3deg and the origin needs to start at the top right. Make sure the position is absolute as well

```
<div class="my-img-box">
  
</div>
```

# 1.39 CSS Float property

With our skewed image complete, our next task is going to be to add the page content. We're also going to learn how we can have a featured image and have the content flowing around it.

This is going to be another really good guide because this is going to be a task that you're most likely going to have to implement in some form or another.

Let's switch back to our app, and then go to the code. Let's start off by adding our content. So you can see this is the skewed header portion. We want to come down below that. I'm going to create a page container class here.

## about.html

```
<div class="page-container">
  <div class="content-wrapper">
    <!-- IMG goes here -->
  </div>
</div>
```

This is where we'll have the image, and where we'll also have all the content.

Now, we can go and add our paragraph tags. So we have a few different types of ways we could do it. You could type a lot of content, you could copy and paste. But I find it's the fastest to use the Emmet tool. So I'm going to type `p>lorem200`

Then if I hit tab, this is going to give us lorem ipsum text with 200 words. Let's do the same thing, except this time, we're going to want a different number of words, say, 350 words. `p>lorem350`

## about.html

```
<div class="page-container">
  <div class="content-wrapper">
    <!-- IMG goes here -->

      <p>Lorem ipsum dolor sit amet, consectetur adipisicing elit. Dicta
      provident sunt laudantium fuga repudiandae, nulla! Debitis ullam culpa
      incidentum quo voluptatibus deserunt, quia velit, eaque quaerat iste
      consequuntur, explicabo voluptate voluptates dolor dignissimos illum
      doloribus distinctio mollitia quam laboriosam. Optio possimus vitae a.
      Dolore obcaecati animi non sit nemo, iusto debitibus quos tempora temporibus
      veniam, accusantium est deserunt aperiam, reprehenderit sed dolorem
      reprehenderit officiis vero corporis totam adipisci perspiciatis atque cum
      architecto voluptatem!</p>

      <p>Lorem ipsum dolor sit amet, consectetur adipisicing elit. Dicta
      provident sunt laudantium fuga repudiandae, nulla! Debitis ullam culpa
      incidentum quo voluptatibus deserunt, quia velit, eaque quaerat iste
      consequuntur, explicabo voluptate voluptates dolor dignissimos illum
      doloribus distinctio mollitia quam laboriosam. Optio possimus vitae a.
      Dolore obcaecati animi non sit nemo, iusto debitibus quos tempora temporibus
      veniam, accusantium est deserunt aperiam, reprehenderit soluta odit. Sint
      ex odio delectus, harum, in voluptatem, explicabo aliquid unde est
      temporibus iusto asperiores soluta ratione excepturi iste reiciendis. Id,
      odio officia ea. Iusto ex cumque sapiente quae animi nulla quo excepturi
      consequuntur, recusandae minima blanditiis libero culpa adipisci dolore,
```

```

veritatis doloremque provident ipsam facere. Fuga quae maiores fugiat
earum, ullam voluptates quia voluptas aperiam ratione natus ipsa aliquid
alias quis ut! Enim voluptatem minima, doloribus sed ullam vitae eius, eos,
officiis ex suscipit id harum voluptate. Nostrum fugit sed dolorem,
reprehenderit officiis vero corporis totam adipisci perspiciatis atque cum
architecto voluptatem!</p>
</div>
</div>
```

Hit save, and now, if you switch back and hit refresh, you'll see that we have all kinds of content.



We have two paragraphs, and they're different sizes, which is good, because if you just copied and pasted the first one, what I've found is that that's not really reality.

If you're building this for a client, or for your company, they're most likely going to have different size paragraphs. Whenever you're using placeholder content, I usually recommend that you do something like this.

Now that we have that, let's go get the image. I'll provide you a link in the show notes. We want to pull in that chef image. Hit `save image as` and then, inside of our images directory I'm going to create a new folder called `people` and that's where we will save the chef jpeg.

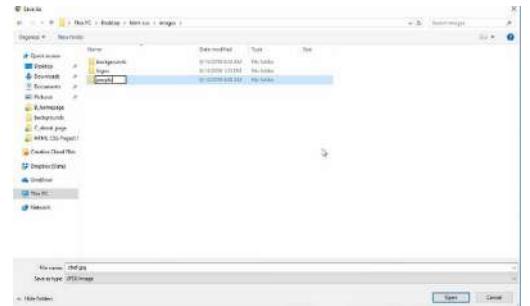
Now that we have access to the picture, let's go in and add that directly into the code base, right where our comment was.

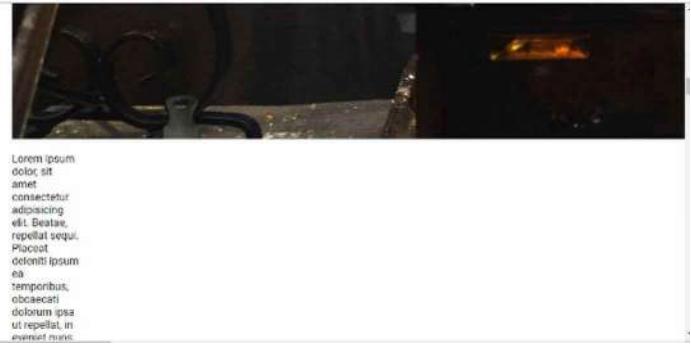
## about.html

```

<div class="page-container">
    <div class="content-wrapper">
        
```

Now let's verify that it's working.





And yes, that image is getting pulled in. It's absolutely gigantic, and that food looks pretty good. But we definitely do not want to keep it like that. So in our CSS, we will customize it.

So let's go back to `common.css`, and down on the bottom here, I'm going to add a new set of styles. We'll just call these our page container styles.

### **common.css**

```
/* Page container styles */
.page-container {
    display: flex;
    justify-content: center;
}
```

Now, if I hit save, and hit refresh, nothing's really changing, because this image is throwing it off. But it is inside of that wrapper. So now, let's go and let's add a content wrapper class.

### **common.css**

```
.page-container > .content-wrapper {
    width: 1000px;
}
```

Go back and check it out.



Now you can see that it's getting a little bit better. We still have quite a bit of work to do. But at least we can see that we're selecting the right items.

Let's keep moving down the line. This is working out well. The next thing we want to do is grab that image. So with the about us, right now, we just have this image tag. And what I'd like to do is I want to be able to grab it and then move it to the left, and then have the content floating around it.

This is going to be one of the cases where I will use an ID for this image. And I'm going to call it chef.

```

```

I'm going to have an ID of chef, because I know there's going to be only one ID of chef on this page. And so it's fine to use an ID in that case.

in our `common.css`, what I can do is I'm going to grab this chef ID. And remember, we grab an ID by giving the hash symbol, and then whatever the name of that ID is.

### common.css

```
.page-container > .content-wrapper > #chef {  
    width: 350px;  
    margin: 25px;  
    float: left;  
}
```

I'm going to float this to the left. And that's how you can have content flowing around the image itself. So now, if you hit refresh, you can see the image has shrunk down.



Now, when I've had bugs, and I fix them, do you notice how many times the way I've fixed them is just by talking it through? Well, that is a very popular technique in the developer community, and there's even a special name for it.

It's called rubber duck debugging. Hopefully, you don't mind if we take a little side trip, because this is worth it. You can google it and research it yourself.

What the concept is, and if you ever are walking around a dev shop and you see on someone's desk a little rubber duck, that is what we're talking about here.

The principle of rubber duck debugging is kind of like what we just went through. One thing that happens many times with developers is they'll run into a bug, and then they cannot understand why something's not working.

But instead of really working through the issue, we just kinda stare at the screen and hit refresh a bunch of times, and we don't really know where to start debugging.

The concept is, you'd have this rubber duck on your desk, and then whenever you run into a bug, you would explain the bug to the duck. And many times just by talking through the problem and explaining it out loud, you find out what the fix is.

That's what rubber duck debugging is, in case you ever hear that or you feel the need to talk it through, that is a principle that I've used through the years, and it works quite well.

Now, to get back to the fun stuff. You can see that everything is working. So the float property is a very powerful tool and it used to be the way that we built entire websites. Years ago, whenever you wanted to have items align next to each other, we didn't have tools like CSS grid, or Flexbox.

We used floats in order to align item, and it was very messy. It was not a fun time. It was kinda like the dark ages of CSS and HTML. But now, floats are really just used for doing exactly what we're doing right here, where we want to have content that is wrapping around an image.

So this is looking good. But now we need to clean it up just a little bit. I don't really want the margin on the left-hand side, so with this margin property, let's play with this a little bit. So I might want a little bit on the top.

### common.css

```
.page-container > .content-wrapper > #chef {  
    width: 350px;  
    margin: 23px 40px 20px 0;  
    float: left;  
}
```

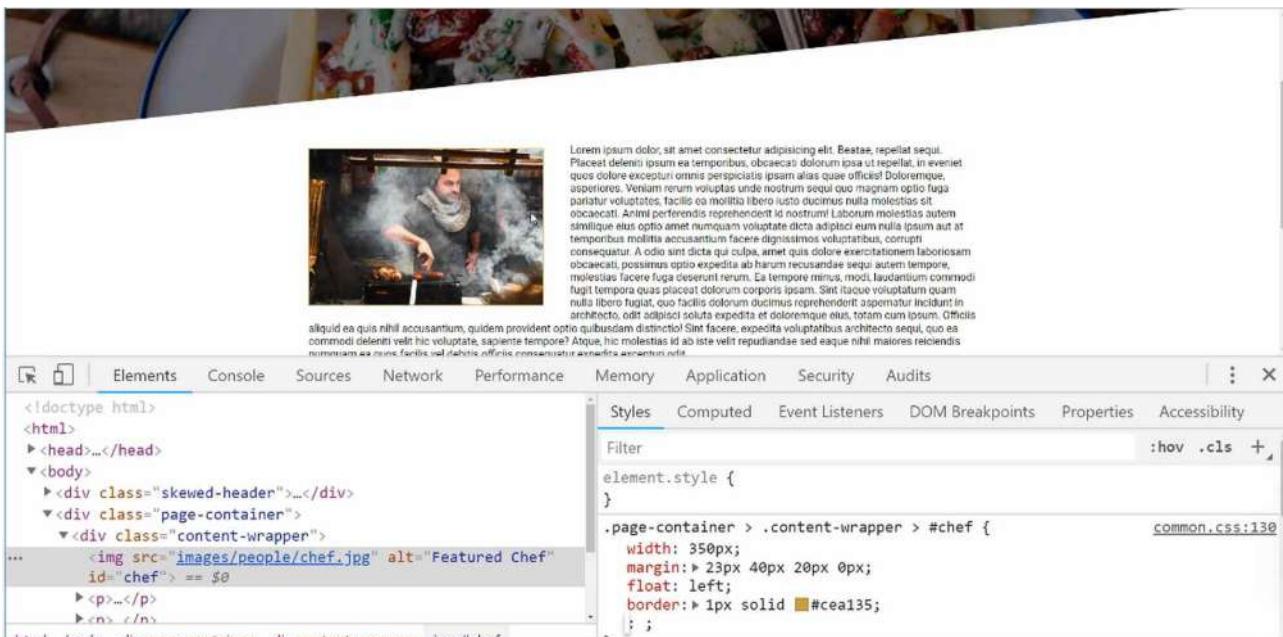
Hit refresh.



And that's looking really good. And feel free to play with that however you want. But I think that that's looking pretty nice right there. And now, the text is just flowing right around the image.

Now, one other thing I would like to add is a nice border. If you look at the finished version, you can see there's a little gold border going around the image. And that's something we haven't done yet. So let's try to do that.

I'm going to just add a border with our browser tools, and I want it to be one pixel wide. I want it to be solid. And then, let's go with that #cea135 color. And you can see, that gives it just a little bit of pop.



I like that. I think that looks pretty good. So let's copy this.

### common.css

```
.page-container > .content-wrapper > #chef {  
    width: 350px;  
    margin: 23px 40px 20px 0;  
    float: left;  
    border: 1px solid #cea135;  
}
```

I think that's something that we should be happy with.

And the last element that I want to update is I don't like the way that this text looks. So if you look at what we have in our design, do you notice how we have additional space? And that extra space, extra line space, really helps to make it more readable.



And it looks more kinda like what you'd expect in a lifestyle website or lifestyle blog, like a restaurant. So let's fix that because it's too closely placed together, and this doesn't look like something I would want to read.

So let's go back, and let's update that.

### common.css

```
.page-container > .content-wrapper p {  
    letter-spacing: 1px;  
    line-height: 30px;  
}
```

If you have ever worked with Microsoft Word and you wanted to have something like double-spaced or anything like that, that's what the line-height property allows you to do.

Now if you hit refresh, you can see that that looks so much better.



I'm really a fan of that type of look and feel compared to what we had. This is starting to look more professional.

So, great job if you went through that. You now know how to have a text running around and flowing around an image, as well as working with other text properties such as line height.



## Coding Exercise

On the below container class, place a margin with the top set to **10px**, right side to **23px**, bottom to **34px**, and the left side to **0px**. Also give it a border that's **2px** with a solid color of your choice

```
<div class="container">  
    <p>Pretend that theres a lot of text in here</p>  
</div>
```

## 1.40 HTML Square Grid

Moving our way down the about us page, you can see the next component that we're going to build is going to be this box grid.

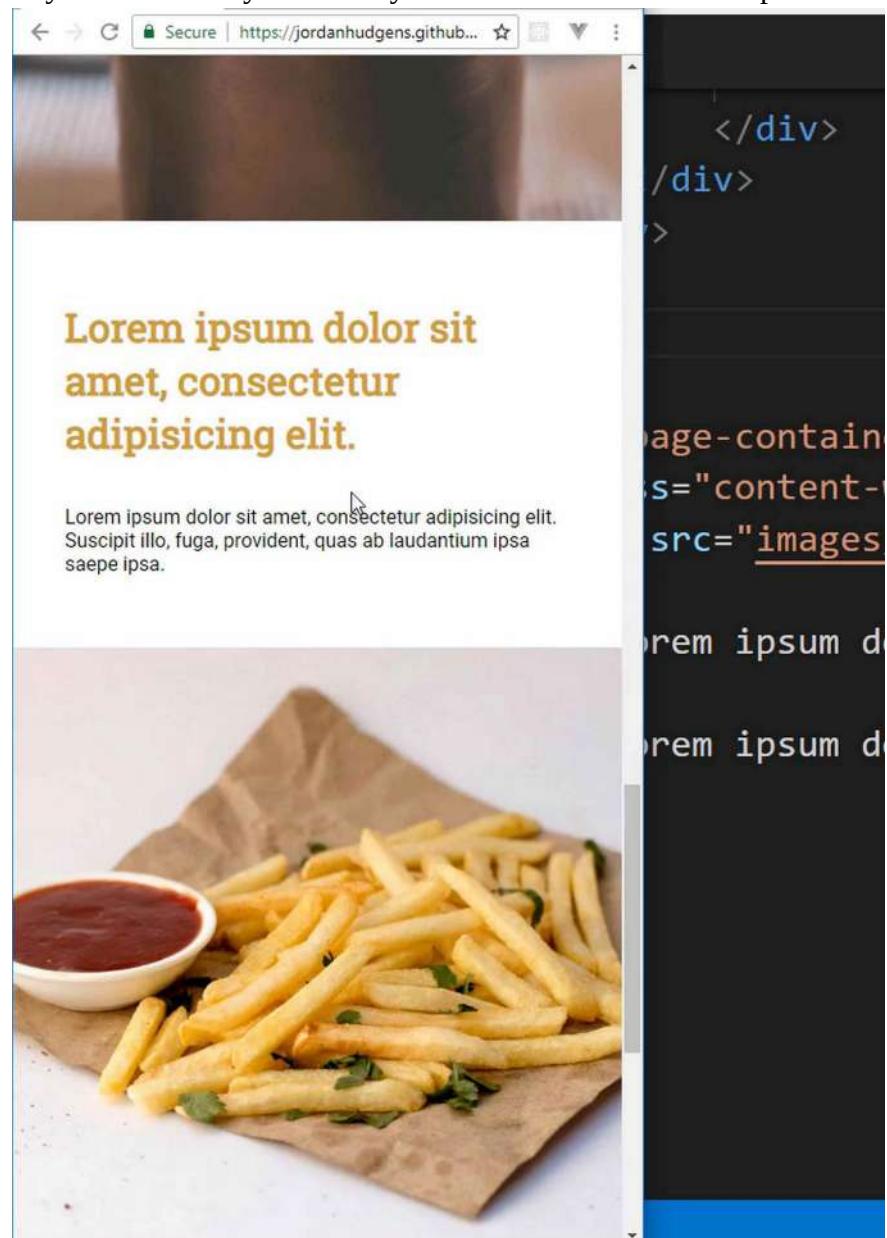
And there is a very important reason why I picked out this specific type of layout and I did it for multiple pages. I did it on the about page and the menu page is pretty much this entire set of images and then right next to the images are the content elements.

The reason why I did that was because one, I've been asked to build this type of layout many times and so I'm assuming that at some point in your career you'll be asked to work on this kind of system as well. In addition to that, this type of layout will force you to really reinforce a lot of the concepts that we've learned so far.

You need to be able to learn how to work with a grid layout, how to implement Flexbox, how to have different text components sit right next to images, how to align them. Then once we get into the responsive section, which is where we're going to be able to make this program work perfectly on mobile devices, you can see how this will work.

If you just shrink the size here, you'll see when this gets to a smartphone size, do you see how the images and the text actually rearrange themselves automatically?

Well, that is going to be something that you are going to have to do many, many times and this type of layout is a great way of learning how to get that accomplished. With all that being said, let's spend the rest of this guide putting the HTML structure in place that we're going to need for this.



Then after that, in one of the next guides, we'll get into adding the CSS. So come down here in the `about.html` and we want this to be below the page container and we're going to create a new wrapper. This is going to be called the `squares-wrapper`, if I can spell it, and there we go.

All the content we're going to be putting together is going to be inside of here. I'm going to create a new class here just called squares and then the square components will be listed inside of here. A very common convention that I'll follow is I will have some type of wrapper container class, then I will have the plural name for whatever I'm using.

So if I have a squares wrapper inside, I'm going to have squares and then I have a singular version of that right afterwards. So here, I'm going to have nested inside of that a square class, and here, we're going to have an image, so I'm just going to put a comment here and say image goes here. Then below that, we're going to have the actual content.

Let's just call this the square text wrapper. Then inside of here, we'll have a `h1` and I want to have just some lorem ipsum text, so we can say `h1 lorem8`, which is my Emmet abbreviation, that's going to give us some good heading content. Then, I want to do a paragraph tag and for this I want a little bit more content. Let's say `lorem50`. That should give us what we're looking for. We can always add more if we want to add more, or take some away if we want to do that. That is going to be our basic square text wrapper.

## about.html

```
<div class="squares-wrapper">
  <div class="squares">
    <div class="square">
      <!-- IMG goes here -->

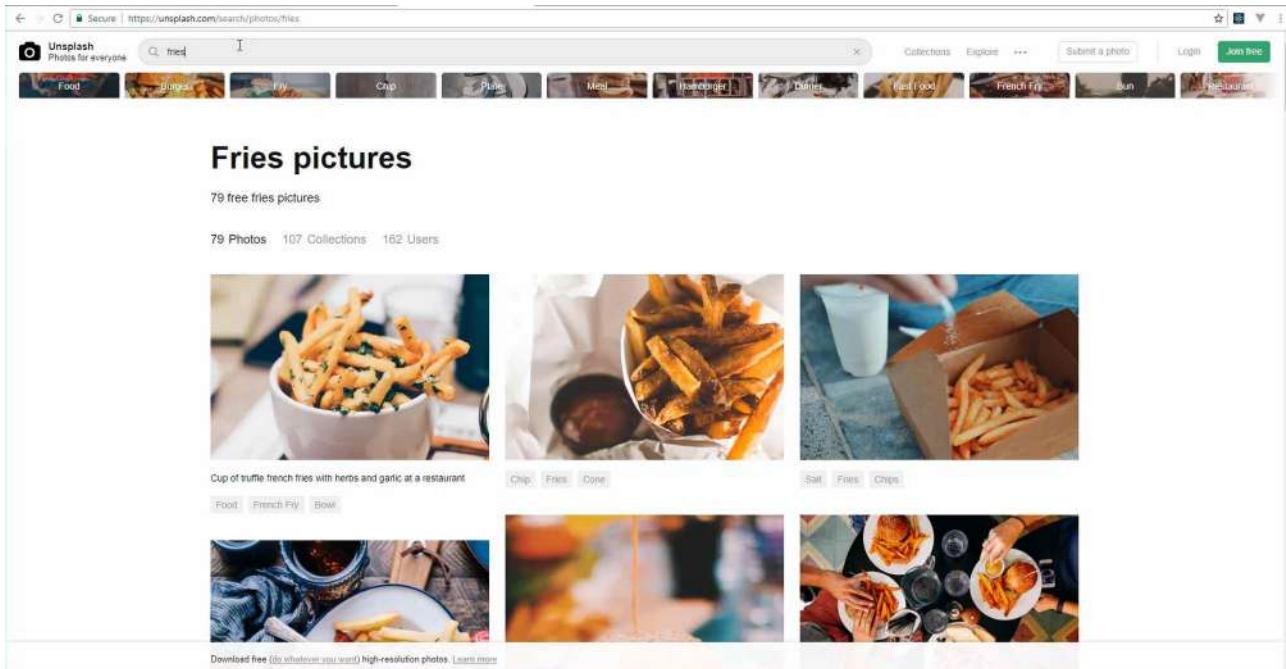
      <div class="square-text-wrapper">
        <h1>Lorem ipsum dolor sit amet, consectetur adipisicing elit </h1>

        <p>Lorem ipsum dolor sit amet, consectetur adipisicing elit. Dicta provident sunt laudantium fuga repudiandae, nulla! Debitis ullam culpa incidentum quo voluptatibus deserunt, quia velit, eaque quaerat iste consequuntur, explicabo voluptate voluptates dolor dignissimos illum doloribus distinctio mollitia quam laboriosam. Optio possimus vitae a. Dolore obcaecati animi non sit nemo, iusto debitis quos tempora temporibus veniam, accusantium est deserunt aperiam, reprehenderit sed dolorem, reprehenderit officiis vero corporis totam adipisci perspiciatis atque cum architecto voluptatem!</p>
      </div>
    </div>
  </div>
</div>
```

Now, let's take care of this image. If I switch over here, I will include these images for you in the show notes as usual. Here, I want to go to the digital literacy guide. I have images and I created an entire directory called squares here and so we will use both of these images. I'm going to click on this and let's save the image as. Go back to images, right click new folder, and we're just going to call this squares, just to keep the naming consistent.

We'll do squares fries-sq-1.jpg and now we'll do the same thing for fries two. Once again, feel free to use any kind of images that you want. I did have these custom made with the right size just to make sure that they render properly on the page.

So I'd recommend, even if you want to use other images, that you will at least keep these same sizes. If you're also curious on how to get access to images for your website, a great resource, it's what I've used here, is called Unsplash. If you go to [Unsplash](https://unsplash.com/search/photos/fries), they are free photos that you can use for social media, for websites, anything like that. You can even see some of the images that we've been using throughout this course if you type in fries.



But, you could type in anything else. I will get coding pictures from here and just it's a fantastic resource for any kind of stock photography that you need and it's all completely free, which is really nice.

Now, we have those images here. Let's come down and let's switch to the code and add these in, so I'll get rid of that comment. Let me create a wrapper for this because I know I'm going to eventually use it. I'm going to say image wrapper and that's a div. Inside of here is going to be an image tag and we're going to grab that in images, squares, and then we'll go with the first one.

We can just say here Fries that should be fine. Then, that's all we need for one of the squares.

## about.html

```
<div class="squares-wrapper">
  <div class="squares">
    <div class="square">
      <div class="img-wrapper">
        
      </div>

      <div class="square-text-wrapper">
```

Now if we hit refresh, then you're going to be able to see a gigantic set of fries and then the h1 and then some content.



**Lore ipsum, dolor sit amet consectetur adipisicing elit.**

LOREM IPSUM, DOLOR SIT AMET CONSECTETUR ADIPISICING ELIT. LABORIOSAM, QUILBUSDAM NON, DOLOR SIT AMET CONSECTETUR TEMPORIBUS DISTINCTIO EVENIET. MAXIME TOTAM ADIPISCI QUOD DUCIMUS EARUM INCIDENT ESSE INVENTORE DOLORE A VOLUPTATIBUS, ELIGENDI MINUS ATQUE EST VERITATIS NOSTRUM EX! AT ACCUSANTIUM AD HARUM VENIAM NATUS EOS SOLUTA ARCHIFECTO ATQUE QUIDEM!

So all we need to do now to get us started is to add one of the other squares and scroll to the left because that's showing everything, so this gives us the square and there we go. Just go down if you have the square class highlighted, go down only to where the ending div is and then you can just copy and paste that.

```
59      <div class="squares">
60          <div class="square">
61              <div class="img-wrapper">
62                  
63              </div>
64
65          <div class="square-text-wrapper">
66              <h1>Lorem ipsum, dolor sit amet consectetur adipisicing elit.</h1>
67
68              <p>Lorem ipsum, dolor sit amet consectetur adipisicing elit. Laboriosam,
69                  </p>
70          </div>
71      </div>
```

Now, the second one is going to be slightly different. The only difference being that we want to reverse the order. Here, I'm going to cut this and then I'll paste the image wrapper at the bottom of this square. If I hit refresh now, actually let me switch it up. We want to use fries square two. I'll just make sure that that's all working. Now, you can see we have that other image and we have our headline and then all of our content.

**Lorem ipsum, dolor sit amet consectetur adipisicing elit.**

...etiam non dolorem odit vel consequuntur temporibus distinctio eveniet. Maxime totam adipisci quod ducimus earum incidenti esse inventore dolore a voluptatibus, eligendi minus atque est veritatis nostrum ex! At accusantium ad harum verilam natus eos soluta architecto atque quidem!

***...etiam non dolorem odit vel consequuntur temporibus distinctio eveniet. Maxime totam adipisci quod ducimus earum incidenti esse inventore dolore a voluptatibus, eligendi minus atque est veritatis nostrum ex! At accusantium ad harum verilam natus eos soluta architecto atque quidem!***

...etiam non dolorem odit vel consequuntur temporibus distinctio eveniet. Maxime totam adipisci quod ducimus earum incidenti esse inventore dolore a voluptatibus, eligendi minus atque est veritatis nostrum ex! At accusantium ad harum verilam natus eos soluta architecto atque quidem!



This may not look great yet, but as hopefully you will trust me now since we've been doing this for a while, we have the structure in place. We have all the HTML code we need and we're going to leverage CSS in order to style it and lay it out on the page. So we will do that in the next guide.



## Coding Exercise

Place an `h2` tag that says "Lorem ipsum". Below that, place an image with its source path set to `app/assets/images/waffles.png` and the `alt` set to "Waffles"

# 1.41 CSS Grid>Flexbox: 2 columns

Now that we have all of our content here, I think we are ready to add our square styles, and I think you'll also be pleasantly surprised by how little amount of code you're going to have to write in order to get this working.

I'm going to come down to the bottom of the `common.css` file and I'm going to say these are the square, let's say square grid styles. And now inside of here, let's add our styles for the squares wrapper. We'll say display flex, and justify content here is going to be center, so that is going to take the entire set of code.

So let's reference our HTML one more time. Remember squares wrapper is what is wrapping up all of the content, and then inside of it we have a single div called squares. And so this is going to take that squares div and it's going to simply align it centered along the horizontal access.

Now that we have the squares wrapper set, now we can grab that once again. And this time we're going to grab the square inside of it. Actually, sorry, squares. Don't wanna get that one wrong. Here we simply want to dictate a width, so here the width is going to be 1,000 pixels, and that's all we're going to need to do for the squares.

## common.css

```
/* Squares grid styles */
.squares-wrapper {
  display: flex;
  justify-content: center;
}

.squares-wrapper > .squares {
  width: 1000px;
}
```

Inside of here, remember, we have our single square. This is going to be where we are going to build out our grid, so we have squares and then inside of that is going to be our square class. And for this I want to use CSS grid, because at the end of the day what we are building out is a grid layout. So I think it is a good fit for using CSS grid.

So I'm going to say display grid, and then inside of that I want to set up our grid template columns. Here we have grid template columns, if you remember back to the last time we used grid, we just used one fractional unit. We explored some other options, but right here I want two columns, so I can say 1FR and then 1FR, and that's gonna give me two equally sized columns.

## common.css

```
.squares-wrapper > .squares > .square {
  display: grid;
  grid-template-columns: 1fr 1fr;
}
```

Now if I hit save, that's going to be the grid layout. It's not really worth looking at yet because the images are simply going to be overriding everything, so let's fix that first. Let's go and we don't have to grab these squares, wrapper squares and square here, because we're never going to be using a image outside of the square wrapper inside of that image wrapper, if that makes sense.

So here, and I'll explain what my choice is here - I'm going to say square and then image wrapper, and then select the image. The reason why I'm going with this choice is because I am never going to use this image wrapper outside of a square. Or I should say the styles that I'm going to be applying here are only for when they're inside of this square class.

Technically I could do squares wrapper and then squares followed by square, but that's really not needed right now, because at the end of the day, the square, whenever I want to have a image wrapper inside of it, is going to have these styles, I'm always going to want it to have these styles.

What I'm going to do here is I'm going to say the width should be 100%. The height should be 500 pixels, and you'll see why I'm doing that here in a second, that's what's going to give us our nice square look. Then after that we need to use a little tool called object fit and then cover. This is an incredibly helpful little attribute, it is very similar to what we did when we added that skew in the top image, where it maintains the aspect ratio of the image and shrinks it down, and essentially crops it, so that it fits inside of whatever container it's inside.

## common.css

```
.square > .img-wrapper img {
  width: 100%;
  height: 500px;
  object-fit: cover;
}
```

If you hit save now, we should be a little bit closer to what we're looking for. If I hit refresh here, you can see you that is looking really good.



Right here you can see that we have these images, they're lining up, they're nice and square, so they are cropped properly, and that's looking excellent. A couple little fixes, obviously we still have to address the content itself. And then also do you see how we have a little bit of a break right here? I'll zoom in so you can see it.



Do you see that little bit of margin? That is just due to the default margin provided by HTML. So I'm going to fix that, and let's inspect it and see what values this needs to be set at. So for our image wrapper here, I'm going to ... Let's see, I'm trying to see if I want to apply this to the square or to the image wrapper, and it looks like it's the square that actually needs it.

So in the square I'll say margin top and then let's say -5 pixels, and that's almost there. Let's move it down one more and there we go, -4 pixels is perfect for that square class. Let's copy that value and switch back, this was just in the square class. Hit save, and now we should be good to go.

#### common.css

```
.squares-wrapper > .squares > .square {  
    display: grid;  
    grid-template-columns: 1fr 1fr;  
    margin-top: -4px;  
}
```

If you hit refresh, everything should still be working, so that is coming along very nicely. I can tell you from experience, when I was first tasked with building something like this, and this was a number of years ago, the designer gave this kind of checkerboard looking design. This took a while for me to figure out exactly how to build.

We ran through it pretty quickly because I've done this a number of times and I know how to implement this now, but I can tell you that this is a little bit challenging to get perfect if you've

never done it before. But the keys are working with that object fit property and then CSS grid. Those two things make this a much easier type of system to build out.

Now that we have all of those items set, now it's time to clean up our heading and our text and everything there. Let's switch back to the code, I have the image wrapper image as the last item, so now let's grab the square text wrapper. To do that I'm going to kind of copy the same process that we followed right above it.

So I'm going to say square and then we'll do the square text wrapper, and inside of this I'm going to make this use a flex box container. I'm gonna justify content in the center, and then align items is also going to be in the center, because remember we want all of the text content to be perfectly centered horizontally and vertically.

I also want the elements to sit on top of each other. So let's see what we have right here. If I hit refresh, do you see how the heading is right next to the content?



If you remember that the flex direction by default with flex box is to put items right next to each other in this kind of format, so we need to use the flex direction column. I'll say flex direction and column, and that will fix that value. We also want to add some padding, so let's add padding of 42 pixels.

#### common.css

```
.square > .square-text-wrapper {  
  display: flex;  
  justify-content: center;  
  align-items: center;  
  flex-direction: column;  
  padding: 42px;  
}
```

Now if you hit refresh, that is looking much better, you can see that that's really the exact look that we are wanting.



**Text**  
Lorem ipsum, dolor sit  
amet consectetur  
adipisicing elit.

Lorem ipsum, dolor sit amet consectetur adipisicing elit. Laboriosam, quod dolore? Quibusdam non, dolorem odit vel consequuntur temporibus distinctio eveniet. Maxime totam adipisci quod ducimus earum incident esse inventore dolore a voluptatibus, eligendi minus atque est veritatis nostrum ex! At accusantium ad harum veniam natus eos soluta architecto atque quidem!

The only difference is we want to grab this heading and add a color to it. So we can just copy this, and inside of the square text wrapper for the heading, what I want to do is update the color. Here we'll say the color should be that CEA135.

common.css

```
.square > .square-text-wrapper h1 {  
    color: #ceaa135;  
}
```

Save, refresh, and there we go, that is looking really nice.

**Lorem ipsum, dolor sit  
amet consectetur  
adipisicing elit.**

Lorem ipsum, dolor sit amet consectetur adipisicing elit. Laboriosam, quod dolore? Quibusdam non, dolorem odit vel consequuntur temporibus distinctio eveniet. Maxime totam adipisci quod ducimus earum incidunt esse inventore dolore a voluptatibus, eligendi minus atque est veritatis nostrum ex! At accusantium ad harum veniam natus eos soluta architecto atque quidem!

1



We have everything that we need on this page except for the footer, and there's one other thing. Do you see how we have this space right here? Well, that is not ideal. I think I would like to have a little bit more space between our squares and the page content.

But the one issue here is, it'd be pretty straightforward if I just added something like, I could say on the page content, give me a margin bottom of 40 pixels. And that would work, but that means that

that margin bottom would be used in every other spot where I use the page content, and I don't want that.

There may be spots where I want it to be more flush, and the same thing with the squares. I could say I want you to take the square wrapper and give it a margin top of 40 pixels, and that would work, but still not exactly what we're really looking for, because then that margin top would follow the squares along everywhere else we used them.

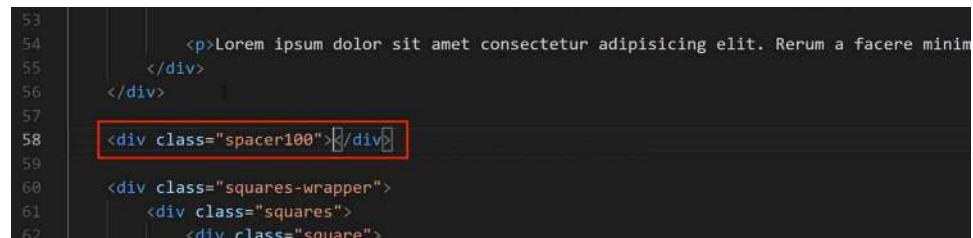
Instead I'm going to create a little helper method here. I'm going to create a special class, and I usually will always have classes like this. Here I'm going to call it our helpers styles and I'm going to add a spacer. Let's add a spacer of, let's say spacer 100 is going to be the class name.

What this is going to do is it's going to give us the ability to have some margin anytime we want. It provides a spacer for us. Here we can just say the height is going to be 100 pixels wide and the width is going to be 100%, something like that.

### common.css

```
/* Helpers */
.spacer100 {
    height: 100px;
    width: 100%
}
```

Now what we can do is just call this spacer 100, and what I usually do is I'll have a bunch of these. I'll have like spacer 100, I'll have spacer 40, and then anytime that I need to, I can call that. So right here I can come in between and say spacer 100, and then ... Emmett isn't helping me out for some reason, there we go.



```
53
54     <p>Lorem ipsum dolor sit amet consectetur adipisicing elit. Rerum a facere minim
55     </div>
56 </div>
57
58 <div class="spacer100"></div>
59
60     <div class="squares-wrapper">
61         <div class="squares">
62             <div class="square">
```

So I'll just give this empty div here with that class name, and now you can see I have the right amount of space, or I should say I have some space where I didn't before.

minima lure eaque fugit, rerum porro quo dolorem quam dicta sapiente vel veniam ipsum delectus doloremque maxime  
ipsam a ratione! Quod accusantium ducimus nihil tempore quaerat natus quas dolor id ipsa cum tempora nisi aliquid a,  
quos inventore voluptates magnam. Illo doloremque aut nobis dolor. Sint natus repellendus autem. Ipsum nihil nemo  
voluptate impedit excepturi earum molestias deleniti sequi velit architecto unde iusto ea possimus rerum soluta laudantium  
eaque voluptatum, veritatis fugit incident quibusdam itaque quam? Tenetur?



**Lore ipsum, dolor sit  
amet consectetur  
adipisicing elit.**

This is a little bit too much space, so let's actually update this. So I want to have this, let's see what 60 looks like, that looks perfect. So let's change the name of the class, I'm going to change this div to be spacer60. Then I'll come here to the css and this is now going to be a spacer of 60, just like that. The height will be 60 pixels.

### common.css

```
/* Helpers */
.spacer60 {
  height: 60px;
  width: 100%
}
```

Hit save, hit refresh, and that's looking really good.

minima iure eaque fugit, rerum porro quo dolorem quam dicta sapiente vel veniam ipsum delectus doloremque maxime  
ipsam a ratione! Quod accusantium ducimus nihil tempore quaerat natus quas dolor id ipsa cum tempora nisi aliquid a,  
quos inventore voluptates magnam. Illo doloremque aut nobis dolor. Sint natus repellendus autem. Ipsum nihil nemo  
voluptate impedit excepturi earum molestias deleniti sequi velit architecto unde iusto ea possimus rerum soluta laudantium  
eaque voluptatum, veritatis fugit incident quibusdam itaque quam? Tenetur?



**Lorum ipsum, dolor sit  
amet consectetur  
adipisicing elit.**

In this guide you learned how to create a full checkerboard type pattern using CSS grid and flex box, and also how to create little helper classes that help us add some shared style components.



## Coding Exercise

Give the below spacer a width of 800px and a height of 45px

```
<div class="about-page-spacer"></div>
```

# 1.42 Review of Code Organization Best Practices

Now that we have our two-column grid layout finished and that's looking good, and everything else on the page is really coming along nicely, the last element that we want to integrate is the footer.

We want to have the same exact footer that we had here on the home page. So, we can simply bring that in, and let's see how that works.

Let's open up our about.html and our index.html and then scroll down almost to the bottom where the footer is. We want to highlight everything on the footer and copy it into our about.html.

## about.html

```
<div class="footer">
  <div class="logo-footer">
    
  </div>

  <div class="footer-phone-hours">
    <span class="phone">555 555 5555</span>
    <span class="hours">10 AM - MIDNIGHT</span>
  </div>

  <div class="links-wrapper">
    <div class="nav-link">
      <a href="index.html">Home</a>
    </div>
    <div class="nav-link">
      <a href="about.html">About Us</a>
    </div>
    <div class="nav-link">
      <a href="menu.html">Menu</a>
    </div>
    <div class="nav-link">
      <a href="take_out.html">Take Out</a>
    </div>
  </div>

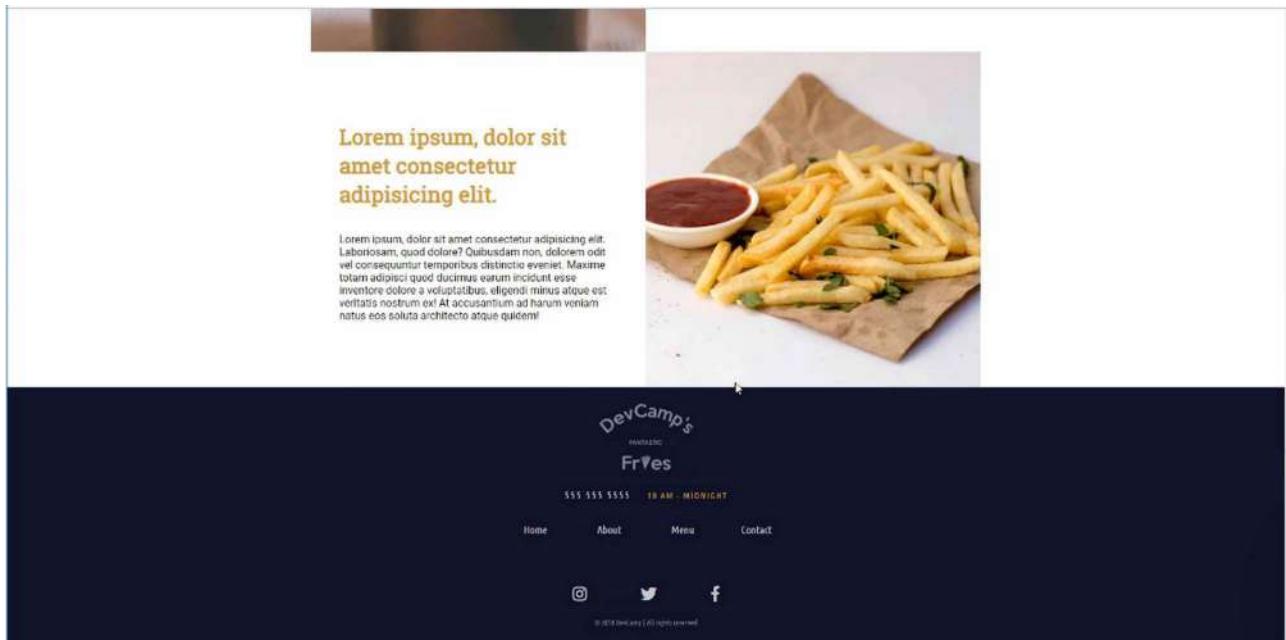
  <div class="social-media-icons-wrapper">
    <a href="#">
      <i class="fab fa-instagram"></i>
    </a>

    <a href="#">
      <i class="fab fa-twitter"></i>
    </a>

    <a href="#">
      <i class="fab fa-facebook-f"></i>
    </a>
  </div>
  <div class="copyright-wrapper">
    &copy; 2018 DevCamp &#124; All rights reserved
  </div>
</div>
```

That should all work. Let's see. We have `common.css` containing all the footer styles. So, this will work for right now. But, after we're done here I would like to clean our `common.css` file, because that file is getting a little bit massive.

First, let's make sure this is working. Scroll all the way down.



You can see that this is working perfectly. So, that is all you have to do on that side. Now, let's go to `common.css`. Let's do a little bit of clean up now that we're done.

I'm going to come up to the top, and you can see that we have all kinds of different comments, and I think the comments are going to be a great way of being able to separate all of these out.

The footer really has a lot of styles. That deserves to have its own file. So I'm going to say, new file inside of styles. This is just going to be `footer.css`. Then, let's come here to common and we're going to just get all of these.

## footer.css

```
/* Footer */
.footer {
    background-color: #11122b;
    color: #cbcbcb;
    font-family: "Ubuntu Condensed", sans-serif;
    display: flex;
    justify-content: center;
    align-items: center;
    flex-direction: column;
    height: 500px;
    margin-top: -4px;
}

.footer > .logo-footer img {
    width: 250px;
    height: 100%;
    /* filter: brightness(50%); */
    opacity: 0.5;
```

```

}

.footer > .footer-phone-hours span {
  margin-left: 10px;
  margin-right: 10px;
  font-size: 1.2em;
  letter-spacing: 2px;
}

.footer > .footer-phone-hours > .hours {
  font-size: 0.8em;
  color: #cea135;
}

.footer > .links-wrapper {
  width: 400px;
  margin-top: 40px;
  margin-bottom: 40px;
  display: flex;
  justify-content: space-between;
  align-items: center;
}

.footer > .links-wrapper > .nav-link a {
  color: #cbcbcb;
  text-decoration: none;
  transition: 0.5s;
}

.footer > .links-wrapper > .nav-link a:hover {
  color: #cea135;
}

.footer > .social-media-icons-wrapper {
  margin-top: 40px;
  margin-bottom: 40px;
  width: 300px;
  display: flex;
  justify-content: space-around;
}

.footer > .social-media-icons-wrapper a {
  font-size: 1.5em;
  color: #cbcbcb;
  text-decoration: none;
  transition: 0.5s;
}

.footer > .social-media-icons-wrapper a:hover {
  color: #cea135;
}

```

We'll import these to our `about.html` and `index.html` as well.

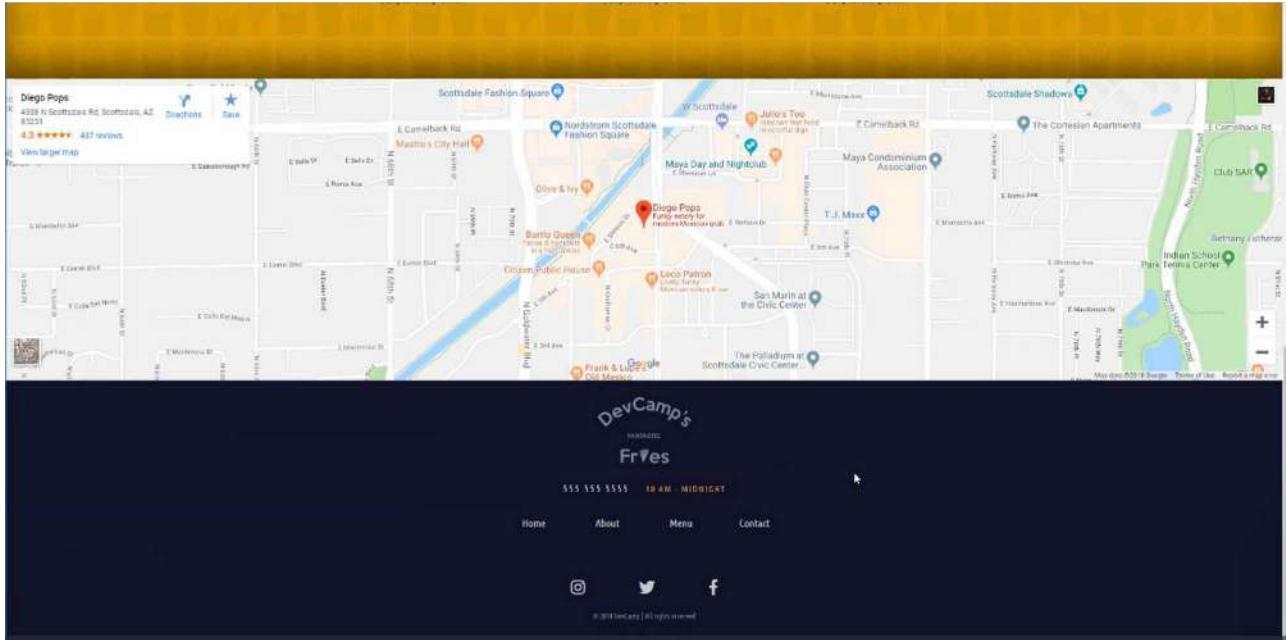
## `about.html` + `index.html`

```

<link rel="stylesheet"
      href="https://use.fontawesome.com/releases/v5.2.0/css/all.css"
      integrity="sha384-hWVjflwFxL6sNzntih27bfxkr27PmbbK/iSvJ+a4+0owXq79v+lsFkW54
      bOGbiDQ" crossorigin="anonymous">
<link rel="stylesheet" href="styles/common.css">
<link rel="stylesheet" href="styles/navigation.css">
<link rel="stylesheet" href="styles/footer.css">

```

Let's verify that we didn't break anything.



And, nope. Everything is still looking good. That means everything is working on that side. If you go to home you can see we still have the footer. So, all we're doing is we're organizing the code so it's a little bit easier to navigate.

Now the skewed header, this is something that is going to be common among just about all the pages but not every one. If you notice the homepage does not have any of those skewed header styles. What I'd like to do is come up with a pages one, or I could just create a skewed header.

I could say `skewed-header.css` and now we can know where whenever we need that skewed header all we have to do is bring it in right here. Let's come and grab all of that skewed header content and paste it in.

### skewed-header.css

```
/* skewed header */

.skewed-header {
  position: relative;
  height: 400px;
  overflow: hidden;
}

.skewed-header > .header-bg {
  position: absolute;
  top: 0;
  bottom: 0;
  right: 0;
  left: 0;
  width: 100%;
  height: 100%;
  background-image: url('../images/backgrounds/about-us.jpg');
  background-repeat: no-repeat;
  background-size: 100%;
  transform: skewY(-6deg);
  transform-origin: top left;
  filter: brightness(50%);
}
```

```

.skewed-header > .skewed-header-wrapper {
  position: absolute;
  display: flex;
  justify-content: center;
  width: 100vw;
}

.skewed-header > .skewed-header-wrapper > .skewed-header-content {
  width: 1000px;
  display: flex;
  justify-content: space-between;
  align-items: center;
}

.skewed-header > .skewed-header-wrapper > .skewed-header-content > .links-wrapper {
  display: flex;
}

.skewed-header > .skewed-header-wrapper > .skewed-header-content > .links-wrapper > .nav-link {
  margin-left: 10px;
  margin-right: 10px;
}

.skewed-header > .skewed-header-wrapper > .skewed-header-content > .heading-wrapper {
  background-color: #cea135;
  color: #11122b;
  border-bottom-left-radius: 2px;
  border-bottom-right-radius: 2px;
  width: 175px;
  padding: 10px;
  text-align: center;
}

```

Now, if you open up the about, we can add this right here.

## about.html

```

<link rel="stylesheet"
      href="https://use.fontawesome.com/releases/v5.2.0/css/all.css"
      integrity="sha384-hWVjflwFxL6sNzntih27bfxkr27PmbbK/iSvJ+a4+0owXq79v+lsFkW54bOGbiDQ" crossorigin="anonymous">
<link rel="stylesheet" href="styles/common.css">
<link rel="stylesheet" href="styles/navigation.css">
<link rel="stylesheet" href="styles/footer.css">
<link rel="stylesheet" href="styles/skewed-header.css">

```

Hit save, and after you refresh, you can see that this is still working properly.



The reason why I wanted to do that is that I don't like having a file called common unless 100% of the code that is in the common directory is used by every page on the site.

Let's see. If we follow that rule, every page is probably going to have a heading. It makes sense for this rule to be here. Then everyone has a body. So, that makes sense. Those are truly common.

But this page container, this is not on every page. So, we should probably come here and add a `page-container.css` file. Then we can just go and grab all of these items.

### **page-container.css**

```
/* page containers */

.page-container {
  display: flex;
  justify-content: center;
}

.content-wrapper {
  width: 1000px;
}

.content-wrapper > #chef {
  margin: 25px 40px 30px 0px;
  width: 350px;
  float: left;
  border: 1px solid #cea135;
}

.content-wrapper > p {
  line-height: 30px;
  letter-spacing: 1px;
}
```

It's just a small amount of content. We're only going to have to bring that into the about page.

### **about.html**

```
<link rel="stylesheet"
      href="https://use.fontawesome.com/releases/v5.2.0/css/all.css"
      integrity="sha384-hWjflwFxL6sNzntih27bfxkr27PmbbK/iSvJ+a4+0owXq79v+lsFkW54
      b0GbiDQ" crossorigin="anonymous">
<link rel="stylesheet" href="styles/common.css">
<link rel="stylesheet" href="styles/navigation.css">
<link rel="stylesheet" href="styles/footer.css">
<link rel="stylesheet" href="styles/skewed-header.css">
<link rel="stylesheet" href="styles/page-container.css">
```

Hit save, hit refresh once again, and everything here is looking good. Everything is still following exactly what we had before.

Next, you're probably going to guess what we're going to do now. We have the square grid. So, let's come to styles and create a `square-grid.css`. Then we can do the same thing here where we're going to grab all of the square grid content and pull it out.

## square-grid.css

```
/* squares */

.squares-wrapper {
  display: flex;
  justify-content: center;
}

.squares {
  width: 1000px;
}

.square {
  display: grid;
  grid-template-columns: 1fr 1fr;
  margin-top: -4px;
}

.square > .img-wrapper img {
  width: 100%;
  height: 500px;
  object-fit: cover;
}

.square > .square-text-wrapper {
  display: flex;
  justify-content: center;
  align-items: center;
  flex-direction: column;
  padding: 42px;
}

.square > .square-text-wrapper h1 {
  color: #cea135;
}
```

That's all we need to do there. Let's do the same thing on the about page.

## about.html

```
<link rel="stylesheet"
      href="https://use.fontawesome.com/releases/v5.2.0/css/all.css"
      integrity="sha384-hWjflwFxL6sNzntih27bfkr27PmbbK/iSvJ+a4+0owXq79v+lsFkW54
      b0GbiDQ" crossorigin="anonymous">
<link rel="stylesheet" href="styles/common.css">
<link rel="stylesheet" href="styles/navigation.css">
<link rel="stylesheet" href="styles/footer.css">
<link rel="stylesheet" href="styles/skewed-header.css">
<link rel="stylesheet" href="styles/page-container.css">
<link rel="stylesheet" href="styles/square-grid.css">
```

Lastly you may think the helpers are common, and they kind of are, but still, they're not master styles. They simply are helpful tools.

If this common file gets any larger, which in a large application your common files still will get pretty large, then what you are going to want to do is to create a `helpers.css` and wherever that's needed, then you can go and pull that out. Let's save that, paste that file in.

## helpers.css

```
/* helpers */

.spacer {
  margin-top: 50px;
  margin-bottom: 50px;
}

.dark-bg {
  background-color: #11122b;
  color: #cbcbcb;
}

.phone a {
  color: #cbcbcb;
  text-decoration: none;
  transition: 0.5s;
}

.phone a:hover {
  color: #cea135;
  text-decoration: none;
}
```

Now to import it. First is `about.css`.

## about.html

```
<link rel="stylesheet"
      href="https://use.fontawesome.com/releases/v5.2.0/css/all.css"
      integrity="sha384-hWjflwFxL6sNzntih27bfxkr27PmbbK/iSvJ+a4+0owXq79v+lsFkW54
      b0GbiDQ" crossorigin="anonymous">
<link rel="stylesheet" href="styles/common.css">
<link rel="stylesheet" href="styles/navigation.css">
<link rel="stylesheet" href="styles/footer.css">
<link rel="stylesheet" href="styles/skewed-header.css">
<link rel="stylesheet" href="styles/page-container.css">
<link rel="stylesheet" href="styles/square-grid.css">
<link rel="stylesheet" href="styles/helpers.css">
```

Now everything should still be working. I hit refresh. You can see that everything's there. Our spacer is still there. Our grid is there. Our footer is working. This is all really, really nice.

One thing that I love about this is imagine a scenario where you give this to another developer and you come back later on and you haven't worked in this type of environment for a while.

When you look at the head tag, and you look at the metadata right here, you can instantly all of the features on that page by just looking at this one spot of the site.

You can see all of the dependencies. Then it also is going to make it a lot easier for you to update styles. If you ever need to update the skewed header, you can go directly to the skewed header file and have access to all of that content right there.

I think that's a really nice way of organizing the code. If you're looking to get a development job, then people are going to want to see code organization like this. Because this shows that you understand, not only how to organize code, but you also realize the importance of it.

So, great job. If you went through that, we are making our way. We now have the home page, which is the hardest page to build out. Then we have the about page. And next is going to be the menu. You can see over here what the menu should look like.

The screenshot shows a menu section with three items:

- Mild**: An image of a dark drink with a straw and a small tag labeled "MILD".
  - Lorem ipsum dolor sit amet, consectetur adipisicing elit.
  - Unde porro maiores numquam illum, distinctio quisquam blanditiis.
  - Sit inventore beatae unde est nobis aliquam, fuga!
  - Commodi inventore odit assumenda recusandae, et que possemus numquam?
- Medium**: An image of a pile of French fries with a small bowl of red dipping sauce.
  - Lorem ipsum dolor sit amet, consectetur adipisicing elit.
  - Unde porro maiores numquam illum, distinctio quisquam blanditiis.
  - Sit inventore beatae unde est nobis aliquam, fuga!
  - Commodi inventore odit assumenda recusandae, et que possemus numquam?
- Hot**: An image of French fries with steam rising from them.
  - Lorem ipsum dolor sit amet, consectetur adipisicing elit.
  - Unde porro maiores numquam illum, distinctio quisquam blanditiis.
  - Sit inventore beatae unde est nobis aliquam, fuga!
  - Commodi inventore odit assumenda recusandae, et que possemus numquam?

And, I can tell you this. We are going to be able to build the menu in a single video because we have already built all of these styles. That is going to be pretty cool.

There's going to be a couple little features that I want to teach you about in them, so we'll spread it out through a couple videos because we're going to learn about concepts like being able to have anchor links and bullet points and numbers and things like that.

I will spread the teaching over a couple of videos. But, we're actually going to get to what looks almost like this all in a single video.

So, hopefully, you'll like that. The next section is going to be pretty short and sweet.



## Coding Exercise

In the provided HTML starter code, make your own copyright message. Be sure to add a copyright sign!

```
<div class="copyright-message">  
</div>
```

## 1.43 “Menu” page with 2 grids

In this section of the course, we are gonna walk through how to build out the menu page. Now, this is gonna be a pretty quick section. And, like I mentioned in the previous video, we actually are gonna build out all of the real functionality for the menu page all in a single video, so that should be pretty cool.

That's what we're gonna do right now. And then, later on, we're gonna learn about things like bullet points and some of the other features that are on this page.

But, to get started, let's come into the code, and I'm gonna go and open the `menu.html` page here. I'll close off the index and I think it's easiest just to get rid of everything that is there, and we're gonna copy everything from the about us page and then we'll make a few changes.

So I'm going to get rid of that, the title here should be menu. And then, we're going to be using most of these elements, but not all of them. So we're not gonna be using the page container, so we can get rid of that. And I don't believe we're gonna be using the helpers, either, we can always bring them back in if we need to, though.

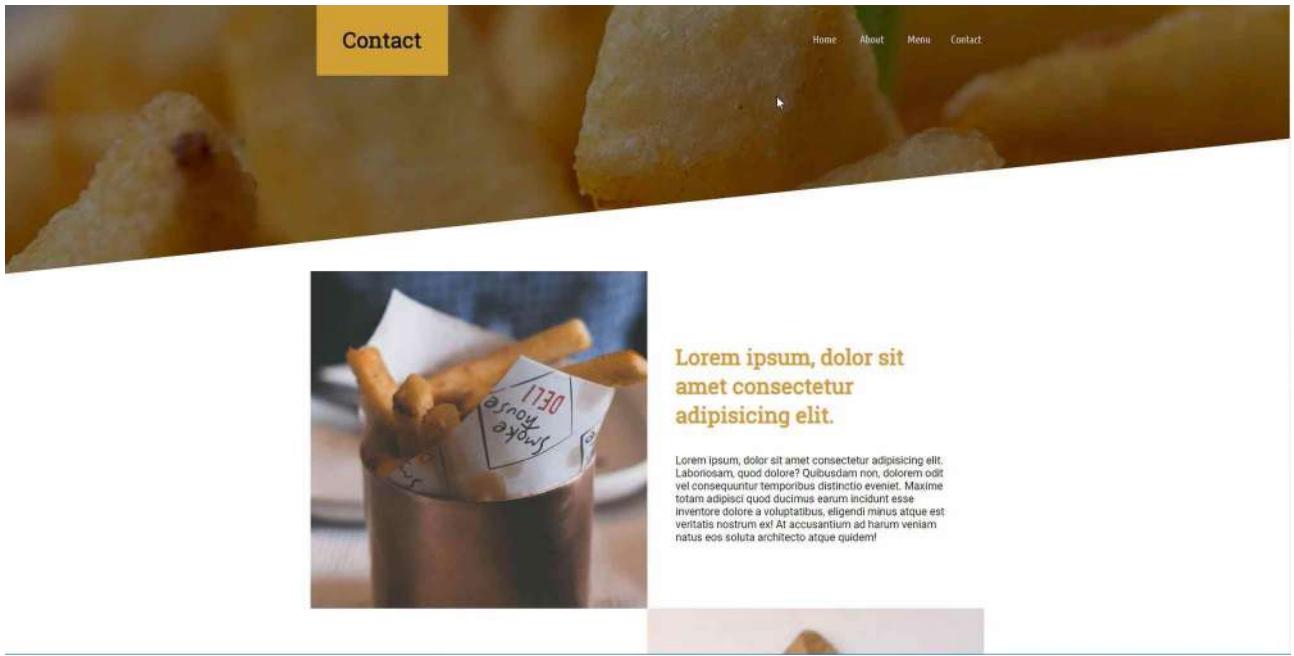
We are gonna be using this skewed header. So we're gonna have a new image to pull in, so let's switch back into the browser, and I'll include this link in the show notes. So we're gonna grab this menu jpeg image here. So if you click save image as, you want to put this in backgrounds. So click on save, and now we can actually use it. So we can bring it in, and it's going to replace it right here on the menu page. So let's come over where it says about us, and now it can just say menu.

So this just ... to kind of review, if you remember back when I said we were gonna use an inline style here, and it was because I wanted to be able to share the functionality of the class, this is why. Because if you want to have different images here, then you would have to go and create a different class for each one of those images.

Whereas with this approach, we can have one shared class, so that header background class, that has all of the core functionality, and then we can just update the background image each time that we want to call it. So I think that's a good approach, that's exactly what I would do in a real-life application.

So now, let's say that we want this to say contact. And you can say contact us, it's up to you. Just say contact. Everything else here is going to be the same. We're going to get rid of all of this page container content, though. We don't need that any longer, along with the spacer, which we may or may not need.

So let's just hit save and see what that looks like. So if I click on menu, you can see that that was a lot easier than some of the other pages.



You can see, it's pretty much all done. Let's duplicate this, we want to have four of these. Just ... not really for any reason except for the fact that I think it's nice to be able to test it out and to keep playing with it. Because the more that you use it, the more familiar it's gonna be to you.

So let's grab these squares, and also, let me make sure I'm picking up the right spot. So we want to have these squares and ending at that div tag and then there at the very bottom, then we can go and we can just clone that. So we'll just copy and paste.

Now if we hit refresh, you can see that that is working perfectly.

And also notice how the alignment is also working very nicely. So our CSS grid is scalable, this is the kind of thing where you can have as many of these stacked on top of each other as you want, and they will all work perfectly. So great job if you went through this. In less than five minutes, you were able to reuse the various tools that we've spent all of this time building and you're able to populate them on a brand-new page. So, very nice.

And I did realize I just ... I just realized I misspelled ... or not misspelled, I grabbed the wrong name. We do not want the heading to be contact. This is supposed to be menu. There we go, that looks better. That would've bothered me to leave that in the video.



So our menu page, for all intents and purposes, is done. What we're gonna learn about in the next few guides are how to work with lists of data, and then also, how to work with things that are called anchor tags, a very special type of link. So I will see you in the next guide.



## Coding Exercise

Use the float property, set to left, to align the below divs side by side. And make their text color red.

```
<div>
  <div class="bagel"></div>
  <div class="bagel"></div>
  <div class="bagel"></div>
</div>
```

# 1.44 HTML Bullet Point and Numbered Lists

In this guide, we're going to walk through two different types of lists that you can use in HTML, such as having a list of bullet points, and then we also have the ability to use a numbered list, and I'm going to show you how to use both inside of this guide.

Let's switch back to the code, and going down into our menu, you can see we have our title, and then right below it, we have the content.

```
57      <div class="square-text-wrapper">
58          <h1>Lorem ipsum, dolor sit amet consectetur adipisicing elit.</h1>
59
60
61          <p>Lorem ipsum, dolor sit amet consectetur adipisicing elit. Laboriosam,
62              </div>
63      </div>
64      <div class="square">
65          <div class="square-text-wrapper">
66              <h1>Lorem ipsum, dolor sit amet consectetur adipisicing elit.</h1>
67
68              <p>Lorem ipsum, dolor sit amet consectetur adipisicing elit. Laboriosam,
69          </div>
70      </div>
```

We're going to get rid of the content here, and we're going to have our list. The first one I'm going to use is a bullet point list, and I mainly use this one just because this is what I use the most in real-world development.

A bullet point, you may think, would have some kind of name that involved a bullet or BL or something like that. But in HTML, it's actually called an unordered list. So even when you think of a bullet point list, in HTML it is called an unordered list, and the tag to use it is `<ul></ul>`.

If you type UL, this is going to give you a wrapper that you can place in each one of the bullet points. Inside of an HTML list you have to put a `<li></li>` tag. Now the LI tag is short for list item.

## menu.html

```
<ul>
    <li>Some menu item...</li>
</ul>
```

Now if I hit save, and come back, and hit refresh, you can see where we have this bullet point:

Now you can have a few more of those, so let me use Emmett right here to give us some examples. If you type in `li * 4 > lorem8`, it will populate it for you.

## menu.html

```
<div class="square-text-wrapper">
    <h1>Lorem ipsum, dolor sit amet consectetur adipisicing elit.</h1>
    <ul>
        <li>Lorem ipsum dolor sit amet consectetur, adipisicing elit.</li>
        <li>Cupiditate facilis tempora mollitia, nulla perspiciatis
corporis quisquam!</li>
        <li>Quis repellat iusto distinctio quas iure eaque aut?</li>
        <li>Explicabo in omnis quam aliquid molestias optio cumque?</li>
    </ul>
</div>
```

Now you can see that that gives us some handy starter data. Hit refresh, and you see that is exactly what we have.

So that is working quite nicely. It's really not that difficult to create bullet points in HTML. Then obviously you can grab these if you want to style them. So you could grab each one of the list items. You can add a class to them or you could add an ID.

I wouldn't really recommend adding an ID because remember, that would mean that you would have to create an ID and a unique one for each list item. Typically you're adding some type of class. So you could do something like `menu-item`, something like that.

But we're not going to do that because the default that we're getting right there is exactly what we need. So that is how you can create a bullet point list. Now if you want to have numbers instead of the little bullet points, then what you can do is create what is called an ordered list.

So a UL is a bullet point list that is unordered. But for list items, like if you want to have numbers by them, it is an ordered list, and the wrapper tag for that is going to be `<ol></ol>`, short for ordered list.

That's all you need to do in order to have those numbers. Now I'm just going to copy our list because it's actually the same exact tag name, and then I'm going to put into in the next menu item.

## menu.html

```
<div class="square-text-wrapper">
    <h1>Lorem ipsum, dolor sit amet consectetur adipisicing elit.</h1>

    <ol>
        <li>Lorem ipsum dolor sit amet consectetur, adipisicing elit.</li>
        <li>Cupiditate facilis tempora mollitia, nulla perspiciatis
corporis quisquam!</li>
        <li>Quis repellat iusto distinctio quas iure eaque aut?</li>
        <li>Explicabo in omnis quam aliquid molestias optio cumque?</li>
    </ol>
</div>
```

If I hit save here, come back and hit refresh, you can see that we now have numbers by the items.

The screenshot shows a web page with a light gray background. At the top, there is a large image of a dark-colored cup filled with golden-brown french fries. A small piece of paper with the word "French Fries" written on it is tucked into the top of the cup. To the right of the image, there is a yellow rectangular box containing the text "Lorem ipsum, dolor sit amet consectetur adipisicing elit.". Below this text is a bulleted list:

- Lorem ipsum dolor sit amet consectetur, adipisicing elit.
- Cupiditate facilis tempora mollitia, nulla perspiciatis corporis quisquam!
- Quis repellat iusto distinctio quas iure eaque aut?
- Explicabo in omnis quam aliquid molestias optio cumque?

At the bottom left of the page, there is another yellow rectangular box with the same text and list. To the right of this box is a smaller image of french fries served on a piece of brown parchment paper, accompanied by a small white bowl of red ketchup.

So these are the two types of lists that you have access to use whenever you're working in HTML. Let me leave this here, just so you'll have it in the show notes, and that way you have access to both

of them. And then I'm going to use this UL just so we have some matching content down in the other four items.

Now, you should have three bullet lists and one ordered or numbered list which is exactly what we have.

So in review, whenever you want to create a set of bullet points, the tag is `<ul></ul>`, which is short for unordered list. Whenever you want to have numbers by your list items, that is `<ol></ol>`, and it's short for ordered list.



## Coding Exercise

Create an ordered and a unordered list, respectively. Both should have at least three list items

# 1.45      HTML Anchor Tags

In this guide, we're going to learn how to work with Anchor Links inside of HTML.

Now if you've never heard of what an Anchor Link is, that's perfectly fine, we're going to do a little demo to see what an Anchor Link does and that may also help explain why you'd want to use them.

Now, we have a menu and imagine that on the link to here from your Facebook page or Twitter, or something, you want to direct your users directly to the `mild` element. So you want to send them to this section of the page.



What we can do is we can use an Anchor Link, so instead of just sharing the menu link, we can have them directed to this part of this page, and so as you can see when I clicked on the word `mild`, it auto scrolled right to where we wanted to be.

Now if I open this up in a new window, you can see that the URL is structured a little bit different. `https://jordanhudgens.github.io/digital-literacy-html-css/menu.html#mild`, so this `#` gives you the name of the Anchor Link.

If I hit enter here, I'm not just taken to the `menu.html` page, I'm also auto scrolled down right to the section that I want to be at, so that's what we're going to learn how to do in this tutorial.

**Mild**

- Lorem ipsum dolor sit amet, consectetur adipisicing elit.
- Unde porro maiores numquam illum, distinctio quisquam blanditiis.
- Sit inventore beatae unde est nobis aliquam, fugit.
- Commodi inventore odit assumenda recusandae, eaque possimus numquam?

**Kids**

- Lorem ipsum dolor sit amet, consectetur adipisicing elit.
- Unde porro maiores numquam illum, distinctio quisquam blanditiis.
- Sit inventore beatae unde est nobis aliquam, fugit.
- Commodi inventore odit assumenda recusandae, eaque possimus numquam?

Now, I have our starter code here and so we're going to start with the very first one, so if I switch over to `menu.html` and at the top I'm going to rid of this lorem ipsum text and I'm going to start off and say that this is going to be spicy, which is going to be our first link.

From here I'm going to create an A tag.

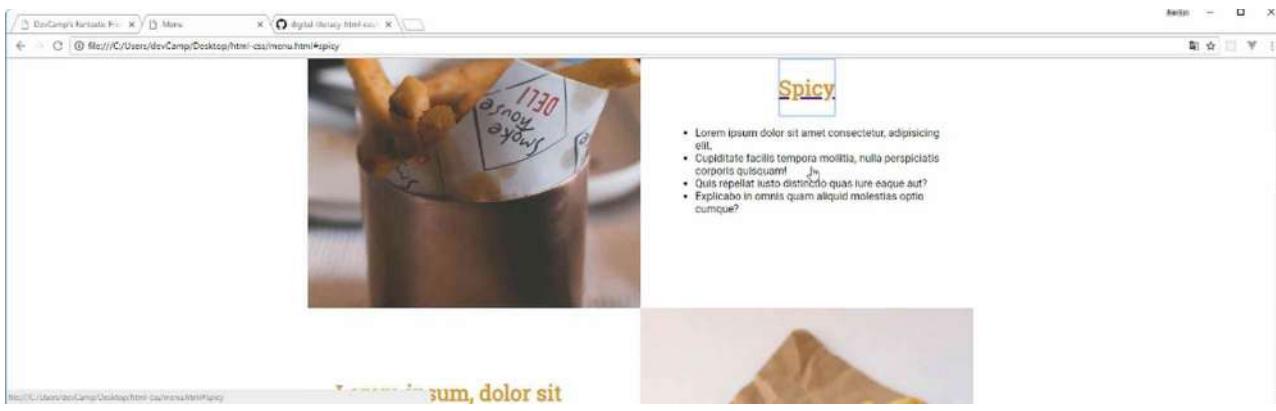
## menu.html

```
<div class="square-text-wrapper">
  <a href="#spicy" name="spicy">
    <h1>Spicy</h1>
  </a>

  <ul>
    <li>Lorem ipsum dolor sit amet consectetur, adipisicing elit.</li>
    <li>Cupiditate facilis tempora mollitia, nulla perspiciatis
corporis quisquam!</li>
    <li>Quis repellat iusto distinctio quas iure aut?</li>
    <li>Explicabo in omnis quam aliquid molestias optio cumque?</li>
  </ul>
</div>
```

What we're doing here is we're saying that this is the Anchor tag's name and then here is the link, so if you put both of those there and then wrap up the H1 tag then this should work. So let's go test this out.

If you come back to the menu, hit refresh, you can see we have some style issues but if I click on spicy I am taken exactly to where I want to be.



Let's fix a couple of these style items. If I right-click and click on inspect, you can see that this is going to the square, then the square-text-wrapper and then the H1 tag. So what I can do here is I can either apply this to the H1 tag or I can go one level above it into the A tag here.

I want to say that I want the text decoration to be none, so that's going to remove that ugly blue line, but did you also notice that kind of blue selection? Well, that is called an outline.

If we click on these hover states, you can play around and see when it is on focus that is when that little outline is going to surround the element, so what we want to do is we want to say when this is focused, we want an outline of none and then that's going to remove it.

Now, we're actually going to have to create two different style definitions here, one for the focus state and then one just for the A tag. So let's go into our `menu.html` and you can see that we don't have any way of selecting this without going into the square-text-wrapper.

I think if I want to use Anchor tags anywhere else in this application, I may just want to have a class dedicated to it. So I'm going to have a class called Anchor Link,

### menu.html

```
<a href="#spicy" name="spicy" class="anchor-link">
  <h1>Spicy</h1>
</a>
```

We're going to put this into our `helper.css`, I'm going to have an anchor selector or `anchor-link` and also the focus state.

## helpers.css

```
.anchor-link {  
    text-decoration: none;  
}  
  
.anchor-link:focus {  
    outline: none;  
}
```

Let's go to `menu.html` and make sure that we've imported our `helper.css`, and it looks like we haven't, so add that in.

```
12  <link rel="stylesheet" href="https://use.fontawesome.com/releases/v5.2.0/css/all.css" int  
13  <link rel="stylesheet" href="styles/common.css"  
14  <link rel="stylesheet" href="styles/nav.css">  
15  <link rel="stylesheet" href="styles/footer.css">  
16  <link rel="stylesheet" href="styles/skewed-header.css">  
17  <link rel="stylesheet" href="styles/square-grid.css">  
18  <link rel="stylesheet" href="styles/helpers.css">
```

Hit save, now hit refresh and it's working.



You can see that there's no outline or line underneath it, so this is giving us a behavior that we're looking for. Now let's go and let's populate that in all of the other sections too,

I'm just going to copy everything here and then we're just going to replace it four times. So the first one was spicy, the next one is going to be medium, after that one we'll have mild, and then we'll just make up one for the last one.

We'll say this one is just no flavor, I'm not sure who's going to order it but someone might. Okay, no flavor, hit save and hit refresh and now we have all of those links.



### Spicy

- Lorem ipsum dolor sit amet consectetur, adipisciing etia.
- Cupiditate facilis tempora molitiae, nulla perspiciatis corporis quicquam!
- Quis repelat iusto distinctio quas lute eaque aut?
- Explicabo in omnia quam aliquid molestias optio cumque?

### Medium

1. Lorem ipsum dolor sit amet consectetur, adipisciing etia.
2. Cupiditate facilis tempora molitiae, nulla perspiciatis corporis quicquam!
3. Quis repelat iusto distinctio quas lute eaque aut?
4. Explicabo in omnia quam aliquid molestias optio cumque?



That is working nicely and it looks like each one of these other elements is there as well. That's how you can create an Anchor Link in HTML.



## Coding Exercise

Create an anchor link with a path of #product and name it accordingly. Also put a h1 tag inside it that says "Product One"

# 1.46 ::Before ::After pseudoselectors

A common question I get from students relates to the before and after pseudo selectors. And so in this guide, I want to walk through a dead simple explanation for how they work.

## helpers.css

```
.anchor-link h1::after {  
    font-family: "Font Awesome 5 Free";  
    content: "\f0c1";  
    font-weight: 900;  
}
```

And what we're gonna do is, we have our links right here on the menu and I want to add a little link icon to the right of each one of them. Now technically I could just go to Font Awesome and then we could put an icon, place it inside of that H1 tag, but then we'd have to do it for each one of these elements.

But what if we want to have every anchor tag in the entire application to have a link there? Well that is a perfect scenario for using an after, or a before pseudo selector. So let's talk about what they are, because they can look a little bit confusing if you've never seen them. But I think that after we walk through this demo, it's gonna make a lot more sense.

Let's switch back to the code here, and we have right here our spicy H1 anchor tag. And we can get access to this, if we just look and we select an anchor link and then we grab the H1 tag inside of an anchor link. That's gonna give us exactly what we're looking for.

Let me open up our helper's file and down below I'm gonna grab the anchor link and then after that, I'm gonna grab the H1 and then I'm going to add two semi-colons here, and I'll say before because I'm gonna show you both the before and after pseudo selectors.

And this is a very different way of working with CSS because usually with CSS we don't render items in content directly on the page, but that's exactly what this pseudo selector allows us to do. The attribute I'm gonna use is called content, and then content takes a string. So right here I'm going to say content, and let me just say I'm before and then I'm going to copy this and now we're also going to create an after pseudo selector.

And here I'm going to add just a little dash and here I'll say I'm after, and I'll put a dash and a space right there.

## helpers.css

```
.anchor-link h1::before {  
    content: "I'm before -"  
}  
.anchor-link h1::after {  
    content: "- I'm after"  
}
```

Now if I hit save and come back and hit refresh, you can see that this added the actual content in the H1 link.



I'm before - Spicy - I'm  
after

- Lorem ipsum dolor sit amet consectetur, adipisicing elit.
- Cupiditate facilis tempora mollitia, nulla perspiciatis corporis quisquam!
- Quis repellat iusto distinctio quas iure eaque aut?
- Explicabo in omnis quam aliquid molestias optio cumque?

We did not touch the HTML whatsoever, so what before and after do is they allow you to select an item on the page and add content to it. This can be incredibly helpful because notice if you scroll down, this has affected every one of the anchor tag H1s.

And if you wanted to say, make a change that populated on hundreds of pages, this is a great way of doing it. If not, you'd have to go and find every single anchor tag in the entire system and then change it so that's really all that you need to do.

Now I want to add an icon, so let's talk about what we can do in order to make that happen. I'm gonna get rid of this before tag here, and in the content I



I'm before - Spicy - I'm  
after

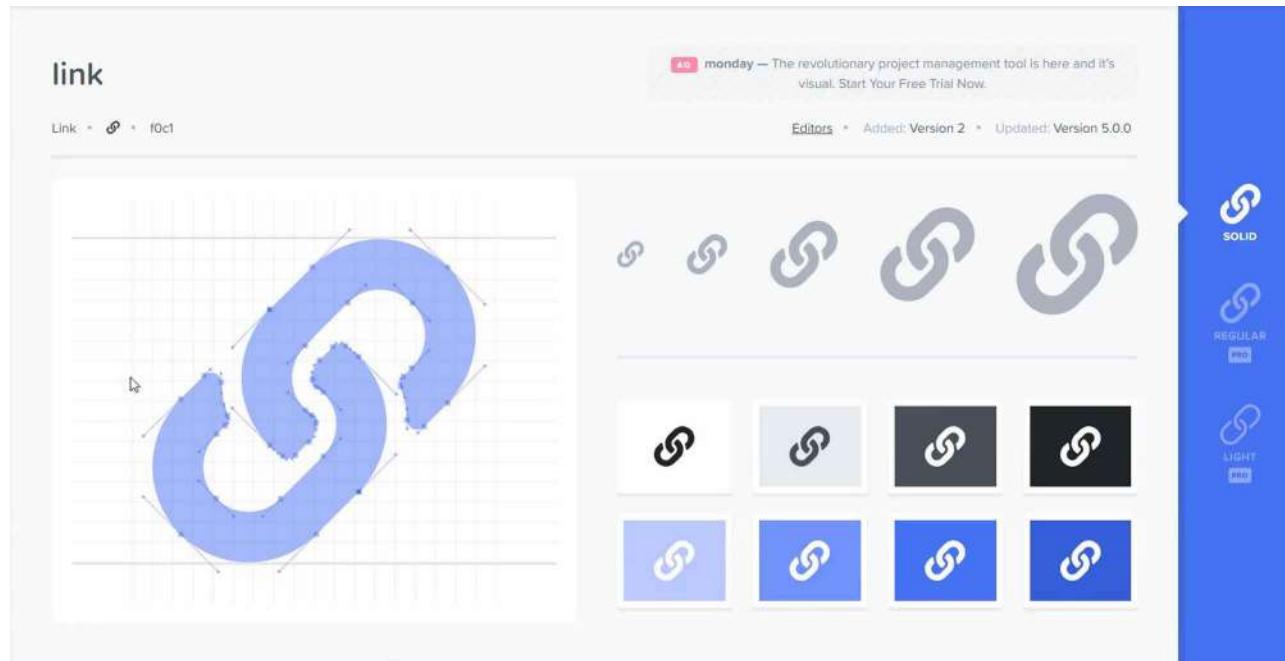
- Lorem ipsum dolor sit amet consectetur, adipisicing elit.
- Cupiditate facilis tempora mollitia, nulla perspiciatis corporis quisquam!
- Quis repellat iusto distinctio quas iure eaque aut?
- Explicabo in omnis quam aliquid molestias optio cumque?

I'm before - Medium - I'm  
after

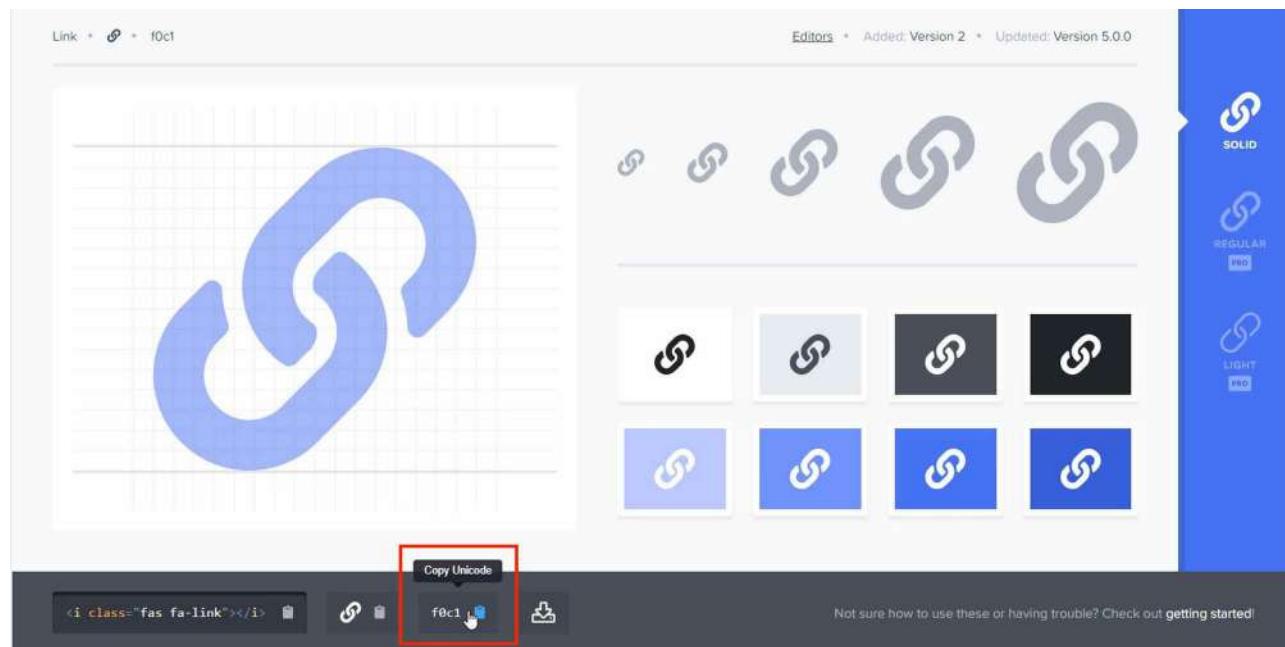
- 1. Lorem ipsum dolor sit amet consectetur, adipisicing elit.
- 2. Cupiditate facilis tempora mollitia, nulla perspiciatis corporis quisquam!
- 3. Quis repellat iusto distinctio quas iure eaque aut?
- 4. Explicabo in omnis quam aliquid molestias optio cumque?



am going to add a unicode character. So we need to switch back to the browser and let's go to Font Awesome here. And we're going to grab a link tag here, so let's type in link and then click on it.



With Font Awesome, the traditional way that you use it is to simply grab their HTML elements in these little i tags. But you also have a few other options, you could get the actual glyphicon which is not something that I do very often. In fact I don't think I've ever done that. But this one is very helpful, this is a unicode representation of the icon.



What this is going to allow us to do is to call this directly from within a CSS file. Because we can't call these class names, so we're gonna call this unicode. And this is going to allow us to treat it exactly like a font.

Now inside of the content, I'm going to use an escape character, so this is gonna be a forward slash, or backward slash. And then from there I'm going to paste in that unicode character. I'm going to also add a space right there, just so we have a little bit of space between the content and our content tag.

## helpers.css

```
.anchor-link h1::after {  
    content: " \f0c1";  
}
```

So that is the content, but one other thing we have to do here is we have to bring in the font family. So we have to use the Font Awesome font family, and this part's a little tricky and this required me the first time I had to do it on an app. This took a little bit of debugging because you would think that you could just do something like this. Where you call Font Awesome, but that technically doesn't work.

```
font-family: "FontAwesome";
```

And their documentation doesn't do a good job of explaining it, so you have to say, Font Awesome and then the version, so we're using version five, and then free, so that is a little bit odd. There's also one other trick and this one was very frustrating because there was nothing in the documentation that explained it. But Font Awesome also requires for their free icons, you to only use a certain font weight. So you have to pass in a font weight and so the font weight needs to be at 900.

## helpers.css

```
.anchor-link h1::after {  
    font-family: "Font Awesome 5 Free";  
    content: "\f0c1";  
    font-weight: 900;  
}
```

Now if we do this and come back to our menu page and hit refresh, you can see we've added that link.



### Spicy 🔥

- Lorem ipsum dolor sit amet consectetur, adipisicing elit.
- Cupiditate facilis tempora mollitia, nulla perspiciatis corporis quisquam!
- Quis repellat iusto distinctio quas iure eaque aut?
- Explicabo in omnis quam aliquid molestias optio cumque?



### Medium 🔥

1. Lorem ipsum dolor sit amet consectetur, adipisicing elit.
2. Cupiditate facilis tempora mollitia, nulla perspiciatis corporis quisquam!
3. Quis repellat iusto distinctio quas iure eaque aut?
4. Explicabo in omnis quam aliquid molestias optio



And the link has been populated every single spot where we have any of our H1 tags inside of an anchor link. And so this is something that, if you look at other code bases, you're gonna see this little after pseudo selector all over the place.

One of the most popular places I see it is within the body tag. And so with in the body tag, they'll put some kind of before and after. And it might look a little weird, especially if you've never seen it before, and so I wanted to create a dead simple example, and a practical example of how it can be used. It literally is doing exactly what it says it's doing. It gives you the ability to put content before or after an element that you've selected.

## 1.47 contact.html creation

You've done an amazing job making it this far through the course, and you've built out a pretty cool website in a relatively short period of time and we're on our final page. I've saved one of the coolest features we're going to build for the very end.

We're also going to learn how we can integrate some really neat animations, as you can see right here, if I click on "your name", do you see how that name label drops down, and if I click out of it, it goes away. We also add a really neat little drop shadow here. It's light, it's not too over the top.



I've seen some shadows that really just are way, way too strong, and they don't look good. But here it's just a nice subtle one. You see Google doing a lot of these kinds of effects. If I click away, then it goes back just to a straight line.

Now we're going to walk through how an HTML form looks by default and it looks very different than this. So we're going to see how to build out this cool animation and how to add the drop shadow, all kinds of cool things like that.

We have two different text fields, then we also have this message field which is a text area field that allows users to type paragraphs in, so we're going to work on that. We'll work on how we can add a button that will have a hover effect when you click on it.

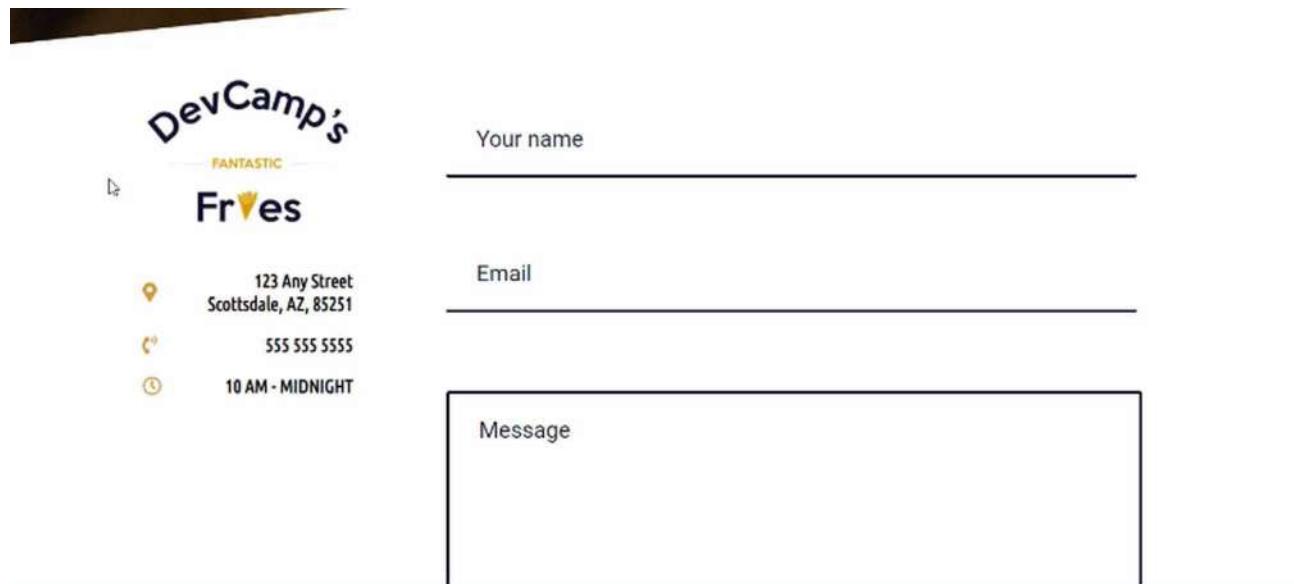
This is also something I've seen Google do a lot of where they add these little click features so it gives some depth as the user is clicking on it, which is, I think, a pretty cool feature.

Then here on the left-hand side, we're going to build out a little grid here that'll have the address, the phone, and then the hours. So we definitely have our work cut out for us, but this is going to be

really

fun.

This is one of my favorite parts of the whole site.



In this guide, we're simply going to add the HTML. Then each one of these features is a little bit advanced, so we're going to take it nice and slow to make sure that you understand exactly what is going on.

Let's switch over into the code, and for our contact page, let's just get rid of all of the content and then we're going to copy everything from the About Us page. So if I paste this in, paste that in, remove the first line and we can say, "Contact Us".

## contact.html

```
<head>
<meta charset="utf-8" />
<title>Contact</title>
<meta name="viewport" content="width=device-width, initial-scale=1">
```

Now, we wanna have our icons and our fonts, and then we also are going to wanna have our common styles, our nav, our footer. We want the skewed header. We do want the page container in this case. We do not need the square grid.

I don't know about helpers, but my rule of thumb is to not include something until I know for a fact that I'm going to use it.

## contact.html

```

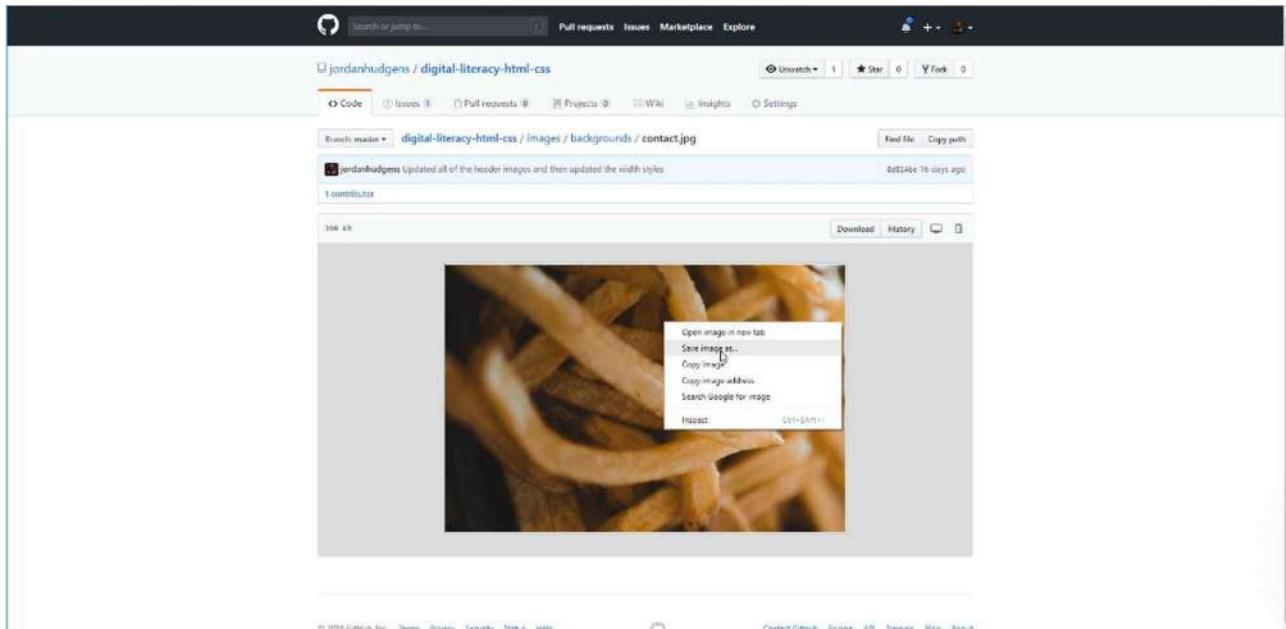
<link href="https://fonts.googleapis.com/css?family=Roboto"
rel="stylesheet">
<link href="https://fonts.googleapis.com/css?family=Ubuntu+Condensed"
rel="stylesheet">
<link href="https://fonts.googleapis.com/css?family=Roboto+Slab"
rel="stylesheet">

<link rel="stylesheet"
href="https://use.fontawesome.com/releases/v5.2.0/css/all.css"
integrity="sha384-hWjflwFxL6sNzntih27Pbfkr27PmbbK/iSvJ+a4+0owXq79v+lsFkW54
bOGbiDQ" crossorigin="anonymous">
<link rel="stylesheet" href="styles/common.css">
<link rel="stylesheet" href="styles/nav.css">
<link rel="stylesheet" href="styles/footer.css">
<link rel="stylesheet" href="styles/skewed-header.css">
<link rel="stylesheet" href="styles/page-container.css">

</head>

```

That is everything we need there. Now, in the skewed header, we're going to use another background image. So if you want, I will include a link to it in the show notes, and it's going to be the contact image. So we'll save this into our backgrounds so we have contact. So now we'll be able to use that.



Let's change the image tag, as well as the title.

## contact.html

```

<body>
  <div class="skewed-header">
    <div class="header-bg" style="background-image:
url('images/backgrounds/contact.jpg');"></div>

    <div class="skewed-header-wrapper">
      <div class="skewed-header-content">
        <div class="heading-wrapper">
          <h1>Contact</h1>
        </div>
      </div>
    </div>
  </div>

```

Everything in the nav is going to remain the same. Now, here in the page container with this, we're going to keep the page container class because I just wanna have that big wrapper.

But we're going to get rid of everything else inside of it, and then we'll keep the spacer and if we're keeping the spacer, that means we are going to need our helpers. So let's bring that in.

### contact.html

```
<link rel="stylesheet" href="styles/common.css">
<link rel="stylesheet" href="styles/nav.css">
<link rel="stylesheet" href="styles/footer.css">
<link rel="stylesheet" href="styles/skewed-header.css">
<link rel="stylesheet" href="styles/page-container.css">
<link rel="stylesheet" href="styles/helpers.css">
```

The reason why I do not like to include items that I'm not going to use is that every time that we include one of these elements, one of these files, there is a small load that comes in from the server with its request.

With each one of these files, the website has to say, "Okay, I need the page container. I need the common CSS file. I need all of these things." The more files that you import the slower your page is going to load.

Now, with files as small as ours, it's not an issue, but it is a best practice that you want to just kinda get in the habit of is only including the items that you genuinely need.

We're going to have page container, I'm going to put a little h1 in here just so we can see it.

### contact.html

```
<div class="page-container">
  <h1>I'm in the page container</h1>
</div>

<div class="spacer60"></div>
```

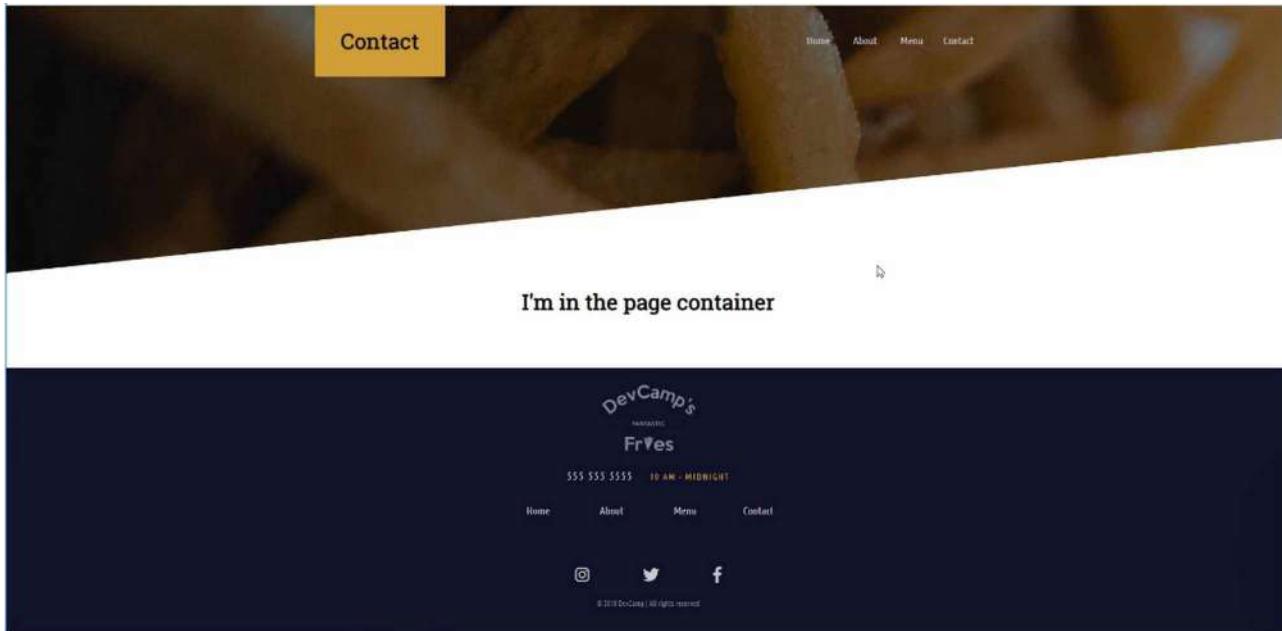
Hit save, and then we also want to get rid of all of our squares, so let's get that. We'll take it down to the footer.

### contact.html

```
<div class="spacer60"></div>

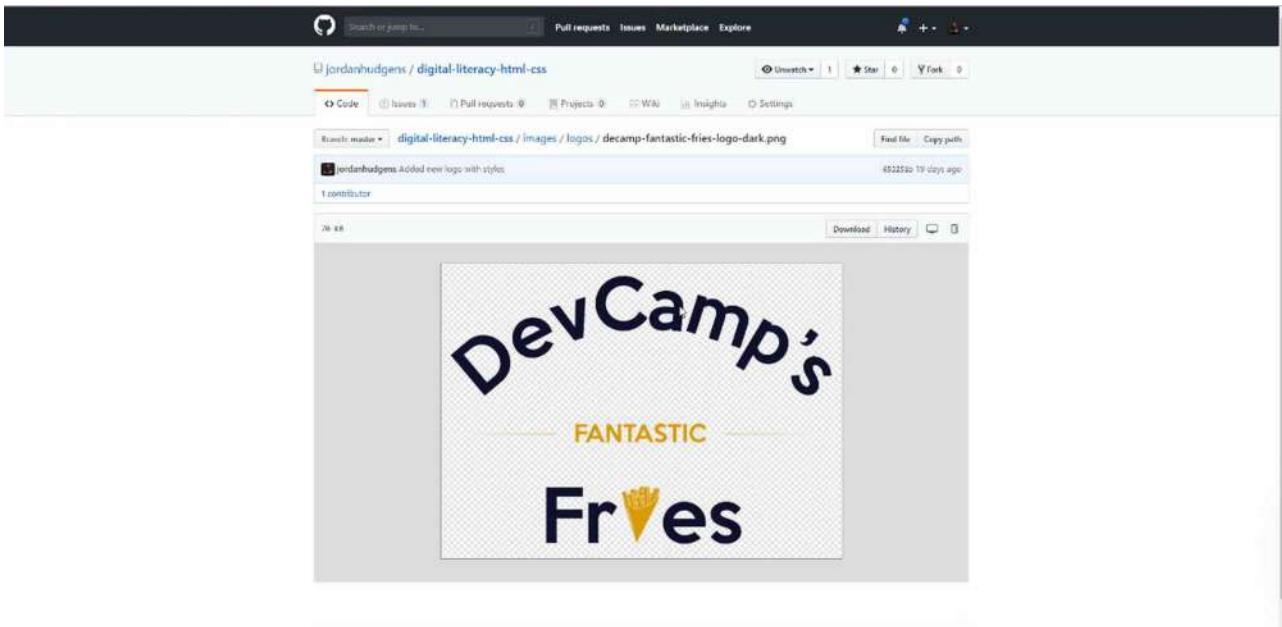
<div class="footer">
```

Hit save. Now if you go back, we should have our contact page, at least a placeholder for it.



And yes, we have the correct image up here with the correct skew. We have our page container and it's centered, and contact. So everything looks like it is working properly, so let's come back and now let's start adding the content.

Now, we know that we're going to have a grid. So let's take a look at the final version.



It's going to be a little bit different than what we've used before because we're going to have this little sidebar item and it's going to be pretty small and maybe taking up something like 20% or so of the whole space.

Then our form is going to take up all of the rest of the space. So this is going to be a little bit different than we've implemented before, which is good.

You can get used to working with the two column layout like this. So we know we're going to have a logo, address, we're going to have icons, we're going to have phone number. Then the hours and then this form element. Let's start adding those.

I'm first going to create a contact grid wrapper class, and everything is going to go inside of here.

## contact.html

```
<div class="page-container">
  <div class="contact-grid-wrapper">
    <div class="company-metadata-sidebar-wrapper">
      <div class="logo">
        <img src="" alt="">
      </div>
    </div>
  </div>
</div>
```

Now, some of the names are a little bit verbose but I kind of like that because it's very descriptive. I can take a look at this code and I'm instantly going to know exactly what it's for.

One of the nice benefits in having a few of your classes have long names is that you can use shorter, easier to follow names inside of it.

Since all of the styles we're going to be applying are only going to be for, say for this logo class, they're only going to be called for the logo classes inside of the company metadata sidebar wrapper.

That's one reason why I like to follow that kind of pattern. This image tag is going to be for a logo. I do not believe that I've pulled this one down yet.

We have a logos directory, but we only have the white logo. We need to bring in the darker one, the one with the blue and the gold. So let's go and find that. Once again I'll keep this in the show notes as well. This one I believe is this `decamp-fantastic-fries-logo-dark.png`.



Save that in our logos folder, now all we need to do is import that one.

## contact.html

```
<div class="page-container">
  <div class="contact-grid-wrapper">
    <div class="company-metadata-sidebar-wrapper">
      <div class="logo">
        
      </div>
    </div>
  </div>
```

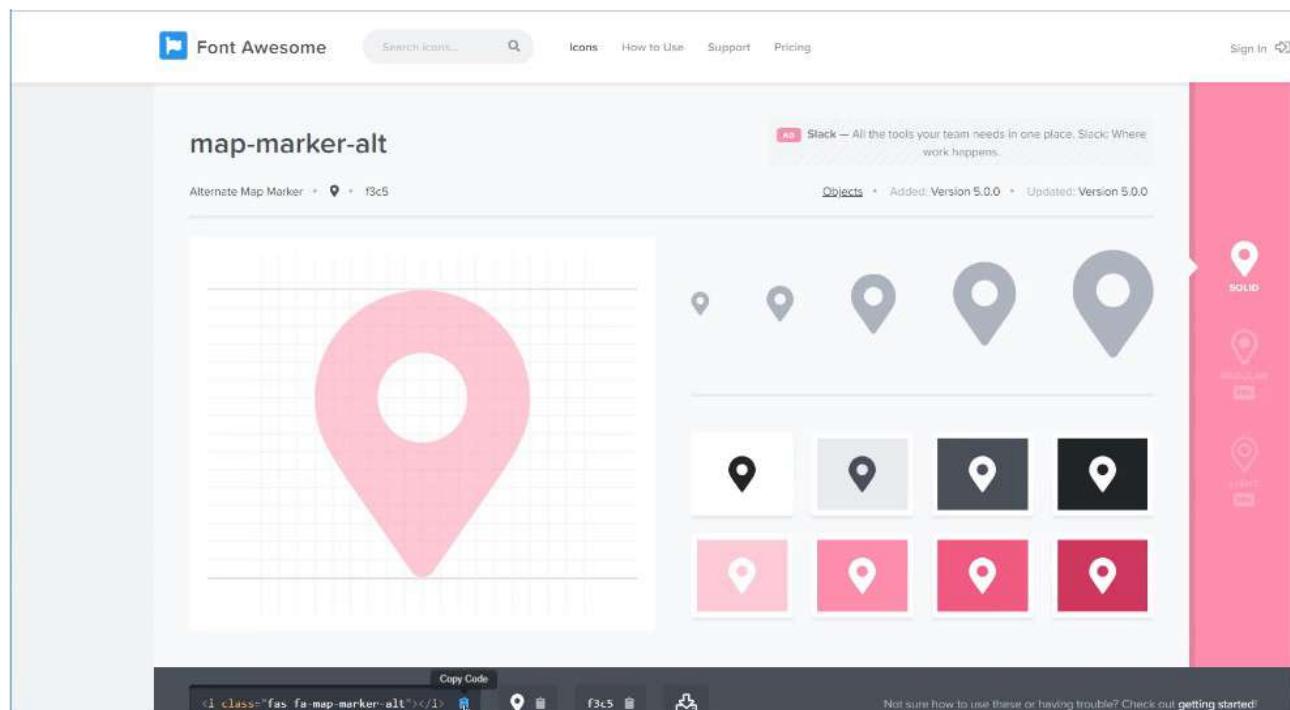
That is our logo class. Now let's create another one.

## contact.html

```
<div class="page-container">
  <div class="contact-grid-wrapper">
    <div class="company-metadata-sidebar-wrapper">
      <div class="logo">
        
      </div>

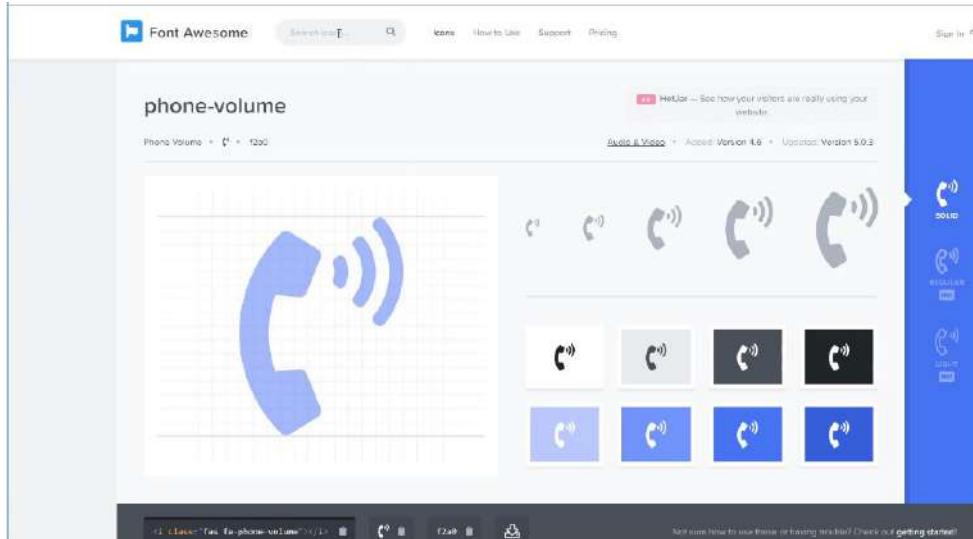
      <div class="company-details-wrapper">
        </div>
      </div>
    </div>
  </div>
```

Now, in the company-details-wrapper, this is where we'll put in the address, the phone number and then the hours. Each one of these is also going to have an icon. So let's grab the map icon from Font Awesome.

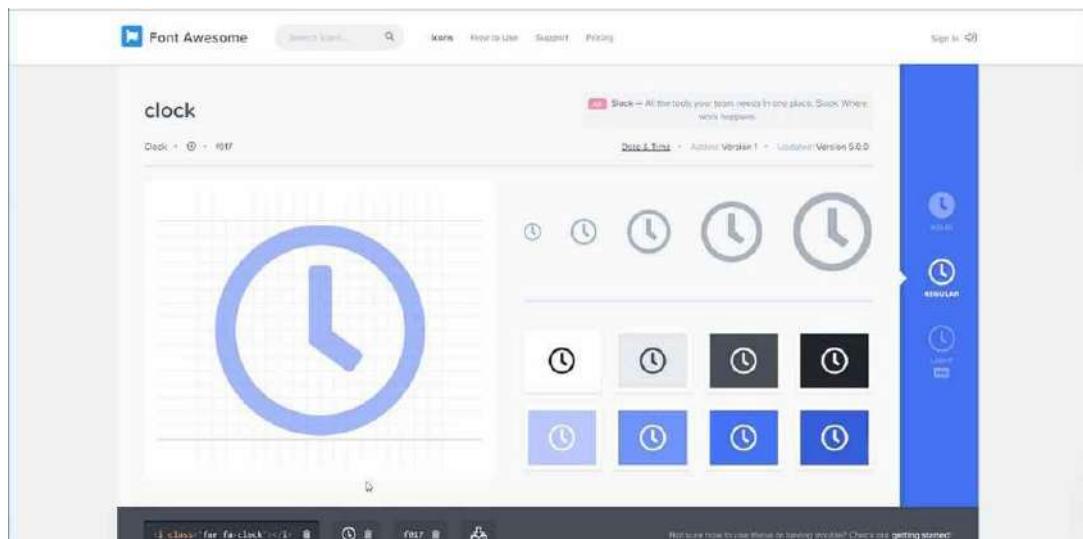


Then while we're at it, let's just grab the phone and then the clock for the hours.

Here's the phone.



And now the clock.



Here's the code.

### contact.html

```
<div class="company-details-wrapper">
  <i class="fas fa-map-marker-alt"></i>

  <i class="fas fa-phone-volume"></i>
  <i class="far fa-clock"></i>
</div>
```

We have all of the icons that we're going to need and we'll just save them right here. As we clone this we'll remove the ones that we don't need.

We have our icon and now I'm just going to add a div here and it's just going to be an empty div. Inside of this, this is where we're going to put the address.

## contact.html

```
<div class="company-details-wrapper">
  <i class="fas fa-map-marker-alt"></i>

  <div>
    123 Any St<br>Scottsdale, Arizona
  </div>

  <i class="fas fa-phone-volume"></i>
  <i class="far fa-clock"></i>
</div>
```

So we'll go with the good ol' 1, 2, 3 any street. We're going to use a break tag, so this goes on two lines, and then well put this in Scottsdale, Arizona, 85251 just like that. That's all we need to do right there.

This is going to be our first company details wrapper, and then let me just grab these so that we're not copying them with each one. Now let's go and let's clone this for each of the other elements.

## contact.html

```
<i class="fas fa-phone-volume"></i>
<i class="far fa-clock"></i>

<div class="company-details-wrapper">
  <i class="fas fa-map-marker-alt"></i>

  <div>
    123 Any St<br>Scottsdale, Arizona
  </div>
</div>
```

So we'll make one and two. So now we have three total, the very first one is for the address. Now let's get one for the phone and then one for the hours.

## contact.html

```
<div class="company-details-wrapper">
  <i class="fas fa-map-marker-alt"></i>

  <div>
    123 Any St<br>Scottsdale, Arizona
  </div>
</div>

<div class="company-details-wrapper">
  <i class="fas fa-phone-volume"></i>

  <div>
    555 555 5555
  </div>
</div>

<div class="company-details-wrapper">
  <i class="far fa-clock"></i>

  <div>
    10 AM - MIDNIGHT
  </div>
</div>
```

That is all of the metadata that we're going to have on that sidebar. So now that we have that, we are ready to build out the structure for our form. So right here, this is the sidebar wrapper. I wanna make sure that I'm going outside of that.

Now we're going to create another div here, and this is going to be a div that I'm going to give a class of form to. Then I'm going inside of here to create a few divs. So I'm going to create three divs for those elements. Remember, we have a name, an email, and a message.

### contact.html

```
<div class="form">
  <div class="form-group">
    Name
  </div>

  <div class="form-group">
    Email
  </div>

  <div class="form-group">
    Message
  </div>
</div>
```

Then, after that let's add a button. So this is going to be the div that is going to manage the button. So I'll call this centered button wrapper.

### contact.html

```
<div class="form">
  <div class="form-group">
    Name
  </div>

  <div class="form-group">
    Email
  </div>

  <div class="form-group">
    Message
  </div>

  <div class="centered-btn-wrapper">
  </div>
</div>
```

The reason why I'm going with this name is because if you have a centered button or you create styles for a centered button, there's a good chance you're going to need to add that in multiple parts of the application, maybe even in spots that you don't have a form, you may just need a button that you want centered.

I'm going to go with a little bit more a generic name. We'll also go with a more generic selector. So inside of there, in this case, I will just put the button.

## contact.html

```
<div class="form">
  <div class="form-group">
    Name
  </div>

  <div class="form-group">
    Email
  </div>

  <div class="form-group">
    Message
  </div>

  <div class="centered-btn-wrapper">
    <button type="submit">Send</button>
  </div>
</div>
```

This is a built-in HTML element here called button. We'll add a class later. That should be all we need to implement from a content perspective.

Let's switch back and see if our HTML is done. Hit refresh, and yes we have our logo. We have our sidebar, we have the first beginnings of our name, email, message form, and then our button.



It looks like we have all of the HTML for the most part that we're going to need. In the next guide, let's start building out our grid so that we can have the sidebar all lined up here on the left-hand side, and then we can segment out the section we want to be dedicated to the form.



## Coding Exercise

Make 3 divs, each with a class name of `social-links`. Inside of each div, put in an `i` tag with what ever Font Awesome class you'd like.

# 1.48 Page Layout with CSS Grid

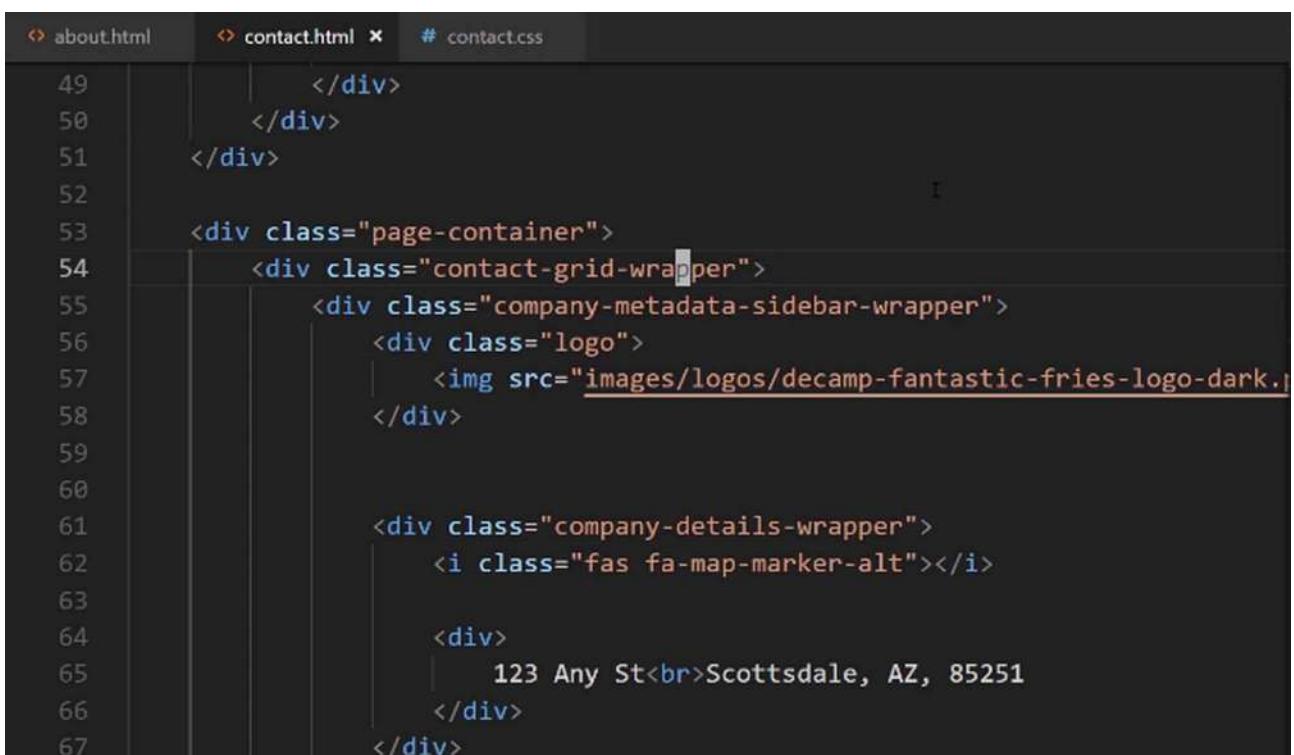
Right now we have this gigantic logo, we have our sidebar content and then we have what's going to be the form, kind of all sitting on top of each other. Let's start with that. I'm going to create a dedicated page just for the contact page, or I should say dedicated stylesheet for that.

We know we're going to need that anyway. We might as well create it now. In **styles.css**, I'm going to create a file called **contact.CSS**. Let's make sure that we import it here at the top. Duplicate that line and now we want **contact.CSS**.

## contact.html

```
<link rel="stylesheet" href="styles/contact.css">
```

Now with all of this in place, we can start adding the styles. If I look at this. Let me just make sure I have all of the names. We went with this **company-metadata-sidebar-wrapper** here, and then we have the **contact-grid-wrapper**. We're going to start with this and then we're going to move down and add this later on.



```
49     </div>
50     </div>
51   </div>
52
53   <div class="page-container">
54     <div class="contact-grid-wrapper">
55       <div class="company-metadata-sidebar-wrapper">
56         <div class="logo">
57           
58         </div>
59
60
61       <div class="company-details-wrapper">
62         <i class="fas fa-map-marker-alt"></i>
63
64         <div>
65           123 Any St<br>Scottsdale, AZ, 85251
66         </div>
67     </div>
```

Let's just grab the **content-grid-wrapper**. I'm just going to copy this, give us a little bit more room, and then paste it in. Now let's add a **width** here for our content and this is going to be **1000px** wide.

I want to use the grid for this so I'm going to say **display** is going to be **grid**. I want to use **grid-template-columns** and this is a perfect example of a situation where grid is a better option than Flexbox because we want two columns.

## contact.css

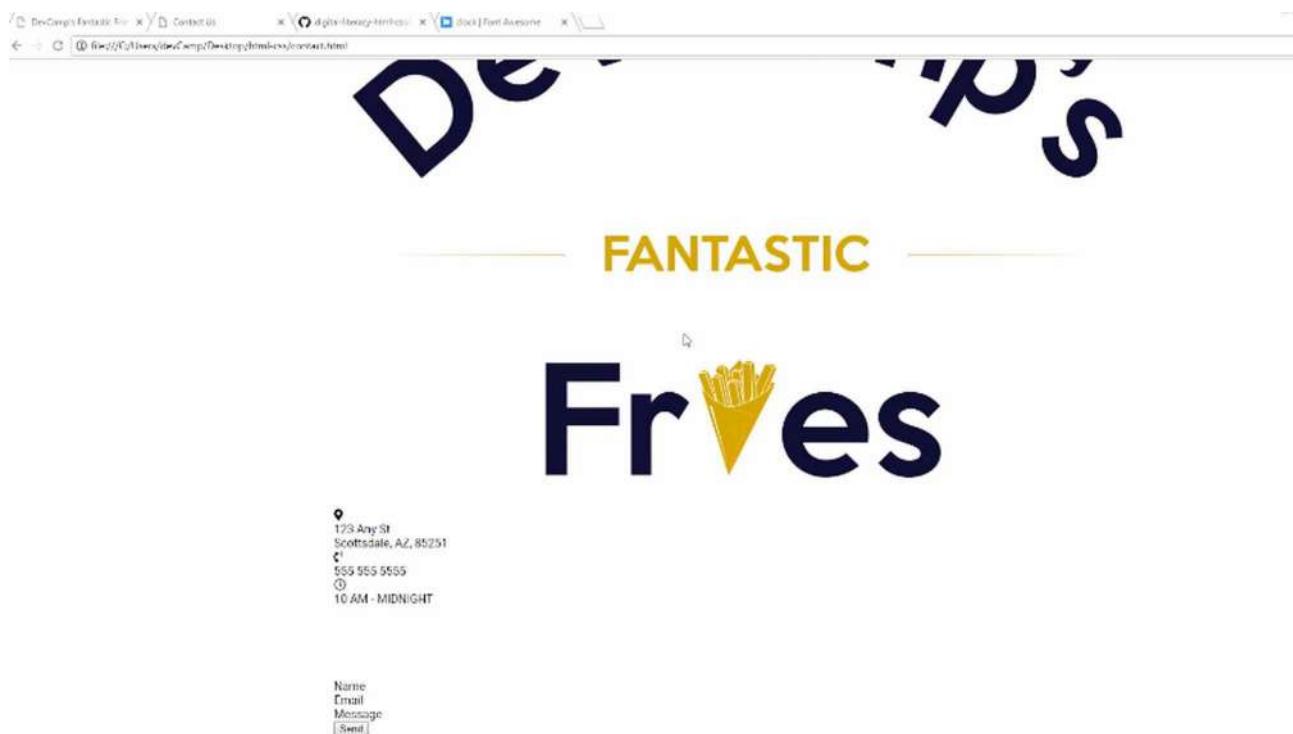
```
.contact-grid-wrapper {  
    width: 1000px;  
    display: grid;  
    grid-template-columns:  
        }  
    }
```

We want one of them to just be the normal size and then we want the other column to be three times that size, which is exactly what we're looking for with the form and then the sidebar right next to it. I want to have a decent amount of `grid-gap` between them, so I'm going to say `grid-gap: 100px`.

## contact.css

```
.contact-grid-wrapper {  
    width: 1000px;  
    display: grid;  
    grid-template-columns: 1fr 3fr;  
    grid-gap: 100px;  
}
```

Now if I go and switch, let's see what that does. Okay, I hit refresh and we still have the issue where this image is too big. Let's fix that first before we do anything else.



We have the next class here, which is going to be this `company-metadata-sidebar-wrapper`. I'm going to copy this, come down and paste that in. I want to grab the logo and then from there or I should say I need to grab the class of logo and then the image.

The way that you can have an image, and this is a very cool part of CSS grid. The way you can have an image fit inside perfectly, whatever size container that you want to give it or whatever size column you want to give it is by just saying that you want the **width** to be **100%**.

## contact.css

```
.company-metadata-sidebar-wrapper > .logo img {  
    width: 100%;  
}
```

It's only going to take up 100% of whatever div it is inside. Let's see if that works. Now, if I hit refresh it still doesn't work. We might have an issue with where that item is. We have the **content-grid-wrapper** here and that is at **1000px**, and then we have the **company-metadata-sidebar-wrapper**.

It looks like I selected the right thing, but obviously, I did not. So let's go and see what the issue is here when we inspect it. I'm going to inspect the image and that is at **100 width**. Now let's look at the **content-grid-wrapper**. Oh, here is the issue. I have display grid and then I have the grid template columns here.

```
<!doctype html>  
<html>  
  <head>...</head>  
  <body>  
    <div class="skewed-header">...</div>  
    <div class="page-container">  
      <div class="contact-grid-wrapper" style="background-color: #f0f0f0; padding: 10px;">...</div>  
      <div class="company-metadata-sidebar-wrapper" style="background-color: #e0e0e0; padding: 10px;">...</div>  
      <div class="logo" style="text-align: center; margin-bottom: 10px;">
```

It is saying that the issue is that we cannot use **1FR** and **3RF**. If you hover over here it says that this is an **invalid property value**. Let's go take a look at the code and see what could be the issue because it looks like...oh, I found it. It's supposed to be **1FR**.

You were probably watching me as I was doing that and yes, it doesn't matter how long you do this you will have typos, and walking through the process like I just did is the best way of fixing them. Now, this is working perfectly. There we go.

The screenshot shows a web browser with the URL [@devcampsfries/contact.html](#). The page content includes a logo for "DevCamp's Fries", company metadata (123 Any St, Scottsdale, AZ, 85251, 555 555 5555, 10 AM - MIDNIGHT), and a contact form with fields for Name, Email, Message, and a Send button. The developer tools are open, specifically the Elements and Styles tabs. The Elements tab shows the DOM structure with a selected element: <div class="contact-grid-wrapper">. The Styles tab shows the CSS for this element, which defines a grid layout with a width of 1000px, a grid of 1fr and 3fr columns, and a grid gap of 100px.

```

<!DOCTYPE html>
<html>
  <head>...</head>
  <body>
    <div class="skewed-header">...</div>
    <div class="page-container">
      <div class="contact-grid-wrapper"> ...
        <div class="company-metadata-sidebar-wrapper">...
          <div class="form">...
        </div>
      </div>
    </div>
  </body>
</html>

```

```

element.style {
}
.contact-grid-wrapper {
  width: 1000px;
  display: grid;
  grid-template-columns: 1fr 3fr;
  grid-gap: 100px;
}

```

We have our image here. Let's add a little bit of margin here at the bottom. Let me select this image and I'm just going to add `margin-bottom` here. Let's go for `20px`. That looks good. Let me copy this, and we could have this apply to either the logo class or the image in this case. Either one of those will work fine.

## contact.css

```

.company-metadata-sidebar-wrapper > .logo img {
  width: 100%;
  margin-bottom: 20px;
}

```

I'm just going to add it directly to the image. Now if I hit refresh, everything is working. This is looking really good. We now have the grid layout that we're wanting. Now let's work on each one of these content wrappers because these are all going to be identical.

The screenshot shows the same web browser and contact page as before. The developer tools are open, showing the Elements and Styles tabs. The Elements tab shows the DOM structure with a selected element: 
<html>
  <head>...</head>
  <body>
    <div class="skewed-header">...</div>
    <div class="page-container">
      <div class="contact-grid-wrapper">
        <div class="company-metadata-sidebar-wrapper">
          <div class="logo">
             .logo img {
  width: 100%;
  margin-bottom: 20px;
}

```

If you look at the final version, we're going to have an icon on the left and then the content itself, the text content, is going to be on the right-hand side. So let's see what we need to do in order to implement that.

We have this `company-details-wrapper` so we have the identical class for each one of these divs. So I'm going to grab this and then let's come down here and I'm going to grab that sidebar wrapper and then that's going to be all we need is the `company-details-wrapper`.

### contact.css

```
.company-metadata-sidebar-wrapper > .company-details-wrapper {  
}
```

Once we're inside here, we can add a custom `font-family`. We're going to want that Ubuntu Condensed, you saw that was a little bit of a different font. That was the one we used in the navbar. Then we want `sans-serif` as a backup, and let's make it bold.

We're going to have a `font-weight: 900`, and then the display it's going to be a Flexbox container. We're going to use Flexbox and then let's `justify-content: space-between`. If you notice we have those two columns and it's separated by whatever available space that we have.

### contact.css

```
.company-metadata-sidebar-wrapper > .company-details-wrapper {  
    font-family: "Ubuntu Condensed", sans-serif;  
    font-weight: 900;  
    display: flex;  
    justify-content: space-between;  
}
```

Right here you can see we have this column here and then on the right-hand side we have the rest of it. So let's come back. That is the `space-between` property that we're wanting, and let's just see what that gives us.

Let's take a look at this. Again, I'm going to hit refresh and that's looking better. It's not perfect yet obviously but it is almost where we want it to be.

```

<!DOCTYPE html>
<html>
  <head>...</head>
  <body>
    <div class="skewed-header">...</div>
    <div class="page-container">
      <div class="contact-grid-wrapper">
        <div class="company-metadata-sidebar-wrapper">
          <div class="logo">
            
          </div>
        <div class="company-details-wrapper" style="background-color: #f0f0f0; padding: 10px; border-radius: 5px; margin: 10px 0; font-family: 'Ubuntu Condensed', sans-serif; font-weight: 900; display: flex; justify-content: space-between; margin-top: 20px; margin-bottom: 20px; text-align: right; font-size: 1.25em; border: 1px solid #ccc; border-radius: 5px; padding: 10px; width: fit-content; margin-left: auto; margin-right: 0; position: relative; z-index: 1; ">
          <div>123 Any St<br/>Scottsdale, AZ, 85251</div>
          <div>555 555 5555</div>
          <div>10 AM - MIDNIGHT</div>
        </div>
      </div>
    </div>
  </body>
</html>

```

Let's just add a few more style elements. Let's do it right here inside of the `devtools` just so we can see it all happening. This way we don't have to keep on switching back and forth.

For `company-details-wrapper` here let's go and let's add a margin to the top and bottom. So I'll say `margin-top` of `20px`, `margin-bottom` of `20px`, and then I want the `text-align: right`. There we go. From there, let's increase the `font-size` a little bit, so let's go with `1.25em`. That's looking much better.

```

<!DOCTYPE html>
<html>
  <head>...</head>
  <body>
    <div class="skewed-header">...</div>
    <div class="page-container">
      <div class="contact-grid-wrapper">
        <div class="company-metadata-sidebar-wrapper">
          <div class="logo">
            
          </div>
        <div class="company-details-wrapper" style="background-color: #f0f0f0; padding: 10px; border-radius: 5px; margin: 10px 0; font-family: 'Ubuntu Condensed', sans-serif; font-weight: 900; display: flex; justify-content: space-between; margin-top: 20px; margin-bottom: 20px; text-align: right; font-size: 1.25em; border: 1px solid #ccc; border-radius: 5px; padding: 10px; width: fit-content; margin-left: auto; margin-right: 0; position: relative; z-index: 1; ">
          <div>123 Any St<br/>Scottsdale, AZ, 85251</div>
          <div>555 555 5555</div>
          <div>10 AM - MIDNIGHT</div>
        </div>
      </div>
    </div>
  </body>
</html>

```

I also just want to make sure, I can't see from here or not, if the entire component itself is aligned in the center or not, so let me take a look. This is a Flexbox item, so I'm going to say `align-items: center`.

# Fries

123 Any St  
Scottsdale, AZ, 85251  
555 555 5555  
10 AM - MIDNIGHT

```
elements Console Sources Network Performance Memory Application Security Audits
[html>
head>
  <div class="skewed-header">...</div>
  <div class="page-container">
    <div class="contact-grid-wrapper">
      <div class="company-metadata-sidebar-wrapper">
        <div class="logo">
          
        </div>
        <div class="company-details-wrapper">...</div>
      </div>
    </div>
  </div>
```

Styles Computed Event Listeners DOM Breakpoints Properties

Filter

```
{  
  font-family: "Ubuntu Condensed", sans-serif;  
  font-weight: 900;  
  display: flex;  
  justify-content: space-between;  
  margin-top: 20px;  
  margin-bottom: 20px;  
  text-align: right;  
  font-size: 1.25em;  
  align-items: center;}
```

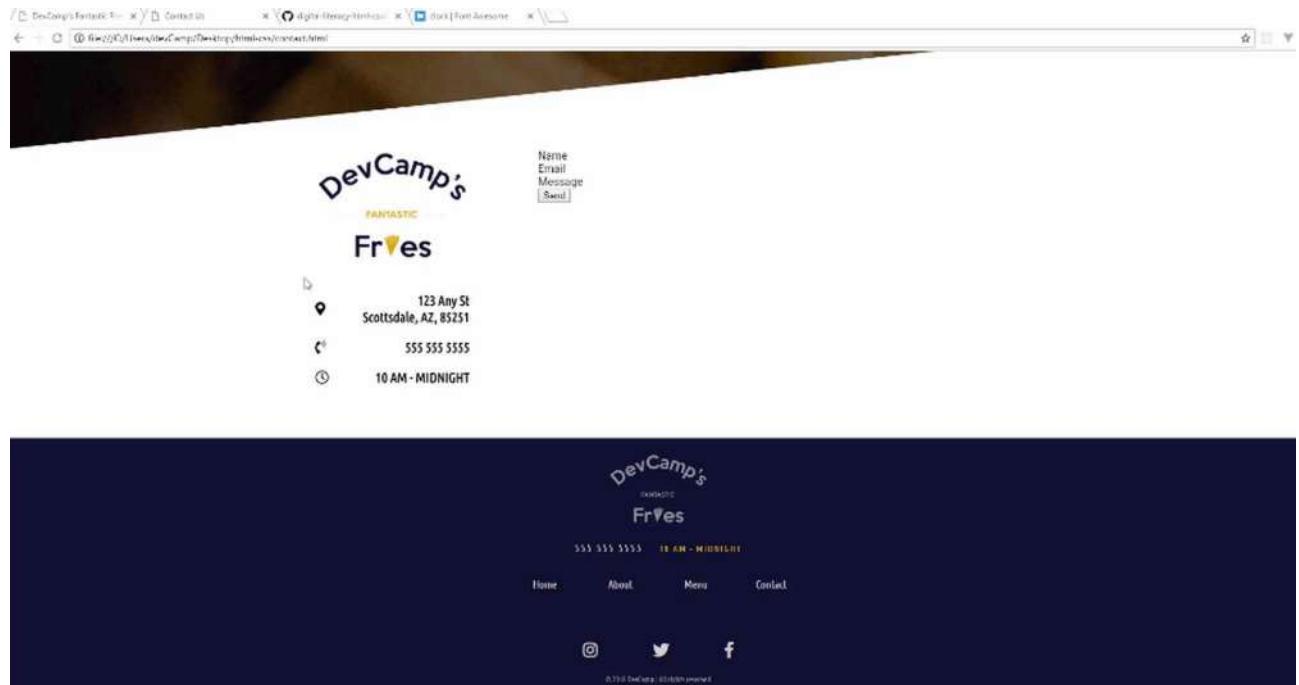
Yes, that did fix it. Did you see how that moved up for those icons? It moved each one of those items into the center. That's what I didn't notice. This icon and this icon were perfectly aligned but this one wasn't. So that is what we need to do.

Now that we have that, we can just copy all of these values and paste them directly into the code. Let's paste that in. I'm going to move the align items up here and the only reason for that is because whenever I'm using a tool like Flexbox or CSS grid, I don't like to have the rules that are specific to them be in different spots.

```
about.html contact.html # contact.css ●
  2   width: 1000px;  
  3   display: grid;  
  4   grid-template-columns: 1fr 3fr;  
  5   grid-gap: 100px;  
  6 }  
  7  
  8 .company-metadata-sidebar-wrapper > .logo img {  
  9   width: 100%;  
10   margin-bottom: 20px;  
11 }  
12  
13 .company-metadata-sidebar-wrapper > .company-details-wrapper {  
14   font-family: "Ubuntu Condensed", sans-serif;  
15   font-weight: 900;  
16   display: flex;  
17   justify-content: space-between;  
18   align-items: center; highlighted  
19   margin-top: 20px;  
20   margin-bottom: 20px;  
21   text-align: right;  
22   font-size: 1.25em;  
23 }
```

It would be really confusing if I had `display: flex` there, and then `align-items` here, and `justify-content` here. If I wanted to go make a change then it would take me a little while to see where everything is. So I like to have all of those items together.

Now if we come back, hit refresh, that is working very nicely. Let me close this and take a look. This is perfect. We have our sidebar now. It is 100% done.



In the next guide, we're going to start building out our form.



## Coding Exercise

Use flexbox on the container to evenly spread apart the `info-card` divs (you may have to do some research on `justify-content` values). Also give each of the child elements a border on the left side that is `10px` thick with a solid color of `black`;

```
<div id="info-container">
  <div class="info-card"></div>
  <div class="info-card"></div>
  <div class="info-card"></div>
</div>
```

# 1.49 HTML Form Elements

In this lesson we are going to build out our HTML form. We're not going to try to style it or anything like that. Forms can be one of the more confusing topics in the world of web development and so I want to take it nice and slow so that you are completely familiar with everything that we're working on.

So right here with the Name, Email, and Message is where we're going to place the form, and I think it's also going to be nice, because you saw how the form's going to end up and if you've never worked with forms before, I think you might be a little bit shocked at what a HTML form looks like out of the box. So that's what we're going to do here in this guide.

Coming down to where we have our placeholders here,

## contact.html

```
<div class="form">
  <div class="form-group">
    Name
  </div>
  <div class="form-group">
    Email
  </div>
  <div class="form-group">
    Message
  </div>
</div>
```

I'm going to start by adding an input. So I'm going to start with having an input here, and this is going to be of type text and then I'm going to set an id here of full name and then placeholder of your name, and that's all we need to do. And so what we're doing here is, we're saying that in this form we have an input tag and it is a text tag. So type is a special reserved word in these input forms.

So we are saying this is going to be a text form, and then we have an id of full name. And we're going to be using this when we build out our animation. And then the placeholder is just that temporary text you'll see that will just give some information to the user on what they should type in there.

## contact.html

```
<div class="form-group">
  <input type="text" id="fullName" placeholder="Your name">
</div>
```

Now if I hit save and switch back, you'll see that we have that little form element right here so that is working.



Now we're going to add a label. So this label is going to have a direct connection to the form and so here I'll say, this is for the full name and the value is going to just say your name.

### contact.html

```
<div class="form-group">
  <input type="text" id="fullName" placeholder="Your name">
  <label for="fullName">Your name</label>
</div>
```

Now if I hit save and refresh, you'll see that we have this little name right here, and if you click on this, it will actually activate the form.



FANTASTIC



Your name	I	Your name
Email		
Message		
<input type="button" value="Send"/>		



123 Any St  
Scottsdale, AZ, 85251



555 555 5555



10 AM - MIDNIGHT

Now just to give you a preview of what we're going to be doing, the label here is the same label that when you click on the form element, is going to drop down. So what we're going to do is, we're actually going to hide the label when the form loads and then once it's been activated, so once we click on it, that's when we're going to reveal it. And we're going to add a little animation there and have it fade and move down. So that is the same element.

And so now let's go and let's do the same thing for the email. So here this is going to be of type and instead of saying type text, this is actually going to be of type email. The id is going to be email and then the placeholder should just say either your email or email, whatever you prefer. And then make sure for the label that you say email.

### contact.html

```
<div class="form-group">
  <input type="email" id="email" placeholder="Your email">
  <label for="email">Your email</label>
</div>
```

Because what this label **for** attribute here does, is the for attribute creates a connection between the ID. So if your email was emailAddress, then the for would need to be changed to emailAddress. So that is what that for, is there for, and so now if I say your email, that is all that's needed.

The reason why I used type email instead of text is something ... let me show you right here if I hit refresh, what this does is, it'll give some validation. So if I were to type in, say something like this, or if I were to type my name in, like this, we could implement some validations. And if you hover over, you can even see right here, HTML, without us doing anything besides just saying this an email form input. It says, please indicate an @ symbol in the email address.

Your name  
jordan hudgens

Your name  
Your email

Message

Send

Please include an '@' in the email address. 'jordan hudgens' is missing an '@.'

📍 123 Any St  
Scottsdale, AZ, 85251

📞 555 555 5555

⌚ 10 AM - MIDNIGHT

And so it gives some hints. The only reason we are doing it, is just because HTML gives us some nice validations, just right out of the box with that. So that is our email.

The next item we're going to do is going to be our message. Now messages in text areas, that's whenever you have paragraph content or anything like that, that is called a text area input, but it's technically not a regular input. So we're not going to copy that. I am just going to bring the label down and then we're going to use what is called a text area tag, just like this.

```
95 <div class="form-group">
96   <textarea name="" id="" cols="30" rows="10"></textarea>
97   <label for="email">Your email</label>
98 </div>
```

We're going to provide a name. So we are going to say message, an id here, also message. And then we are going to skip this column and rows, this is where you could hard code the value and the size, but we're going to do that with CSS, so we're not going to even worry about that at all.

And we will though, add a placeholder here, and then you can just say something like, message in the placeholder. And then do the same thing down here, so this is going to be for message and the placeholder is going to be message.

## contact.html

```
<div class="form-group">
  <textarea name="message" id="message" placeholder="Message"></textarea>
  <label for="message">Message</label>
</div>
```

Hit save and that is all of the HTML we need to have this form. So if I hit refresh you can see that we have our form here.



Your name

Your email

Message

**Send**

📍 123 Any St  
Scottsdale, AZ, 85251

📞 555 555 5555

⌚ 10 AM - MIDNIGHT

It looks slightly different than what we're going to be building out.



Your name

Email

Message

**Send**

HTML forms by default are some of the ugliest elements that you will see. So we're going to see how we can customize all of these different values.

Now, before we leave I want to add one little caveat to this, and that is that typically when you're building out a form, you would actually wrap the whole thing in a form tag. And so you'd do something like this, where instead of this div being just a div, you would say this is a form and then it would have things like actions.

And what actions are, is they give you the ability to contact a outside server and the reason why I am not doing that is because it's not needed for this particular application. There is no server that we're contacting, this is just a placeholder form. And depending on what framework you would use to make this active and dynamic, you're going to have to change the way the form works anyway.

So building it isn't even that necessary. So if you're going to use ruby on rails, you're going to have to convert all of this code in to the ERB format, or the slim format. If you're doing this in react, all of this is going to have to get converted in to JSX and don't worry, if you've never even heard of any of those words yet, that doesn't matter I'm just wanting to give you a head's up.

Because you may find some other tutorials and you'll see a form tag here, and you'll see a bunch of other attributes just right here. And if you're wondering what those are, a regular HTML form right out of the box does have those. But because HTML is stateless and you can't do anything with it anyway, it always has to connect to a backend server.

I just want to show you how to implement the design and then later on, if you do want to apply this to a real world application, then that applications framework will dictate how the form needs to be set up. And all of the styles that we're creating here, all of those will carry over perfectly.

So I just wanted to add that one little note, because you may look at HTML forms and say, hey we didn't create a valid form, and no it's because we're not trying to create something here that'll contact the server, because we're not connected to any back end server. The goal of this is to show you how to customize the look and feel, and then however you want to implement it in your real world app, that will be dictated by the framework you're using.



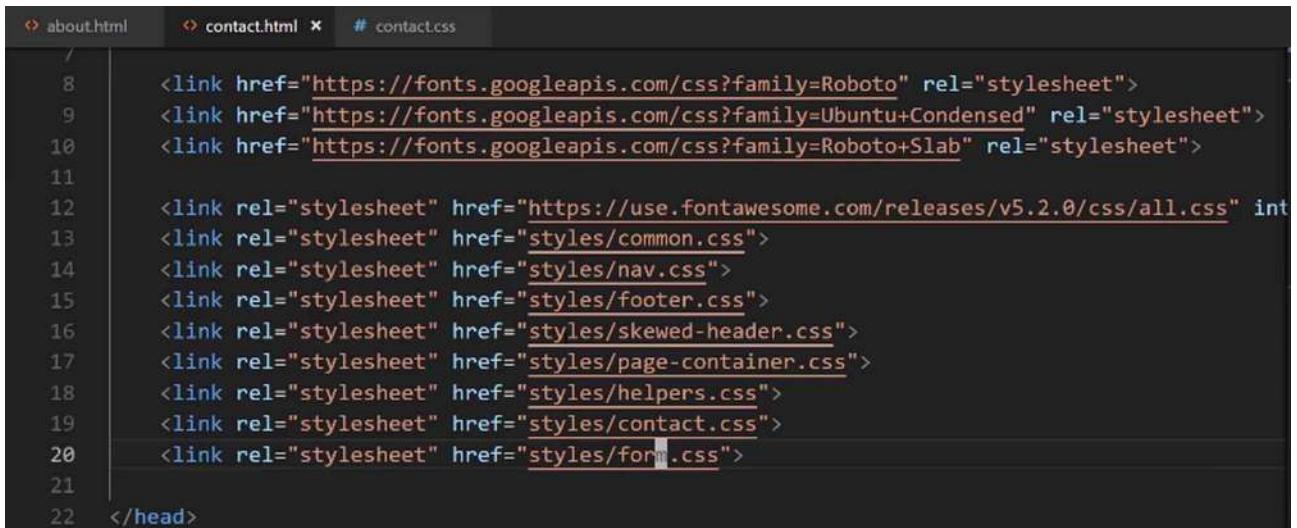
## Coding Exercise

In the below div, place a input tag with the type set to `text`, id of `occupation`, and the placeholder set as `Your occupation`. Below that, place the label for the input and have it say "Your occupation".

```
<div class="form-group">  
<!--place your code here-->  
</div>
```

# 1.50 Text Input styles with CSS

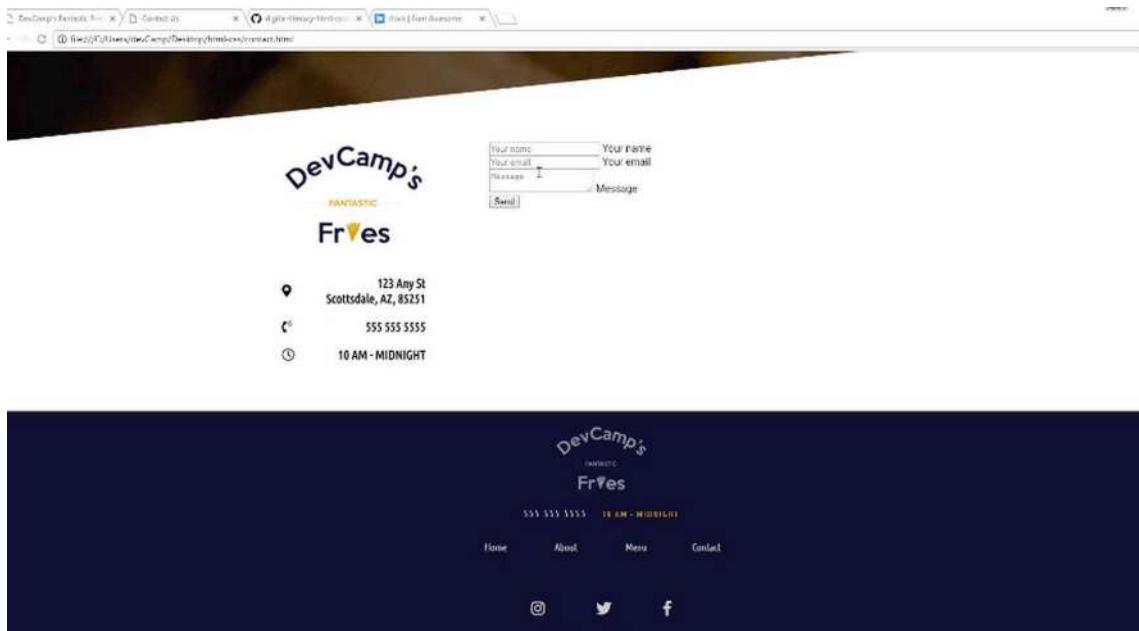
I'm going to go up to the top, and I'm going to import another style sheet. It's one that we need to create. This is going to be just dedicated to forms. This is going to be the form style sheet, and if we open up the sidebar, let's go and create this file. I'm going to say form.css.



```
about.html contact.html # contact.css
8 <link href="https://fonts.googleapis.com/css?family=Roboto" rel="stylesheet">
9 <link href="https://fonts.googleapis.com/css?family=Ubuntu+Condensed" rel="stylesheet">
10 <link href="https://fonts.googleapis.com/css?family=Roboto+Slab" rel="stylesheet">
11
12 <link rel="stylesheet" href="https://use.fontawesome.com/releases/v5.2.0/css/all.css" int
13 <link rel="stylesheet" href="styles/common.css">
14 <link rel="stylesheet" href="styles/nav.css">
15 <link rel="stylesheet" href="styles/footer.css">
16 <link rel="stylesheet" href="styles/skewed-header.css">
17 <link rel="stylesheet" href="styles/page-container.css">
18 <link rel="stylesheet" href="styles/helpers.css">
19 <link rel="stylesheet" href="styles/contact.css">
20 <link rel="stylesheet" href="styles/form.css">
21
22 </head>
```

The reason why I'm putting this in a dedicated form file instead of the contact one is just because the form is something that's usually shared across other pages. You don't want to have to import the entire contact library every time that you want to have a styled form.

Taking a look, and we can close the **about.html** page. If we go down, let's see what we need to select. We have the class of form, and then we have form groups. Let's go back to the finished version and see what we have.

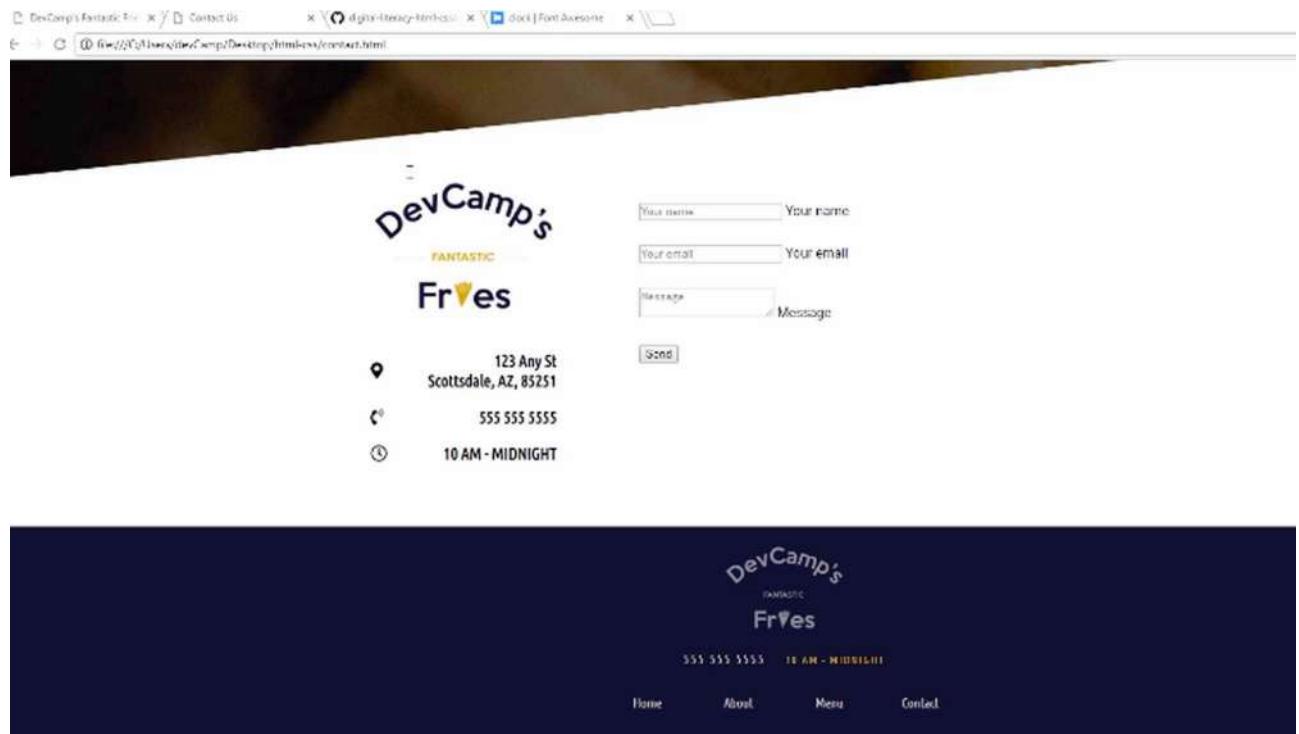


We have these inputs here. To start off, let's just make sure that we have some space in between those. We'll also get to test out our selectors that way. We have a **form-group**. I'm going to say, for a class of form and any **form-group** inside of that is going to need to have some margin on the top. Let's make this 30px, and then we're going to have some margin on the bottom.

## form.css

```
.form > .form-group {  
  margin-top: 30px;  
  margin-bottom: 30px;  
}
```

Switching back, if you hit Refresh, you can see that gives us some nice space. This should be all we need on that side. We also know that our selector is working.



That's kind of a pattern I follow, is anytime I'll add a new file or anything like that, I'll add a very basic test, just like this, just to make sure that everything is working properly.

What I've discovered is if I do not do something like that and then I had a typo or anything related to that, something small, unrelated to the code, then that becomes a lot harder to debug. If I go and I'm coding for 10 or 15 minutes without testing it, then it's going to take me longer to trace back to see where the bug occurred.

When you are writing code, write very small, tiny bits of it, and test it continually, to make sure that everything that you're doing is working. Now, let's go, and we're not going to actually have to write any styles for the form directly. So let's actually keep the form group here at the top.

The next element we're going to do is we're going to grab the `form input`. The form input here is going to have quite a few styles. We'll walk through them one-by-one. We first want to make sure that our `font-size` is pretty big. So I'm going to go `1.5em`.

## form.css

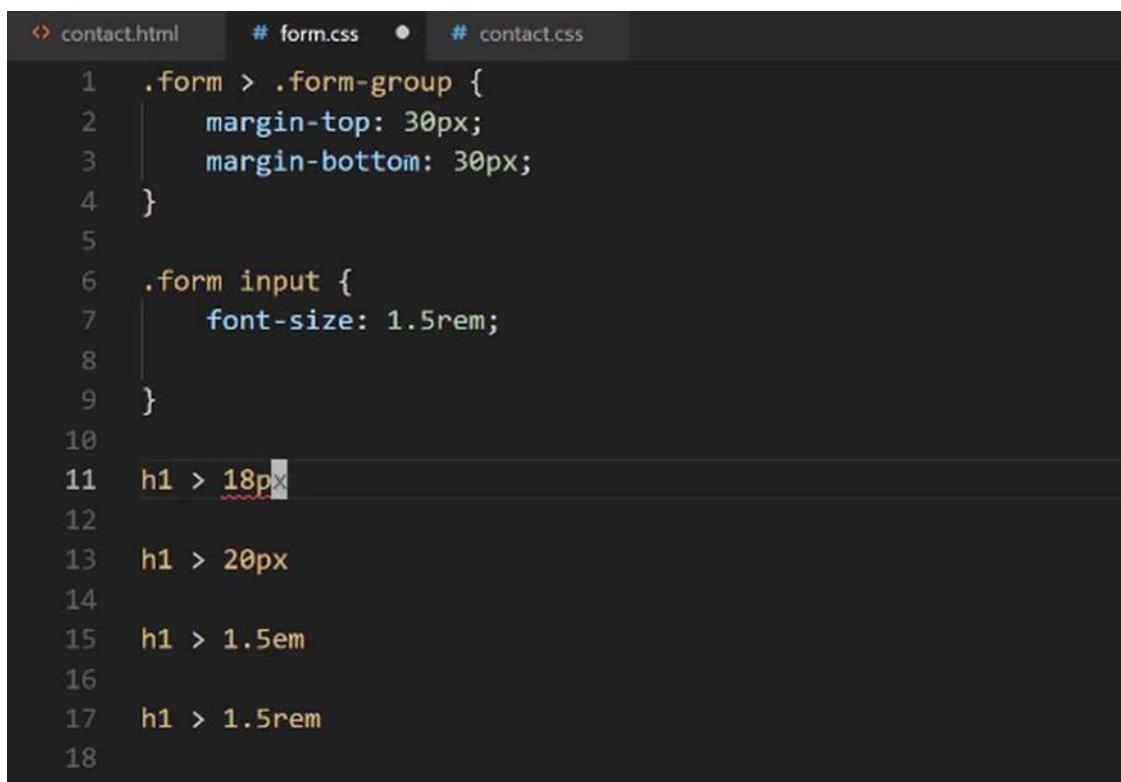
```
.form input {  
    font-size: 1.5em;  
}
```

Actually, let me show you one other size. We have `em`, but we also have another unit of measure called `rem`. What the difference is, usually there's not a big difference, but what `rem` does is it gives you access to the root value, where `em` is a little bit different.

For `em`, If there's some other style definition that precedes it, `em` is simply going to give you a percentage of that. I know that's kind of hard to understand if you've never seen it before. So let's actually kind of walk through it here.

Say that you have an `h1` tag, and you start with it something like `20px`, some size like that. If you have `20px`, that is a custom size you have given. We'll say, at its root, `h1`s have `18px`, just for the sake of argument. You say, "Okay, in my application, all `h1`s should be `20px`."

Well, if you use the `em`, so we say `h1` should be something like `1.5 em`, this `em` is going to be related to this custom value of `h1 > 20px`. This is going to be about `150% of 20px`. If you come down and say, "Okay, well, for this other element, I want to do `1.5 rem`," this is going to go back to the root. That `r` is going to go to the root, and so it's going to be `1.5 on the 18px`.



A screenshot of a code editor showing a file named `contact.html`. The editor has three tabs: `contact.html`, `# form.css`, and `# contact.css`. The `# form.css` tab is active. The code in the editor is:

```
1 .form > .form-group {  
2     margin-top: 30px;  
3     margin-bottom: 30px;  
4 }  
5  
6 .form input {  
7     font-size: 1.5rem;  
8 }  
9  
10 h1 > 18px  
11  
12 h1 > 20px  
13  
14 h1 > 1.5em  
15  
16 h1 > 1.5rem  
17  
18
```

That's what the difference is. If you've never heard of that or if you do see it later on, that's what the difference is. If you need to make sure you're going back to the root value, then use `rem`.

We have our `font-size`, and I'm going to use a `font-family`. I'm actually going to use a value that you may not have seen before. This is going to be called `inherit`. This just means that I want to inherit from whatever the last value was.

### form.css

```
.form input {  
    font-size: 1.5em;  
    font-family: inherit;  
}
```

If that's a little bit confusing, the only reason I'm doing this, I don't think it'd actually make a difference in our situation, but if you're working on an application that has a ton of different forms and maybe they're styled differently, this is going to give you a lot more control. This will make sure that you don't run into a situation where you have a `font-family` overriding your `form input`.

That's all it's doing. It's going to go look at the root input value or font-family, and it's just going to say, "Okay, we're going to use this." Then I'm also going to do the same thing for the color. So I'm going to say `inherit` for this as well.

Then let's add some padding. For padding, I want to do `1.5 rem`, and then let's go with `2 rem`. The same rules apply as the font-size for the padding. In a form input, you have very little padding, by default.

### form.css

```
.form input {  
    font-size: 1.5em;  
    font-family: inherit;  
    color: inherit;  
    padding: 1.5rem 2rem;  
}
```

What I'm saying here is whatever the root padding value is, I want this to be, top and bottom, 150% of that and, left and right, 200% of that, or approximately. That's what we're doing with the padding.

The reason why I'm using these `rems` is not just to teach you, but also, in my own practice, when I'm building out applications, when I'm working with forms, they can be very tricky. You can have all kinds of kind of sneaky overrides that come in, maybe from a framework you've installed or anything like that.

I will always try to make sure that I'm protecting myself and I'm going back to the `root value`, to essentially say I'm very confident that I'm starting from scratch with these values. That's the reason why we're doing that.

I'm going to add a `border-radius` here of `2px`, which we've been using throughout the entire application. Then let's add a `border-bottom`. By default, all HTML forms have a border around all edges. I want to say that the `border-bottom` here is going to be `3px solid`, and it's going to be that nice dark blue color, so `#11122B`.

### form.css

```
.form input {  
    font-size: 1.5em;  
    font-family: inherit;  
    color: inherit;  
    padding: 1.5rem 2rem;  
    border-radius: 2px;  
    border-bottom: 3px solid #11122B;  
}
```

Now, this, by itself, is not going to work perfectly. We also need to, right above this, say that we want a `border: none`. The way CSS works, it's just going to go line by line. It's going to set the `font-family`. It's going to set all those values.

### form.css

```
.form input {  
    font-size: 1.5em;  
    font-family: inherit;  
    color: inherit;  
    padding: 1.5rem 2rem;  
    border-radius: 2px;  
    border: none;  
    border-bottom: 3px solid #11122B;  
}
```

It's going to come to line 12 here and say, "Okay, they don't want a border." Then, on line 13, we'll say, "Okay, they do want a border, but only on the bottom, and put in these values." That's how we're able to create a form that doesn't have those ugly little gray edges. It just has a little line on the bottom.

```
1  .form > .form-group {  
2      margin-top: 30px;  
3      margin-bottom: 30px;  
4  }  
5  
6  .form input {  
7      font-size: 1.5rem;  
8      font-family: inherit;  
9      color: inherit;  
10     padding: 1.5rem 2rem;  
11     border-radius: 2px;  
12     border: none;  
13     border-bottom: 3px solid #11122B;  
14  
15 }  
16
```

Now that we have that, I want to have a `width: 100%`. Then `display: block`. Now, `display block` is the default behavior for divs, but we are declaring it here. This is going to be necessary for us to build that cool animation.

### form.css

```
.form input {  
    font-size: 1.5em;  
    font-family: inherit;  
    color: inherit;  
    padding: 1.5rem 2rem;  
    border-radius: 2px;  
    border: none;  
    border-bottom: 3px solid #11122b;  
    width: 100%;  
    display: block;  
}
```

We're just saying that I want this form input to essentially act like a div, in that sense. From there, I want to say `transition`, and I want to capture `all` transition types because we're going to be performing some different types of transitions. You'll see why I need to do that here in a second, and I want it to take `0.3s`.

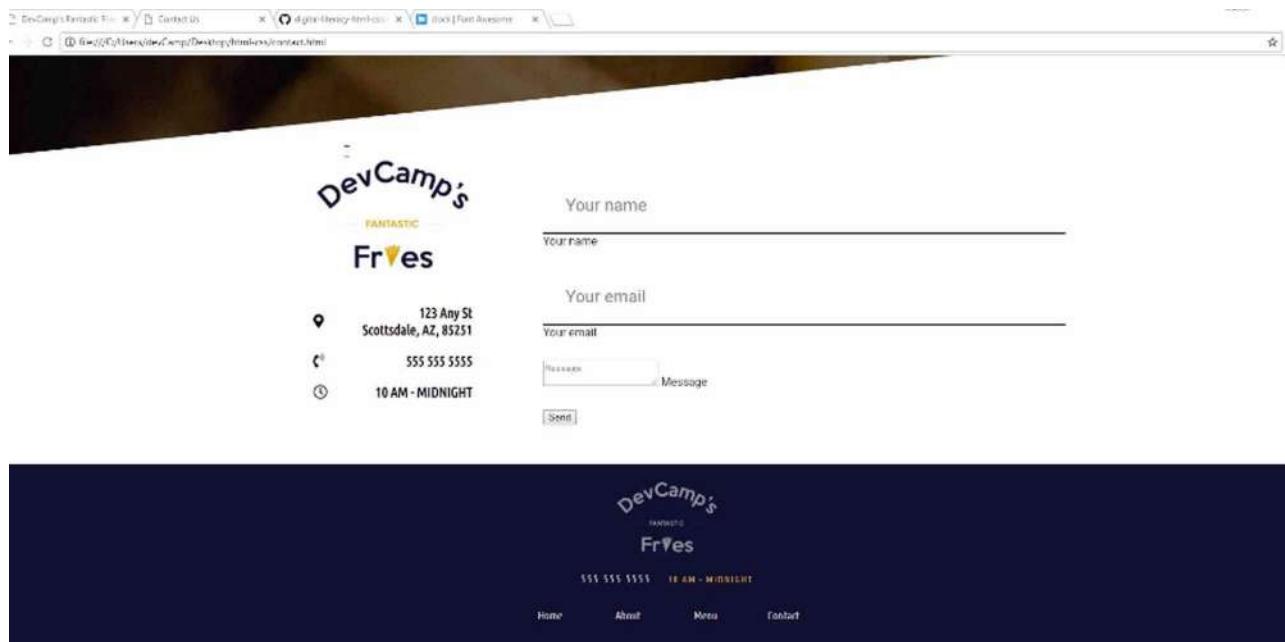
### form.css

```
.form input {  
    font-size: 1.5em;  
    font-family: inherit;  
    color: inherit;  
    padding: 1.5rem 2rem;  
    border-radius: 2px;  
    border: none;  
    border-bottom: 3px solid #11122b;  
    width: 100%;  
    display: block;  
    transition: all 0.3s;  
}
```

These are going to be pretty quick. That was a lot of code, but hopefully, we went line by line, slow enough so that if any of it did not make sense, then hopefully it makes more sense now. If after that, you're still a little unclear after we're done with the video, what my recommendation is is to pause it and not go onto the next one.

Play with these values and see exactly what happens when you change some of them. Switch between `rems` and `ems`, and remove font families and color and inheritance and remove the block and those kinds of things. Hopefully, that'll help to kind of reinforce what we're doing.

Now if I do that, you can see we have much better-looking form elements here. Now, we still have a lot of issues and a lot of things we need to do, but hopefully, this gives you a good feeling for the first few items that we needed to implement.



This is getting a little bit closer to what we have here. In the next guide, we're going to continue on down the line, and we're going to work on the text area.



## Coding Exercise

The below input tag needs a block display and the padding needs to be set to 10px on top and bottom, and 15px to the right and left.

```
<input type="email" placeholder="Your Email" name="email">
```

# 1.51 Form textarea / Button styles

Now that we have some of the base styles for our two text inputs, I think it's natural to move on to putting in styles for the textarea and the button.

Let's start back in our `form.css`. We have the form input, and so what I want to do next is I just want to grab that textarea.

## form.css

```
.form textarea {  
}  
}
```

You may notice that here we did not select the textarea like we normally gone through this, and that's because we don't want just the initial child elements since we'd have to say `.form > .form-group > textarea`.

But here in this situation we just want to have all of the text area tags inside any form.

## form.css

```
.form textarea {  
    margin-top: 20px;  
    width: 100%;  
    height: 250px;  
    border-radius: 3px;  
    border: 3px solid #11122b;  
    font-size: 1.5rem;  
    font-family: inherit;  
    color: inherit;  
    padding: 1.5rem 2rem;  
    transition: all 0.3s;  
}
```

Hit save. Come back. And if we hit refresh you can see we have a much better-looking textarea.



I think this looks a million times better than the default one. So that is coming along quite nicely. So I think the next area that makes sense to work on is going to be the button.

And now we could put the button here in the `form.css`, but my personal preference whenever working with buttons, just based on my experience, is many times buttons are going to be used throughout an entire site. So I think they are worthy of their own dedicated style page.

So I'm going to say `buttons.css`, because usually, you're going to have multiple buttons. We're only going to have one. Then let's go to `contact.html` and let's just make sure that we have imported this.

```
8   <link href="https://fonts.googleapis.com/css?family=Roboto" rel="stylesheet">
9   <link href="https://fonts.googleapis.com/css?family=Ubuntu+Condensed" rel="stylesheet">
10  <link href="https://fonts.googleapis.com/css?family=Roboto+Slab" rel="stylesheet">
11
12  <link rel="stylesheet" href="https://use.fontawesome.com/releases/v5.2.0/
13  <link rel="stylesheet" href="styles/common.css">
14  <link rel="stylesheet" href="styles/nav.css">
15  <link rel="stylesheet" href="styles/footer.css">
16  <link rel="stylesheet" href="styles/skewed-header.css">
17  <link rel="stylesheet" href="styles/page-container.css">
18  <link rel="stylesheet" href="styles/helpers.css">
19  <link rel="stylesheet" href="styles/contact.css">
20  <link rel="stylesheet" href="styles/form.css">
21  <link rel="stylesheet" href="styles/buttons.css">
22
```

There we go. So here we have a button of type submit. Let's add a class to this. Now if we go into buttons, I can add a dedicated button class.

```
contact.html x # form.css # buttons.css # contact.css
93      <input type="email" id="email" placeholder="Your email">
94          <label for="email">Your email</label>
95      </div>
96
97      <div class="form-group">
98          <textarea name="message" id="message" placeholder="Message">
99              <label for="message">Message</label>
100         </div>
101
102         <div class="centered-btn-wrapper">
103             <button type="submit" class="btn">Send</button>
104         </div>
105     </div>
106 </div>
107 </div>
108
109 <div class="spacer60"></div>
110
111 <div class="footer">
112     <div class="logo-footer">
113         
```

## buttons.css

```
.btn {  
background-color: #cea135;  
color: #11122b;  
font-size: 1.25em;  
padding: 10px 25px;  
border: 1px solid #cea135;  
border-radius: 2px;  
outline: none;  
transition: all 0.3s;  
}
```

Now we added a border but you won't actually see the border visually because we're going to use the same exact gold color.

The reason why I'm doing this is that in certain browsers if you do not put a border around the button, certain browsers will automatically do that actually, and it makes the button kind of look ugly.

So the best practice is to either put a different colored border or to simply put a one-pixel border with the same color that you're wanting.

Later on we're going to add a very cool little animation where we add in a fade in and out with a box shadow whenever we click on the button, which is something that will work really well on mobile and on web, and it's something that's pretty popular right now from a design perspective.

Hit save now and let's make sure that worked.



There we go. Now we have a much better-looking button. We've built out the styles for our textarea message, and then for our button. In the next guide, we are going to go back up and we're going to start adding styles for our labels.



## Coding Exercise

Make a text area that has 6 rows and 60 columns, and set the name to "challenge".

# 1.52 Label and Form Element Drop Shadow Styles

Now that we have the base styles for our text inputs, our message text area, and our button all in place, I think it is time to work on the next major item, which is going to be the label.

We're going to work on each one of these labels and then we'll be able to see what those styles need to end up looking like. Let's switch back to the code here and go to the form, and I'm going to go to right below where we have the text area.

The way we can grab this is by saying `.form label`, and then let's update the `font-size` and `font-weight` first. We'll say font-size is going to be `1.2rem`. I want to still use rems, I want to keep that consistent. Then the font-weight, we'll go with `700`. We don't want it quite as thick as `900`.

## form.css

```
.form label {  
    font-size: 1.2rem;  
    font-weight: 700;  
}
```

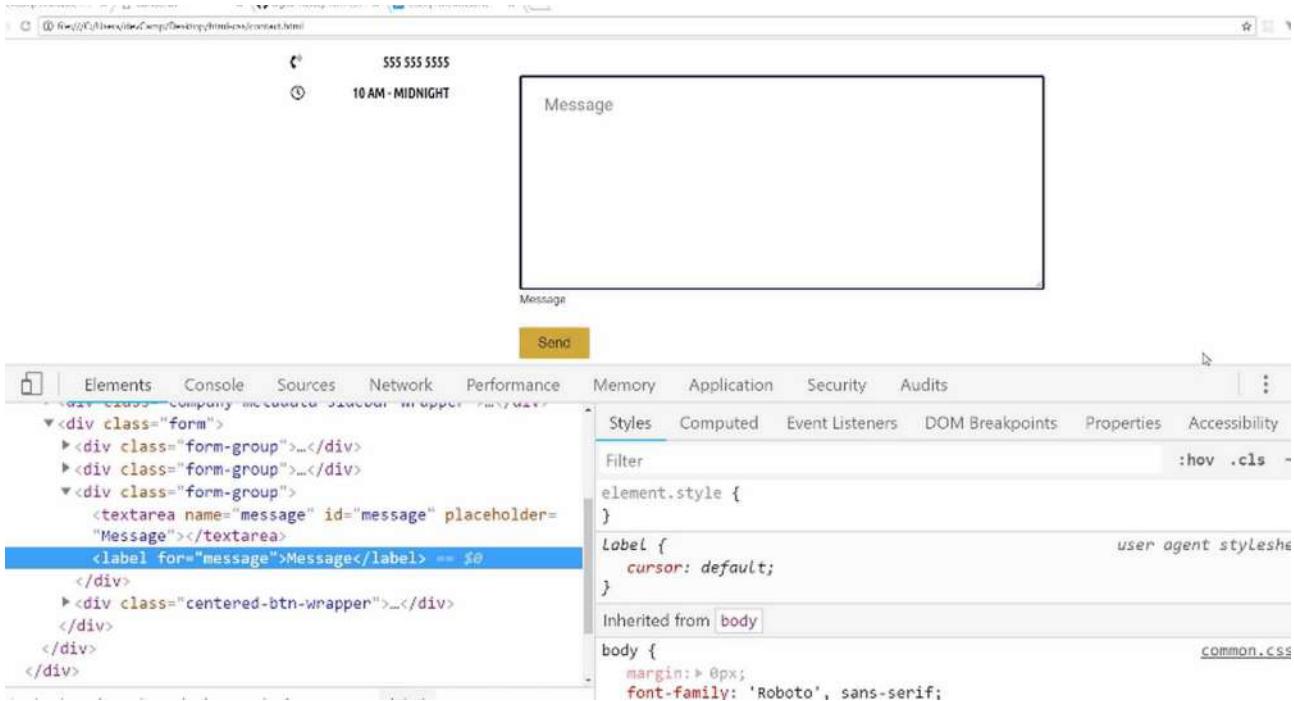
That's one of the nice things with font-weights. You can give them weights anywhere from I want to say `200` or `300` all the way up through `900`. So you kind of have a sliding scale with the level of boldness that you want to give it.

From there, let's give a `margin-left: 2rem` and then a `margin-top: 0.7rem`. Actually, yeah, we want `7` here. These are just some values that when I was building up the prototype, I worked with that until I got it to the point where I was happy with it, so that's where I'm getting those.

## form.css

```
.form label {  
    font-size: 1.2rem;  
    font-weight: 700;  
    margin-left: 2rem;  
    margin-top: 0.7rem;  
}
```

In a normal development environment, what I would probably be doing is what I did when I was building this out. I would be adding these directly in the inspector. That's my normal workflow when I'm wanting to build them. I will select one of the items, so here I would select the label and then, and I would start adding each one of the values there.



I think it's a little bit easier for you to read as I'm going here, especially because I'm talking through them. Now that we have the `margin-top` there, now we want `display: block`. Once again, this is going to give us the ability to treat this element like a div, as opposed to just a normal label.

Then, I want to add `transition: all 0.3s`, which matches all the other animations that we have. We'll add a color here and we'll make this `#CEA135` and then just a couple other items.

### form.css

```
.form label {
    font-size: 1.2rem;
    font-weight: 700;
    margin-left: 2rem;
    margin-top: 0.7rem;
    display: block;
    transition: all 0.3s;
    color: #CEA135;
}
```

We need to work on the `visibility` and then the `opacity`. I'm going to leave these as comments, I'm not going to fill them out yet because we technically want to hide this label first, but before we do that, I want to make sure that it actually matches the styles we're looking for.

Now, if I come here and hit refresh, you can see that our labels are looking great. This is exactly what we're looking for. The only difference is we want to be able to animate them, so we will take care of that later on.

The screenshot shows the DevTools Elements tab with the following details:

- DOM Tree:** Shows the HTML structure of the page, including fields for "Your name", "Your email", and a "Message" input.
- Elements Panel:** Shows the current selection as "`label`" with the class "form".
- Styles Panel:** Shows the CSS styles for the selected element:

```
element.style {  
}  
.form label {  
    font-size: 1.2rem;  
    font-weight: 700;  
    margin-left: 2rem;  
    margin-top: 0.7rem;  
    display: block;  
    transition: all 0.3s;  
    color: #ceal35;  
}
```

I think the next best step is to work on these inputs. Right now, as you can see, when I click on this input, it adds that ugly, little blue outline and I don't want that. I want when you click on the input for you to click on it and no outline except for this cool, little, soft blur.

The screenshot shows the DevTools Elements tab with the following details:

- DOM Tree:** Shows the HTML structure of the page, including fields for "Your name", "Your email", and a "Message" input.
- Elements Panel:** Shows the current selection as "`label`" with the class "form".
- Styles Panel:** Shows the CSS styles for the selected element:

```
element.style {  
}  
.form label {  
    font-size: 1.2rem;  
    font-weight: 700;  
    margin-left: 2rem;  
    margin-top: 0.7rem;  
    display: block;  
    transition: all 0.3s;  
    color: #ceal35;  
}
```

Let's work on that effect. Let's switch over to the code, so for our form, and I want to move up the focus will transition here. I want to put it right below the `form input` because that's just going to be the natural place I would look for it if I ever go back and have to make any changes.

I'm going to say `form input` and then I want to grab the pseudo-state of focus like we've talked about before. Here, I want to remove that ugly outline, so `outline: none` and then I want to add a `border-bottom` and then a `box-shadow`. Let's add that `box-shadow` first.

If you remember back to when we had the full dedicated guide on the box-shadow, we're going to pass in a few different values. If what I'm going to type in is still a little bit fuzzy and doesn't make sense, I recommend for you to go back and watch that video because we went line by line.

We even had that one visual to be able to see what each one of these values represented, I'm just going to type it out here, though. I'm going to say `1rem`, `2rem`, and then `rgba`. There we go. For these values, I'm just going to want to go `0, 0, 0, 0.1...`

## form.css

```
.form input:focus {  
  outline: none;  
  box-shadow: 0 1rem 2rem rgba(0, 0, 0, 0.1);  
}
```

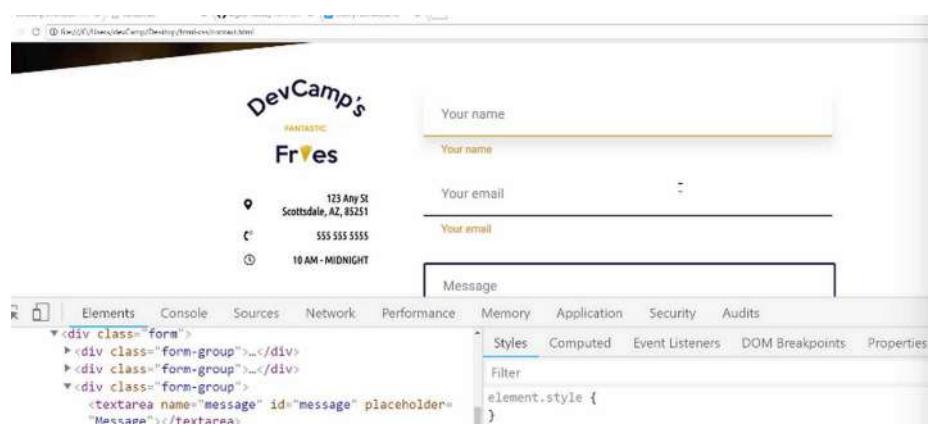
That's going to give us just a really nice, light shadow. Then, I also want to change the `border-bottom` color. Let's change this to be `3px`, which is the same width, solid, and then we're just going to change it to our gold color, hit save.

## form.css

```
.form input:focus {  
  outline: none;  
  box-shadow: 0 1rem 2rem rgba(0, 0, 0, 0.1);  
  border-bottom: 3px solid #cea135;  
}
```

Now if we come back and hit refresh, now we should have that effect. If I click on that, there we go. Look how much better that looks. Kind of hard to see for some of these inputs with the Google auto-complete, this is just something in the browser, that's what's doing that little pop-up. That is occurring the same way.

That looks fantastic. I am loving that. This is one of my favorite forms I built out in a while, so I'm enjoying it. Now that we have that, I think it would make sense to go and just do the exact same thing for the `textarea`.



Now, there are a couple different ways we can do that. One is going to be the `form input`.

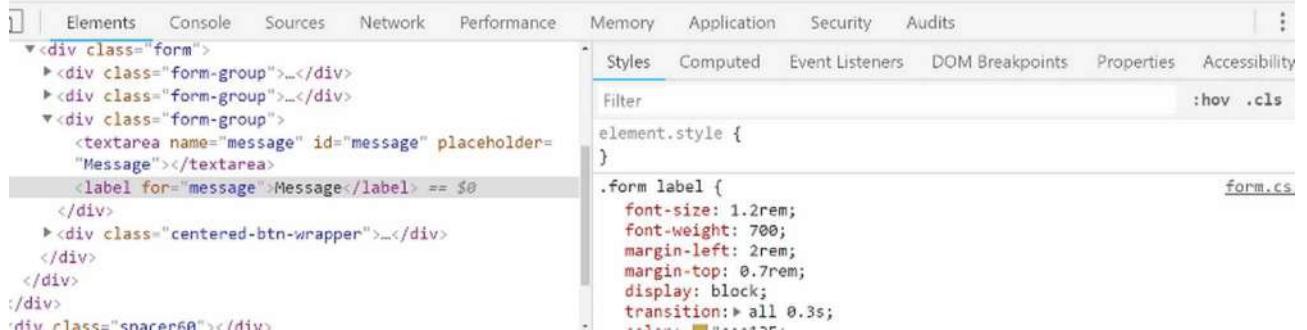
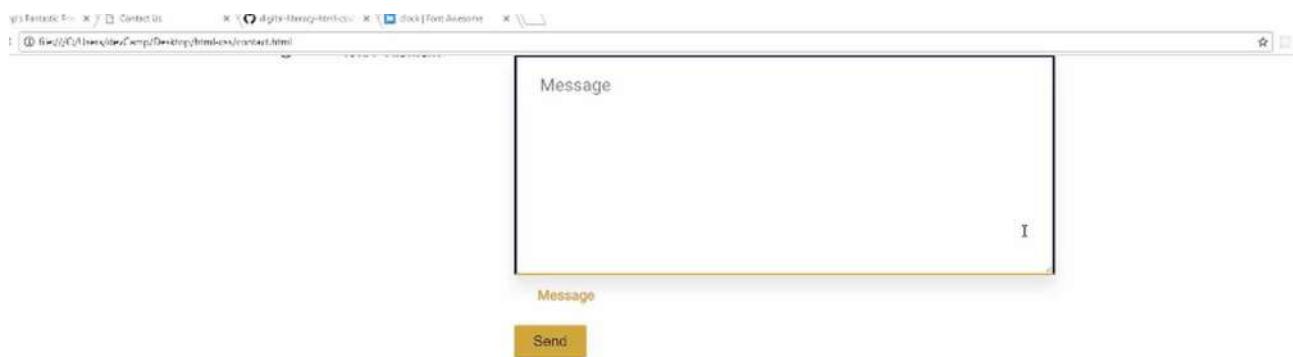
We could just copy and paste this entire thing. I think it makes more sense and is a little bit more efficient to just add a comma here and say we also want this to apply to the form and then `textarea`. We want to grab the text area tag and then that `focus` element as well.

### form.css

```
.form input:focus, .form textarea:focus {  
    outline: none;  
    box-shadow: 0 1rem 2rem rgba(0,0,0,0.1);  
    border-bottom: 3px solid #cea135;  
}
```

If I do this, this should just go and add that. Hit refresh. Now, if I click here, that's working perfectly. Oh, but you know what? I just realized, because I was looking in my notes and I realized that I had duplicated it and it's for a reason, which you may notice.

Do you see how this is adding the gold border at the bottom? I do not want that, so never mind on that. That would have been a cleaner way of doing it, but we wouldn't get the exact behavior we're looking for, so that's fine. It's not a lot of code duplication.



We're just going to paste this in and now, instead of form input, it will be the form text area and the only difference that we're going to have here is we're going to say that we just want the entire border to turn into that gold color.

### form.css

```
.form textarea:focus {  
    outline: none;  
    box-shadow: 0 1rem 2rem rgba(0,0,0,0.1);  
    border: 3px solid #cea135;  
}
```

I think this looks good. Let's go back and take a look. If I hit refresh now, then that's looking really nice.

The screenshot shows the Chrome DevTools Elements tab. At the top, there is a preview window showing a yellow-bordered input field with the placeholder "Message". Below the preview, the DOM tree shows the following structure:

```
<div></div>
<div></div>
<div>
  <input id="message" placeholder="Message" type="text">
  <div></div>
</div>
```

The input field has the ID "message" and the placeholder "Message". The styling for the input field is defined in "form.css:4" with the following CSS:

```
.form label {
  font-size: 1.2rem;
  font-weight: 700;
  margin-left: 2rem;
  margin-top: 0.7rem;
  display: block;
  transition: all 0.3s;
  color: #ceai35;}
```

Now, I might want to change one of these values now that I'm actually looking at it, so if I come here and let's click on **hover** and then, **focus**. That's going to give us our nice shadow. For this **box-shadow**, let me see what happens if I change the second value to **2**. That looks a little bit better.

The screenshot shows the Chrome DevTools Elements tab. The input field now has a box shadow, appearing as a yellow-bordered box with a slight drop shadow below it. The styling for the input field is now:

```
.form label {
  font-size: 1.2rem;
  font-weight: 700;
  margin-left: 2rem;
  margin-top: 0.7rem;
  display: block;
  transition: all 0.3s;
  color: #ceai35;
  box-shadow: 0 2px 2px rgba(0, 0, 0, 0.1);}
```

In the DevTools sidebar, under "Force element state", the ":focus" checkbox is checked, and the "outline" and "border" properties are also checked.

It's completely up to you on what you want to do. This is still your application at the end of the day, so you can have it at **1** or **2**. I think I'm going to change mine to **2**, just because I think that because it's such a large region. That gives it a little bit more visual depth. Now, if we try this again, that is working.

DevCamp's  
FANTASTIC  
**Fries**

Your name

---

Your name

---

123 Any St  
Scottsdale, AZ, 85251

555 555 5555

10 AM - MIDNIGHT

Message

Message

Send

Very nice job. If you went through this, you now know how to create some pretty impressive looking box-shadows and animations inside of your HTML forms.



## Coding Exercise

The below input tag needs the outline disabled and a custom border of 2px with a solid blue color.

```
<div class="parent">
  <div class="container">
    <input type="text" name="phone" placeholder="Your Email">
  </div>
</div>
```

# 1.53 Custom Form Placeholder Text Styles

Right here, we're increasing the font, but we're not really stylizing the placeholder the way that we'd like to. You can see the difference between how our placeholder text looks in the design.

The screenshot shows a web page with a dark header featuring the text "DevCamp's FANTASTIC Fryes". Below the header is a sidebar with icons and text: a location pin with "123 Any Street Scottsdale, AZ, 85251", a credit card icon with "555 555 5555", and a clock icon with "10 AM - MIDNIGHT". To the right of the sidebar is a form area. It contains three input fields with placeholder text: "Your name" (black font), "Email" (black font), and "Message" (black font). A yellow "Send" button is located at the bottom right of the form.

Compared to how we have it styled in our project.

This screenshot shows the same web page and sidebar as the previous one. However, the form fields now feature placeholder text in a bold, orange-yellow color. The "Your name" field has "Your name" in the placeholder, and the "Email" field has "Your email" in the placeholder. The "Message" text area also has "Message" in the placeholder. The yellow "Send" button remains at the bottom right.

You see how the design has that dark blue color for the text area and the input? We're going to style our forms to match the design.

Let's switch to the code, and I'm going to come down below the text area and the input because this is going to be shared. This is going to be one where we're going to be able to update the input placeholder for the text area and the input all at the same time.

### form.css

```
.form input::-webkit-input-placeholder, .form textarea::-webkit-input-placeholder {  
}  
}
```

What this is going to do is this is going to give us the ability to stylize that placeholder value.

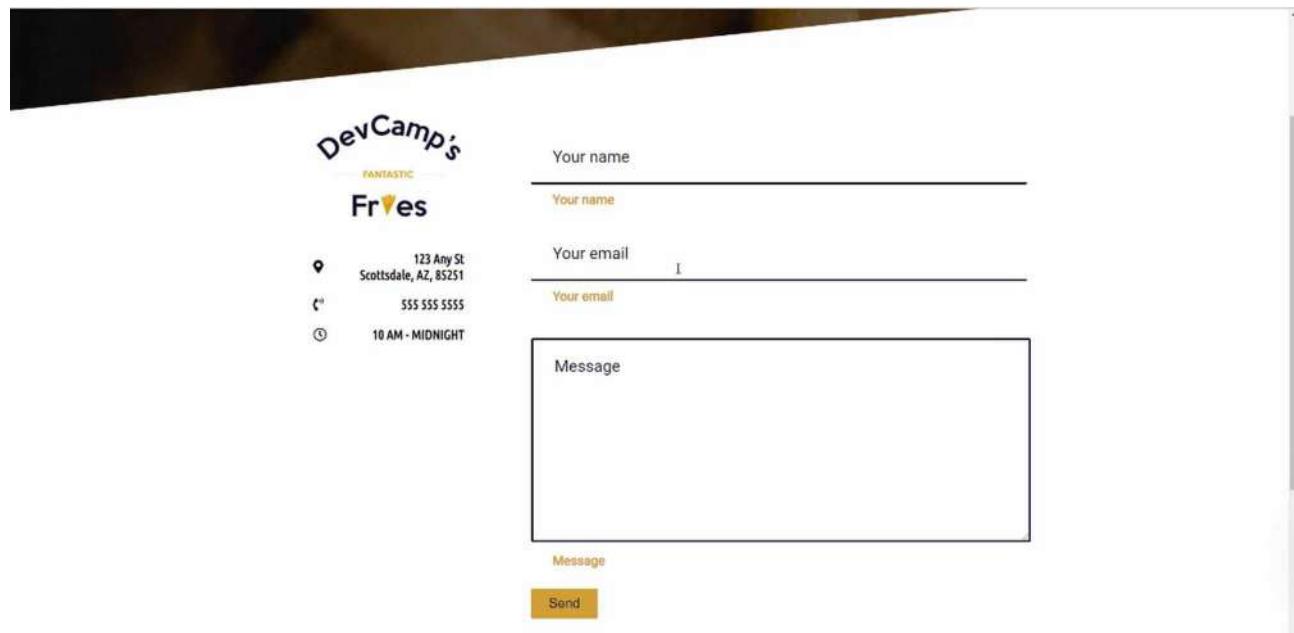
This is only available in certain browsers, so if you do this and you're on Firefox, this will most likely not work, or if you're on, say, Microsoft Edge right now, that's not available, but it is available on Chrome and Safari. This should work for the majority of people accessing the website.

Let's add in our styles.

### form.css

```
.form input::-webkit-input-placeholder, .form textarea::-webkit-input-placeholder {  
    color: #11122b;  
}
```

If you hit Save and refresh, you can see that that worked perfectly.



I am really loving the way all of this is coming along.

We just have a few more things that we are going to work on, especially this label animation that we're going to be building.

I think that that is definitely worth its own video because that's going to be probably one of the more challenging topics that we're covered throughout this entire course. We're going to take a break now and come back and we'll build that out.



## Coding Exercise

Target the input tag without using the ::webkit selector and make the text color pink, and the font size 15px. Also, try it out in your favorite text editor or on [CodePen!](#)

```
<input type="email" name="email" placeholder="Your Email" id="email1">
```

# 1.54 Form Labels Animations

Okay, it is time for the moment of truth here. We are going to build out the full animation so that when we click on any of the form inputs, the placeholder is going to go away, and then our little label is going to slide down.

Now, it works as far as the animation on the box shadow, but we are currently showing the label. We want to hide that by default, and then we want to show it, and reveal it, make it look like it's sliding down whenever the user clicks on the input or the text area.

Let's start building this out. It actually isn't going to take a lot of code, but it is going to cover a few topics we have not covered up to this point. We're going to take it nice, and slow. Let's come down here, and let's first update the visibility, and opacity values here.

## form.css

```
.form label {  
    font-size: 1.2rem;  
    font-weight: 700;  
    margin-left: 2rem;  
    margin-top: 0.7rem;  
    display: block;  
    transition: all 0.3s;  
    color: #cea135;  
    visibility: hidden;  
    opacity: 0;  
}
```

What these styles are going to do is they're just going to make it look like our label is invisible, if you hit refresh, the labels are now hidden.

The screenshot shows a web page for "DevCamp's Fries". On the left side, there is a sidebar with the DevCamp's logo, a "FANTASTIC" badge, and a "Fries" badge. Below these are icons for location ("123 Any St, Scottsdale, AZ, 85251"), phone ("555 5555 5555"), and operating hours ("10 AM - MIDNIGHT"). The main content area contains a contact form. The "Your name" field is a simple text input. The "Your email" field is another text input. Below these is a large text area labeled "Message". At the bottom right of the form is a yellow "Send" button. The overall design is clean and modern.

Now that we have that, let's take care of the form input first.

### form.css

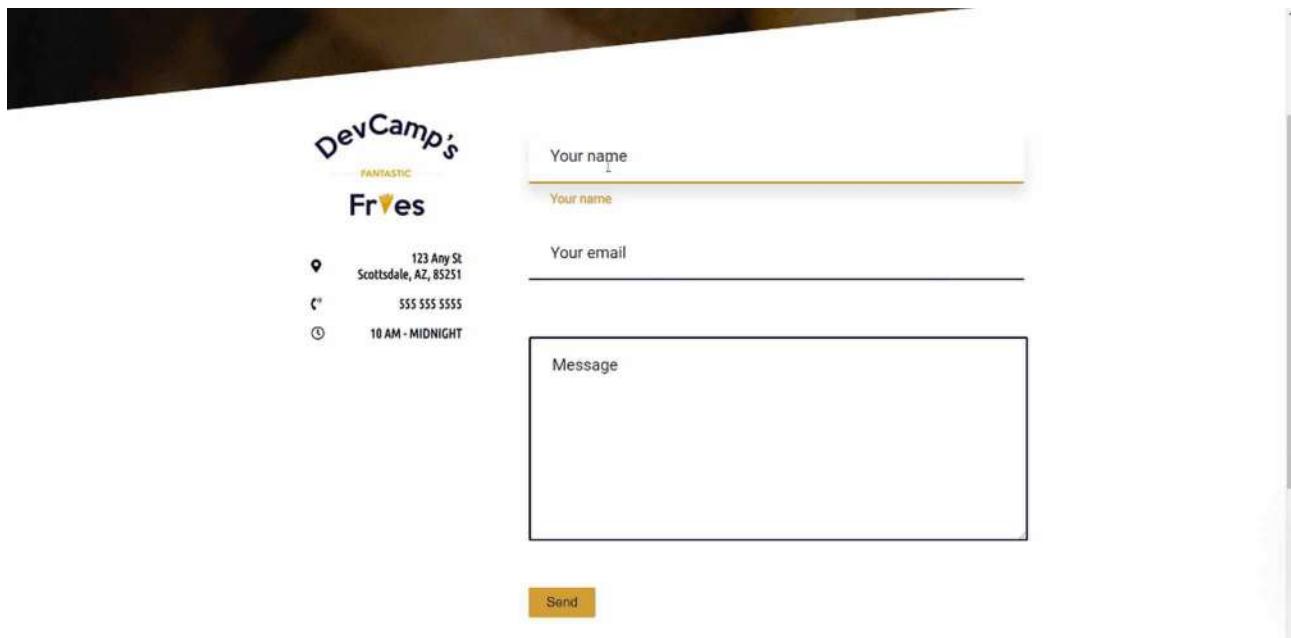
```
.form input:focus {  
}
```

I'm going to show you another selector that we haven't used before, which is the plus sign. This is going to go, and grab the very first value of label, and then from here, it is going to go and apply whatever type of behavior we want. So, it's going to apply these styles.

### form.css

```
.form input:focus + label {  
    opacity: 1;  
    visibility: visible;  
}
```

Let's just see if that works before we even add the rest of the animation.



If I click on your name, you can see that that pops up. So, that is working nicely. This is good. Now let's build in that animation.

In order to do that we are going to use a tool that is called transform. We've worked with transform before, but in a different way, because here we are looking to transform, and translate a value as opposed to just using transform to perform a skew.

### form.css

```
.form input:focus + label {  
    opacity: 1;  
    visibility: visible;  
    transform: translateY(1rem);  
}
```

What that means is essentially it's going to take the entire value, the entire label, and it is going to move it down the Y-axis, which means it's just going to slide it down one full value.

If it is five pixels, or ten pixels high, it's going to start at the very top of where it would be, and it's going to slide it down by one rem or one value. So, it's just going to essentially take it exactly to where it's at right now, but it's going to move it all the way up to the top.

It's going to make it seem that the placeholder is actually sliding down, changing colors, and is moving. That's all that we're doing. So, let's hit refresh, and now if you click here, you can see that that is working perfectly.

The form consists of several input fields and a text area:

- A text input field labeled "Your name" with a placeholder "Your name" in orange.
- A text input field labeled "Your email" with a placeholder "Your name" in orange.
- A large text area labeled "Message".
- A yellow "Send" button at the bottom.

This is exactly what we want it to do. So, this is pretty neat. I'm really liking this, and it works on the email, so that is awesome.

Hopefully, that wasn't too challenging. What I want to do now is apply this to the text area. So, we can just add a comma.

### form.css

```
.form input:focus + label, .form textarea:focus + label {  
  opacity: 1;  
  visibility: visible;  
  transform: translateY(1rem);  
}
```

I click on it, and that looks like it's working. We need some space here, but the animation itself is working perfectly.

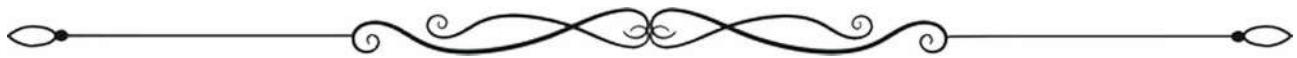
Now, I just want to test it out. I'll type in a name, that's looking good, an email, that all looks really nice, and then for the message that is all perfect.



I do not see any changes that need to be made on this whatsoever. So, congratulations if you went through that. You were able to leverage the tools of transform and translate in order to create a really dynamic form experience.

It makes it feel like the user is interacting directly with the form, and it's changing based on their behavior. So, great job.

In the next guide we're going to polish up a few items, and then you're going to be completely done with this entire project, very nice work.



## Coding Exercise

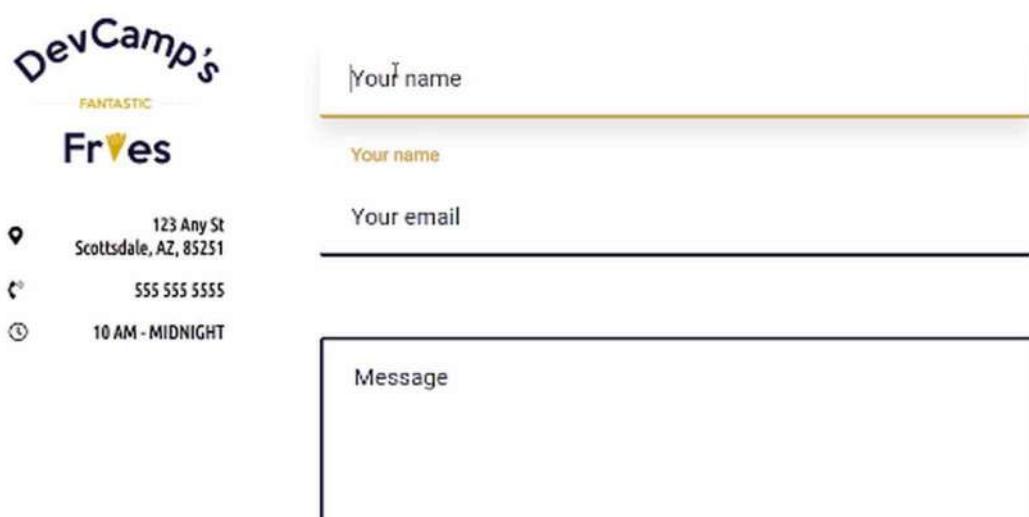
Make the below span tag's text disappear with the visibility and opacity properties!

```
<span>I've always wanted to be part of a magic trick!</span>
```

# 1.55 Finalizing Contact Page Styles

In this video, we are going to finalize the styles on the contact page. I've seen three items that I'd like to clean up.

The first one is that, if you notice, whenever you click on one of these elements do you notice how we have duplicate labels? We have the placeholder here and then we have the label moving down and I'm not really a fan of that. I'd rather have the entire placeholder disappear whenever this `focus` event occurs, so that's one item.



Then I also want to move the send button in the center and move it down a little bit, and then each one of these icons should be our gold color. Let's address each one of those items and we'll be done with the desktop version of the application.

Right here I'm going to come all the way down to the bottom, and let me take a quick note to just make sure that I'm not duplicating any selectors. This is what we're wanting to update, this `form input::-webkit-input-placeholder`, but we want to capture the `focus` area.

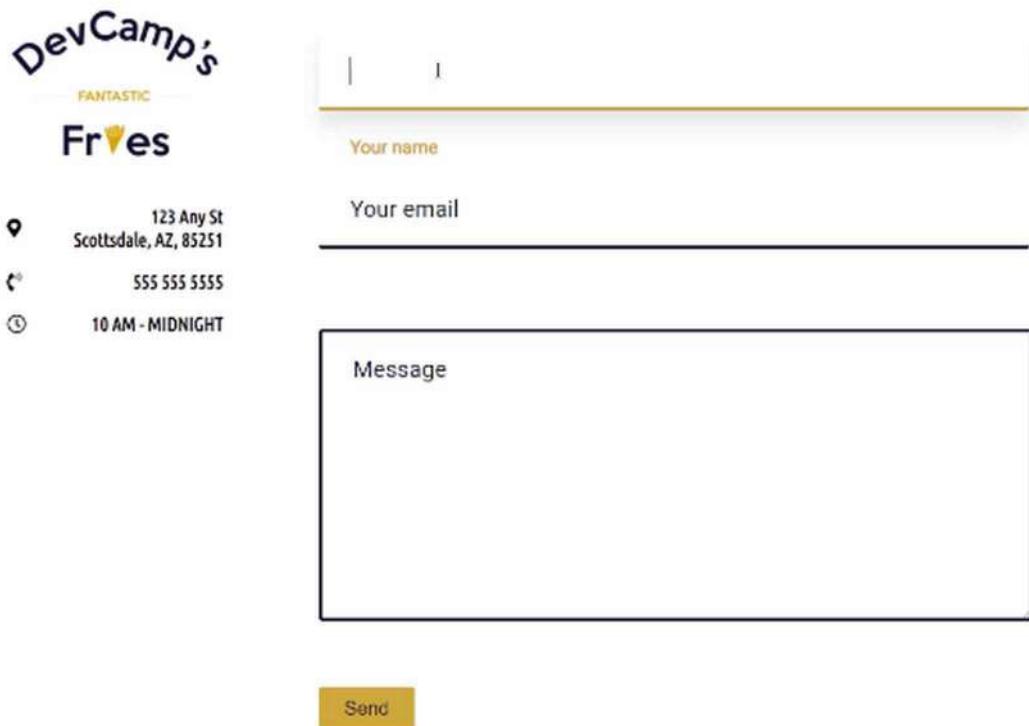
Let's just copy this and now we're going to say `input` but then we're also going to just capture it when the actual focus event is occurring. The way that you can do that is you can actually chain on your selectors just like this, so it can say `input: focus`, and then `textarea: focus` as well.

## form.css

```
.form input: focus:: -webkit-input-placeholder, .form textarea: focus:: -webkit-input-placeholder {  
    color: transparent;  
}
```

Then what I want to do is I can just make this essentially disappear by making the color `transparent`. I'm going to make the placeholder color transparent and now if I come and refresh

if you put on the name you can see that that disappears and that's working. Same thing with email, same thing with message.



This is working very nicely, and all we had to do was update the **pseudo-selector** so that the placeholder was listening for that focus event and changing the color. That looks really nice. Moving down, let's also now update this send button so it gets moved down here.

We want to add our spacer which, remember, in our helper file we created a spacer, so we can do that. I also want to center it. We'll use Flexbox for that. Coming to our buttons, we could either put this in **buttons.css** or in our **helpers.css**.

I think it makes sense to put it in **buttons.css** because this is going to be related purely to how buttons behave. Let's go and find this class name if I scroll down here. It's **center-btn-wrapper**, that is the name of the class. So we want to say **.center-btn-wrapper**.

### buttons.css

```
.center-btn-wrapper {  
  display: flex;  
  justify-content: center;  
}
```

This is going to be just a basic flex container so we've created a lot of those throughout this course. We're going to say **display: flex**. From there, we'll just say **justify-content: center**. Hit save, come back, hit refresh. It looks like we have a little issue, let's see what's going on.

DevCamp's

FANTASTIC

Fries

📍 123 Any St  
Scottsdale, AZ, 85251  
⌚ 555 555 5555  
🕒 10 AM - MIDNIGHT

Your name

Your email

Message



Send

Let me just double check my spelling, `center-btn-wrapper`. Oh centered, there we go, I kind of like `center-btn-wrapper`. That could be the issue and there we go, that's working. That is looking nice.

📍 123 Any St  
Scottsdale, AZ, 85251  
⌚ 555 555 5555  
🕒 10 AM - MIDNIGHT

Your email

Message

Send

DevCamp's

I forgot one other thing. We have to add the cool little `box-shadow` whenever we press on this so we'll do that. We also need to move it down just a little bit, so let's just add a really small spacer. If you open up your `helpers.css` you can see we have a `spacer 60`.

Let's just add an even smaller one. We'll add a spacer here, we'll just call this `.spacer10`, that should give us what we need. For this, it's just going to have a `height` of `10px`.

## helpers.css

```
.spacer10 {  
    height: 10px;  
    width: 100%;  
}
```

```
95     </div>  
96  
97     <div class="form-group">  
98         <textarea name="message" id="message" placeholder="Message"></textarea>  
99         <label for="message">Message</label>  
100    </div>  
101    <br>  
102    <div class="spacer10"></div>  
103  
104    <div class="center-btn-wrapper">  
105        <button type="submit" class="btn">Send</button>  
106    </div>  
107    </div>  
108    </div>  
109 </div>
```

Now we can call that directly in the view. Then we'll just say `.spacer10`.

Hit save, refresh, and that just gave a little bit more space. I like having just a little bit more room there. Before we finish out this box-shadow, let's also take care of these icons. If you come back here we'll find out where we need to select, so that's going to be in the `contact.css` file.

It's in the `company-details-wrapper` and then we can just select the icon. That should be relatively straightforward. We have the `company-details-wrapper` just like this. I'm going to copy that and then we can just grab that `i` tag. I'll just grab the `i`, and then say our color is going to be the `#CEA135`.

## contact.css

```
.company-metadata-sidebar-wrapper > .company-details-wrapper i {  
    color: #cea135;  
}
```

Hit save, and now if you hit refresh those icons are properly set up. All of that is looking really good.



Your name

---

Your email

---

- 123 Any St  
Scottsdale, AZ, 85251
- 555 555 5555
- 10 AM - MIDNIGHT

Message

Send

The very final piece to what we need to implement is going to be this button right here. Notice if I click on it nothing really happens? I'd like to update the status of it and add that box-shadow whenever that happens.

The way we can do that is let's go into the **buttons.css** class here. Right under **btn**, I'm going to use a button pseudo-selector called **active**. What this means is that a user has clicked on the button, or if they're on a phone, or something, they have pressed on the button.

#### **buttons.css**

```
.btn:active {  
}
```

We just want to add a box-shadow and let's add 5px, 5px, 30px, and then rgba, and we're going to go with our same 0, 0, 0 and for our transparency, we'll say 0.4. Hit save now, hit refresh, and let's see if this works when you click on the button.

#### **buttons.css**

```
.btn:active {  
  box-shadow: 5px 5px 30px rgba(0, 0, 0, 0.4);  
}
```

That is gorgeous, I love that. When you click here you can see that it almost looks like we are having a true feeling of pressing on the button because it adds that depth. I really like that.

555 555 5555  
10 AM - MIDNIGHT

Message

Send

We have completed the entire desktop version of this entire site. Fantastic job in going through all of this. We have the about page, the home page, the menu, we've seen how to use all kinds of tools, and then we finished it up with the form.

The last thing that we need to implement in the next section, is going to be how we can make this entire website **responsive**. What that means is that we're going to be able to make it so that if someone accesses the site on a phone or a tablet, it's going to automatically respond and readjust the content.

We're going to learn about tools like **media queries** and we're going to be able to update column orders and all kinds of cool steps that you can take. Now in the modern time, if you're building a website you really have to make it responsive.

There are many times where you're actually going to be getting even more traffic from a smart phone than you might be getting from the web. A restaurant website's a great example of this.

If the website gets most of its clicks off of a site like **Yelp**, most of those users are on their phones. Making sure that this works in a responsive manner is definitely going to be a critical skill to have. I will see you in the next and final section.



## Coding Exercise

Space apart the letters in the below card by 2px. Also have the color set to grey.

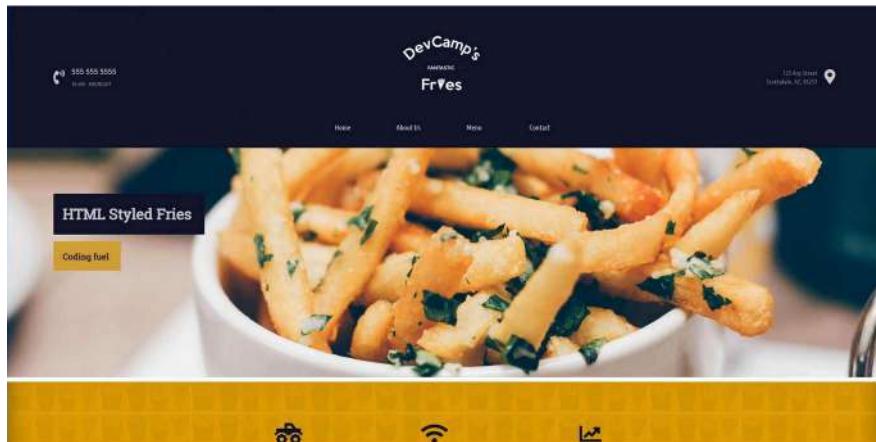
```
<div class="card">
  <p>This is the cards content!</p>
</div>
```

# 1.56 CSS Media Queries

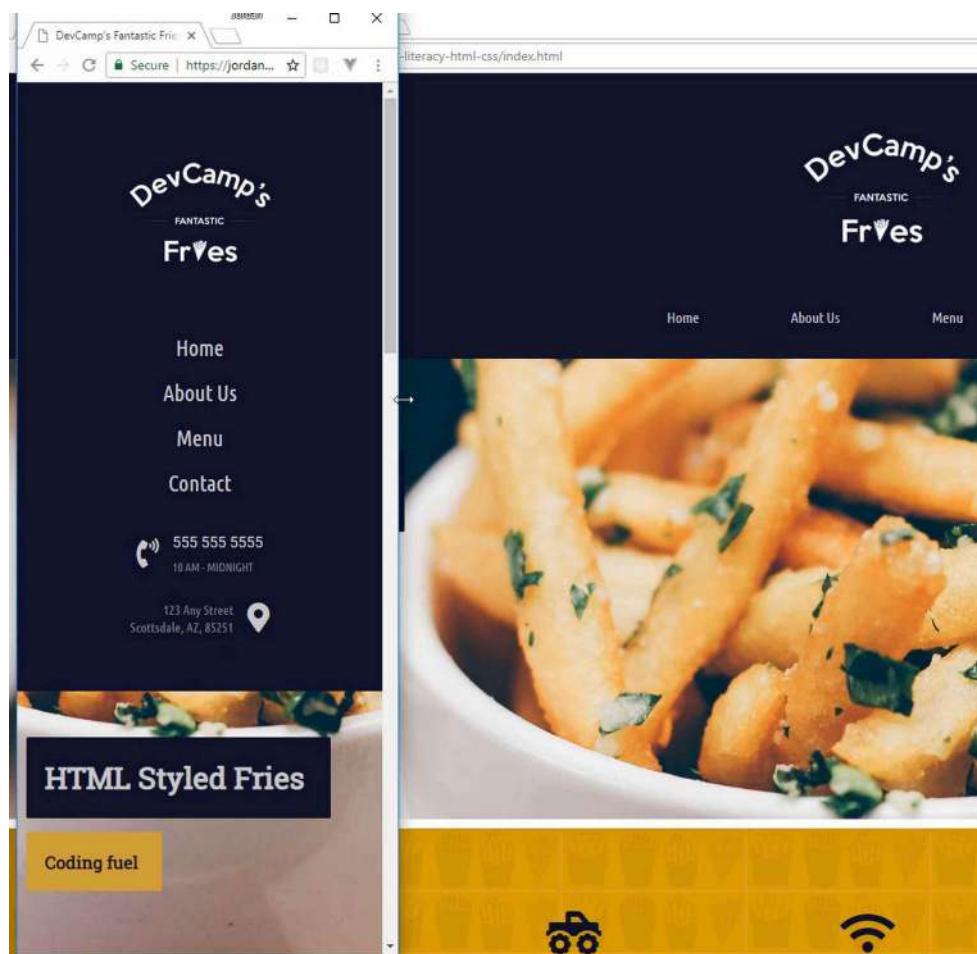
Let's talk about media queries. And if you've never heard of what a media query is, that's perfectly fine. We're going to start from the ground up.

A media query is a tool in CSS that allows for us to implement responsive design elements. When I say responsive, what I mean is what I'm going to show you right here. I have two nearly identical sites, one is responsive and uses media queries, and the other one is not.

After this demo, we're going to walk through the code and we're going to implement a full media query for this homepage.



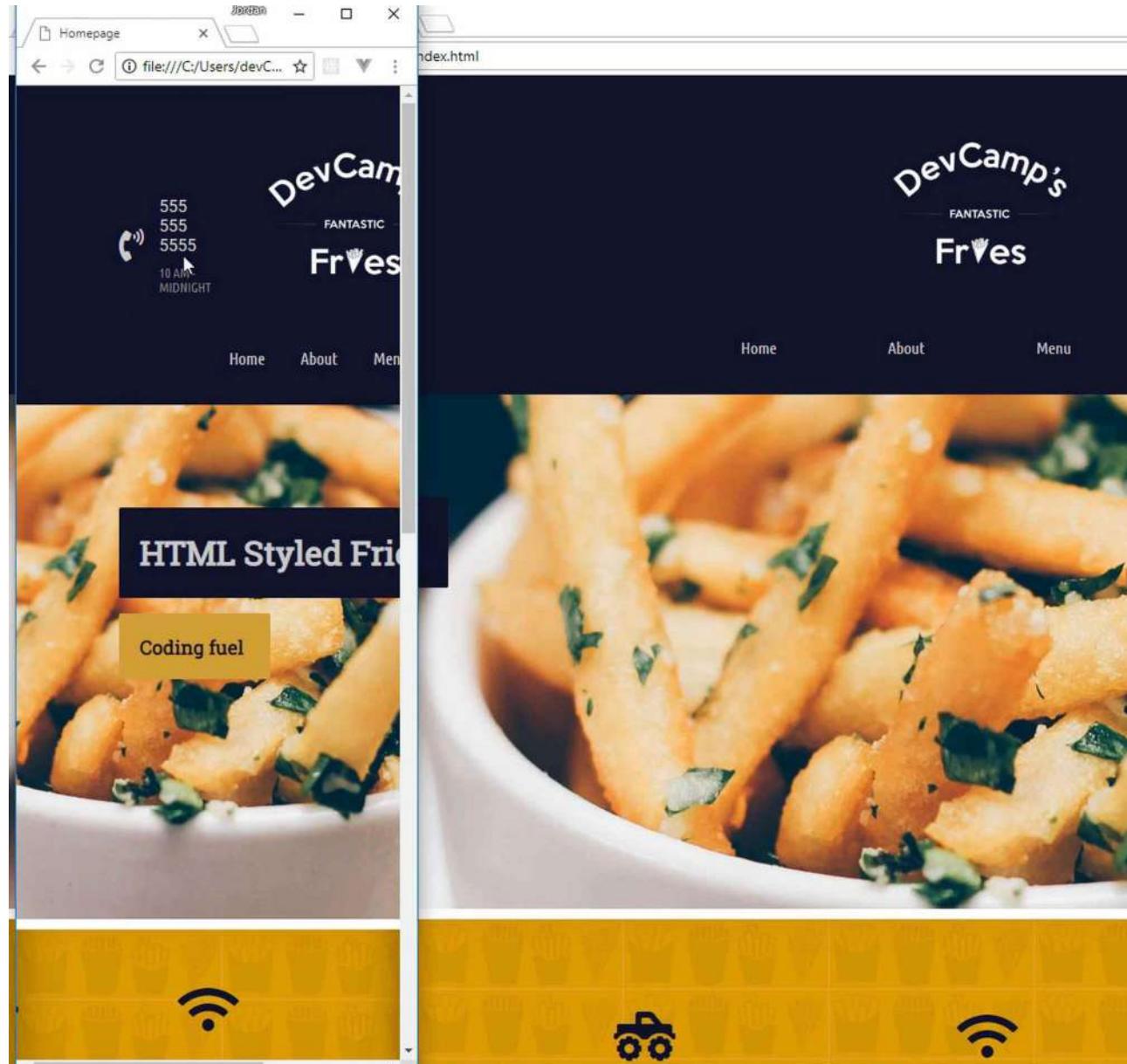
Let's first look at a media query based site. If I open this up, what I can do is it looks good on desktop and then if I were to access this on a mobile device, which I can mimic by just bringing this down here, you can see that the entire site readjusts.



This is what someone coming on an Android or an iPhone would see. Do you notice how the logo has readjusted, it's now at the top, and then we have the navigation elements stacked on top of each other, and then the same thing with the contact information?

Then everything else has also readjusted so this is looking really good, this is the kind of experience you'd want to see on a mobile device. Now, if you open up a site that does not have the media query though, and then try to perform the same action, you're going to get very different behavior.

Notice now, if I take this down to the same size, that it does not readjust.



So someone accessing the site's going to load it up and they're going to see this distorted looking page. They're going to have to scroll to the right, a bunch of things are kind of shrunken and overlapping, and this is really not a good experience. This is the reason why media queries are so powerful.

Now that we have a good idea on what they are, let's walk through how we can implement them. I have this page open and I have the code open for it right here. So I'm going to start by creating a

new style sheet here called media queries. And this isn't necessary, this is just because I want to be able to have all my media queries in one spot. It will also be easy for us to see them.

Let me open that up and in the styles directory, I'm going to save a file called `media-queries.css`. Then here, we're going to be able to add all of those. Now, the syntax for using a media query is ... it looks a little bit different if you've never used it before. It starts with an `@` symbol, then you say media, then this is a method or a function so that means it takes in arguments.

The very first thing that we have to provide is the breakpoint. So if I say medium max-width and then I'm going to use 615 pixels.

### media-queries.css

```
@media (max-width: 615px) {  
}
```

What this is going to do is this is going to say that whenever we have a screen that is below 615 pixels wide, which is a pretty standard size for using with smartphone devices, then I want you to apply these styles.

Now, whenever you're using media queries, a trick to make sure that you are following is your media queries should be at the very end, they should be the last styles that you include. The reason for that is because if you have media queries and then you call other styles after that, they will override it.

And so you need to make sure you call that at the very end. And what this is going to do is the browser's going to look, it's going to see all of these media queries, and it's going to check and say, "Okay, these are the styles I am going to apply, and I'm going to run these if the screen is 615 pixels or less." And you could do this for any of them, so let's just test this out.

If you look at the `index.html` here in the navigation wrapper, you can see that we have navigation wrapper and then we have all of these items inside of it. We have a left column, a center column div, and then a right column. I think this is going to be a great place to start.

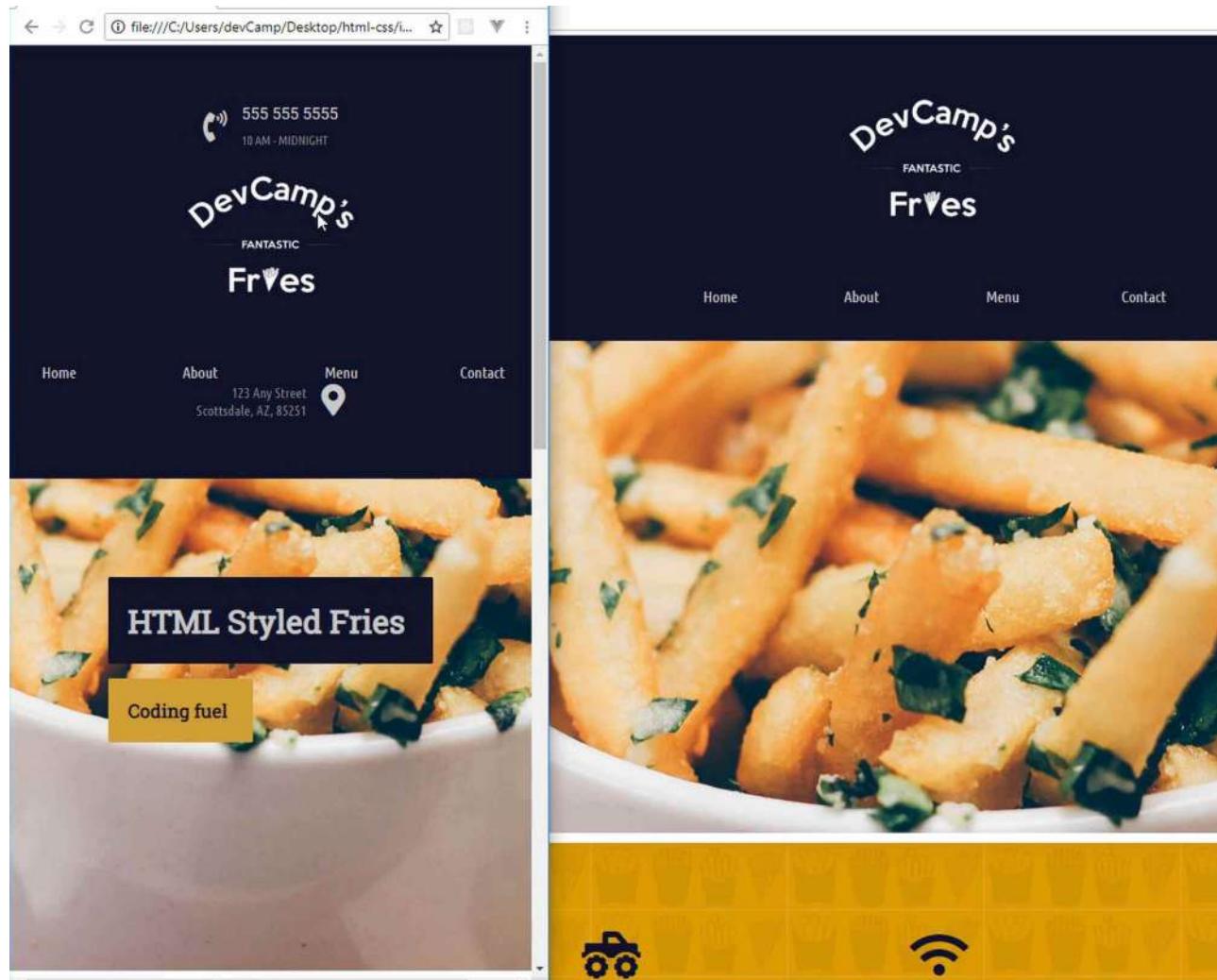
So if I say navigation-wrapper, then inside of here, I'm going to change the flex direction. Instead of having the default row, I'm going to change it to column, then I also want to update the height so it's 100%.

### media-queries.css

```
@media (max-width: 615px) {  
    .navigation-wrapper {  
        flex-direction: column;  
        height: 100%;  
    }  
}
```

Let's see what this does for us on our site that previously was not working on mobile.

Let me open this up in a new browser window. Now, if I shrink this down, you can see that when we hit that breakpoint, you can see this gets readjusted.



Before, that was not working. Before, it just simply went to the side and then it didn't shrink down and it didn't stack it on top of each other like that. So that means that our media query is firing and it's working. Now, we still have some work to do, because this doesn't look very nice, we still have to adjust some of the other items.

I don't want the phone number here at the top, I want to have the logo at the top and we're going to be able to leverage Flexbox in order to make that possible. So I'm just going to take this down to about right here, about this size. Now, when we switch back, we're just going to be able to look at that and I'm also going to keep the code open at the same time, so that we can save some time and just look at it simultaneously.

The screenshot shows a code editor with two tabs: 'index.html' and '# media-queries.css'. The CSS file contains the following code:

```
1 @media (max-width: 615px) {  
2     .navigation-wrapper {  
3         flex-direction: column;  
4         height: 100%;  
5     }  
6 }
```

To the right of the editor is a browser window displaying a mobile version of a website for 'DevCamp's Fries'. The site features a dark header with a phone number (555 555 5555), a logo, and navigation links for 'About' and 'Menu'. Below the header is a large image of fries in a container with the text 'HTML Styled Fries' overlaid. A small button labeled 'Coding fuel' is visible at the bottom of the image.

Okay, now that we have our navigation wrapper updated, I want to talk about a very powerful tool inside of Flexbox that you may or may not be aware of. So Flexbox gives you the ability, not simply to line out and align your items in a really nice and easy manner, it also gives you the ability to change the order.

And so that is how we are going to readjust the order that these elements are shown. This is one of my favorite parts about Flexbox. So right here, if you remember we have our left column, our center column, and then our right column. So what I can do is say navigation wrapper, then I want to grab the left column, just like this. Then inside of here, I want to change the order to two.

And that's going to allow us to change the default order, because usually the order is simply the order that it is declared in the HTML. But what we can do is we can actually override that using Flexbox. and I'm also going to add some margin top and some margin bottom, so let me do 10 pixels on the top and then margin bottom, I'm going to go with, let's say 15 pixels and that's all we need the left column.

### media-queries.css

```
.navigation-wrapper > .left-column {  
    order: 2;  
    margin-top: 10px;  
    margin-bottom: 15px;  
}
```

Now, for the center column, remember the center column is the logo. If you want to take a look at it, this is the final site, but this also is what we started with. So here you can see that the left column is the phone number and the hours, the center column is the nav component and logo, then the third column is this address here, and that's what we're looking to adjust.



I want to take the center column and I'm going to change the order here so that the order's going to be one, then I also want to change the width. The width here is going to be 100%.

### media-queries.css

```
.navigation-wrapper > .center-column {
  order: 1;
  width: 100%;
}
```

Then lastly, we want to grab the right column and, as you may have guessed, since we used two on the left column, for the right column, we want to use three here. Then I'm going to also just add some margin on the top, so here I'm going to say 15 pixels.

### media-queries.css

```
.navigation-wrapper > .right-column {
  order: 3;
  margin-top: 15px;
}
```

Now, this isn't going to be perfect yet, because we still have those links that we need to fix, but this might get us a little bit closer. So let's see, hit refresh and you can see that is looking much better.

```
@media (max-width: 615px) {
  .navigation-wrapper {
    flex-direction: column;
    height: 100%;
  }

  .navigation-wrapper > .left-column {
    order: 2;
    margin-top: 10px;
    margin-bottom: 15px;
  }

  .navigation-wrapper > .center-column {
    order: 1;
    width: 100%;
  }

  .navigation-wrapper > .right-column {
    order: 3;
    margin-top: 15px;
  }
}
```

Do you see how we have this logo now that is at the very top? Our images, or I'm sorry, our links here, these need to get fixed, that's what's pushing everything over. But we can

see that everything now is in the correct order, and so that is looking really nice. As you can see, all we have to do is because we leverage Flexbox, we're able to alter that order and control exactly how we wanted the page to be rearranged.

So now let's switch over to the `index.html` page. And let's see, we have a class of links wrapper and then a nav link inside of each of those. So that's going to be the next class that we're going to add to our media query here.

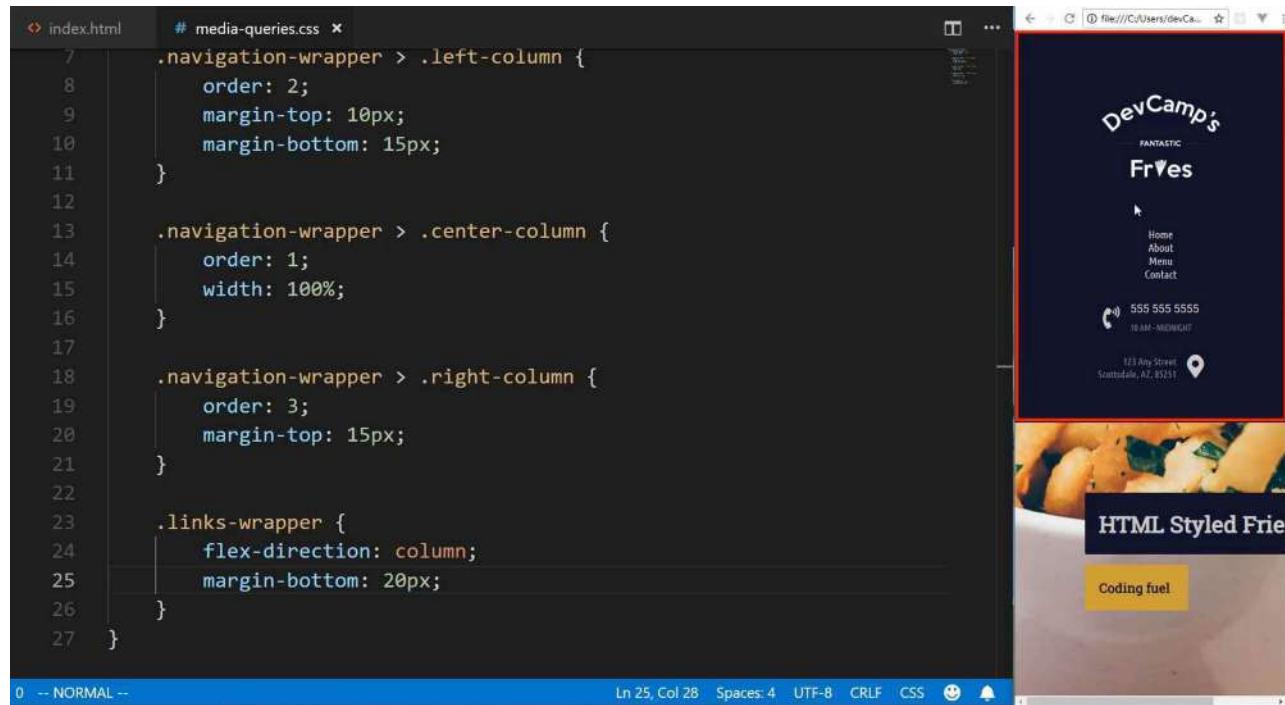
Now, what we can do is we'll say links wrapper, and then here inside of it, if you're using Flexbox and you want to change the order so it's no longer showing from left to right, but instead it goes from top to bottom, you can simply change the flex direction.

We're going to change that to column and then I'm also going to add some margin to the bottom. And for that, we'll go with 20 pixels, then let's also update the nav link, but before we do that, let's just see where we're at.

### media-queries.css

```
.links-wrapper {  
    flex-direction: column;  
    margin-bottom: 20px;  
}
```

If I hit refresh that is looking much better.



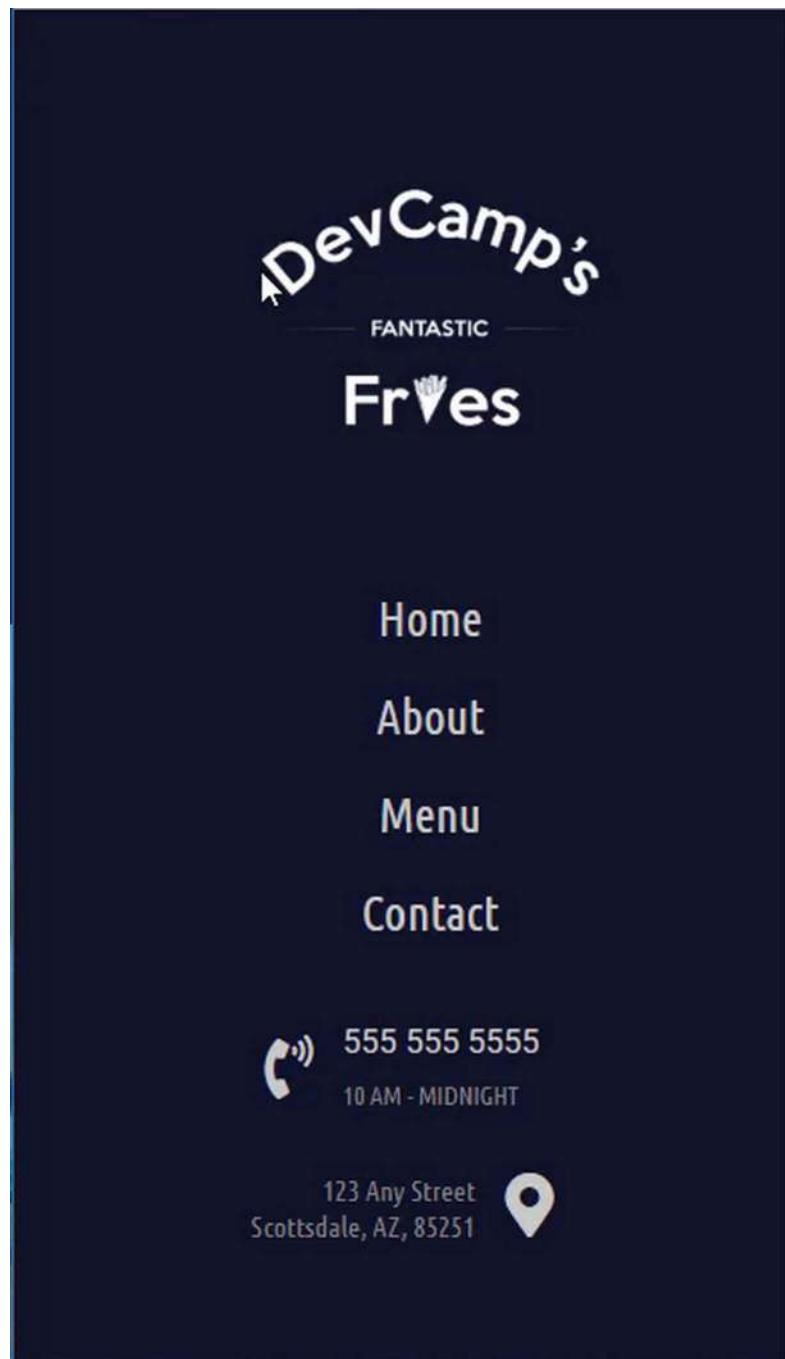
The screenshot shows a code editor with two tabs: "index.html" and "# media-queries.css". The "# media-queries.css" tab contains the following CSS code:

```
7 .navigation-wrapper > .left-column {  
8     order: 2;  
9     margin-top: 10px;  
10    margin-bottom: 15px;  
11 }  
12  
13 .navigation-wrapper > .center-column {  
14     order: 1;  
15     width: 100%;  
16 }  
17  
18 .navigation-wrapper > .right-column {  
19     order: 3;  
20     margin-top: 15px;  
21 }  
22  
23 .links-wrapper {  
24     flex-direction: column;  
25     margin-bottom: 20px;  
26 }  
27 }
```

Below the code editor is a browser window displaying a dark-themed website for "DevCamp's Fries". The navigation bar includes links for Home, About, Menu, and Contact. Below the navigation is a phone icon with the number 555 555 5555 and the text "10 AM - MIDNIGHT". A location pin indicates the address "123 Any Street, Scottsdale, AZ, 85251". The main content area features a large image of fries and the text "HTML Styled Fries" and "Coding fuel".

See, that this is exactly what we're looking to do. So that is how we're able to leverage media queries to be able to adjust dynamically how the page is laid out, so I'm really liking that. Now, let's go to our links wrapper, we're going to select all of the nav links, so the nav link class.

And inside of here, I'm going to just add ... They're stuck a little bit close together. That would be hard for someone on mobile to click the right button. Let's add some margin to the top and bottom. Then from there, we're going to update the font size. Let's go with 1.5em, then lastly, we're going to adjust the width to be 100%.



## media-queries.css

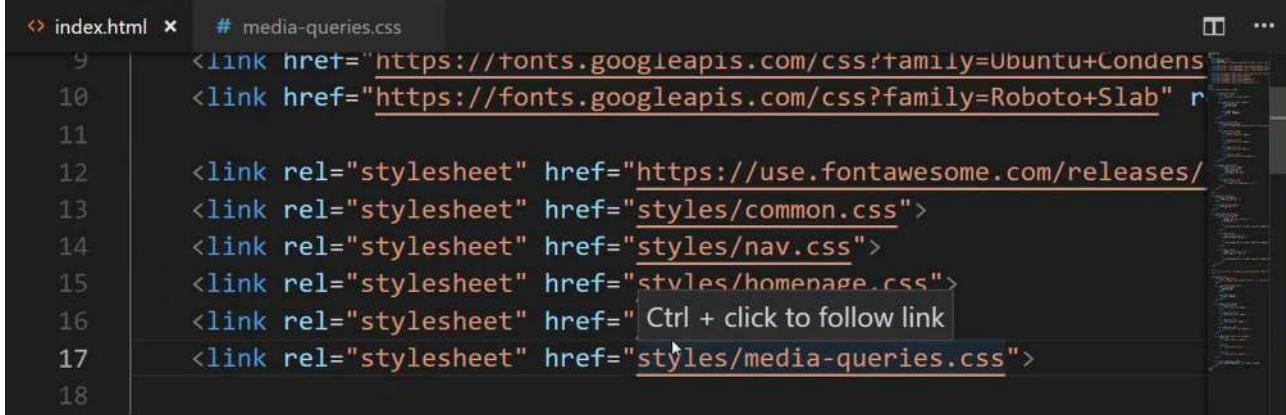
```
.links-wrapper > .nav-link {  
  margin-top: 10px;  
  margin-bottom: 10px;  
  font-size: 1.5em;  
  width: 100%;  
}
```

Now, if you hit save, come back, hit refresh, there you go. Now, what we've done is we've completely changed the layout of this nav bar so that it matches exactly what we're wanting on a mobile device.

Imagine you are building out this website and you're building it for a restaurant, just like we have here, and someone clicks on the link from Yelp. They are going to want to see something that looks like what we have here, not a distorted site where they don't even know where to click or anything like that. They want to have something that really fits with that mobile user experience, and we're able to leverage media queries in order to do that.

Let's just review, really quick, what we've done. The syntax, once again, for a media query is using the @ symbol, the word `media`, then defining the width. So you can have multiple media queries, you could have a media query that is for smartphones. Then you can adjust and have a media query that is for iPads or tablets. You could have as many media queries as you want, and then the page would dynamically change.

So this is almost like a conditional, you can think of this as being a way that you can tell the browser that these are different rules that you want it to follow whenever the user is on a different screen size. Then from there, you simply have to call that in the HTML, just like we did right up here and it's going to work and it's going to readjust.



A screenshot of a code editor showing the `index.html` file. The file contains the following code:

```
<link href="https://fonts.googleapis.com/css?family=Ubuntu+Condensed">
<link href="https://fonts.googleapis.com/css?family=Roboto+Slab" rel="stylesheet">
<link rel="stylesheet" href="https://use.fontawesome.com/releases/">
<link rel="stylesheet" href="styles/common.css">
<link rel="stylesheet" href="styles/nav.css">
<link rel="stylesheet" href="styles/hompage.css">
<link rel="stylesheet" href="styles/media-queries.css">
```

The line `<link rel="stylesheet" href="styles/media-queries.css">` has a tooltip "Ctrl + click to follow link". The code editor interface shows tabs for `index.html` and `# media-queries.css`, and a sidebar with various icons.

Then lastly, using tools like CSS Grid and Flexbox give you the ability to control how you want the layout to change. If you weren't using a tool like that, it'd be much harder to change the order, like we have right here. But because you have those kinds of tools in place, you can simply readjust the entire page based on the screen size.



## Coding Exercise

We need some space around the below div. We need 23px to the left and right but no space on top or bottom. We also need the position property of the div to be fixed.

```
<div id="contact-us-card">
  <p>Lorem ipsum</p>
</div>
```

# 1.57 Responsive Styles

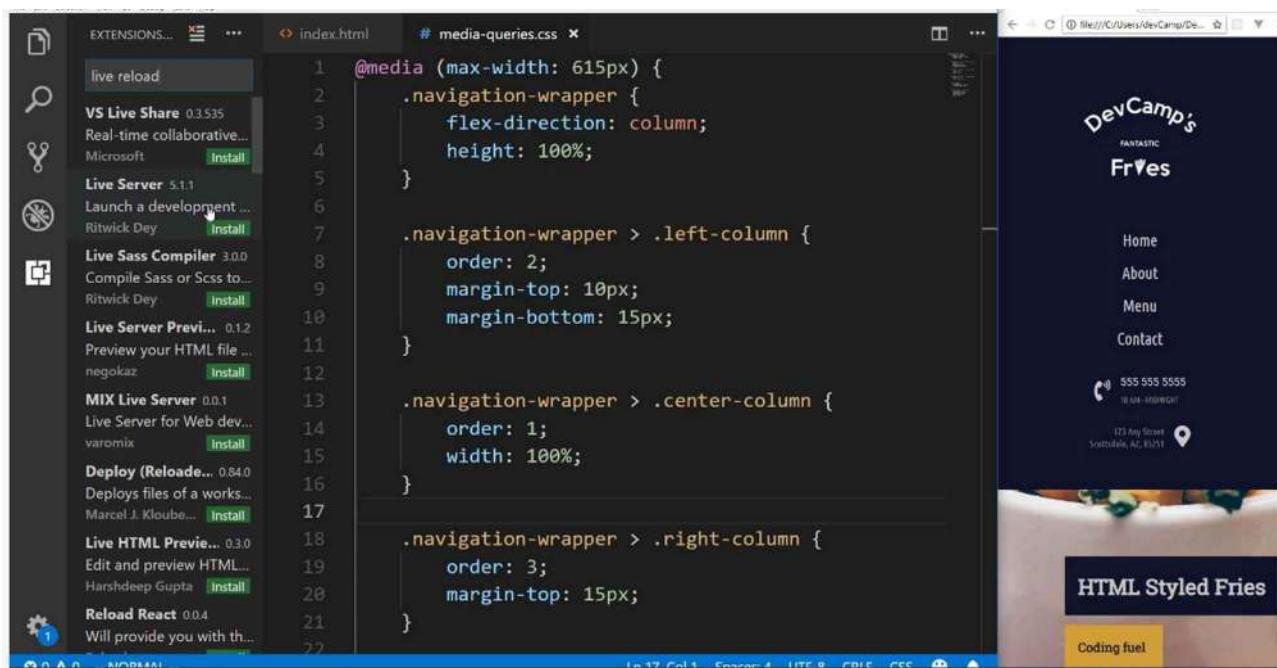
In this guide, we are going to continue adding media query styles and finish up the styles for our Homepage.

The way that I like to think about this, if media queries are still a little bit foreign or hard to understand, is I like to think about media query as almost as if they are a site within your site.

So think about these styles as being styles that you would build if you had to design a page that would render at certain screen sizes. That's the way that media queries really made sense to me.

Now, I also want to show you a helpful little tool if you are using Visual Studio Code, and it's going to help automate our process.

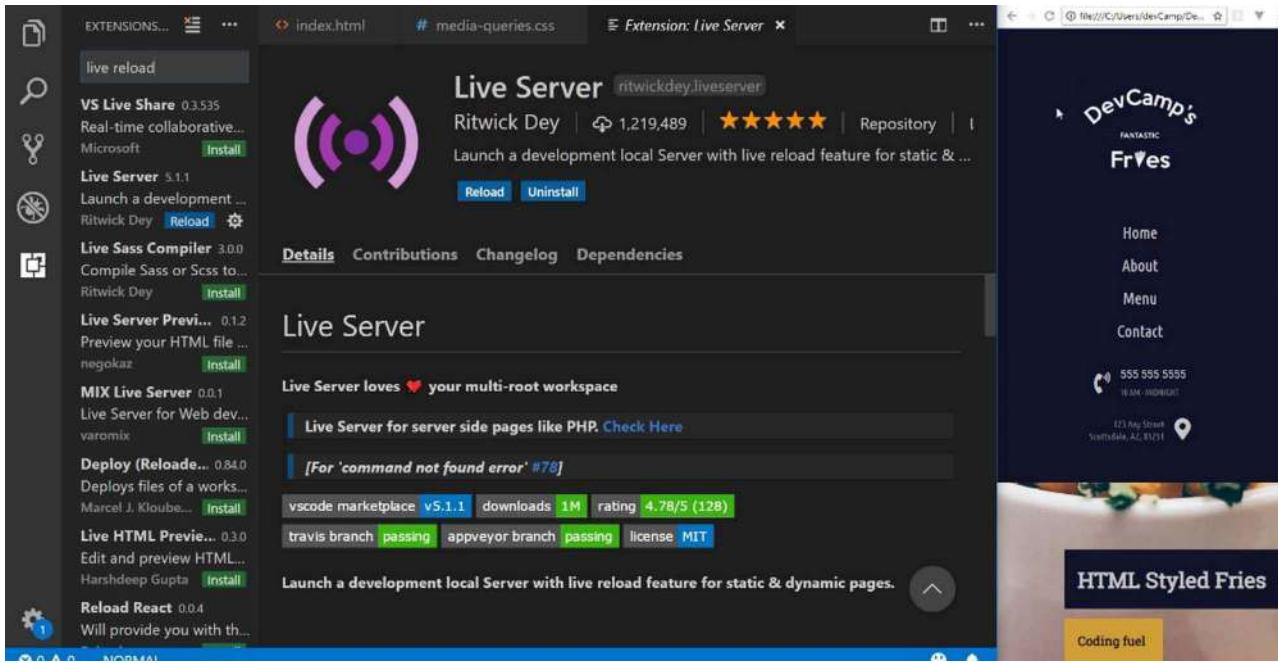
Up until this point, for the most part, we've had the layout where we have our screen here, we have our text editor, and then we'd switch back into the browser and then hit refresh, but since we have the browser right here, then it makes sense to use a tool in Visual Studio Code called live server.



What this is going to do is it's going to give us an automated way of updating, so every time that we hit save it is going to automatically update the page, so we're not going to have to switch over and hit refresh.

It's just a nice way of being able to automate our workflow. And after it's done, just click reload like I just did right there, and then you should be good to go.

If you click on it, it'll give you some details for exactly how it works, and so you can see that you have the ability to start it up and it has a bunch of little commands, but we'll walk through those as well.



Now let's open our `index.html`, if you right click on this page, you can say open with live server, and this opened up in another window, and let me pull that down here just so it's something we can see.

This is just going to really help to make our process more efficient because now all we're going to have to do is hit save and then we're going to see the results of our work. That's kind of the workflow that I usually have whenever I'm building these out, but I'm usually working on a larger screen.

This is looking good, and we can test it out. So let's say that we changed the phone number to 3333 just like that, and if I hit save, it will automatically update in our browser. So that is how it works, and it's a really nice little tool.

Like I said, throughout this course, I always want to make sure that I am giving you not just the best advice on how to build out your applications, but I'm also teaching you various workflow techniques that have helped me through the years.

Now that we have this in place, let's keep on moving down the line. So we have everything done that we need to have done for our nav bar. Now let's update our hero unit.

I'm going to come down right below the links wrapper, and I'll say hero-section, and inside of here, I'm just going to update the padding.

## media-queries.css

```
.hero-section {
  padding: 50px 10px;
}
```



You can see that that re-adjusts it perfectly and that's looking really good.

That is literally all we have to do for the hero section. You can see the parallax feature is still working nicely.

Let's keep on moving down the line, and now under the hero section we have the features section, and for this, I'm going to change the height so that it is going to be 100%.

### media-queries.css

```
.features-section {  
    height: 100%;  
}
```

Now, that's not really going to change it because we first have to go and we have to adjust our columns and everything that we have going on there.

We have the features section and then the columns wrapper, and you're going to notice a pattern here and we're going to be able to start building out a lot of these styles pretty quickly because they're going to be very similar.

### media-queries.css

```
.features-section > .columns-wrapper {  
    width: 100%;  
    flex-direction: column;  
}
```

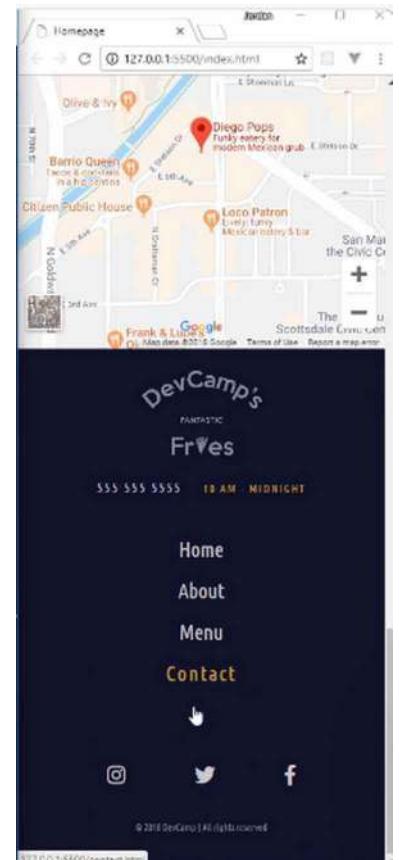
Let's look at that.

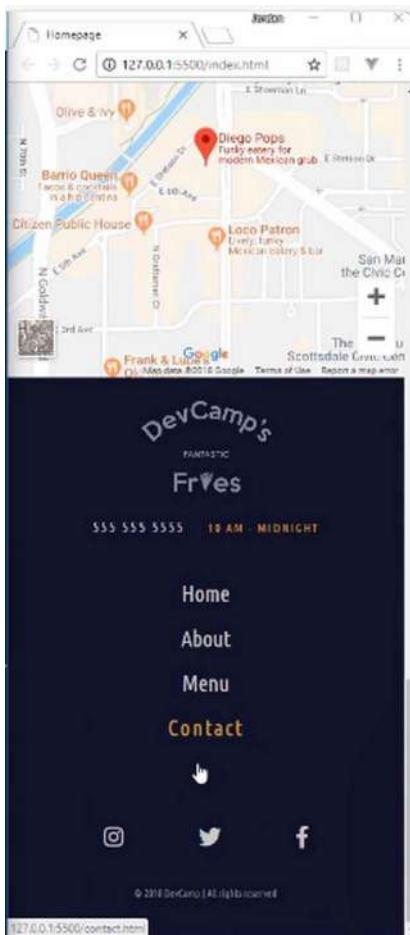
We now have adjusted our features section so that they are now stacked on top of each other when they are on a smaller screen. So that is looking really nice, and let's just keep on moving down the line.

You can see the map was already set at 100%, so that's working perfectly, and then we have a few little issues here with the footer. So let's see exactly what we need to do in order to fix those. I think one of the first things I'll do is I'm just going to give it a height.

### media-queries.css

```
.footer {  
    height: 100%;  
}
```





If I hit save, there we go. That's all we needed.

And so you can see just another example of why it is so helpful to have some of those classes, like the links wrapper class. Do you see how that automatically was populated here on the footer and that works absolutely perfectly?

That is the full set of styles for the homepage. That is all that we needed to do. So in the next guide, we're going to keep on moving down the line and we're going to work on the other pages.



## Coding Exercise

Make the below div 125px wide and make the opacity 0.5.

```
<div class="theDiv"></div>
```

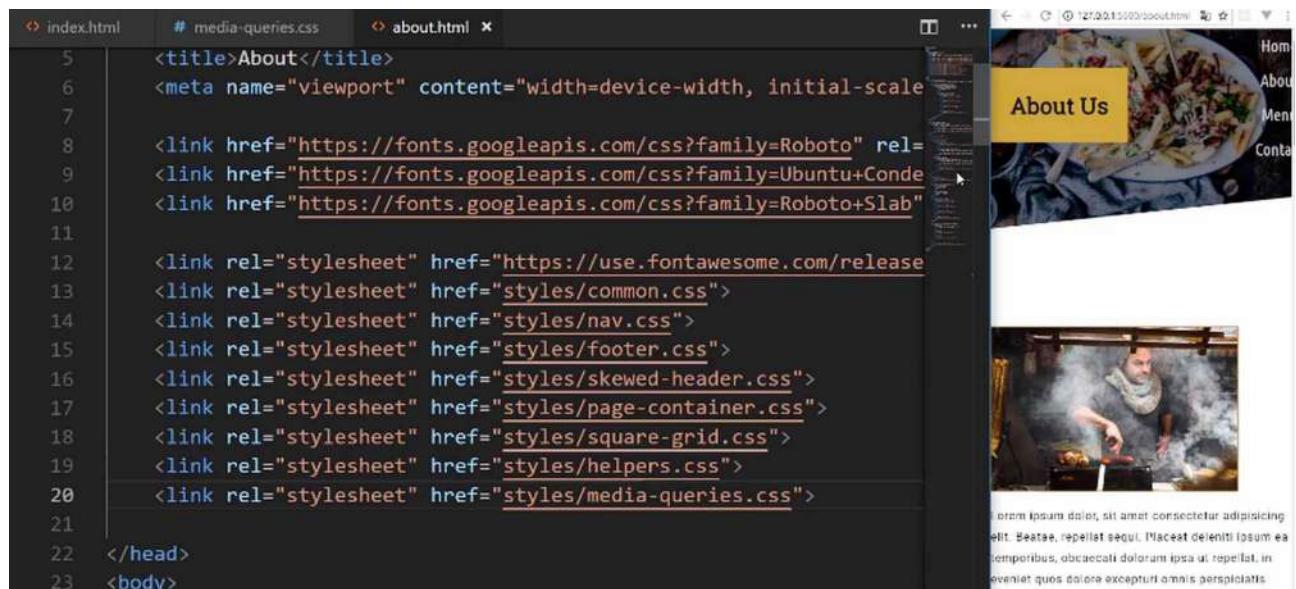
# 1.58 Implementing Responsive Styles to the Square Grid, Image Skew, and Form Elements

Now, this looks quite a bit different because it doesn't have any other responsive elements yet. The very first step that we need to take is to open up that **about.html** file, and import our media queries. I'm going to pull this down and import the media queries file.

## about.html

```
<link rel="stylesheet" href="styles/media-queries.css">
```

If you hit save, you can see that it's adjusted and you have the nav elements stacked on top of each other. Now we need to update this entire skewed header component.



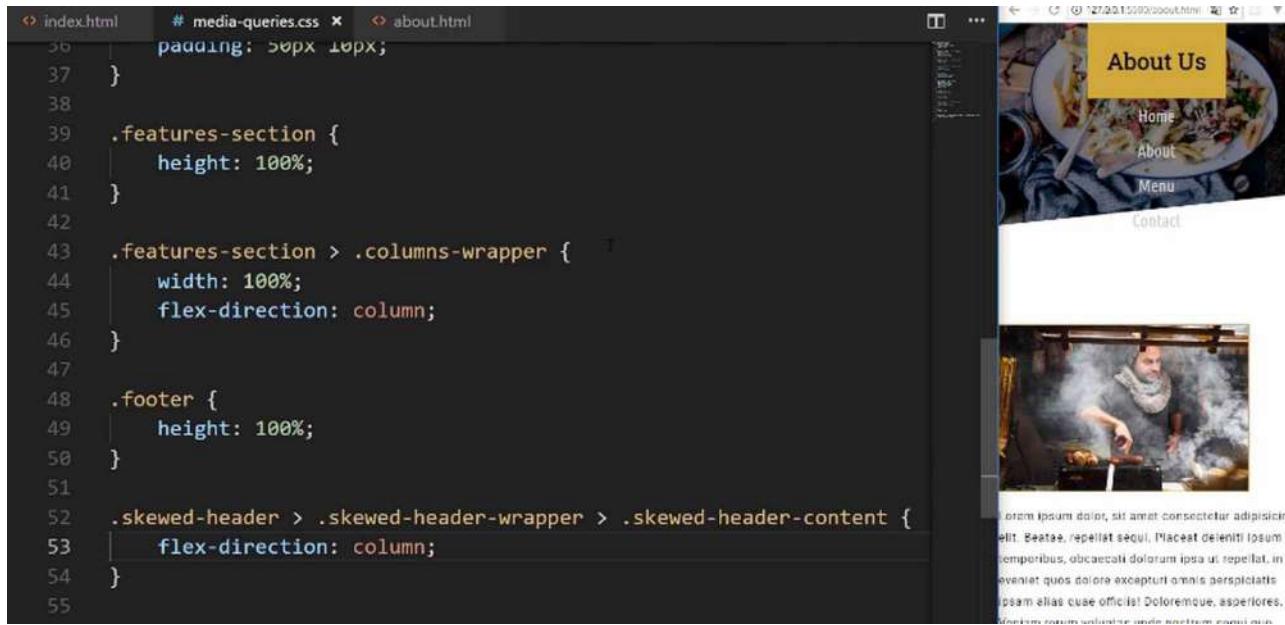
Let's come down into the **media-queries.css** file, and I'm going to select the **.skewed-header**, and then the **.skewed-header-wrapper**, I believe the name is. Then let's just verify what these class names are. We have **skewed-header**, **skewed-header-wrapper**, **skewed-header-content**.

What we're wanting to grab is the **.skewed-header-content**. Inside of here, we just need to update the **flex-direction**, that was using Flexbox. We can say we want this to be **column**.

## media-queries.css

```
.skewed-header > .skewed-header-wrapper > .skewed-header-content {  
    flex-direction: column;  
}
```

There you go. That is looking really good. The only change that I'd like to make is to have the image to grow a little bit so that all of that content fits inside of it.



The screenshot shows a code editor with three tabs: index.html, # media-queries.css, and about.html. The media-queries.css tab contains the following CSS:

```
26 padding: 20px;
27 }
28
29 .features-section {
30   height: 100%;
31 }
32
33 .features-section > .columns-wrapper {
34   width: 100%;
35   flex-direction: column;
36 }
37
38 .footer {
39   height: 100%;
40 }
41
42 .skewed-header > .skewed-header-wrapper > .skewed-header-content {
43   flex-direction: column;
44 }
```

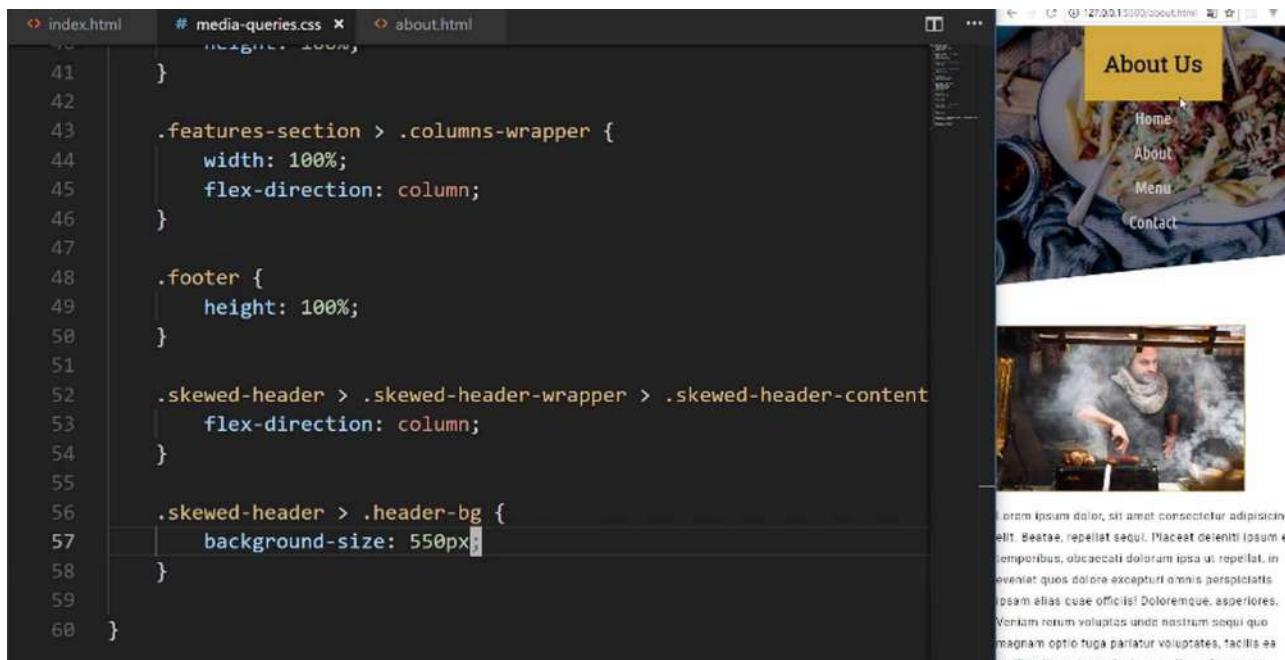
To the right of the editor is a browser window displaying a website. The header features a yellow "About Us" button and a navigation menu with links to Home, About, Menu, and Contact. Below the header is a large image of a person cooking. The page content includes some placeholder text (lorem ipsum) and a footer section.

We can do that by grabbing the `skewed-header` and then the `header-bg`. Inside of here, we're going to use an attribute called `background-size`. This is going to use `550px`.

### media-queries.css

```
.skewed-header > .header-bg {
  background-size: 550px;
}
```

Hit save and look at that. Now we have a really nice looking header here that has all of the same skewed styles. We didn't lose anything here, and it now works perfectly on mobile, so this is looking really good.



The screenshot shows a code editor with three tabs: index.html, # media-queries.css, and about.html. The media-queries.css tab contains the following CSS, with the `background-size` rule highlighted:

```
10 margin-bottom: 100px;
11 }
12
13 .features-section > .columns-wrapper {
14   width: 100%;
15   flex-direction: column;
16 }
17
18 .footer {
19   height: 100%;
20 }
21
22 .skewed-header > .skewed-header-wrapper > .skewed-header-content {
23   flex-direction: column;
24 }
25
26 .skewed-header > .header-bg {
27   background-size: 550px;
28 }
```

To the right of the editor is a browser window displaying the same website as before. The header now has a larger, more prominent background image that covers the entire header area, fitting all the content inside.

Moving down the line, let's now work on our image here. This is going to be the image that is, I believe, the id of `#chef`. This is going to give us an opportunity to learn about another skill in CSS.

I'm going to grab the `content-wrapper` and then I'm going to grab `#chef`. If I add some styles here, watch what happens. I'm going to add `margin`, in my ideal scenario I would have a little bit of margin on all sides, and it would just fit in nicely, but that would take up all the space.

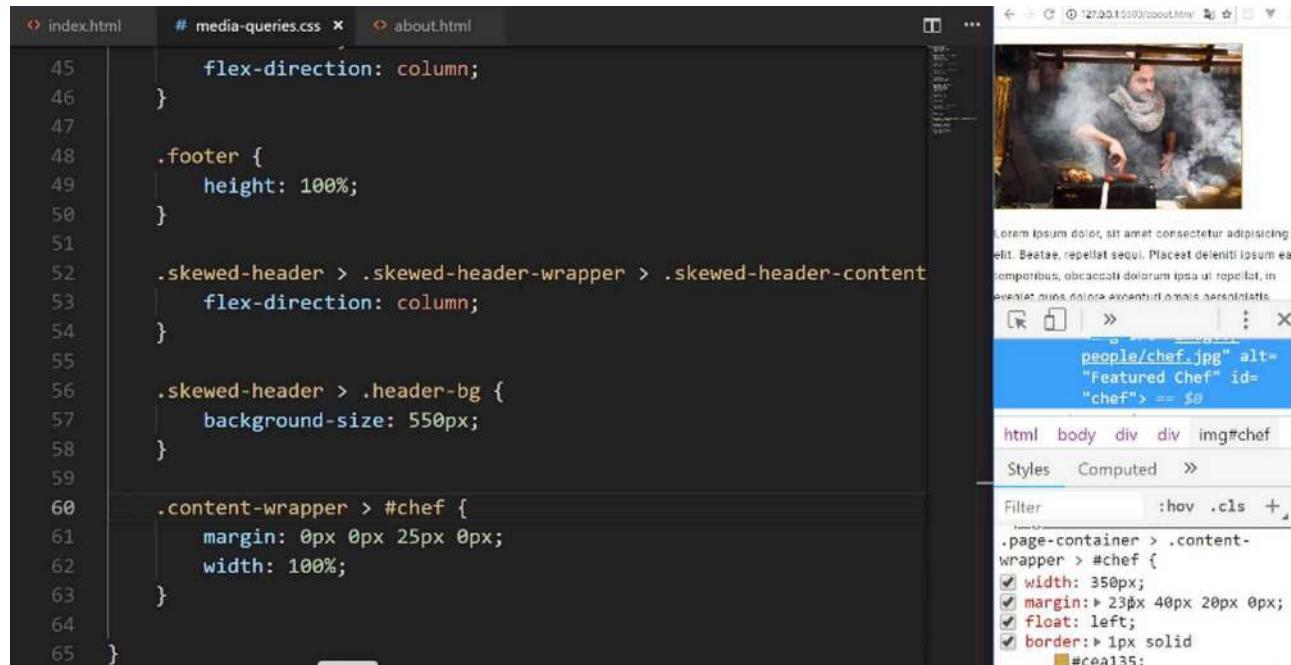
So I'm going to use `0px 0px 25px`, because I want some space on the bottom, then `0px`. Then I want to have a `width of 100%`. Now if I hit save, nothing happens. Okay, so what gives here?

### media-queries.css

```
.content-wrapper > #chef {  
    margin: 0px 0px 25px 0px;  
    width: 100%;  
}
```

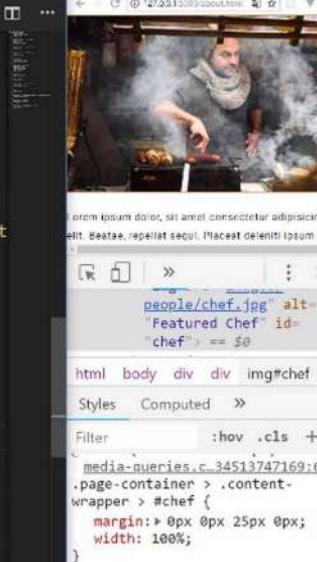
Well, this is going to be a very good lesson because there's one skill that we have not talked about in this entire course. I purposely haven't because it can be considered a bad practice, but it is still something that is important to learn. I'll walk through exactly what it is right now.

Let's select this, and scroll up so you can see the values. If you look down into the tools here, you can see that we have a `width of 350px`. We have the `margin`, and these are the things that we set in our `page-container` class, in our CSS file.



This is an issue because it looks like it is ignoring our values. It's technically doing that for a very good reason, and this is the reason: do you notice how we have the `page-container` class and then the `content-wrapper`? This is following the rules of CSS.

There are two options that we could do. One, I could just come here and I could say .page-container > .content-wrapper > #chef just like this. You see how that worked?



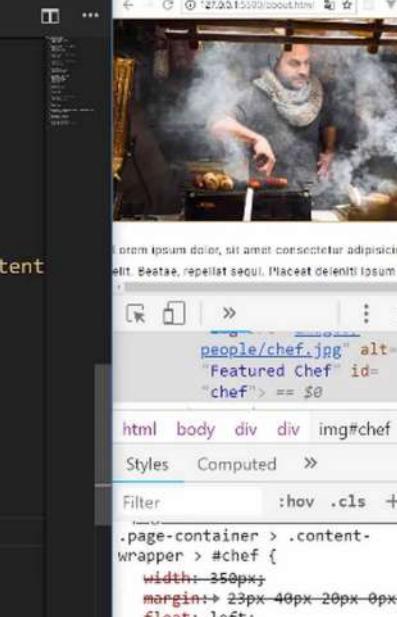
The screenshot shows a browser's developer tools with the Styles tab selected. It displays the CSS rule for the image element with id="chef". The rule is: .page-container > .content-wrapper > #chef { margin: 0px 0px 25px 0px; width: 100%; }. The computed style panel shows the same properties with their values.

```
index.html    # media-queries.css    about.html
45     flex-direction: column;
46   }
47
48 .footer {
49   height: 100%;
50 }
51
52 .skewed-header > .skewed-header-wrapper > .skewed-header-content
53   flex-direction: column;
54 }
55
56 .skewed-header > .header-bg {
57   background-size: 550px;
58 }
59
60 .page-container > .content-wrapper > #chef {
61   margin: 0px 0px 25px 0px;
62   width: 100%;
63 }
64
65
```

That's perfectly fine, but I wanted to teach you another skill. Whenever you have a situation like that where maybe you weren't the one that created the styles, maybe you brought in some template or framework and you have these media queries, and you want to force your styles even when you have a less specific selector.

That's the entire reason why this isn't working, is because remember in CSS, the most specific selector wins the battle. You can always see what the most specific selector is by looking at the devtools right here.

We can either update it so that it matches the same level of specificity, or we can use a little tool here called !important. I can add a !important just like this in front of each of these values if I hit save, you can see that it still works.



The screenshot shows the developer tools again, but now the CSS rule for the image element with id="chef" includes !important declarations: .content-wrapper > #chef { margin: 0px 0px 25px 0px !important; width: 100% !important; }. The computed style panel shows the updated values.

```
index.html    # media-queries.css    about.html
45     flex-direction: column;
46   }
47
48 .footer {
49   height: 100%;
50 }
51
52 .skewed-header > .skewed-header-wrapper > .skewed-header-content
53   flex-direction: column;
54 }
55
56 .skewed-header > .header-bg {
57   background-size: 550px;
58 }
59
60 .content-wrapper > #chef {
61   margin: 0px 0px 25px 0px !important;
62   width: 100% !important;
63 }
```

Now, part of the reason why I did not teach this earlier is because using `!important`, is usually the sign of someone not understanding how CSS works. They don't understand how you need to have the most specific selector, in order to have priority with your style definitions.

There are times where this is helpful and is needed. I would not be doing my job if I did not teach it to you. This is the way that you can get that to work.

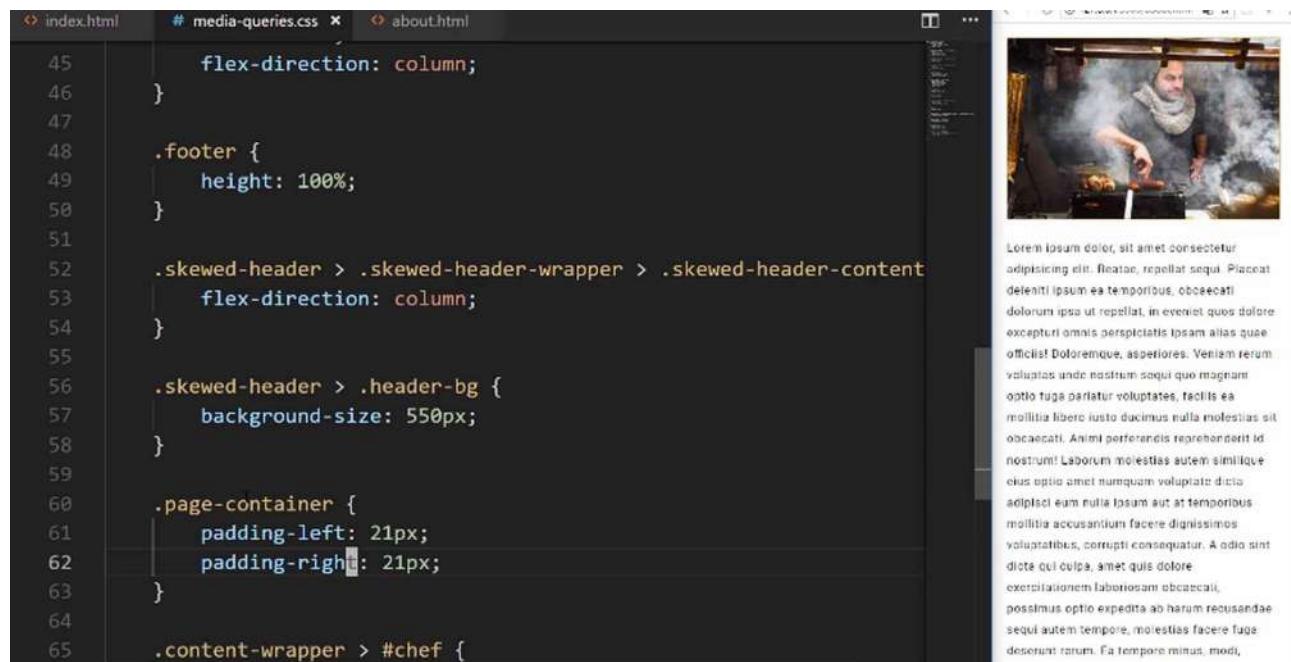
I also want to add some `padding` around the entire page. I'm going to close out of here, and I want padding on the side, not just of the image but of everything here on the page. That will make it easier to see and to read.

I'm just going to add a little bit of padding on the left, so we can do something like `21px` on the left. Then we'll do the same thing on the bottom on the right.

### media-queries.css

```
.page-container {  
    padding-left: 21px;  
    padding-right: 21px;  
}
```

There we go. That cleaned it up nicely and this is making it much easier to read on a phone and see how it also moved it up. I'm really liking the way this is coming along.



```
index.html      # media-queries.css      about.html  
45             flex-direction: column;  
46         }  
47  
48     .footer {  
49         height: 100%;  
50     }  
51  
52     .skewed-header > .skewed-header-wrapper > .skewed-header-content  
53         flex-direction: column;  
54     }  
55  
56     .skewed-header > .header-bg {  
57         background-size: 550px;  
58     }  
59  
60     .page-container {  
61         padding-left: 21px;  
62         padding-right: 21px;  
63     }  
64  
65     .content-wrapper > #chef {
```

Now with all of this, it is time to attack our squares. Now, this is going to be really cool. Now when I originally was building out the prototype, I was really excited with how this worked. We'll also use `!important` here just so you can see how it works.

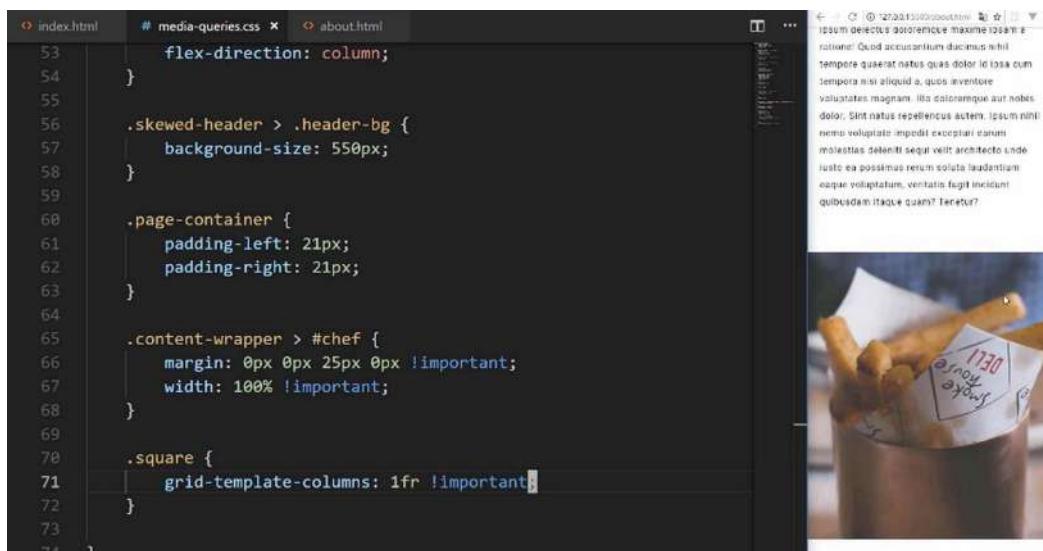
I'll use a less specific selector. Just know that in your real-world applications, you usually just want to be area specific with your selectors so that you can always have control over what's going on. It's still good to practice and understand this process as well.

I'm going to grab just the `square` class just like this and let's add our `grid-template-columns`. Usually, we have been using, for our squares, we've been using `1fr`, and `1fr`. I want to just change it so now it's only going to be a single row. We can just say `1fr` and `!important` once again.

## media-queries.css

```
.square {  
    grid-template-columns: 1fr !important;  
}
```

Now if I hit save, you can see that automatically readjusted it. This is really cool how this works. So imagine that you've come to this site, and you're looking at the restaurant and then you have all of these cool, little, square rectangular types of images.



Now you may also notice that we kind of have a little bit of a bug here, where we have the image, then we have the heading and the content. That's followed by the image, or we want to follow it by the image, but instead, it's the heading again. This is not what we really want.

What we can do is we can force the order. So just like we changed the order of the elements in the navbar on the homepage, we can do the same thing with our squares. Let's go and verify what kind of classes we have here and what class names.

I believe it's `img-wrapper`, and then `square-text-wrapper`. That's what we're going to need to override here. So I'm going to say `.square` and then `img-wrapper`. I always want the `img-wrapper` to be second. I always want the `img-wrapper` to come afterward.

## media-queries.css

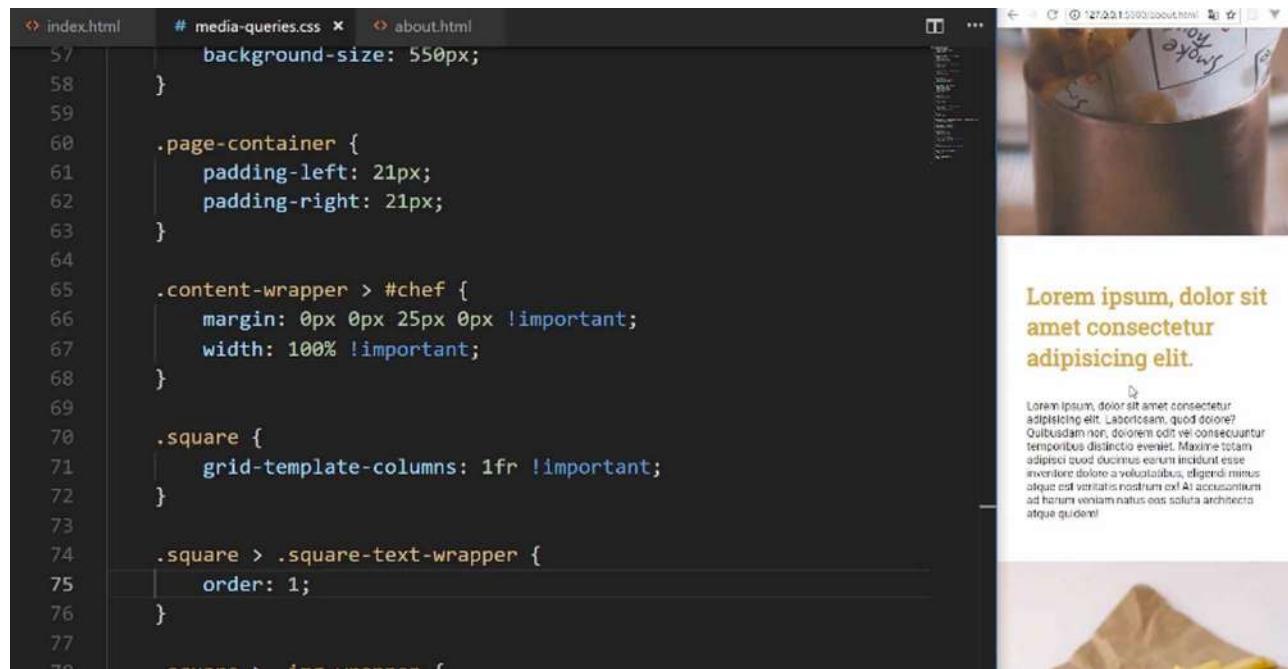
```
.square > .img-wrapper {  
    order: 2;  
}
```

Then I want the text to be the first one. For this one, let's go, and I am horrible at remembering these names, `.square-text-wrapper`, `square text wrapper`. We want this one to have an order of 1.

### media-queries.css

```
.square > .square-text-wrapper {  
    order: 1;  
}  
  
.square > .img-wrapper {  
    order: 2;  
}
```

Now let's see if that fixed it. Now you're scrolling down, you see the `text-wrapper` here, then the image, then we have the text again, the image, and then the footer.



This looks really good, and look how cool that is. We didn't have to change any HTML or anything like that. We didn't have to use JavaScript or rearrange the items, we were able to do 100% of this, in this case, with `CSS grid`, and then just changing the order.

Let's move to the menu because this is going to be very similar. Let's open up the `menu.html`, and then from here, let's import our `media queries`. I'm going to import our media queries just like this, and you can see this automatically fixed everything.

The screenshot shows a code editor with several tabs: index.html, media-queries.css, menu.html, and about.html. The index.html file contains the following code:

```

12 <link rel="stylesheet" href="https://use.fontawesome.com/release/v5.15.1/css/all.css" type="text/css">
13 <link rel="stylesheet" href="styles/common.css">
14 <link rel="stylesheet" href="styles/nav.css">
15 <link rel="stylesheet" href="styles/footer.css">
16 <link rel="stylesheet" href="styles/skewed-header.css">
17 <link rel="stylesheet" href="styles/square-grid.css">
18 <link rel="stylesheet" href="styles/helpers.css">
19 <link rel="stylesheet" href="styles/media-queries.css">
20
21 </head>
22 <body>
23   <div class="skewed-header">
24     <div class="header-bg" style="background-image: url('images/headers/1.jpg');">
```

Look at that. We have all of these just gorgeous images, and they're all in the right order. We didn't have to touch a single line of code on this page. That is pretty awesome.

Whenever you notice that you're writing code, that automatically takes care of other pages, that means that you're writing **scalable** code. You're writing code that you could use across an entire system. So that's working really well.

Lastly, we have our contact form. Let's finish this off and when we do we're going to have this entire course completed, which is pretty exciting. Let's open up the **contact.html**, and as you may have guessed, the very first step is going to be to import our media query style sheet.

## **contact.html**

```
<link rel="stylesheet" href="styles/media-queries.css">
```

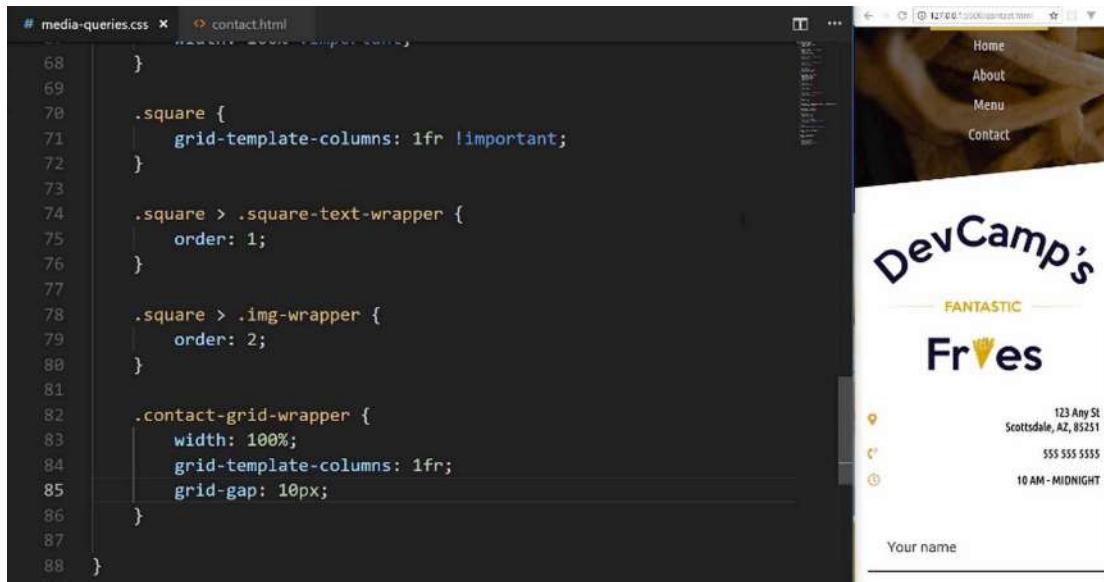
We bring in our media queries, and that takes care of the top. Now we have to fix what we have here in the form. So let's go down, and let's see a few of those names. We have the **page-container**, **contact-grid-wrapper**, and I think that's going to be the first one that we're going to address.

Let me close out our other HTML files, and now moving down, we have the **contact-grid-wrapper**. Let's adjust the **width** to **100%**. Then we want to add the **grid-template-columns** so that it's just **1fr**. Let's also add some **grid-gap** here and let's make that **10px**.

## **media-queries.css**

```
.contact-grid-wrapper {
  width: 100%;
  grid-template-columns: 1fr;
  grid-gap: 10px;
}
```

Hit save, and you can see that it is looking much better. We could adjust the image if we want to. It's completely up to you. I think it looks fine just like that.



The screenshot shows a code editor with a file named 'media-queries.css' open. The CSS contains rules for a grid layout, specifically targeting '.square' and '.contact-grid-wrapper' elements. To the right of the code editor is a browser window displaying a website for 'DevCamp's Fries'. The website features a navigation bar with links to Home, About, Menu, and Contact. Below the navigation is a large banner with the text 'DevCamp's' and 'FANTASTIC Fries'. On the right side of the banner, there is contact information: '123 Any St, Scottsdale, AZ, 85251', '555 555 5555', and '10 AM - MIDNIGHT'. At the bottom of the page is a form with a single input field labeled 'Your name'.

```
# media-queries.css x contact.html
68 }
69
70 .square {
71   grid-template-columns: 1fr !important;
72 }
73
74 .square > .square-text-wrapper {
75   order: 1;
76 }
77
78 .square > .img-wrapper {
79   order: 2;
80 }
81
82 .contact-grid-wrapper {
83   width: 100%;
84   grid-template-columns: 1fr;
85   grid-gap: 10px;
86 }
87
88 }
```

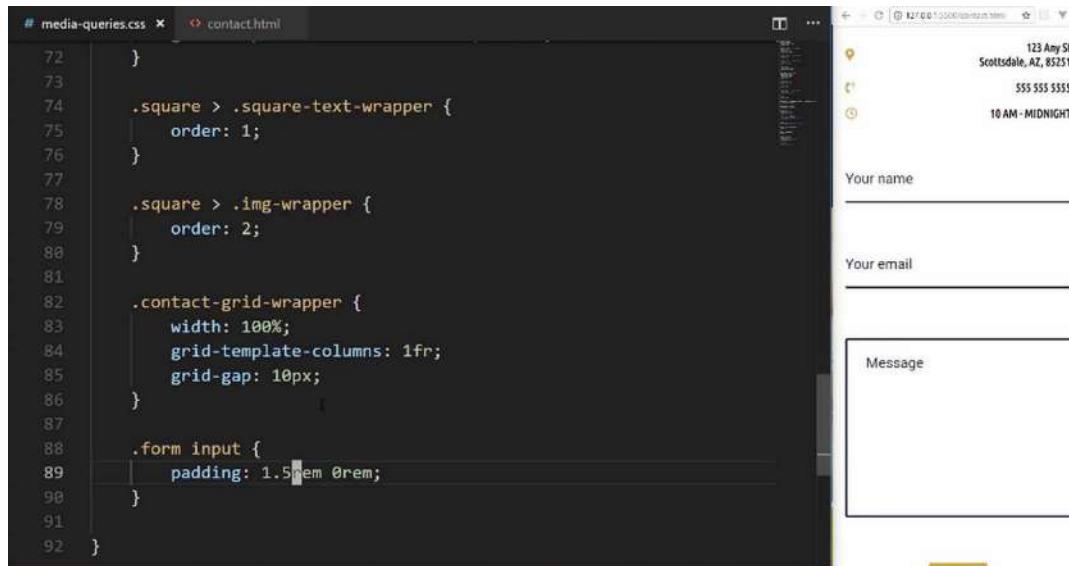
Now we have to adjust this form. We just have to update these form elements because it is going too far to the side. We want it to end right here. We don't want it to go all the way past it. Let's see. We have the `contact-grid-wrapper`, now let's go and grab our `form` class.

We don't really need to work on the form class, we need to work on the `input`, and so with this, let's update the padding. I'll say padding should be `1.5rem` on the top and bottom. Then we want `0rem` on the left and right.

### media-queries.css

```
.form input {
  padding: 1.5rem 0rem
}
```

You can see that, that worked perfectly. Now it's fitting in exactly where we want, and then we just have to do the same thing on the `textarea`.



The screenshot shows the same code editor and browser setup as before. The 'media-queries.css' file now includes a rule for the `form input` element, setting its padding to `1.5rem` for vertical space and `0rem` for horizontal space. The browser preview shows the updated form fields: 'Your name', 'Your email', and 'Message', all properly aligned within their respective input boxes.

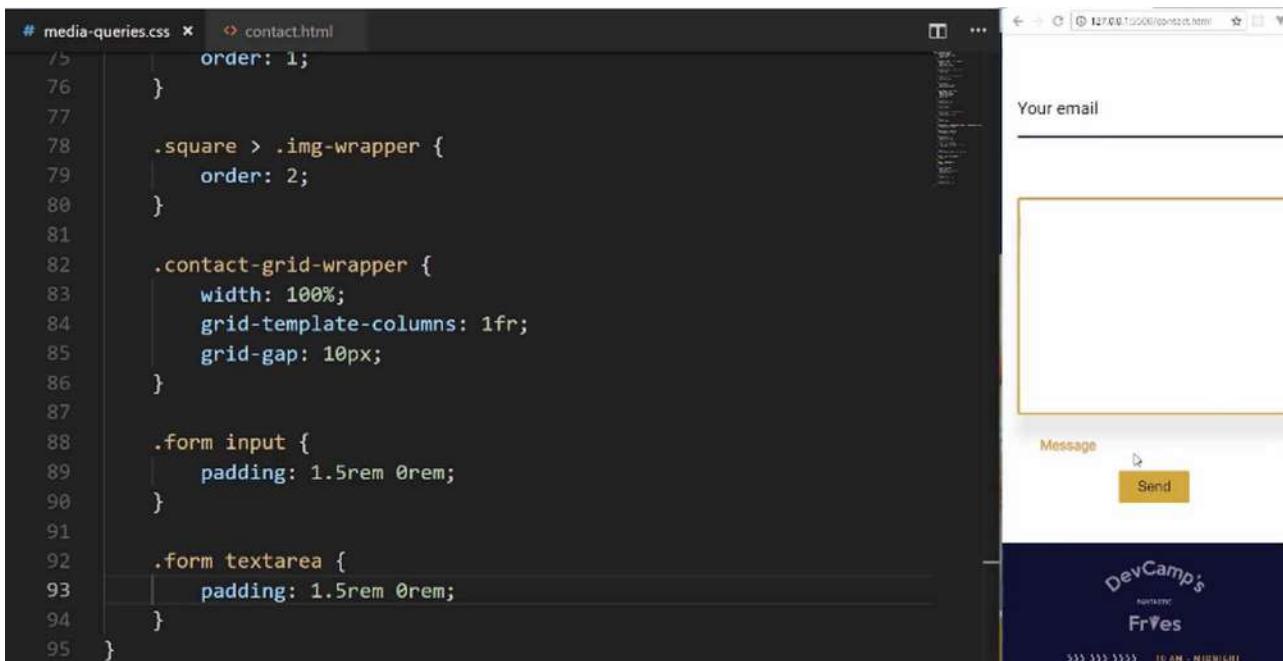
```
# media-queries.css x contact.html
72 }
73
74 .square > .square-text-wrapper {
75   order: 1;
76 }
77
78 .square > .img-wrapper {
79   order: 2;
80 }
81
82 .contact-grid-wrapper {
83   width: 100%;
84   grid-template-columns: 1fr;
85   grid-gap: 10px;
86 }
87
88 .form input {
89   padding: 1.5rem 0rem;
90 }
91
92 }
```

So we're going to say `textarea` just like that, and for this padding, it's the same thing, `1.5rem`, and `0rem`. Hit save, and that's working. Then let's see if our animations are still working, everything there is looking really good.

## media-queries.css

```
.form textarea {  
    padding: 1.5rem 0rem  
}
```

Wow, I'm excited. This site's looking great, and now that you know all of these skills, you're going to be able to build out any kind of website.



What we walk through, even though this may be a four-page website, the skills that we walk through, really are the fundamentals you need to build any kind of site out there.

You can use this to build a reporting dashboard for a business. You could use this for a marketing site, or a portfolio site. Anything that you want to build, you now know how to do. You know how to use tools like Flexbox, and CSS grid, just to be able to align anything on the page wherever you want. You know how to pull in images, and how to work with links, routing, icons, and custom fonts.

So congratulations in going through this. I know this is a very comprehensive course. You were able to do quite a bit of work, and hopefully, you enjoyed it. You got a lot out of it, and now you're going to be able to go out and build your very own projects.



## Coding Exercise

The below parent container needs to use grid with 5 template columns to position the links. The width should be set to 300px as well.

```
<div class="nav-links">
  <div>
    <a href="#"></a>
  </div>
  <div>
    <a href="#"></a>
  </div>
  <div>
    <a href="#"></a>
  </div>
  <div>
    <a href="#"></a>
  </div>
  <div>
    <a href="#"></a>
  </div>
</div>
```