Introduction to Web Development

by Brian Holt and Nina Zakharenko

Introduction

Tools

(Oriental music) So, I've been working as a developer for the past eight years, and it's a job that I find really rewarding, because I feel like I get to go to work and solve problems, and I find it really fun, and so I'm hoping to share this, some of this skill set with you, we're kind of scratching the surface but, it's going to be enough for you guys to kind of get a grasp of everything, and know enough to know how to find more information, and how to learn some of these things in a more in-depth kind of way. So, the first thing that we're going to talk about is our tool set, and it's really really important, and why? It's because, you know, the tools of our trade are designed to help us, to help us solve problems, to help us do things faster, and you need a good text editor, and a way to debug HTML, CSS, and Javascript, and there are lots of browsers out there, but Chrome is, kind of, it's the one that's really geared towards developers. It's much easier to find your way around, the developer tools are built in, unlike some of the other browsers, and the text editor we'll be using today is Sublime Text, but you can use any editor of your choice. The reason we use a text editor that's specific for code is because we have syntax highlighting for all the different programming languages that we use, HTML, CSS, so, in this example, the brackets are highlighted in a different color, and the string, which is the thing between those quotes, is a different color as well. And that's a visual indicator that things are correct, and so your brackets have to match up,

and so if they're colored it's much easier to see if you've missed something, same with your parentheses. So, Sublime Text has a great feature, where your parentheses will, when they match up they will be highlighted, so if you type parentheses into Sublime Text, it will put underscores underneath the matching pairs. So in the example above, we have an extra parentheses with no match, and so, when we type it and put our cursor on it, we'll see that no other parentheses highlighted. It also has a great tool called Replace All. So, for example, if you want to change a string in a whole block of text, you can go to Edit and Replace, type in your new thing, hit Replace All, and it'll find all of the instances and swap them out for you. So, we're going to use Chrome to debug our webpages, and if you guys could visit Google. com, and right click anywhere that's not an image, and click on View Page Source. So when you guys do that, you're going to get something like this. And it's like, whoa, what is all this stuff? There's a bunch of numbers, not a ton of words, it's not really clear at all. So, a better way of debugging a page, and you guys can try this out by right-clicking on the Google image, and going to Inspect Element. Okay, that's much better. By doing this, you're scrolling right into the source of the code that does that specific thing. And so here you can see it's an image tag, which we'll talk about later, but there's also a link to the image that's being used, and if we hover over that link, we'll see the Google logo. So, a great thing about Chrome is that you can actually go ahead and update the values. So, in that, if you have this up here, you'll see that there's a height tag. You can double-click in it and update the value. So you can enter a new value, press enter, and you'll see your changes. So one thing to note about this is these changes are only on your computer. If you refresh the page, you'll lose them. Let's try and change the Google logo, so we're going to navigate to Google. com, we're going to right-click and inspect the Google image element, and then we're going to set the width to 700, and the height to 30. So, I'll go ahead and do that with you guys. Just have to go back. Back? Yeah, just (indistinct) click to the top. Is that fine? Yeah. Okay. So, we've got Google. com, we're going to right-click, Inspect Element, it'll pop up this console here, and let me make this text size bigger so that you can see it. So, we're going to double-click in the height, change that value to 30, and double-click in the width, and set that value to 700, and press Enter. And you guys should see something like this. Does everyone see that, does anyone need help? Great. Okay, so that was our result. The other great thing about Chrome is it has something called a Javascript Console, which we'll be using a lot later in the class, and to do that you go to View, Developer, and Javascript Console. So, can everyone open that up? So, it's View, Developer, and Javascript Console. For the purpose of this class, it would be nice to open that up in a brand new tab so that if there's anything going on with the web page that you have this console open on, it's not interfering with your work. For example, like Google here. This is totally fine, we could just press, go in here and press Enter a few times to clear it up. Press the X in the top left, though. Not on

the front-side, but at the top-side. Just up, up, down, there you go, that's it. This? Ah, there we go. Yep, you can hit this right here to clear out your console. So, we're going to type something out here, and see our result. So, let's type console dot and then log, and in the process, this developer console should be giving you hints, so as you type you will see things that match. And then we're going to put a quote and "Hello World", match that quote, and match that parentheses. Once you have that and you hit Enter, if you scroll up, you'll see that "Hello World" was printed right here. So, the other thing that we'll be using the console for a lot is if you have an error in your Javascript files, you'll see an indicator here in the developer toolbar, it'll show a big red X and the number of errors that you have.

Client and Server

Let's talk a little bit about the Client and Server, and what happens when you're on a webpage. So, you're on a webpage and you click a link. What happens behind the scenes is the client is creating an HTTP request, and it's sending it over to the server, so it's like "Knock knock Server, I want this. " And the Server's going to come back and say, "I have it, or I don't". So the Server finds the thing that you asked for, bundles it up, and sends it back to the client. After this, the client decides what it's going to do with it. If it's an image, or an HTML page, it's going to go ahead and display it. So things that run on the Client's side are HTML, Javascript, and Flash. Things that run on the Server's side are things like PHP, MySQL, anything that does generation. So, this is a really brief example of an HTTP request, so GET is an HTTP verb, so you're sending this instruction to the server. You're saying "Get me this link. " The version of HTTP I'm using is 1. 1, which is not important. The Host is, like, which server these resources will live on, and the Accept is what the client is expecting to get back. So, the Server will handle the request, and it's going to send back something like this to the Client. The number 200 and OK is important, 200 is an HTTP status code. So, the Client and Server don't need to know, they don't need that OK message, they know purely by that number that everything went exactly as planned. And so, you'll get back in this example, some HTML, and then your Client, your browser, knows how to translate that into what we see, so colors, images, different font sizes. Now, here's a page that we've all seen a lot, and why does this happen? It means that the WebServer couldn't find the file that you asked for, so you asked for a file that doesn't exist. So, some really common HTTP status codes, ones we probably see almost everyday. 200 means OK, 404 means that that file wasn't found, and 500 is a server error. So, a good way of kind of figuring out what your status code means, if you see it, ones in the 100s are purely informational. Ones in the 200s indicate success of some sort. 300s

are redirection, so that means something has moved. 400s are Client errors, and 500s are Server errors.

HTML

Hypertext Markup Language

We're going to go over three languages, that are going to cover what we call client-side development, front-end development, which is Frontend Masters right, that's what it's referring to. Front-end, when we say front-end development we're referring to what is actually executed on your computer, because like what Nina was saying, you request something from the server, the server sends something back, right. We're going to talk about the something sent back. Later we're going to talk about what's actually happening on the server, that's the Node. js part of this course, but, the server's actually going to send you back code, and then your computer is going to, or your browser rather, is going to execute that code. So of that code that is being sent back to your computer we kind of have three different pieces. We have the HTML, which I'm sure many of you are familiar with, we're also going to later talk about the CSS and the JavaScript. So the first part, HTML, is kind of the, answers the what, right, like it's the actual content that's on the page. So if you have a giant blog post or a blog, right, the HTML is going to be the actual words in your blog post. It's going to have the author's name, the date, it's kind of the what, it has like, it doesn't care about what it looks like, it just cares about what is actually on the page. The what it looks like is the CSS. And then if you have things moving around, or you click a button and it pops something up, that behavior that you're seeing, that's the JavaScript executing, that's the actual code. So HTML in and of itself is not actually code, it's called markup, right, so it stands for Hypertext Markup Language. That's because HTML doesn't execute, rather it's just describing what's happening. Kind of think of it like a Word document, right, you don't put behavior into your Word document, your Word document is not executed, it's rather just read. Same diff here with the HTML, it's just a different format for it. Yeah, so it's going to actually have the words on your blog post. Doesn't have any style, that's all in the CSS. So, if any of you ever have seen HTML before, it looks like a bunch of nested tags, right, it goes in and out in a bunch like that, so, you'll see a lot of that. And it starts with the most general, and then it nests further and further. So let's look a little at the, like this is a contrived example, this is not valid HTML, this is just to show you kind of what it would look like. Like you have a car tag, right, and so you see the car at the top,

and then the ending tag that's /car, means it's like that's the end of it, right. So everything inside of it is describing the car. Then you have the engine inside the car, so, the engine is part of the car, right, that's why it's nested inside of it. And inside of the engine, you have a transmission and radiator, right. So if we look at, for example, the cd-player right there, the cd-player is nested inside the stereo, so the cd-player is inherently part of the stereo, and the stereo is inherently part of the car, right. So that's kind of how HTML works, is, you're saying that, this is nested inside of here, so I know the cd-player is part of the stereo and the stereo is part of the car. Kind of get where I'm going with that? So if we have any sports fans, particularly Seahawks fans, let's look at another example. So you have a team, right, and on the team, we have a couple wide-receivers down there, and the wide-receiver is part of the wide-receivers, and the wide-receivers are a part of the offense. So it's just kind of that nested description further and further up, so inherently the wide-receiver is part of the offense and the offense is part of the team, or you can ask, is this wide-receiver part of the team? Yes, it's part of the team, right. Kind of a weird example, but, let's start talking about HTML now. Looks a little weird, right, but as you'll see, they're just a bunch of abbreviations that eventually you just kind of memorize because you'll use them so much. So p stands for paragraph, h1 is a really big title or header or something like that, and then you'll notice on every HTML document there's a head and a body. You'll never see what's actually in the head, the head is all the meta-data. For example, like when you click a tab, right, if I can bring this up right here this, what's showing up here in this tab, that's what's comes from the title. So it's not actually displayed on the page, nothing in the head will ever be displayed on the page, it's all that meta-data, you can say like, I'm about to give you an English document and this English document is encoded this way, and it has these kind of style things to apply to it, that's the kind of stuff that goes in the head. The body is actually what you're going to display, right, so in this case I have a header and a paragraph. So, I got down here a CodePen link, hopefully most of you have these slides open so you can click on this, if not, I'll give you a second to, does anyone, at least here in the room, not have the slides open, because otherwise you can just click on the link. Just in case, for us, I will give you that link again real quick if you want to open the slides, the slides are right here. Right, some people are still typing, so I'll give you a sec. Okay, so we're on slide 36. Okay, so, go ahead and just open that.

My First Web Page

Alright, so this is called CodePen. It's essentially like a little web app, that you can write your own code in it and it'll instantly execute it for you, which is kind of fun. So, this is exactly what you were seeing before, on the slides. So you can start editing this, and the part on the right will

reflect it. So as you can see here we have our html, which is kind of like the entire HTML document will be wrapped in the html, that's just how you're letting the browser know, I am giving you HTML, please render it. Something you'll find out as you get more and more into programming is that computers will always do exactly what you tell them to do, right, like there's no inference, there's no intelligence about this, your computer is only as smart as you are, and probably a little bit dumber, so. But, I mean, let's go ahead and just add another paragraph, so you can put this p tag, right, and say, I love this so much, and then, you have to close it, you have to let the browser know, OK, I'm done telling you about this paragraph, please stop displaying the paragraph, and there you go. Or you can put another tag in there, right, h1, and say like, THIS IS GREAT. So does it matter that this is your account on CodePen, are we changing it, just on our browser? You're changing it just on your browser, so, please feel, like you can't overwrite me, so, I did that on purpose. (audience laughs) Brian, can you bump the file? Sure. Is that good, better? OK. Yeah that brown's kind of hard to read, I also don't know how to change it, so. So, I mean, feel free to toy around with that, we have lots of examples coming so there'll be plenty of opportunity to mess around with this. Cool, so, you'll notice that there is some style to this, like, the browser knows if it's an h1, make stuff really big, right, if it's a paragraph then make sure stuff is spaced out, because like, if I just start typing in here, which you can, right, but I try and put spaces in here, it's not going to give them new lines, right, that's because the browser's just, like doesn't know what to do with it, so it just kind of smashes it together, it's like, I don't know why you're giving me lines so I'm assuming that you don't want new lines. (laughs) So that's why you have to put it in paragraphs, paragraphs kind of separate stuff. Cool, any questions on that? Alright. By the way this is a really bad idea, like, what I just did right there, please do not do that. Okay.

HTML Tags - Meta, Content

Let's talk more about these meta tags that we are seeing. html needs to be present, always needs to be there, again, it's just letting the browser know like this is an HTML document, please treat it as such. The head tag is where all your meta-data goes, nothing in here gets displayed and the body is like, that's your actual content on the page, so typically your head's going to be, 30 lines, your body will be, you know, if it's a short page only a couple of lines but, often counts will be thousands of lines, so, tens of thousands of lines, even. Alright, so we saw some h1s right, there's actually h1 down to h6, if you're using older browsers, and even in the newest browsers you can actually go to h8, h10, stuff like that. I don't know why you would, I don't ever use much more than h3 or h4, but, it exists, it's there for you, and like we saw before, the browser has some default styles for it, like notice it was bold, it was larger, there was spacing around it, the browser

just assumes that like, He just gave me an h1, I think he wants it to look like a title, but, we'll see later how to change that. Paragraphs, just denotes like, here is a block of text, that is a paragraph, please display it separated from other things, it's just like a paragraph, exactly what you would think of in terms of semantically what a paragraph is. And then a div, div is, like it's both very useful and very frustrating, but it's a separation of content, like it just, it's a container, just like, you can't, if I give you a piece of Tupperware, right, you can't assume anything about what's in the Tupperware. It's kind of like the same idea of a div, it's just a container to hold other things, it has no meaning. Like a paragraph means, I'm going to have text in here, like it's a Tupperware specifically for text, right, that's the weirdest example I've ever used, but, we'll stick with it. A div is a container for anything, right, and then you'll kind of define what's going to be in it, using other means that we'll discuss later. But, you'll see lots of it, and your temptation, when you're doing web development, like I still do it, so, you'll definitely do it, is to put everything in a div, like everything just like, we call it div soup, right, or divitis, that, just like, you just have this total, huge mess of divs, so. You'll kind of figure out the balance, or you won't like me, and you'll just continue doing it for the rest of your life, so, whatever, Okay, (laughs) confessions of a web developer. (audience laughs) Okay, so, we'll just go over some of the more used tags, ul stands for unordered list, you'll find out that there's a lot of abbreviations, it's annoying, but you just get used to them, so, there's a unordered list, so, like you notice in my slides I have a bullet point list, that's an unordered list. It's just kind of saying, I'm about to give you a list of items, and there's no specific ordering to it, like you could rearrange stuff and nothing would really change, right, no order is implied whatsoever. Ordered list is saying I'm about to give you a list, and please maintain the order of this list because it's important, right. Like, there's steps to, I don't know, bathing someone right, I, never mind, I'm not going to go there, let's think of something else, like, huh? Driving somewhere? Driving somewhere, like, open the door, get in the car, turn on the car, and go, right, you can't really mix those up, like, if you try and start the car before you're in it you're going to have a bad time. Thank you, I didn't like where that other one was going, so. (laughs) This is a family friendly audience I think, right? Right, Okay. (laughs) So yes, please, like it implies order, please maintain it, and you'll notice most of the time ordered lists start with numbers or letters, like one, two, three, A, B, C, or something like that, something to denote that it has some order to it. And then an li goes inside either one of those and it's just saying this is an element inside of here. So why is this really cool? Like if you're doing an ordered list, and say, we're going back to the car driving example, we forgot to put in the open the door step, and we had already numbered everything like one, two, three, four, five, six, seven, right. Say we had to add a second step in again, right, we would not have to go back and change all those numbers, I'm sure all of you have done that before, like you're making a list on a document and you have to go back and

change all of the numbers because you've added a new second element right? You can have it so your styling will just like, oh, I have five elements so I know I have to put five tags and so the second one always get the second one. In the chat they're asking for a link to some references on these tags, I guess maybe the documentation on listing them all? Yeah, some, what's that? I can send one. Yeah, I have a link of it as well. Sure. Just post it in chat. Yeah, that'd be great, looks like Nina will take care of it, but, the best place for documentations for everything web, in my personal opinion, is the MDN, or it's the Mozilla Developer Network, it's, I'm sure many of use Firefox right, great browser, great company, and so they put out something called the MDN, which is essentially just documentation for how to be a web developer. It's like, at least for me, it's the authoritative source of truth, so if I do not know something you can go there, and if it says it on the MDN, it is just truth, just solid truth. So, yes, you can search for MDN li, and then, I bet you the top result will be, well, let's just try it, alright. So, MDN li. Right there, just get rid of this. So, I don't expect you to read this, but, this will tell you everything that you have ever wanted to know about the li. Yeah, great source. I will recommend not using W3Schools, that's usually one of the top results and it's usually the worst one, so. The link, I posted. All 73 schools? Yeah. Thank you Nina. Just kidding. No, it's fine, like it's very simplified and, so it might be good to start out with. (audience chatter) Cool, so do we have any questions about lists, we're about, I think we're about to do an example with them, so, you'll get some hands-on experience with that. Oh, here's a good example. So, my favorite social media sites, the top one obviously, and the best ever would be reddit. com, thank you. But, let's talk about the fact I'm using ol versus ul here, like this is implying that there's an order, it's like, if you put Instagram on the top it's not the same list any more, right, this one implies that Reddit is my absolute favorite, and then Instagram is my third favorite. That's what that ol means, if I'd put a ul there it's like, here is a list of sites that I like, in no necessarily order implied. Now, you might be asking yourself, well, why does this matter, like ul versus ol, for the most part it's just going to be the developer that sees it, right, there are some default stylings to it but, when we get to the CSS part, I'll show you how to change it, you can make an ol look like a ul, and it'd be totally transparent to the user. So why, why do we painstakingly make sure that we're using the correct tag for every single thing that we do? This concept is called semantic HTML, and it's essentially, it's going to make your life easier, because you're going to write code, then you're going to go back and do something else for three months and you're going to come back to it, and if you look at it it's like, why did I do it this way? If you do get into this habit of doing semantic HTML, it makes everything just real easy, because you come back and you say, Oh, I'd made this list and there's order to it, so I should not change this. So it's kind of like you communicating either, a, with your future self, or b, with one of your co-workers, right, because your co-worker's going to come in and say, Oh why did he do this this way? And you'll find out a

lot of the habits I'm going to try and get you into, or we're going to try and get you into, is for that. It's for service to your future self, because you write something once, and it's like, this is the greatest thing ever, I'm never going to have to touch it again, and then you're just going to be so sad when you have to touch it again (laughs). So, it makes the difference to get into good habits right now. There's a question that they want you to explain semantic markup a little bit more. Sure. Let's see, it's, semantic markup, and we'll get into more of it as we go further and further, but it's using the correct tag for the correct situation, so like, for example, there's another tag that we have not talked about yet, it's called article, right. As you might imagine, an article is like, describes an article, so if you're on a news site, you would expect your article to be inside of an article tag, right, and then everything inside of it you knew was a part of an article. Now, article doesn't have any special power that a div doesn't have, right, it's just that an article is a special Tupperware just for articles, and so when you look at an article, you know this is an article, this container can only contain articles, right. But you look at a div and you're like, I don't know what's going to go in here, you have to go through and read all of the code, to figure out, what did I stick in this Tupperware and why is it moldy, right, or something like that. (audience laughs) So, it's just like, this is, I'm sticking with the Tupperware example, I'm really proud of this one, not as proud as I am as the bath one, whatever. It's like putting a label on your Tupperware, it's letting your future-self know, this is what's in here, because it's kind of a black box, and you kind of have to poke around, and figure out what's in there, it's a labeling system for it. Doesn't it also make it a lot easier and more powerful when you start doing dynamic pages with CSS, because you can reference those tags in the code? It's also, that's also true, so it makes those tags addressable by CSS. If that doesn't make sense to you right now, that's OK, we'll get to it, but, it's also giving more hooks for your CSS to pick up on. Thank you. Cool, does that, kind of make sense, the Tupperware labeling sort of example, OK, we'll stick with that.

Exercise 1: Creating a New Page

Exercise time. Let's open--so go to CodePen. So you can go to codepen. io/pen, it'll take you straight to creating a new Pen. I'm pretty sure you can go to codepen. io, and just, like there's a New button there as well. Let's go back here and let's get the requirements again. So, I want you on this new page, so first of all, let's talk a little bit, just specifically about CodePen. You don't have to put that html tag on CodePen, CodePen actually automatically does that for you, so if you want to be lazy like I am, you don't have to, if you want to actually be complete and kind of get the full experience you're welcome to as well, it won't hurt anything. Anyway, so what I want you to do in this new page is I want you to make a title, right, if you remember titles were like the h1

through to h6, and the reason why you would choose an h1 versus an h2 versus an h3 is it just implies hierarchy, right. So the h1 should be the most important title, typically you only have, there's kind of different schools of thought on this but, you have one h1 on a page, like, this is my blog, like, my blog would be h1 and like, blog, like, this is blog post number one, that would be an h2, and blog post number three, would be an h2 as well, right, and then you would have different headers inside your blogs, like, the first part of my day, h3, second part of my day, h3, right. It's just a, it's hierarchy that you define yourself, whatever makes sense to you, that's what you should do, OK? So, in your new CodePen, create a new page that has a title, and then I want you to create an unordered list of things that you look for a car, and then I want you to create an ordered list of your favorite cars, right. And then after you've done that, give each one of those a title, right, it could be like, my favorite car is and, things I look for in a car. I'm doing half your work for you, just kidding. Actually writing copy is probably like one of the hardest parts of my job, because I'm just really bad at it. OK, I'll, are those instructions clear to everybody? They're asking more, to explain the heading tags. Oh sure. Alright. So, you can click these nice little arrow things, like you can collapse it, right now we're not worrying about anything CSS and JavaScript, we'll come back to that later. So, just to talk a little bit about h1s right, like this is a title, h2, a smaller title, h3, yet smaller title. Yet smalled, I can spell, OK. So, it implies hierarchy, right, like as we've seen with HTML, it's very hierarchical based. You'll find that a lot of this stuff that I'm teaching you, it just teaches you kind of how to defined your own hierarchy, right, there's no special rules of how you can use h1s, like there's nothing to say it's like, you can't have an h2 on a page if you don't have an h1, like it's purely just what it means to you. So, for example, we'll talk about the project that I work on, which is Reddit GIFs, right, and h1, there's only one h1 on a page and it defines like, this is what this entire page is about, right. And then I'll have h2s, like this is what this particular section, of this whole page means, and then an h3 would be what this particular section inside this section of the whole page is. But, that is all defined by me, right, it's, what does an h1 mean to me? So they're essentially giving you tools, like different markers, right, for your Tupperware, of how to define things. So in this particular case, they don't really have any meaning, it's only meaning that you assign to it. Or like you'll look at different websites and they'll have like an h4 next to an h2, right, and they just, there's no meaning to it, and then in that case, it's just, it's not very useful because it's just chaos. But, anyway, they're just titles, and then the number only has significance as much as you assign to it. And I guess, the other difference is that the browser has different default stylings for it, but very rarely will you just leave it with the default stylings, you'll change it to be whatever you want it to look like. Does that make more sense? Hopefully as we keep using them, it'll make more and more sense to you. OK. So, how did we do, people still working? Looks like it, I'll give you a couple more minutes. I'll give you, like

three more minutes, is that good for people? I have a question. Sure. So, when we are creating the unordered and ordered lists, it's in the body, not the paragraph, right? Right. So yeah, it's just kind of to, you don't actually have to put the body, like, when you're making HTML documents you do have to have a body, but CodePen actually wraps everything in the body for you, just to be nice to you. (laughs) So you're welcome to put, (mumbles) in this particular case. OK. Were those two in the same code, in the-- Yeah, let me throw those back up. Yeah. So do you use div to basically wrap anything or everything in, can you explain the difference between a div and a span? Sure. So, something we have not addressed yet, which is actually a really good point to bring up, is span. Span and div are similar because they are simply like generic containers for something, right. The difference is, is like a div is like a big container that contains lots of smaller containers, whereas span is just like, it's actually like the smallest container, right. So, you can have a paragraph, and it can have a couple of spans that will sit around certain words, so, I will go over it after we finish this example. Maybe you're covering it later, but, he's asking about block-level elements as well, he wants those explained. Yeah, we'll definitely go over those, but, we'll address that more in the CSS part, which is a little bit later today. Okay.

Exercise 1: Solution

You can put the html here if you want to, not required, I probably will stop doing it after this, but, we'll just do it just so you know what it looks like. So. Brian, you couldn't change your theme by just clicking your username and going into settings? That's probably a good idea. Or to add group settings? Especially for like, Classic or Tomorrow Night. Yeah, let's just, let's actually do Classic, that looks good to me. OK, SourcePro, yeah, cool. Now, and then I can just, Grandmas use CodePen too, let's go For projection, how funny. Did any of you ever read CSS-Tricks, it's, first of all, awesome blog to read for CSS-related stuff. CodePen is done by the guy that does CSS-Tricks, or he is a part of that company. Super great guy, but he's really cheeky sometimes, which is also great. So now do Control + zero to reset your browser. There we go, oh that's nice, cool. Control + zero and you reset your browser. Delicate, legible. Yeah, much better. Cool. I just totally messed that up, alright. Alright, and then you put head, alright. So, we have to give it a title, right, so this title is going to be an h1, and it's going to be MY SUPER AWESOME LIST OF CAR RELATED GREAT THINGS, because I like to yell when I type. Cool, so, something else you'll notice with CSS is like, you notice how I'm indenting everything and not indenting some other things, like, that it kind of implies hierarchy. You can put this back in like that, and, it's not going to affect the way the CSS or the HTML works. We do it, again, for our future selves, when we're reading it, so we can see the hierarchy, that makes it much more readable for yourself, right. Like you look at this,

you see no hierarchy, right, you assume that the h1 is in a higher hierarchy than the body is when in reality it's not, it's actually nested inside the body. So, the rule of thumb here is that, if something is nested in something else, it gets further indented, right, so the h1 is nested inside the body, so the body's here, then the h1 goes here. Or if it was nested, there is the body, a div, and then the h1, that's why you kind of see that hierarchy. Or in other words, white space likes tabs, spaces, new lines are not important in HTML. Right, like I can also do this if I wanted to do, that, which you'll see a lot, right, so I have this h1, and then I have my text inside the h1, see, this white space right here is not important, the new line. It's, just do whatever is the most readable for you, and, in my opinion, putting the text on a new line is the most readable. Alright, so, we have an h1 here that lets us know, like this is the title of my page, right. I believe the next one was an unordered list of what you look for in a car, Okay. So if we're going to do an unordered list, we're going to use a ul, right. So, ul, and then I'll, something else that I prefer to do is, every time I create my opening tag I just write my closing tag right at the same time, it just makes it really easy to not forget to do that, Okay. And then now, since I'm doing a ul, I'm going to nest some li's inside of it, so, I'm going to indent a little bit further, Okay, so my first list element is, like, I don't know, price because I'm cheap, right. Just, do you like my beautiful styling, it has this huge bold header and, little tiny price, Okay So let's put another thing in there, like, it needs to have, a huge stereo, so we're going to say stereo, and maybe some sweet rims. I'm kidding (laughs), you guys probably think I'm an idiot, which is totally fine, you can think I'm an idiot, OK, so. Does it like, (laughs) that's what I look for in a car apparently, price, stereo and sweet rims, probably going to end up driving like a 92 Corolla, with these blinged-out Walmart rims or something like that, and a huge boombox right, not even like a stereo in it. OK, so, that's what I look for in a ul, is like you have the ul, and then you have the list elements inside of it, which is what that li is right, it's a list element, Okay. So now let's do an ordered list of my top three favorite cars, Okay, so ol, and again, we'll just close it right now, because that's easiest. So this one, it's going to have some order implied to it, so I'm just going to do li again, and say like, I don't know, 92 Corolla since apparently that's my favorite car, I don't know if I'm spelling that right, Okay. Maybe after that's like a 91 Corolla, it's my second favorite car, and then my third favorite car is probably like a, let's say like a Tesla Model S, it's my third favorite car after the two Corollas. (audience laughs) Okay, so, notice that these look really similar right, like you have the ul and the ol, and, they just have li's inside of it, but, one of them just has bullets and one of them has like numbers next to it. It's kind of the semantic difference there, is that, you could put sweet rims on top of price, it wouldn't make any difference, right, like it's all just kind of a conglomerate of information, whereas the ol's like steps, right, like this is number one, number two, number three. But they both just take li's right, because let's say, Oh my gosh, I forgot my, really my second favorite one is the 88 Corolla, right, and then

it just does all that renumbering magically for you, so you don't have, many of these things is also because programmers are lazy, like, I don't want to have to go renumber stuff, like that's dumb, right, so, let's make the computer reorder stuff for us. So laziness and then flexibility, right, because then, say like dynamically later, I want to be able to add things to this list, using JavaScript or something like that, I'll say, Append this as the third element, so it goes in there and then, like I don't have to really use JavaScript to programmatically renumber everything, the browser just automatically does it for you. So, it's like, getting free programming for yourself. Alright, so, specifically on this, do we have any questions, so far, I think I have to do titles for each one as well so I'll do that, right, real quick, Okay. So, we're going to put in some h2s here saying like, Sweet stuff for my car, and then this is like, Sweet cars for my stuff, (laughs) apparently I make myself laugh, that's great. Yeah, so you're just giving titles, right there. So, notice this is kind of all mixed together, there's really not, the HTML's kind of, not divided very well, so, in order to make this easier for yourself to read later, something you might do is you might wrap these in a div, right. So, I'll put a div around each one of these, and then I need to indent these as well, so I'll show you a little trick that works in Sublime as well as in CodePen, so just, select all this text, right, that's inside the div, so, you might not be able to see it but I've selected that h2 and the ul, and then I'm going to hit Command and, actually can you hit Tab, you can just actually hit Tab, let's just stick with that, that's easier, because I think it will work in both browsers. So if you'll hit Tab, it'll actually indent it a little bit further for you, right, because now it's nested inside that div and needs to be nested a little bit further, so it's easier to read. We follow, does that make sense, that work for you? Shift + Tab moves it back out. Shift + Tab, nope, OK, yeah. And then we'll do the same thing for this one. Oops. Command hit, what, what for ones? Sorry? Hold this key. I didn't. Instead of command, you're saying instead of command. Oh, sorry, you don't have to hit Command you can just hit Tab. Oh OK. Yeah. There's a couple ways to do it, but that one works. So, again, reason why I did this, is because, like you notice the styling over here actually didn't change anything, right, but now you look at it and say, This one, this div right here, is all related, like everything in here is related to one thing, and everything in here, this other div up here, it's all related to a different thing, so you're putting separation in your HTML. And it's going to save you so much time if you just kind of start thinking like that, like, everything in here should have its own container. It's also going to make your CSS way easier, you're not going to have these crazy CSS things.

Audience Questions

And why would you use a div instead of a span? So, I'm going to address that here in just one second, but the answer is is you can't use a span there. OK, so I'll address span right after this, is there any more questions as related to what just happened here? OK, I'm just going to give a, throw out a disclaimer, which I'll probably say several more times, over the next like, 48 hours, if you feel like you're drinking from the fire hose, you totally are, right. This is just like, spitting tons of information out here, because the, to get a basic foundation of web development you have to understand a lot, right, and then it's just like, it's kind of like, once you have the foundation then you can start moving up, so if you don't grasp everything right now, which is why my wife hates talking to me about web development, it's OK, and like, it's just repetition, practice, and like, you're just going to have your aha moments like oh OK, that's why that idiot was up there bumbling about this, right, OK. So, don't get too frustrated with yourself if you don't feel like you're getting it right away. In my opinion, no one gets it right away, I certainly didn't, like I, and, I have to remind myself of that because I've been doing this for years now, so, we're OK, (exhales) breathe, OK, this is not yoga with Brian. (audience laughs) OK, cool, let's talk a little bit about spans, since it seems to be the hot topic. So we have divs and we have spans, they're related in the sense that they are these containers that are just like, they're just containers that have no description to what's going to be in them. The big difference here is that a div is meant to encompass other things, right, so in this case, our div is encompassing our h2 or our ol like it's going to encompass many things. The span just goes over a bit of text, right. So like, for example, like say, I really like, yeah, let's do that, like we're going to put it over all the Corollas. Or even better, models, let's go over models. So, you don't have to follow this if you don't want to, but, oops, the space there. OK. And model, that's right there. So in this particular case, what I've done is, I've put a span around all of the models. So say later, we'll talk about CSS, so if this doesn't make sense right away, that's OK, but, say I wanted to make the models look different than the makes or the years in this case, right. In this particular case, I could do that because I've put spans around all of the models, and, what I'm essentially saying here is span is just going to go around like a little bit of text that you want to be different from the rest of the text. So, in this particular case, again, don't worry, like we'll explain CSS in a couple of hours, but let's say, color red, or something like that right. So now notice that I've made all of the models look different than the rest of the text, right, that's why you would use a span. Kind of make sense a little bit, right, like it's meant just for a little, like a little span of text, huh, there you go, that's why they call it span, a little span of text. Cool, moving on, moving on. Did I address your question, there in the back, with the awesome hat? I guess, except there's still that block and inline. Yeah, so we'll go over that, like ad nauseum, when we go over in CSS, but divs are meant to encompass blocks, right, and a block's just like other elements that are blocks. Like for example a paragraph is a block, a div is a

block, article, what I mentioned earlier, that's a block, or else there's span, there's like, there's one called em, E-M, which just stands for emphasis, like you notice this one is no longer red, it's now emphasized, so it's italics, or there's also strong, which, as you might guess, is, come on, no, /strong, bold, right. So those are inline elements, things that are meant to go around just like little pieces of text, like spans, strongs, ems, there's a couple of other ones. So that's kind of the difference there, is, that a div is a block-level element and that a span is an inline element, meaning that it kind of stays inline. But yeah, we'll address that more in CSS, because that kind of talks more about style, as opposed to, yeah, content, which is what HTML, HTML is to describe content. So I guess what I'm saying is, I'm going to give you an incomplete answer, that's what I'm really trying to say. OK, Leave this Page. Something else, I learned a lot about teaching, oh sorry, go ahead. There's just a question on why you used span instead of div, for that particular. I closed it, alright, well, let's just open CodePen back up, but that's a good, that's a great question. Here's your solution, I was following along while Nina was coding so. Go away, OK. So let's say I have, just a paragraph that says like, a paragraph of text, right. OK, so I have these just like, random words, this is just going to be a contrived example, and let's say I wanted to do something special to the of, like, every proposition needs to have a span around it, so, or needs to be like red, instead of something else, right, so I put a span right there. First of all, going back to semantic HTML, this is the more semantic way to do it, it's because you're just letting it know, I'm putting something in here and I just want to affect this particular piece of the text, that's what a span semantically means. I say that because, I can go in and I can make a div act like a span, right, like it's possible using the browser, like the browser, just like, here's some good ideas but go ahead and shoot yourself in the foot, I don't care, which is kind of like a general theme for programming. But in this particular case, if I make that a div now, it's actually going to split the text up, because the div is saying, like a paragraph, I want this to be different from the rest of my text, like it's a different container, and so as soon as a different container wants a new line, so, you know, it's like, over here, right, it's actually split on different lines now, which is not what we were intending to do, we just wanted to make it stand out a little bit. Furthermore, I don't think you can put divs inside of paragraphs, so this is actually not valid HTML. Right, because spans can go inside of paragraphs, divs cannot go inside of paragraphs, and that's actually one of the rules of HTML, so, there you go. But again, as a general theme, these are just general rules to help yourself, and you're welcome to break them, right, like, no-one's going to stand over your shoulder, well, you might, your three-month self from now, might stand over your shoulder and say You're an idiot, why did you do this?, but, just about all these rules can be broken if you want to. Alright. Any more questions, on that exercise? Cool. So, oh, so this was just, a completed example if you wanted to see it. Brian? Yeah. Homework question. Sure. So, in the top here, and there's this little, you have

bullets and then, under My favorite cars you have numbers, how did you, I guess I missed that part, how do you change from bullets to numbers because all mine are listed under numbers? OK, yeah, great question. So you notice here that I have a ul and I have an ol. Yeah. So the ul is going to be bullets, because it's implied that there's no order, it's an unordered list. So my guess is both of yours are on ol's right? Oh, uh-huh. (laughs) Yeah, so, if you put them in ol's like if I change this for example right now, to be an ol, all of a sudden now price is more friendly, like more important than eco-friendly. Oh, Okay. Make sense? Absolutely. (laughs) Cool.

Review

Let's review. These are inline tags that go over text, right, so we have strong, which is, usually means bold. And you notice that I say usually in all these? You notice how I changed the color of the text? All that is available to change. So, the question, oh, what you need to ask yourself is like, this could have, the HTML makers could have made this bold instead of strong, but the issue here is like, what might be strong, could be different in different contexts, right, say for example, we're making a page and we don't want, we want to really emphasize this one particular word right. We would put strong around it, and then, say later, that we wanted to make, instead of it being bold, we wanted it as like, you know what, it should be like, bigger and green, right. So if we had it, say bold right, we would want to change it to be something else but the tag would say bold and so you would have this kind of cognitive dissonance, like, this says it's bold, but I changed my mind about it, and so now I don't want, no longer want it to be bold, but the name of the tag is still bold, right, so your tag is kind of lying to you. Instead, you can call it strong, right, and in both cases, the word is now strong, like it's, the bold is strong, and the large and green is strong, it's just kind of, it gives you flexibility to change it later. We kind of follow that, this is, again, going back to that semantic HTML, as opposed to saying, This is my green div, you should say, This is my navigation, right, because your navigation is always going to be navigation, it might not always be green. Did that kind of make sense, or like, you don't, go ahead. So it functions in conjunction with my CSS also, so I mean if it's already bold we don't want CSS. We change it, I mean to be a bold like pink, blue, like that, we could make it, or changing it, I don't want to be strong, but OK. Yeah, yeah, you're grasping it, that makes me happy that you get it, so I don't feel like a bumbling idiot up here, well I already kind of do, but, it's cool, I'll deal with my own psychology in my own time, OK. So, strong, it's for bold emphasis, it's like, typically you use italics for emphasis like 95% of the time, but, maybe that 5% of the time you want emphasis to be like, I don't know, have more spaces between the letters, or something like that, or it's gray, or something like that. You're free to change it, and it's not like lying to you right, because if you set

italics and you change this, it's no longer italics and it's gray, but it says it's italics, you just like, well what's going on here, right. And then span is like divs for inlines, it's used for something you want to separate from other things, and like I said, it becomes very useful with CSS, because just putting something in a span will not do anything to it, it just has a span around it and the user has no idea that there's a span around it. Make sense? Cool. So, let's just take a look at another example of this. This is what everyone in the room is thinking right now. OK, so I have this paragraph, it's like, this class is awesome, right, and I love it, so, it's just wrapping individual elements inside of there, just things that you want to put emphasis on right, like something in, you would do in Word and you would just highlight and hit bold on, that's essentially what this is doing. Or you're just saying like, This text is different from the other text, it's special, this is my special text Questions about that? Cool. Let's just close out stuff, I'll leave that one open in case I ever need to do some other stuff.

HTML Tags - Void Tags

Void tags, or self-closing tags, or essentially tags that do not need a closing tag, that's what void tags, or self-closing tags mean. So if you notice, like for every div I have there's a closing div, right, the /div. There are some elements that, that doesn't really make sense on, because they're kind of like, they would never encompass something else. Going back to the em and strong tags, they're just wondering, can you overwrite that with CSS? Absolutely. Okay. Totally.. Cool, so, void tags. Like you know when you're filling out a form and they have a little piece of something that you can fill out, like it says here, enter your name, enter your last name, enter your address, those are input elements, it's something that the user can click on and then start typing into. So, you would never have anything inside of an input, and the fact of the matter is that you can't have anything inside those input right, like you're not going to have a paragraph inside of your input. So, instead of, being contrived and just doing input /input, they just said, well let's just save four keystrokes, and we'll have a trailing slash at the end, right, so that's what this trailing slash right there means. It means that this is a tag that is self-contained, that needs no closing tag, it closes itself. So, I think I have, get rid of that, I have an example, right here, OK. Oops. You'll find that we have a lot of cat pictures in this workshop, so if you like cats, get excited. (audience laughs) So, here I have two different self-closing tags. Oh, apparently I put in attributes, so ignore the image for right now, OK, I'm just going to talk about the input up here. So, the input is this thing right here, right, and, you can type things in it, OK. Again, it doesn't make any sense to put anything inside of there right, like you can't put a paragraph in here, you can't put a div in here, it's just a tag that exists by itself, and so what they have done, is they have just made this a self-closing tag. Any more you

actually don't even have to put that slash, this is actually valid as well, and it's just a tag that just never closes, that's OK, you'll see it quite often, you'll also see this, both of them are valid. I'd recommend at least until you really understand web development just putting it there, it's really just kind of a useful indicator to yourself, this is what this is, right. OK, so, down here, I have the img, which is an image tag, as you might imagine. An image tag is just like, it's a tag that, I'm going to put a picture on the page, right. And as you notice, it is a self-closing tag, you're not going to put something inside of the image, or rather you can't, you might want to, but you can't. If you want to, maybe at the end of the class I'll show you how, but, it's not intuitive. OK, so, that's what the image tag does. Now, in order for an image tag to work, you have to tell the browser, This is the picture I want you to use, like you can't just say, Please give me a picture, right, asking the internet for a picture probably is not going to end well, so, you want to define the picture yourself. So, that's what this src is right, it's just saying, Here is the picture I want you to put here. In this particular example, I used this placekitten website, where you can just say like, Please give me a 400 by 400 image, 400 is pixels, it's 400 pixels, so if I say like 300, that'll give a different picture. You're so lucky right, you get to see cat pictures all day, but what were you expecting from someone from Reddit, it's just tons of cat pictures on Reddit. OK, so, like that's, the placekitten part's not important, right, they're just placeholder pictures. So, that's what this src is, it's called an attribute. So sometimes, like-- Hey Brian, they're asking what's the difference between the bold tag and the strong tag? So, there is a bold tag, it does exist, just don't use it. It's old, and, like it, I was describing the very problem with the bold tag earlier, that sometime later you're going to want your bold text to not be bold but be green instead, or just like, you're going to want to change it to be something other than bold, and in that particular case, you now have a bold tag that no longer makes things bold, it just doesn't make any sense, it's bad, don't use it, use the strong tag. They do, they accomplish the same thing, but the strong one is going to make more sense later. Alright, good question, thank you anonymous internet stranger. Cool, so-- There's questions about like XHTML versus HTML5? So, in this class we're going to be talking about HTML5, so there are different revisions of HTML, any more, HTML5 is supported everywhere, so just don't even worry about old stuff, at least for this class, like you might go to a job and you might work on Windows XP and have to use Internet Explorer 6 and then you might have to care about XHTML, and, if that's the case I'm really really sorry, but, for the purpose of this class we'll be talking about HTML5, and in practical difference they're like 90% the same thing, for the most part. So-- Then on that note on HTML5 they're talking about, self-closing tags, is that considered bad practice in HTML5? The self-closing tags, like meaning that having the trailing slash versus not having the trailing slash? Or just like having self-closing tags? Yeah, I believe so. OK, well you have to have self-closing tags, right, like you can't make an image any other way.

Right, like, if I did this, this is not valid HTML, like the closing img, so, either you have to have this, no closing tag, or space like this, either one's fine. That make sense? Yeah, it just doesn't exist, like that's not, it's wrong, so, don't do it. And it makes sense right, like you have an image, and just an image stands by itself, it's its own entity, it wouldn't, it's not a container, it's just a thing, that is, that is. So some you have to have the closing complete tag, some you have self-closing tags all in, some are void tags and self-closing? Those terms are interchangeable, self-closing and void tag, it means exactly the same thing, and, some tags are self-closing, others are not, like for example you cannot have a self-closing div, that's invalid as well, a div must have an opening tag and it must have a closing tag. Like paragraph, you can do both with paragraph then? So, just have like a, like that, is that what you're saying? That's not valid either, it has to have, that. Now, I say not valid, your browser is pretty smart, it can kind of guess what you're trying to do, but it's just like, don't, right, you don't want your computer guessing what you wanted, because it's usually going to get it wrong. I think the main difference is, like, a paragraph is going to wrap something, a div is going to wrap something, h1's going to wrap something, you know, there's contents inside of those tags, versus an image is an object, you know, it's just-- It's an entity, that exists. It's an, yeah. You're not going to wrap-- Something in an image. Yeah, you're not going to wrap text in an image tag. Yeah, does that kind of make sense? I think it's just one of those things with practice comes. And, another disclaimer that I'll throw out here is, like I've learned a lot of things from teaching my wife to code, because she yells at me when she doesn't like stuff. You all are very nice that you don't yell at me and throw things, my wife is not quite that nice when I'm teaching her things, or maybe like, I deserve it probably, so, it's not her problem, it's my problem. (laughs) I make my wife sound abusive, she's not, so, I love her. Yes, so, my wife wants to understand every little piece of something right now, she's like, well, what is that title doing right now, right, like sometimes there's just things that like, it's better just to ignore them right now and try and come back and understand them later. Because like I said, there's a very wide foundation you need to get to understand HTML, and so to try and understand every micro piece of data right now, it's going to paralyze you, right, because, sometimes, this has to be there, but it's not going to make sense until you understand this other piece right, very common thing in web development. So just learn to be, like, ignorance is bliss sometimes, I guess that's what I'm trying to get at, is like, learn to ignore something until it's the right time to learn it. OK, so, other questions before I start talking about attributes? They're just wondering why you're using the doctype, in here for this HTML5, or is that just because it's an example in CodePen? Yeah, and, it's just another one of those things that like, I don't want to explain right now, I will explain them. So there's a thing that's called doctypes that kind of let the browser know what type of HTML you're going to give it, that's the basic story. I mean, omitting it, don't omit it when you're doing it in code but, it's OK

to omit it when we're talking about it. Because right now actually technically, it's like, everything is already wrapped in html and body, so you have nested html and body which is not correct, you can't have two body elements, anyway. OK, so, src right here, OK. Like an image needs to know where to go, so sometimes these HTML, let's see if I can even get it on the same page. So, notice like, you have line numbers right here, so this src actually is on the same line, it's just too long to fit it on the same line. Sometimes these HTML tags need additional information that are kind of meta-data about the tags, right, we'll talk about classes and IDs later, which are also attributes, but in this particular case you need to tell it where the image is, and that's what this src thing is, and so notice it's enclosed within the tag. So most of the time we've just seen like head, and head is surrounded by angle brackets right, here, image is surrounding, not only img, but it's, and the first thing is always going to be what type of tag it is, and then after that the order is not important, right, so the src tells it where the image is, and then we close the tag afterwards. That kind of make sense? OK, well I think we'll, again, we'll repeat this ad nauseum, like use images all over so you'll see attributes, a ton, as we go forward. But it's essentially like, here's more information about this tag that you'll need, that's kind of the point of an attribute.

HTML Tags - Grouping

So, grouping, kind of talked a little bit about this with our car example, but let's take, let's look at another example of this, right. It's a good idea to group like ideas together, so if you were doing a blog post, the, kind of the semantic, like that makes sense, kind of idea is to group every blog post together right, like, you have blog post one, and it's contained in its own container, and you have blog post two and its contained in its own container, it's to group like things together. Alright, so, this is a lot of text, we kind of get the point let's just make this a lot bigger, OK. So, this is just a contrived example, notice I have blog post one, and I have blog post two right, and they just both have a bunch of Lorem Ipsum text in there, Lorem Ipsum text for those of you who are not familiar, is essentially text that is there to hold space but doesn't actually mean anything, it's just meant to be a non-distraction, but, there for styling purposes. OK, so, I have a div right here, and this div encompasses this h1, and another div, right, and inside of that div I have all of my paragraphs, so I'm kind of grouping those together right, because the h1 is different than the paragraphs, and the divs, this div right here, encompasses all of the text. So notice that I'm using divs just to kind of group like ideas together right, so this is like the first div, and it encompasses one blog post, and there's the second div here that encloses the second blog post, and I also have divs encompassing these paragraphs. So notice again, we talked about divs kind of like Tupperwares right, like it's, in this particular case, this Tupperware is a different kind of

Tupperware from this one, all I'm saying is they're both inside, they both have some sort of grouping to them. We kind of follow my logic there, it's really circular I realize, but, eventually we'll start putting labels on these, these are kind of like black box Tupperwares, right, like you have no idea what the hell they are, but, there's something in there and they're all alike, I'll teach you how to mark your Tupperware here in just a second, in fact I believe it's the next slide. It's like those lunchboxes that have little Tupperwares inside. Yeah, exactly, bento boxes kind of, right, there you go, we're making bento boxes. (audience laughs) I like that example too, this is going to make me really hungry, really fast. So, yeah, that's kind of the idea here is, we're grouping like things together. Does that kind of make sense, it's organizational, right, like I could remove this div right here, I could remove this div right here as well, in fact, all the divs could be gone from this page and it would look exactly the same. But, here, like you know, even if I haven't told you what it is, you know there's some grouping here, some grouping going on, and this is going, again, to save you just bundles of time later. These are very small examples, when you start working on a web page that has ten thousand lines, right, grouping stuff together makes it much easier to find the thing that you're looking for. OK, questions on that? These are kind of best practices kind of things as opposed to, like, requirement to make it display what you want it to display. As well later when we start getting into CSS right, the grouping becomes very important because, I want to separate my blog posts, like I want to have them look different right, so say I wanted to give them a gray background, and, split them like, you know, 15 pixels from each other, those divs allow you to say like, Take this group and separate it from this group, so we're kind of building up to this, I don't know, some grand reveal, where there's like, and surprise, everything was useful. Now, continue doing it.

HTML Classes

Classes, so here is your marker, right, like here, now we're actually going to give classes, which are essentially labels. I'm going to say like, This is a div right here encompassing this, a morphous entity to you, and guess what, it's a blog post, and so you look at it and say, Oh, this div is a blog post, and now you know why that div exists. So, again, to someone who had never seen that code before they'd look at that and say, Why the hell did they put a div here, right, like, what is this? And so that's what we're going to use and they're called classes, and, to preempt the question that I know is coming from the online people, we're going to talk about IDs in just a little bit, right now we're going to talk about classes. So, here's a very small example. We now have a div, right, that's surrounding two images, but we wouldn't even have to know, we wouldn't have to see the images to know like, oh, this is a group of pictures, right. So this is an attribute on that div class

that, on that div, called a class, and that's just saying like, you know, it's, you're taking your marker to your Tupperware and say like, This is like, I don't know, spaghetti from last night, or something like that, that's what the class is for. And then again, you can address the classes later using CSS, which is super useful. So let's take a look at this example. And I know you're all excited because you get to see more cat pictures. Alright, so, expand that out. OK, so, that's what that class is for, right, so it's essentially just saying, you don't even have to see what's underneath here to know what's in it. Does that kind of make sense? Cool. I'm sure you're all really happy you get to see more cute cats. Alright, think that, yeah-- So, I have a question. Sure. Can you hold that, yeah, right here, OK. So here, you have a div class, and you have your image, but then your image has that forward slash, at the end of the image, but then you have, oh, I can shut the close out, OK, never mind. OK, yeah. (audience laughs) Sometimes you just got to like, sound it out loud, there's this concept in computing called a, what is it, rubber ducking, yeah, so, and we actually did this with one of my co-workers, I really hope he's not watching (laughs), at one of my former jobs, so he would always come over my desk, and he'd just ask me, Hey man, like how do you do this, and he'd just start explaining to me his problem, and like halfway through he'd solve it himself and just go back to his desk, right. So, rather than, have him come over we bought him a rubber ducky, and we put it on his desk, and it's like, before you can ask anyone any questions, you must explain your problems to your rubber ducky, and so, I'd hear him, mumbling over the wall, just like, OK rubber ducky, like, I'm doing this, and this is going around, he's like, Oh I get it, and you'd just hear this pounding on the keyboard right. So, it's actually extremely useful right, like just enunciating your problems will help you solve a lot of them. So, I'm serious, go buy yourself a rubber ducky, I feel like I should get Amazon Affiliate fees for that. OK, now I do have another question. OK. You started with div class, but you don't have to close it as div class? Correct. Why? So, your closing tags will never have attributes, right, like it's actually only ordinance. Oh, OK. Yeah. OK. Because this one knows that, I mean if you want to talk about the way the browser look at it, it says like, I have now opened a div, so whatever happens after this is inside this div, and it looks until it finds another closing tag and then it closes out all of that. OK. So, it doesn't have to match. Yeah gotcha. OK, good question though. That make sense to everyone else? Yeah, you don't need the attributes as well on the closing tags, closing tags will never be anything but like slash, the name of the tag, closing, that's it. Great question. I actually referred to that, like, if you're looking at a photo app, so you have your photo album here open, now your photos are here, then getting it in is nothing. Yep, there you go. So yeah, OK, I'm a little muddled up. (laughs) Ha, there you go, I appreciate that. That's a good example though.

HTML ID's

IDs are quite similar to classes, that you're going to put them on there, it's just kind of like another way of labeling what you're having on there. The big difference in IDs is that they are unique to your page, right, so, in the previous example we saw picture groups, we can have 10 picture groups, we can have 10 blog posts, we can have 10 paragraphs, you know, they're kind of things that you can have multiple of and they're reusable, right, I guess that's the biggest thing I'm trying to get at is that they are reusable. IDs are different, that they are totally unique to your page, so like, for example, if I had 10 blog posts, and I needed to address only the second one, I could call that one blog-post-2, right, you're only going to have one blog-post-2 on your page. That kind of make sense, a little bit? Yeah, oh yeah, that's the example I used, good for me. (laughs) So yeah, classes are made to be reused, you're creating classes because you say like, "This is going to be useful", and you want to make things as generic as possible right, because the idea with writing this HTML is you want components to be reusable so that, when I go create my third blog post, I don't have to redo everything I did for my second blog post, I can just reuse everything I used for my first post, and it goes back to developers are lazy, and we want to do the absolute least amount of effort to get the thing done. Again, this will become more applicable with CSS, or rather, probably we'll just make it more concrete, anyway. So, let's just go straight to the CodePen. Resolving host. Here we go, OK. So, as you can see here, more cat pictures, surprise surprise, OK. So, here I have two groups of cat pictures, right, and say, I, for whatever reason, like I needed to hide group-2, right, and I only wanted to hide the second one, I didn't want to hide the first one. I now have the ability to kind of hook into that and say like, "Only hide group-2, and keep group-1", or something like that. So they're just kind of like, again, labeling your Tupperware, but being even more specific. So, I cannot put another group-2 on this page, so, here's the reason why this name is actually a bad name, group-2 is really not that descriptive, because let's say I have blog post groups, and I say group-2 right, am I referring to group-2 of the picture groups or group-2 of the blog post group, so you want to be specific enough, but as general as possible. There you go, that's your contradiction for the day. Be as specifically vague as possible. Brian, on the IDs and classes they're asking about like, how many IDs can you use versus classes, can you just explain that? Totally, you can only have one ID, that's like, hard and fast rule, only have one ID right, because IDs in themselves are unique, so there's no point having multiple groups. You can have as many classes as you want, which is kind of fun because now you can mix and match them, like let's say, I have, let's just modify this a little bit, so, I'm just going to put another group here, this is now group-3 right, OK, so all I've done is just added more cat pictures. But let's say I really wanted to emphasize both the first group and the third group for example, what I would do

is I would say like, emphasis-group, or something like that, I don't know, some name like that right. And then, here on the picture group I can also put that as well. So now you can kind of mix them together, right, so you can have like, I don't know, going back to our contrived Tupperware example you can have, this is a vegetable, and this is also meat, like it has both of these things in there, so, I'm going to label it with both of these things, right, so in this particular group I want it to be, I want it to have all the attributes of being a picture group, but I also want you to know like this one's actually really special so I'm going to emphasize that one as well. That's a great question. But, now the second one, it lacks the emphasis-group so it's not going to have any of the emphasis-group qualities, it's just going to have the picture group qualities. And that's only if you have CSS? If this is going to be used a lot in CSS it means a lot as well in JavaScript. But within the browser, that class label does absolutely nothing except-- Correct OK. Yeah, there's no special classes, right. Everything that you do with class is going to be something you define, either style that you define or behavior that you're going to define. That's what they're, they're one for like labeling your containers, like, for example, I see picture-group and I know that one's a picture group, right. Or like I see emphasis and I know this is a special picture group, right. So that's one of the uses, but the second use, like the part that actually the user's going to see, that's going to be from the CSS and the JavaScript. That's a good question. When the DOM interpreter is going through, and parsing the page, what does it do if it finds two identical IDs? I believe it actually ignores them, so, it's going to be different based on different browsers, like I know, old versions of Internet Explorer will handle it differently, like they're going to go into quirks mode which is a really bad thing, it just does weird stuff to try and make your code work. Varies depending on what-- It will vary based on your browser, but for the most part I think it will actually honor it, like you could have two group-2s and it would just apply all, everything to-- So it would treat it more like a class then. Actually, I'm thinking about it, I think what it'll do, it'll apply it to the first one only, that's my guess. And I guess, the point of all this, just don't do it, right, yeah.

HTML Tags - Naming

This is going back to semantic stuff. If you feel like I'm talking about this a lot, it's because I am and I meant to, so, I'm really hammering this home because, it's really tempting as a beginning developer to not do these kind of things, and you're going to find out the hard way that you should, so it's better just to start by doing it this way. Naming is just, like it's just, like you look at something like I need to name this, I do not know what I need to name it, it needs to be a good name, but I don't know what the good name is. Very, very tempting in HTML and CSS to say like, I have a container, it's purple, so I'm going to call it the purple-container. Or this is my right

navigation, so I'm going to, like it's a navigation bar and it's on the right, so I'm going to call it right-navigation. This is bad, do not do it, you're going to make yourself sad later. Because say you're redesigning your site, and you have your navigation on the right, but your redesign is like, it calls for it to be on the top. You now have a thing called right-navigation that is on the top of your page. You're just going to confuse yourself, others, it's just bad. So, rather than naming something based on its presentation, you're going to name something based on what it is. So, instead of calling it right-navigation or side-navigation you're going to call it like minor-navigation, right, or like, I can't remember what we call it, might be like inferior-navigation or something like that, so like superior and inferior or something, something to that effect, right. Something that implies hierarchy or something like that. So, it's very tempting to give those names, don't, call them something based on what they are, rather than based on how they look. Again, if you call it purple-container, and you have your purple container which is green, like, I guess what just makes it really different is that, you're going to go look at your code, you're going to say, "There's something wrong inside of this container", and you go look at it in your CSS, and you're not going to find it because you're going to be looking for something called green-container, when in reality it's called purple-container, right, something that, they just make for weird things that are hard to find. And the point is that, when you change things later, you don't want to have to rename things, and that's just a question you need to ask yourself, is like, "If I need to change this in three months, "am I going to screw myself up if I call it this? " Or if I need to find this later, and my co-worker has modified this, or someone else I'm working with has modified this to be different, does this--is this name still valid, right.

Exercise 2 - Giving Classes

Remember that blog post that we were looking at earlier? I want--open the code, this CodePen right here, I believe it's the same one, and I want you to go give names to everything. So let's just take a look. Any questions about it? So when I say names to everything, everything inside the body, right. You're not going to ever give classes to things in your head or something like that, your body will never have a class because there's only one body, your html will never have a class because there's only one html on your page. So everything inside the body needs a class. Another caveat that I will give you is that most things will have a class, and I would say most things do not have an ID. I only, I mean, this is kind of getting into like, the way that Brian develops versus the way that other developers develop, I only use IDs when I absolutely have to address something individually, right. Like I'm not going to go and give, like I have shown you, like blog-post-1, blog-post-2, I would never actually do that, unless I actually had to always address blog-post-2, or

something like that, then I would go do it. Whereas everything will have a class, right, like even if I'm not even styling or doing something to it, based on class, I will give it a class, just to label it later. Kind of just like if you have non-see through Tupperware, you don't want to have to open it every single time to see what it is, you just want to stick a label on it so you can pull it out, say like, you know, "This is meatloaf", or something like that, and then you can just stick it back, or something like that. So, don't worry about IDs, you can skip those, just give everything a class. And if there's also any of those that you don't know what they are, first of all, that's kind of a good thing because then you'll kind of like hit home the point that it's good to label everything so when you come back to it, or like, right now you're looking at my code, and like, I wrote bad code because I didn't label anything, so, it will kind of drive home the point that when you're writing code for other people, you'll want to give them indicators of what you're doing. Another point to hit home is that, this paragraph, and this paragraph, they're just paragraphs on a blog post, so, ideally, they would have the same class name, right, because you want to treat them the same. Or, this, if you're talking about like this blog-post-1 and blog-post-2 titles, they're both blog post titles and so they should both be named the same thing, right, because we want to treat them the same way. Make sense? Cool. But you did not come to this class for comfort. Frontend Masters is not a comfortable experience. (audience giggles) You should like make that your tag line, I'm just kidding. (laughs) Yeah? So if you've got conflicting attributes in your CSS between a class and the ID, which one takes priority? The ID. Which we'll also talk about later. Cool. Talking about the exercise, he's asking, for naming blog title, or the h1, would you name it like something blog-title? Yep, that's a really good one. It's about what I would call it. I'll give you like, I see there's still lots of typing, so like, four or five more minutes, does that feel good? And then he's kind of asking, I'm not quite sure if I understand but, if he wants to use the same styling in another part, how would he do that, just by using that same class name, right, that applies. Definitely. I think they're answering. Cool. And again, we'll go in depth once we hit the CSS section, which is next, probably after lunch. So you can go back to the requirements if you want to. Yeah, worry more about two, if you get to three then, that's cool too, but, I'm more concerned about classes, that's the more important part here. Is anyone having trouble discerning what, like something, what the container is actually? Yeah, a bit of that. So which one's giving you issues? I'm probably not doing it right. We're naming the class, right, the h1s and paragraphs, right? Yep. Just giving them names? Mm-hm. So essentially, I've given you a marker and told you to go label all the Tupperware in the fridge, that's kind of the exercise here. But it's already labeled, I guess I'm just-- So, it is, I mean, in as much that, like you know there's a div, or you know that it's an h1, right. Yeah. But what kind of h1 is this? Like, let's say I have a page, and this is only like, 1/10 of what's on the page, right, and I have 50 different kinds of paragraphs on there, what kind of paragraph is this? So, this is a blog,

right, and so, these kind of paragraphs are blog paragraphs, right. Might be kind of a good thing to call it, you can call it blog-paragraph. I'm just throwing out answers left and right. That would be for p, right? Yeah. She said they're already labeled, they're already labeled Tupperware. Yeah, I mean they're, it's like I go give you like, 50 different Tupperwares, that have, I mean they're all Tupperware, you can say, you can just go and give a mark and say, This is a Tupperware, this is a Tupperware, this is a Tupperware, not very useful, right. Rather you want to say, This Tupperware contains this. So this paragraph contains a blog paragraph, right, like it contains blog type text. Or like that first div up there, right, like what's inside of that div? It's an entire blog post, right. Might be you could call it that. Something else I didn't quite, so let's see, yeah, we'll just start going through it. Alright, so does anyone need more time? Alright, let's do this, cool. So. Yeah. Just before you start, he's saying, you want separate classes for the blog posts, but titles and paragraphs can share the same classes? I want, so, like different, I want the blog posts to all have the same class, right, like everything that is semantically the same, should have the same class, but they might have different IDs, in fact, if they have IDs they should be different IDs.

Exercise 2 - Solution

So class, this is a blog post, right. Everything inside of this div, contains a blog post, or is a blog post right, so, likewise here, this, is as well, a blog post, because, these are two Tupperwares that contain exactly the same thing, they should have the same label. Also, as a side note, notice I'm using hyphens, that's just kind of like the way that CSS is typically named, you don't have to, like underscores work, you can do camel casing, that all works, and there's nothing inherently wrong with it, like just the standard way of doing it, the way the community has done it, is this way, is separated by hyphens. And you'll find once we start talking about JavaScript it's camel case, and if you do Python, it's underscores, like it's, every language has its little different semantic way of doing things, OK. So, what is this, it's a title of some sort right, so this could be a blog-title, that would make sense to call it that. OK, what's inside of this div, I feel like seriously like we're sitting going through things like OK, this has this, this has this, I mean that's essentially what this process is right now. This contains a bunch of paragraphs right, this contains all the blog posts, so you could call this like blog-text, or blog-paragraphs, something to that effects, we'll go with text, right, because this contains all of the text in the blog post. And then here, class equals what, this is a blog-paragraph, or, if you're lazy like me, you just know that, this is a paragraph, so that's what p stands for, so you can just say blog-p. They have a good rule over at Google, that I've kind of adopted is that, shorten things as much as it makes sense to right, so like in the example that they give is like, if it's called navigation, if you call it nav, everyone knows what nav means, but if

you call it, if you're trying to like label an author and you say atr, that makes sense to you, it's not going to make sense to someone else, so stick with author, like write out all of author. So, I implore you to shorten things as much as it makes sense to, and I think in this sense, blog-p makes sense. So these are all the same right, so we're just going to give them all the same classes, OK. Now the key here is that we're going through the second post, but we've already given these names out right, like we don't, we should not, definitely not reinvent the wheel. So this was a blog-title right, because I believe that's what we called it up here, right, blog-text, blog-p, blog-p, and blog-p, whoops. Eduardo's asking, I thought that a blog post can target the p tag using like. blog-post-p, why use classes on the paragraphs, isn't that using a lot of extra classes, bad design? That's a good question, so we're going to start talking about Brian's opinion versus everyone else's opinion, there's certain people that would definitely say that. There's a couple of good reasons why I feel like it's a good idea to give everything a class, the first one being that, it's nice to just give everything a label, right, like, just assume that everything you stick in the fridge is going to have a label, because when you pull it back out later, you're going to know instantly what it is. The second part of that, and let's just delve into the way the browser works real quick, when you're talking about CSS, I guess we'll address more of that later but there is CSS performance implications, that make it better to use a class rather than to use an ID. It's a minor consideration because the browser's CSS engine is really really fast, but, it makes your CSS faster to use classes. But going back more towards, like the semantic thing, it's just easier to label everything. It's more verbose, for sure right, like you have to type a lot more, but, in my opinion, this makes things easier. And let's say like, I have a blog post right, but now I have, let's say I put a paragraph at the top that's like a TLDR, do you know what that is, too long, didn't read, it's essentially like a little summary, right. So, say I put a little summary text at the top, right, and I said, like up here, like p. blog-summary, oh, sorry, there's a special trick that you can use, I'm super lazy, I don't like typing things, so you can hit p. blog-summary and hit Tab, it will auto-complete it for you. Anyway. Is that just this CodePen, or-- You can get it to do, you can get Sublime to do the same thing, but yeah, it's called Emmet, you have to install it, but yeah. E-M-M-E-T, like I can't live without it, I use it every day so, force of habit. OK, so, discussing what the, what Eduardo, I think asked, is that, now I have two different kinds of paragraphs in here, right, this was one I added later. If I styled everything based on just paragraphs, or the p tag, now these would receive the same style, which is not what I want, right, I want them to have different styles, so now that I have different classes on each one of them, if I add another different type of the same tag later, I can style it differently. If that doesn't make complete sense to you right now, it'll make sense later, so, but that kind of addresses that. And the other thing is like, you are learning like Brian Holt-flavored CSS right now, which I think I'm pretty good at, someone pays me to do it, and I've done

it for a while so, it can't be that bad. Or maybe it can. (laughs) Yeah. My question is about the new container tags in HTML5 so, at what point do you leverage those in this approach, or do you, is the trade-off to do it yourself versus use those? So, that's a good question. HTML5 has a bunch of new, kind of like container tags, and when I say container tags, something like div, one of them that we've already talked about is article. These are actually kind of articles, right, blog posts are like individual little articles. It would be better to call these articles, for sure, so rather than call this like, div, call this article. And again, you would have to change this down here, because now your Tupperware just got a little bit more specialized, right, that, like you look at this, he's like, oh, this is like some sort of complete article in here, that's why you would use it. As far as like, you notice like, it's still rendering this exact same thing over there, nothing has changed over here, that is, it's like, it's just little cues for yourself later to pick up there's like, oh, that's what this does. It also has, like accessibility implications, so like, e-readers right, like when someone that's blind is using the internet, they have things called e-readers that go through and essentially read the internet to them. And so they, these e-readers pick up really well on these article cues, it's like, oh, there's an article right here, this is important, this is what he is going to want to read, right, because the e-reader doesn't want to, like, here's the home page, here's like, it doesn't want to go through the navigation so it will skip the nav part of the site, but it will go straight to the articles. Which is awesome, we want to make the internet accessible to everyone. Great question. So, article, section is another one, like if you have different sections of your website, it's a good one to use. Nav is another one, like where you have navigation, again, the e-readers will just automatically skip over nav, because, who wants to know about your navigation unless they want to use it. Stuff like that. Are you still going to use classes within those new tags then? Definitely. Just so that you can, point to those things uniquely in CSS? Yep, precisely. And, again, I have an article right there, that's called blog-post, but let's say, two years down the road I want a different type of article, like this is a guest article, or something like that. They're now individually labeled, I do not have to go back and change things, like that's, I hate going back and changing code that didn't have to change, right, like if I had just named it better the first place I wouldn't have had to change it. So that, I'm always guarding about doing future work, right, I'd rather do like, five minutes of work now rather than two hours of work later. Good questions, more questions? Cool. So was this, like, painful, was this really shitty to do, it should be, because I don't like it, but, this becomes extremely useful once we get into CSS because like, now that this is all labeled up and all the Tupperware's labeled, we can just like, now send CSS to go and in like, change everything, and it's really easy to do. So again, spending a little bit of time now, will make things a lot easier later. Oh so we didn't do the IDs, I mean what I would do here, let's just go, do it real quick. So first of all, I'm just going to change this back to div. Oh, something you might notice I've been doing as well,

is, you can have multiple cursors in CodePen, as well as in Sublime, which is the thing we had you all install. So like I can select this, and I can select this, and this, and I can change them all at once, so now this is thing, right. I'm just doing that with Command-click in Mac, I'm pretty sure it's like Control-click in Windows, maybe your window button, I don't remember. Anyway, if that's weirding you out that's what I'm doing. OK, so I said, let's go give the blog post IDs, so let's just say, this would be like, blog-post-1, id="blog-post-2", something like that. Something that is, you can guarantee to your future self there will never be another one of these on the page right, so if I said like, blog-post-primary, or something like that, that's harder to say like I can guarantee I'm never going to have a primary blog post on the page again, that's kind of the difference there like, if you say, There's only going to be one first blog post, so that's a good one to use. That make sense? Again, I only put IDs when I absolutely have to use them, on the whole I try and just make everything a class, because, IDs are inherently not reusable, right, because they are unique you can never use them again. Classes are not, and I'm always looking to write one piece of code and have it apply in many places. Which again, once we arrive to CSS, will make more sense.

Cascading Style Sheets (CSS)

What Is CSS?

Let's talk about Cascading Style Sheets, or CSS as I will hereto refer to them as. Sometimes referred to as California Style Sheets but that's just for fun. Makes me laugh, all right, so CSS. Before, we've been talking about HTML, it is the content, right. Like it's actually like the words on the page. Like, if you noticed in the previous section there was no style it was just like black text on a white page. Purely the content on there. This is going to be taking the content and making it like styled differently, so there's like going to be no text rendered from CSS, instead you're going to say like there's text on the page, now make the text red, or something like that right, purely presentational. So, in HTML you're describing your content. In CSS you're giving this content rules and saying, If you are this kind of content you look like this. So, that's why these CSS things are called rules, like they're blocks of rules. Yeah, and this is all called declarative programming, if you want to use the technical term. We're about to get into imperative programming which is what JavaScript is, but here you're just saying like, I declare, like, this is what it is. You're just saying like, Here is the current state of things. Whereas, imperative is much more like if this happens then this

will be this. Yeah cool, so lets talk about some CSS rules. We kind of already saw this a little bit earlier, but like let's just kind of go over this. Let's go back to CodePen. So, now we're going to start using this little CSS block down here and here on, I made a rule, and this rule affects the body tag, right. So, it's saying everything in the body now will have to adhere to these rules, right. So, now everything is red, or everything can be green, or it can be blue, right. So, CSS understands a bunch of colors, like, there's like slate grey, yeah, random ones right. It's got like I don't know, probably like 100 colors, or so, you can also give it HEX values. If you don't know what a HEX value is don't worry too much about it, but like if I say like 999 that'll be some shade of grey, right. Or you can give it RGB values which are red, green, blue, and you can do like 125, zero, zero, or something like that, and that was red I guess, I didn't know that. I guess it makes sense cause it's mostly a red value, anyway. There's like a dozen different ways to give colors to CSS and just, like, I don't know how to generate them for the most part. Like, I just let, like, Photoshop do it for me and say like, "Ask me like what the HEX value is for this, and it'll give it to you. Anyway, for the sake of this class we're gonna stick to like blue, right. Because that's just easier to remember, but I guess what I'm getting at is like there's a way to like fine tune that color if you really want like a specific color. So, that is what a CSS rule looks like. This is, the color part is the property, right. It's the past of the CSS, or the whatever you're modifying that you're modifying, right. So, in this case we're modifying the text color of the entire body, and then the part of it that's like the red part is the value of it. So, the property is color, the value is red. Would that effect every type of sub-tag that was inside the body that had that color property then? Everything in the body will now be red, and we're about to get into why it's called Cascading Style Sheets, right, and the reason for that is that there's kind of like, and we'll talk about later, it's called specificity, but for now, it's enough that that makes every piece of text in the body red. Questions on that? Makes sense right? So, now let's have two different kinds, or two different rules rather. Okay, so now we have an (h1) here that's being effected by this body tag, right. So, everything in the body is now red, but I now have this <p1> that tells the color to be green. So, now this is red, which note right, like if I delete this, what color is it going to be? It's now red, right, so why is that? It's in the body. Yes, it is in the body. So, that top modifier, the body modifier, does effect it, but the P is lower on the style sheet, therefore it is green. So, we'll get into specificity late, like that's, there's other ways to override it but for right now, these two things, the one that comes last is going to be the one that effects it, right. I'm pretty sure, and let me just double check this to make sure that I'm not leading you astray. Okay, and that's kind of what I was worried about. So, in this particular case the, it still does effect it because I don't think the body tag has specificity to it. Anyway, for the most part the one that's going to be lower on the page is going to be the one that effects it. If everything else is considered equal it will take the one that's

last on the page. It's basically because you're applying it to the paragraph instead of the body tag. Like the body is the parent of the paragraph. Right, so it's going to take the closer one. So, I'm not sure how to explin that but. The most specific one. It's the most specific one. And again, we're going to throw this term specificity around a lot, which it's just how the rule engine works, but Mark nailed it that the P tag is closer, right, like it's the one that's closer. This one is the parent so it's further away, and so it's going to take the green one. Kind of an edge case, like I wouldn't worry too much about that.

CSS Better Practices

This is the one we were in before, right? If I add another paragraph what color, it's green right, but what happens if I want like these are different paragraphs, right. Like, this one's like the leading paragraph and this one is a trailing paragraph or something like that, like and I want them to be styled differently, for whatever reason. Using these P colors, or like these P tags, you can't do that. You can't differentiate between the two. So, what you need to do, is that's when these classes, all that stuff that I just made you do, and you were like banging your head on the table, saying, "Like, why am I doing this? " It's going to start making sense now. So, collapse that right there. I now have leading paragraph, and secondary paragraph, and here, why is this so weird? That's weird, okay. leading-p, which is, refers to this one, is green, and secondary-p is blue, right. Does the. just indicate that it's a class? Yup. Okay. That's exactly what... Would you use a. for an ID then, too? Nope. You use a #. Yeah, which I think I talk about later. We'll get to that when we get to that, but notice here that I have a leading period in front of the class, that's just indicating to CSS I am styling right now based on an ID, cause in theory, I can have both an ID and a class that have the same name and they are totally separate right, they're kept in totally different spaces So, if I say like here ID, by the way, this is a bad idea, but just for demonstration purposes, if I wanted to address this what I would say is leading-p, right, and that would address this, not this. So again, contrived example, really bad idea, but that's the concept at play here. So, my question is here if I add another paragraph with no class, no class, I crack myself up, what color would it be? This is all the CSS, right here. It's going to be black, why? It's black because it has no other options, right, like it's, nothing is setting it to be anything else, and the default color for the browser is black. So, it is black. Again, cause no rules are affecting it. So, again we kind of see where this is going, right. Classes are extensively used for styling, and for some people, I'm mostly in this train of thought, but exclusively, I will prefer to never style based on ID, and the reason being is that it's not reusable, and very rarely do I try and write things that are not reusable. Right, like, if I spend all this time writing this beautiful CSS to make like this little widget really pretty, I don't want that to

be unique, right, like I want to be able to rip that out and use it on another page. It's all about code reuse, and IDs are not reusable, for the most part. Cool, so this is kind of what we've been leading up to is there are different rules for styles winning out, right, or stated differently, when two styles conflict, which one gets applied, and we're about to talk about that here in a second. Right, like if I have going back to this example up here, let's just close these, leave this page, leave this page, thank you CodePen. Leave this page, and I'll just leave this page, okay. Like, if this has both leading paragraph, and secondary paragraph, which one wins out, right. That question is not intuitively obvious, so if it's not intuitively obvious, then don't worry because it's not, but that's again, leading up to this thing specificity that we're going to talk about. Can you just go over once again how you differentiate the classes and the IDs in the CSS? Sure, so let's just say this one up here is id = paragraph-1, right. I think I talk about, I have slides on this, so we'll go over it a couple times, but this, it would just be paragraph-1. Right, so the # versus the. It's the #, right this # means it's an ID. Then, I say like color: yellow. It should end up being yellow, yeah. Alright, more questions? Please. Is there a way of setting up your CSS so it's relational? So, you could say, The second paragraph following a div, I want it to have this style. Yeah, so there definitely is. It's kind of beyond the scope of this class, that's a bit more of an advanced technique. They're called pseudo-classes, and like I touch on them, but I don't actually go into them. So, like I'll show you. That's not going to work, anyway. It has to do with this nth-child selector. This is actually incorrect, so don't, but that's what you're going to use, you're going to use : nth-child, and then in parentheses, you're going to tell which one you want. So, it would be the nth paragraph. The nth leading paragraph, well the nth-child that is, yeah. So, it becomes kind of a mess, right, because we have different paragraphs in here, and we wouldn't necessarily be referring to the same one, so. That's the general gist though, is that you're going to use an nth-child pseudo-class. What I would recommend, like while you're trying to, like, wrap your mind all this CSS is just give it a different class, and style based on that class rather than trying to do relations. That tends to be more clear anyway. In fact, I catch a lot of flack at that at Reddit, because I use nth-child a lot, yeah. What's the long form of the. for class? The long form? Like would I just say, class leading-p, or something, instead of the.? There is no long form. Okay, so I have to use the. to refer to a class. Yeah, and it's just one of those things you just get used to. Alright. Good question, though. I like where your heads at, like trying to be succinct. That's a really good quality to have as a programmer. Cool, let's open another CodePen here. Alright, expend out here. Okay, so I believe, yeah, this is just, like that one that you gave all the classes to earlier, same HTML, right. blog-post-p, blog-post, blog-post-title. This is just kind of an example of like, kind of putting some concepts together, right. So, here I have the blog-post, right, which is just like everything, right. Everything inside an individual blog post, I gave it a border, which is like this little black thing around it. I gave

it some margin, and I gave it some padding. So, margin and padding, it's like it's something that confuses people for the longest time, it's okay if you find it a bit confusing. We'll go over it several times, but essentially both of them are used for spacing, right. So, in particular, let's try taking out the margin-bottom, for example. These are now going to bump up against each other, right. So, now that line is solid, but as soon as I put that back, there's going to be just that little bit of space between the two, okay. Whereas, the padding here, that's kind of like interior padding, right. So, I have this blog post, and I don't want, if I take this padding out everything, the text is going to push up against the walls, right, I don't want it to do that. I just wanted, like, a little bit of space to give it some room to breathe, make it easier to read, so again, margin is exterior, padding is interior, right. Again, we'll go over, I actually have, like, a whole slide just talking about, like, all the different measurements that we have to worry about. And also, don't worry, like, you're seeing, like, border, margin, bottom, like, if you, don't try and like memorize these. They're just like, don't try and memorize anything you can google for and find out, like, within like five seconds, right, that's kind of my rule of thumb. Like, for example, like what are all the things I need for a border, right? This is actually the short hand for border, but it's also the way that everyone writes it, right, but you can also do, like, border-color: red, and then all of a sudden, like, everything will be red. And again, like notice it says black right here, and it says red right here, why is it, it's just lower in the CSS, that's all.

CSS Text Properties and Measurements

We're just going to go over like a handful, there's a ton of CSS properties. Ton, ton, ton, ton, and there's good resources on it. We talked a little bit about the MDN, MDN's great for CSS. Again, stands for the Mozilla Developer Network. It's like the bible of front-end web development, like, if it says it on the MDN, it's truth. CodePen, the creator has another site called CSS-Tricks, he has like a glossary of CSS, super useful. The MDN gets really technical sometimes, and CSS-Tricks is really good at like putting it, like, in layman's terms, and giving examples, and stuff like that, so, for CSS I highly recommend Chris Coyier's website, which is CSS-Tricks, okay. So, let's talk about some ways just to modify text, right. Now there's umpteen billion, this is just some of the highlights. Color, think you can kind of figure that one out, it modifies the text color Font-weight, so you can say normal, and that'll be like just normal text. If you say bold, then it'll bold your text for you, right. So, that's kind of what font-weight does. You can also give it numbers, and again, something you'll discover is like there's many ways to do this, and just kind of pick your favorite. Whatever makes the most sense to you. So like, font-weight, if you say 900, I think, that's bold, whereas if, like, you give like 100, it's like really thin text, but not every font has it, anyway we're

not talking about fonts right now. Anyway, but that's what font-weight is, I would just stick to normal and bold, that's what I do. Font-style, you can have normal, or italic. If you want to make your text italicized. Text-align, so like we talked about, like, divs being like containers, right, like if you wanted to bump up against the left side, versus bump up, versus the right side, or you want it to center, that's what text-align is. Text-indent, so like if you have like a paragraph, but you don't want it to, like, move all of the text over, you just want move it the first line over, kind of like, you know, paragraphs have indentations, that's what text indent does. And you just give it, like, "I want you to indent like 20 pixels, " so it'll just indent that first line like 20 pixels. And then, font-size, again this is just like, "I want like really small text, " so I say like, "Be like nine pixels tall, " right, or, "I want you to be really huge, "so be like 40 pixels. " And there are, we're going to go over measurements of, or, yeah, like units of measurement here in just a sec, so. Like right now, I feel like I do that a lot. Measurements. So, CSS has a whole bunch of different measurements for legacy reasons, and you'll see that legacy reasons, reason very often, like it's, the reason why JavaScript has so many, like, weird things about it is that, like, they're trying to be compatible with code all the way back from the 90's, which is just a huge pain, and CSS has kind of the same issue, right. So, there are pixels, points, ems, exes, rems, inches, millimeters, centimeters, and I don't think I'm even covering them all. Just like a total mess, right. Like, totally different ways of measuring things. So, like for font-size right, I can say it's 10 pixels tall, or I can say it's two ems, or I can say it's three exes, or I can say it's four rems, or three inches, or two millimeters, all of those are valid, and they will work. So, which one do you use then, is the correct question. And the answer is there are a few. So, for example don't use points, don't use inches, don't use millimeters, don't use centimeters, those are all like super old, no one uses them. They were more for like print anyway, as opposed to the Web, because CSS has also been used for print stuff, like setting up like magazines and newspapers, and stuff like that. So, and as far as like, you can use pixels, you can ems, and rem is kind of new, but also kind of useful as well. And they have different usefulnesses. So, pixels is kind of the de facto standard for most things. It's not actually a real pixel, just so you know. It's just an arbitrary unit that they've just called a pixel. Which is good, because like a pixel on an iPhone is very, very small right, and so if you said it was 10 pixels on an iPhone, it would be like this big, right. Whereas, like, if you were on like a big desktop, or even worse, like an old CRT monitor, it'd be like this big. Not really, but it'd be much larger. So, it's kind of each, up to each individual manufacturer to make sure that a pixel looks good on their own screen. With that being said, like, text can't be much smaller than like nine, eight, nine, probably seven the smallest, or it's not readable after that point. So, just kind of as an FYI. Ems are relative, I won't talk too much about it, but what it is, is like, you have a font right, and every font has different. An em is the size of one M, like the letter M, right, a lower case M, how like wide it is. So,

it's a relative measurement of unit. So, let's say you have, like, an image, and then when you make the text different sizes, you want the image to resize with the text, so it's always the same, like, relative height or width, or something like that, you would use an em, right, because an em is always relative to the font size of wherever it is. An ex, just so you know, is the height of a lower case X, whereas like an em is the width. Ex is not super well supported, so just stick to em for the most part. Rem, is a root em, so em can kind of be convoluted a little bit because if, like, you have like a body that's like three ems, and then you have, like, a paragraph inside the body that's like two ems it has, like, this kind of like elastic width that kind of happens, right, because it's relative to something that's relative. Does that kind of make sense? If it doesn't it's totally cool, but just so you know, like, if you nest em measurements, they get kind of out of wack, and that's what rem is, it's root ems, so it's like when the page loads, how big was em, right, and then base everything off of that. So, you kind of lose that elastic effect. Is that for kind of responsive type design? Yeah, so responsive design is like when you're trying to make things work on desktop, tablet, mobile, yeah, so it's for that, and ems are very useful in trying to do that. Em, the measurement of M. Good question. Yeah. So, if we want our CSS to be responsive do we need to use a certain one of those, or what? Not necessarily, again kind of, a little bit beyond the scope of this class, but it'll be a combination, just like certain things make sense in certain measurements, right, like you're never going to do, or very rarely are you going to do widths in ems, like, widths are nearly always going to be pixels, but like text-sizes are often in ems, so they're different tools for different jobs, and it's kind of hard to blanket them. I'd say, when in doubt, use pixels. That's pretty good rule of thumb. Default to pixels, and then like if you know better, like, "Oh, I'm doing font sizes, I should probably use ems. " Then, you use that. Okay. Yeah, pixels while not necessarily uniform across browsers, it will be uniform across your page, right. So, you're not going to have one pixel that's wider than the other, if you're in Chrome, right, but Chrome might be a little bit different than FireFox, and in particular, phones are going to be different than normal browsers. Ems, it's how wide like the literal lower case M is in the current font, and it's good for sizes that scale with font.

CSS Boxes

We're going to get into something a little complicated right now. It's called the box model. That's kind of, like, how CSS is based on, it's like boxes that kind of flow next to each other. Also called like document flow, or something like that. So, there are a lot of CSS properties that specifically affect boxes. The most obvious of that is going to be width and height, right, So, if I tell something, You are only 300 pixels wide, it's going to make itself 300 pixels, and if I have text in there the text is going to flow inside of it right, it's going to flow to that constraint. Like, your text

is not going to try and extend outside of it, it's going to stay within it's parent's constraints. It's kind of weird to explain, but I think it kind of makes sense once you start actually doing it. So, border is one of those, like, your telling it to put a border, please, around my box. So, all those are valid borders, one pixel solid black, three pixels dashed gray, five pixels dotted, and then like a hex number, right. Does the border go on the outside of the box, or does it flow into the box? So, that's a really interesting question. Let me see if my next slide is that, yeah, we'll talk about that in just one second. That hex right there is purple, in case you're wondering. Background-color, you're saying like, "I have this box, I want the inside "of my box to be this particular color. " Background-image, so say I like wanted to, like, make like an annoying tiled background, or something like that, then you can use background-image for that. So, let's talk about widths, heights, borders, everything like that. It's complicated, right, so just so you know, if this doesn't initially make sense work on it. Will border properties be covered? I'll mention it right now, yeah we'll talk a little bit about it, sure. Let's talk about it. So, this is the box model. So, first thing you have is height and width in the interior there, right, don't worry about background-color, the background-color will just only cover what's inside of there, right. So, height and width, that's just saying like, "Here's my container, and it is like this big, right. " Then, outside of your width and height, and right now we're talking about content-box versus border-box, I'll address border-box in just a second, but if you don't tell CSS to do anything differently, it assumes what you want is content-box, which is this model that I'm showing you right here, okay. So, you have height and width, and then on top of the height and width, so say I have like 100 pixels and 100 pixels, and that's my square, right, then I say I have 10 more pixels of padding on every side, it will go outside of that and it'll be, you'll have your 100, 100, and now you have 120 by 120, right, cause you have 10 pixels on either side. So, that's the padding, the padding is what's immediately around it, and then outside of your padding is where your border goes, okay. So, say, like, I have like border three pixels solid black, okay, so then I'll have like a border left, a border top, a border right, and a border bottom of three pixels, so now I have 126 by 126 pixels, right. If it seems weird, it is, okay. And then outside of the border, then you can tell it like, "I want you to be this far away from anything else, " right, "So, I want you to be 10 pixels away from "like, my nearest container. " So, you say like, "I want 10 pixels on every side. " So, then it pushes all the content away from it 10 pixels on any side. That kind of make sense? Okay. The model I just described to you is known as content-box, and is a huge pain in the ass, giant, giant pain in the ass. Recently, they came up with something called border-box, that makes a little bit more sense, and the biggest difference is if I say my content is 100 pixels wide, and 100 pixels tall, that includes both border and padding. So, using this, in fact, I'll go to my mouse. So, using this particular model, if we say, "Make this border-box, " width is now up here, and height is now down here, and it includes, so if I say, "I have 100

pixels by 100 pixels, "and I have 10 pixels of padding. " The 10 pixels in padding go inside of it, and now actually your content only has 80 pixels to fit in, right, as opposed to going on the outside and you have 100 pixels to fit your content in, but your box is actually 120 pixels wide. Why is this easier? Because if you're trying to do relative widths, cause you can do like 80, like I want you to take up 80% of your width, having the padding outside of that 80% made it just a giant pain in the ass, right. You had to like, if I wanted to make my, like I had two containers, one was 80%, and one was 20%, I had to account for padding and border on top of that, so I had to be like 78. 3% wide, plus my padding, right. It just made it, like the math just horrendous. So, you can say things like, "I want you to measure things "in terms of border-box, " and then you just say 80%, 20%, it just fits. That's the big difference, it's extremely useful. I recommend just putting everything in a border-box. Particularly if you want to do responsive, like this is huge if you're trying to do responsive web design. I think you mentioned document flow earlier, can you just explain that? Yeah, we're going to talk about that when we get to floats. Okay. Yeah, so it's coming up super soon. Cool, any questions about border-box, content-box, widths, heights, borders? This is kind of like clear as mud probably, right, like it's kind of convoluted to speak on like a theoretical level, and it kind of just makes sense as you keep going with it. And if you mix up padding and margin, don't worry, everyone does, it's cool. Okay, further questions? I have a question about bordering. So, when you put the CSS property values, does it matter, like, the order that you place them? Like say you put background on top, and while you keep on adding it, it gets pushed down to the bottom. Is there a specific order that you have to maintain while? Nope. The rendering, it doesn't make a difference? It makes no difference. Well, the only difference is like if I have like, if I repeat the value, right, if I say padding 10 pixels, and then I say padding 30 pixels right after it, the 30 pixel one will be honored, because it's the last one in there. Other than that, like if I want to have like font-color and font-weight in there, it makes no difference if I put color first, or if I put color second.

CSS Box Questions

Can you go back to the CodePen for this? I think I have one right now, actually, that's an example of this in particular. I want to ask you a question about the code, I wanted to use two properties. Is it this one? Yeah. So, in CSS, in. blog-post there, I added overflow, and I wanted to use a scroll, then how do I get it to? When I added more text, it just made the box bigger. So you have to give it the height, so if I say like height: 100 px, okay, now you can scroll through it. So, overflow applies to the box model, it's just saying, "What do you want me to do if I overflow? " Like, "If I have more content than I can display, what do you want me to do? " So, scroll is one of them, you

can say hidden, and that just says like, "I don't care what goes beyond it, just throw it away. " It's actually not thrown away, it's just not displayed, right. And the reason why it's constrained like this is I gave it a height of 100 pixels, and it's just asking like, "What do I do if I can't fit "all my stuff in here, in 100 pixels? " Whereas if I take this off, it'll just expand to however big it needs to contain all of its stuff, and then it finishes. So, if I was going to style a page, and I wanted the box to be a certain size, but to be the same size on different screens, would I use pixels for that? Or what would be a better way to do the height? Definitely pixels. Yeah. Pixels. Yup, yup, yup, yup. The weird thing about, so if I said like, "10 ems" or something like that, height 10em, something like that. So, first of all, if you don't tell it what to do with overflow, it just assumes like, "I want my box to still be this tall, "and I only want it to take up that much space, "but let my content flow outside of it. " Which is why you see like this awful text thing. So, you have to tell it like, "No, please actually just do hidden, " or something like that, right. Or scroll, or auto, I think auto is pretty much the same as scroll, it gives it the scrolling things right here. Okay so, what's weird about em is like, you know how like old people get on their computer and just like turn up like the magnification of it, right, by if you hit Command-Plus, or Control-Plus, makes it huge. Ems will mess with the width at that point, right, because it's relative to the font size, and they're making the font size bigger. Whereas pixels will stay the same, right, where the pixels don't actually get bigger, but the ems do. That's why, for the most part, you want to avoid doing widths and heights in ems. Wait, wouldn't you want them to get bigger, if someone wants to make the font size bigger? So, it depends right, like it just depends on what you want them to do, but if you have like two boxes that need to be side by side to each other, if they start getting too big, then they're going to stack on top of each other, right, cause they're going to run out of room to go sideways, so it depends on what you're trying to accomplish. Cool. A couple questions from kind of far back. There's one on the CSS, are we going to cover how to debug it with Chrome DevTools? Yeah. (mumbles) and then the flexbox model. We're not going to go over flexbox. Okay. No way. (laughs) Flexbox is kind of complicated, it's brand new, and it's not very well supported. So, we're kind of brave, we kind of use it in some places on Reddit, but most companies cannot, because they have to support old browsers, and we don't care, so, it's not supported in old browsers, so I'm not going to go over it, and it's kind of hard.

CSS Positioning

Talking about, are we going to go into positioning in CSS, learning the best practices for top left, bottom right. Sure. So, something that's kind of even beyond this box model right here is that you can actually use this tag, or property called position to move stuff around, right. So, let's say you

have, like this box, and you want it to be just three pixels to the left, right, you can do position, and you can move it across. So, let's show you a couple things right now, actually. So, let's go back to this Google thing right here, right. So, you're just going to right click on here, and I'm going to say Inspect element, just like what Nina was doing earlier. I'm going to make this just a little bit smaller, no that's probably, we okay right there? Okay, so you notice we got this thing over here, it says Styles. This is actually the CSS that's being applied to that img, right. These are the rules, like, that we were just barely writing, and what is being applied to that. So, up here we can start doing things like background-color: red, pretty cool, right. So, we can actually start messing around with the CSS, and you can actually just say like, "I don't want this to have so much padding on the top. " So, you can just like uncheck that, and it's no longer padded on the top. So, just what this question is asking, is there's another one called position, and position, you can kind of tell it, "This is how I want you to determine "where you are on the page, " right. So, if I say position: absolute, it's saying like, "I'm going to tell you absolutely on the page where I want you to be. " So, in this particular case I can say top: 0px, so now it's on the very top of the page, and I can say right: 0px, right, and now it's absolutely, if you look at this page now, absolutely positioned at the top right of the page. Or I can say, you know, 100px, now it's 100 pixels from the right of the page. You can also go negative as well, right, so if I say negative, it starts going off the page, right. So, there's several ways you can do this as well, like there's position: absolute, which just like leaves it there, right, but if I say, let's take this off says right, if I say fixed now, okay now if I start scrolling, look, it actually like stays in place. It's fixed within your viewport, right, so you have to think of your page in two different heights, right, there's actually like the height of the entire page, including what's not on the screen, and there's also height in terms of like what is your viewport, what can you actually see. And so, position: fixed is in terms of your viewport, position: absolute is in terms of the entire page, altogether. So, even though, your position on the page is changing, your position on the viewport, like, doesn't change, cause your viewport is just your viewport, it's always where it is. Okay, now there's position:, sorry, relative. We'll do, leave that 10px, or something. Position: relative, come on, relative. Okay, so relative is kind of an interesting one. Because what relative says is like, "Okay, go ahead and put yourself wherever you were going to put yourself, "and then from there, go here, " right. So, in this particular case, I have position: relative, it puts itself like right here, and I say, "From there, go 10 pixels down. " or, "From there go 100 pixels to the left. " Let's just leave this in terms of, okay. "Go to where you're supposed to be, " so in this particular case, let's just uncheck these, right, so that's where it would be if nothing was affecting it, right. Then, "From there, I want you to go right 100 pixels. " Does that kind of make sense? A little bit? So, I have now set all of this, right, and like you now know how to reposition things. People that are new to web development want to abuse this, right, they

want to use position: relative, or position whatever for everything, that's not necessarily the best way to do it, because then you start kind of getting into like weird flows of your document. We're going to go over document flow in just a second, but suffice to say that it makes putting elements next to each other really difficult. So, if you notice here, like the one that I checked off that Google already had on there was position-top, or padding-top, right. You want to stick more to padding and margins to get things together, because then it like make things take up the space that they're supposed to take up. So, in this particular case, we now have this padding-top on top of it, and now nothing can flow into that padding-top, right. It's taking up that space so nothing will flow into it, and in your case, it's totally seamless to you, like, you have no idea that that's padding, right, but if you look at it, if you put a background color on it, then you can tell that it's there. But, I mean, you might ask like, "I want it to have background-color, "like, just on the logo, but I also want it to be down that 112 pixels. " So, if you just change that to margin, it'll have the same position, but as you know, background-color doesn't go out to the margins, it just covers the padding, so, yeah, that's how you do it. That's why you would use margin. Do we follow? Does that make sense? He had, kind of, a list of more questions, but I think you may have covered most of them, I guess. So like, when to use floats versus position properties. So, we're about to go into floats. Okay. It's the next thing we're going to talk about. And then, I guess yeah, so if you're going to cover that it might cover most of that, and then just kind of general question on what CSS frameworks do you recommend? That's an interesting question, so there's kind of this concept of a CSS framework, we haven't actually gotten to frameworks, we're actually going to talk about jQuery, which is a JavaScript tool set, the difference there is minor, so don't worry about tool set versus framework. Anyway, it's essentially code that someone else has written that makes your job easier, right, like don't reinvent the wheel if someone has already, you know, invented it. So, the one that's popular is Bootstrap, we're not going to talk too much about it, but let's get rid of this, right. Like some guys over at Twitter and GitHub made a bunch of like really useful CSS, like this looks really nice, right. They just give you a bunch of styles and says like, "Here's a bunch of good ideas, why don't you go ahead and stick to that? " It's great for if you're just starting out, like if you just want to get a site out really quickly, and you don't want to worry about the styles, just throw some Bootstrap on it. With Bootstrap, all you have to do is you throw on, like, "I want this to look kind of like, " I'll just show you, right. Come on. Let's see. "I want like buttons, " or something like that, right, so, like, you think of a button, like you think of like a normal, really ugly, like grey button that your operating systems has, right. If you don't want it to look like that, all you have to do is throw on like button, button default, and bam, it just looks like that. Like, you didn't have to write any CSS, someone wrote CSS that you can just throw on there. Or like, "I want it to be, like, a warning button, " right, so you do button, button warning, kablam, right. Suddenly, you

have a pretty button that has like a nice little hover state, so if, like, I hover over it, it gets darker, if I press it, you know, it looks different. Right, that's kind of the idea there. So, this is a CSS framework, it's called Bootstrap, there's a couple of them, there's like Foundation, and Inuit, some other ones. Which ones do I recommend? I recommend none of them, if you're actually going to be like building a big product that you want to control how it looks. If you're just like building like a proof of concept that you want to build in a weekend, you don't want to worry about how it looks, Bootstrap is great. It's the only one I really have any experience with. I know the guys who do it, they do an awesome job. They're all designers, so it all looks really nice. Yeah, Bootstrap, if you want to try something like that, that's the one I would recommend.

CSS Shortcuts

Let's just open another CodePen right now. CodePen. io, just kind of wing this. Alright, so, we're going to have a div here. In case you're wondering, I do lots of shortcuts, cause that's just, like, second nature now, so right now, I just wrote (div), right, your browser is aware of which one its like waiting to close, so if you hit, I don't know how to do this on Windows, so I apologize in advance, if you hit Option, Command, Period, it'll close whatever tag you currently have open. Nice little trick, or like I was showing him earlier with Emmet, you can just hit Tab, you can type out whatever the tag names, and hit Tab, and it'll also make it as well. Just fun little tricks, alright, so. We're going to do a div rule, okay. It's going to have a height of like 100 pixels, and a width of like 200 pixels, and we'll just give it like a border: 2px solid red, right. Okay, so now we have this box that we can start toying around with. Okay, we also talked a little bit about border-box, right, so you're wondering, like, how you actually do that. So, let's just do that real quick, and you just do box-sizing: border-box. Border-fox, what does the border-box say? Okay, cool. So, that * right there means apply this to all of everything, right. Apply this to everything on my page. I would recommend never using it, except in this one use case, right. This will make everything into the border-box frame of mind, instead of the content-box, which is the old way, and is awful, and don't do it. But yeah, just grab this snippet and just throw it on every page that you work on. That's just my suggestion. Okay, so. There are shortcuts for a lot things, so I can, for example, I can say margin-top: 10px, right, and well, let's maybe, like, do like 50px so it's a little bit more pronounced, okay. You notice it's moving down the page a little bit, we'll just do like a background-color of orange or something like that, okay. I'm a designer, I make pretty things, okay, so. We have margin-top right now, and you can do that, and then you can say like margin-left: 200px, right, and it's going to move it in 200 pixels, and let's just put some text in here, I am here, right, and so now it's going to have a little text in here, right, and like. I don't like it that it

pushes right up against it, so I can say padding-top: 50px, right. Can you make the CSS so it shows up top. Am I good to just like close this one out? Like, it's just going to be that div and some text, so that's all it's going to be. Okay so, padding-top, now it's moving it down 50 pixels, or I don't know, like 30px, or something like that. Okay, so as you notice, like, if I have to add, like, margins to the top, to the left, to the right, to the bottom, it's verbose, right, like I have to write like four lines of it. So, they made it easy and lazy for you. So, let's just do that real quick, so you can just say margin, right, and let's say I want to have the same margin on my left and on my right, and I want to have the same margin on my top and bottom, which is a really common use case, right. Top and bottom will have the same margin, left and right will have the same margin. So, if I say 10px 20px, or maybe, let's do 10px 200px, so it's like really pronounced, right. Okay, 10 pixels is going to be top and bottom, 200 pixels is going to be left and right. Or if I just like delete it, then it's all sides. All sides now have a margin of 200 pixels. Where'd it go? Maybe 100px. That's interesting. There it goes, it just took awhile, alright. Okay, so, it's cause the server's taking a long time, that's why. Okay so, but that's what that margin does right, like it's saying, if you have one number it knows like, "Okay, apply this to all sides, " if you have two numbers it says, "Top and bottom, left and right, " there is a way to do three numbers, don't worry about it, but, because no one ever does three numbers, but there's one for four numbers that's top, right, bottom, left. Which would be opposite for you right. But if I say like 50px 20px 30px, right. Top is 50 pixels, right is 10 pixels, bottom is 20 pixels, and left is 30 pixels. It's just shorthand, if it's super confusing for you, don't use it, right. Just like write it out, and that's totally fine, but just be aware that like that's, that exists, that's what's happening, you'll probably, eventually get to a point where you just want to do it, because you're lazy like me. So, that applies to margin, that applies to padding. Padding works exactly the same way. There's a bunch of like little shorthand stuff for it. In fact, we can do that if you want. Padding, we'll just do like 10px 20px, right, and then now we'll have that top and bottom 10 pixels, left and right 20 pixels. Questions about that? The stars, sorry, is that just like a Unix star, where it's saying everything? Yup, wildcard, that's what it's called. It just means apply this to everything on the page. Every element ever that put on the page, put box as in border-box on it. And I would again, just recommend putting it on every CSS you do. It'll make your life a lot easier. And again, I would not recommend it for anything else, right. Like for example, if you wanted all of your text to have a different font, or something like that, I would recommend putting that on body, as opposed to the wildcard, and say like font-family:, which is what you do if you want to change fonts, and you can say like monospace, or something like that. Change the font, or Verdana, or something like that. Something like that.

CSS Order and Specificity

Would that apply to every single div on an HTML? Yup, so if I put another div in here, pretty weird, but let's do that. Other dive. And if you change it to div class, would that be different, or would it still apply? It would still apply because it's still a div, right? Well, what if you wanted to change it, what if you didn't want, like, those two boxes to match? Because you have this div in CSS, put it in the class and everything. That's a good question, so let's do one for like this-class, and say background-color: green, okay. Now we're starting to get into specificity a little bit. This-class, so kind of interesting what's going on here, right. So let's, first of all, you've got the HTML, right. Like, this one is a div with no class, this is a div with a class that says this-class, okay. Alright, so, collapse that, let's talk about what's happening here. This div does apply to this, right, like this is still a div, so it's getting all that style applied to it, but then we have one down here that says, this-class, and it doesn't contradict all of those properties, it only contradicts one of the properties, which is the background-color. So, now you have a conflict here on the background class, so it needs to ask itself which one wins, right? Like, which one gets applied when these two are conflicting? This is this idea of specificity that I keep referring to. This-class is more specific, thus it gets applied. So, it's background color is green, and so therefore, it would be orange otherwise, like if I took this property out, it would go back to being orange, but because it is more specific it gets applied. So, let's talk a little bit about specificity now. Let's go back up here to our HTML, okay. I'm going to have another div underneath it, I don't want that one. Actually, you know, why not, it's kind of fun, we'll put the other one right here. I am also here, okay. So, we now have four divs, and I'm going to give this one ID, let's say, call this one another-class, and then down here we're going to have one that's going to have another-class, but it's also going to have id = "special", or something like that. Okay. And let's actually, let's go and change these back out. So, this is going to have no class, this is going to have one class, also, no class, just so it's kind of visually apparent to you which is which, this is going to have two classes, and this is going to have two classes + id. Okay, so no class, one class, also no class, two classes, and two classes, and an ID. And let's expand this one out, okay, so. These ones that have the class on it are all getting the green background, right. The ones with no class are getting the orange background. So, if I do, well first of all, let's do like, if I do another-class, right, cause some of these, these ones with two class, all have another class on them, right. So, if I say background-color: pink. Nothing changes, right, why? Because this-class, and another-class are the same specificity, because they have one class on them that match. So, it's going to pick the one that's lower in the file, the last one that gets declared. But let's do this-class. another-class, make sure that there is no space there, that's key, that's a very key difference, and say background-color: blue. Two classes, both of them down

here, now have a blue background. I can put this above, like, sorry. I could put this above it, right, it wouldn't matter what order in the file it came, because now it has two classes that it's referring to and therefore is more specific, okay. Now, just for fun at the top, like right up here, I'm going to say special right here, and I'm going to say background-color: yellow, or something like that I guess, I guess there's not a lot of huge difference there. Purple. It's background color is now purple, why? It's because IDs, no matter how many, and it's actually not completely true, but for really all intents and purposes, no matter how many classes you have in there, an ID is always more specific because you're addressing just one. So, ID always trumps classes, even if there's like 30 classes on it, way more than is actually possible. I think actually if you have 256 classes it will finally beat an ID, but that's just like totally and completely impractical, and will never happen, or if that's happening to you, you really need to come back and watch this class again, cause that's not good. Okay so, we've kind of demonstrated to you these principles, yeah. So, there's kind of a hierarchy right, like if div gives, let's say, a red background they all have red borders, sorry not background. And it only replaces the things that are, so it's kind of pulling from both of them. Yes, so it's cascading. Cascading. It is a Cascading Style Sheet, this is exactly why it's called CSS, a Cascading Style Sheet, is cause the effects cascade down. Kind of like a waterfall, right, that's the idea here.

CSS Specificity Best Practices

I like hesitate to even show you this because it's so bad. I'm going to show you a really bad idea because you'll probably see it somewhere. Oh my God, everything just went orange, right, but it's from div, like what the hell happened? So, there's this idea of important, that you can throw it on a property and it will override anything, it does not matter how specific it is, it will override it. Now, there are even more special things for accounting for like what if two importants conflict, but if you have importants conflicting, you have a serious problem with your CSS. So, do not use important, like I really can't think of very many use cases where you should be using important. What you should do is like don't use important, and rather like, if you need like this one with like no class, to like override it, give it another class, until you have enough classes, until you can actually like override it using specificity, right. So, the reason why here I said don't use a space, if you put a space there, then you're implying nesting. So, like for example, let's just come up here and say all of these divs are inside of a like class="parent-div", okay. Move all these in a little bit, come on. Okay, so now all of these divs are now nested inside of a parent div, right. What can I actually, this is why you don't style based on divs, right. Cause now it's all like weird looking, but guess what we can do now. We can say I think it's inherit, so it should lose its background color,

and border-color: transparent, and height: auto. So, what have I done here? Really bad, right like I'm writing bad CSS for you right now, but... You just like hid it away. So, the issue here is that you should not style divs as just a tag, right. You should not give divs height or something like that, that's awful because you're going to use divs all over your code, and you don't want to have to override it every single time, right. Rather what I should've done is I not should've styled on div, and I should've just give them all the same class, but what I did here is background-color, I told it was like, "Something else is going to give you a color, and I want you to just totally ignore it. "I want you to get whatever it was supposed to be in the first place. " That's what inherit means. Border-color, I said, "You have a border, just make it transparent. " So, the border actually is there, it's just not being used, and then the height I said, "I want you take whatever space that you need to take. " That's like the default for height, right. Like I have 10 divs inside of me, each one of them has 100 pixels, so I should 1, 000 pixels right, cause I have 10 divs that are 100 pixes. It just says, "Size yourself accordingly. " That's the point of height: auto. So, that was a lot of bad CSS just to teach you that. But notice like all the spaces here, anyway. I said all of this to say like if I wanted to say parent-div div, and I said color:, I don't know, I think cyan's a color. I want background-color though. To get rid of that. Okay, so now no class, also no class, and one class are all getting overridden, but two classes, and two classes and ID are not being overridden. So, this is where we start getting into like these weird specificity edge cases, okay. So, let's go over it. It's overriding no class, right, because now it has a class, right. I'm referring to the parent class, like I don't have to necessarily be referring its class, but as long as its, it essentially just takes a tally of like how specific is this, and from there, you know, apply these styles to that. So, right now it is more specific than div, because there is one class, plus one tag being referred to. So, that's why it's overriding one-class, or no-class, and also no-class because it is referring to a class, right. Now, this-class is being overridden, this one class one right here, because it has one class, plus a tag, so it has a parent div, plus a div, that are being referred to, right, so essentially it has two things now being referred to. Does that mystical magic make sense? So, I'm going to give you kind of a rule of thumb now that should work 95% of the time, I'll just like. So, just picture it like a, I'm just going to write it up here, like a four digit number, or we'll just say a three digit number. So, the first one is how many IDs does it have referencing it, how many classes does it have referencing it, and how many tags does it have referencing it? So, for example, if I have one that has just one tag referencing it. So, when I say tag I mean like , <p>, , something like that, right. And then it would be zero, zero, one, right. That would be like div, right. But if I had like class, like this-class being referred to, it would be zero, one, zero, and that would be like this-class. Or, if I had like this-class and another-class, it would be zero, two, zero, and that would be like this-class. another-class. So, what I'm getting at is like, 10 is bigger than one, so 10 wins, right. So, I would

need a lot of, it's actually not even that, but like it's just kind of a rule of thumb. You would actually need 255 tags to override one class, you need 255 classes to override one ID, but again what I'm getting at is if I had this-class. div, so it would be like zero, one, one, it would be this-class, actually it would be like div. this-class, right. So, that's saying like, "I want this class, but only want this-classes that are on divs. " And so that would have 11 specificity, so it would beat 10 specificity. I'm using this mostly as a mnemonic for like if you have classes and divs, not that if you have 10 tags that it overrides one class, right. That's not what this illustrates. Again, then, like, if you have one ID, it doesn't matter how many classes, how many tags referencing, the ID is always going to be more specific. And if you have an important, right, just think of it like this. Like, it's 1, 000, right, like it just overrides everything. And there's actually even one more on top of that. So, let's go back to our HTML. Right here, so you can actually, what's called inline styling. So, if I say style = background-color: pink, or something like that, except let's put that one on the ID, just to illustrate it. So, if we go down here, notice that now it's pink, even though I'm styled based on ID. Inline styles even override IDs, but they do not override important. Now, you might tempted to use inline styles. There is never, not even one, don't even do it, do not inline your styles, it's just the worst thing you can do for yourself. Because now I've split my CSS among my HTML file, and my CSS. So, if I have to go change, like I see, like, this was a pink div, and I need it to be grey, I don't know if it's in my CSS file, I don't know if it's my HTML file, and it takes me, now, forever to find where is this style coming from? Keep all of your CSS in your CSS files, they should be separate. Do not inline your styles. We've got a few questions queuing up here. Alright. Do you think you could put a link to this CodePen out? So they can. Yeah, let me save, okay. And I will just, let's see where am I, right there, add a new slide. Is there a way of using the Google Development Tools to look at what styles are being applied to a particular inline? Absolutely, we looked at it, but we'll look at it again here in just one second. Let's see, text. Yeah, you gotta make it not black, good job. Black, red, I don't know, something like that, there we go. So, it's not pretty, but it's on the slides now, it's slide 67, is that showing up for everybody? Yup. Okay. I think one of them was just kind of doing some different things with these, how would I go about making these boxes appear beside each other, and then like fixing them to the footer? Fixing them to the footer is a trick. It requires some some kind of advanced techniques, so I would say look that one up on your own. Okay. We're going to talk about document flow and floating here in just a sec, so that question will be, I think it's literally the next set of slides, so we'll talk about.

CSS Resets and the Developer Tools

The other question was on Resets, on why it's useful or not. So, there's such a thing as CSS Reset, it's just a little tiny, in fact, let's just go find it. I like Meyerweb's CSS Reset. This is just a really smart guy, don't quote me on it, I believe he is a Mozilla employee, Eric Meyer. He just gives you like this really general CSS style sheet that takes all of those default stylings, like you know how, like, the sorry, lost my train of thought, you know how the headers are like bold, and different sizes, and stuff like that, it takes all of that default styling, and just does away with them, and so everything just starts from like a very normal, very sane place. Why is that useful? Because different browsers have different default styles, which is really frustrating. Something you're going to find out, if you go in further front-end development is supporting different browsers is hard because they all do things differently. So, what this does is it gets all the browsers on equal footing. Now, what do I recommend about it? I recommend using them, you should use something like this, in fact I recommend this very one. But like, notice here's like margin zero, padding zero, bording zero, font size 100%, font inherent, vertical line, base line, like, just very sane defaults that kind of bring all the browsers onto a level playing field. So, just search for like Meyerweb CSS Reset, you'll find it, or you can even search for CSS Reset, and I think it's still the top result. That's the CSS Reset, it's just like a little tiny style sheet. I use Normalize. Normalize works really well, yeah. All of them work. And they're all pretty similar too. Cool, and then we asked a little bit about like the Developer Tools, let's just like check it out for a second. So again, inspect element, right. So, I have also no class right here, see that its background color is cyan, border red. So, you can kind of see the cascade effect here, right, cause it says parent-div div, and this one shows up, and then if you scroll down, this one is still trying to do, like trying to enforce this background-color, but it's being out-specified by this one, so it gets crossed out. So, if I like undo this one, if I uncheck it, now it's going to be back to orange, right. That kind of make sense, what's going on there? Keep in mind that as soon as I refresh the page, everything that I do in this like inspector goes away. Or like another interesting thing, like if I look here at this two IDs plus classes, and I inspect element here, notice that it says element. style, that's the element's own style, so I did like the inline style, that's what this element. style is, it's pink. And again, if I get rid of that, it goes back to purple, what the ID was styled on. Good question. I think that, explain how you can get the boxes to line up like using float, I believe, and in vertical align, if you can explain that, is what they're asking. So, vertical align is another tough one, we'll talk about floats here in just a sec, it's very easy to like, so I have like this two classes right here, right, but notice that it's left align, so if I go down here to like the div one, right here, so this is the one that's div, right, and I add a new thing that says text-align: center, notice all the text now is centered in all of these, right. Now the next question you might ask is, "How do I vertically align this? " Vertical aligning is tough because there's, like five different ways to do it, and there's no one way to do it. It really

depends on what you're doing, where. There is a property that's called vertical-align, but like if you know, like I'll put it in here, vertical-align: middle is what you would put, but notice it has absolutely no effect. Why does it not work? It only works on tables, so that has to be like, it has to be put into a table, and then have the table displayed in the middle. It's a total mess, I would recommend probably never doing it that way. There's a lot of like hacks to it, like making the line height equal, like there's 1, 000 ways to kind of like hack it together, I'm actually, I'm not going to go into them because they are really hacky and difficult to kind of wrap your mind around. So, I would say there are lots of resources on Google, so just google that when you need to cross that bridge. I'll give you the reason, the reason is that document flow in, because like the document's trying to fit as much as it can on one line, so it's for the most part agnostic for how tall things are, and it's really worried about how wide things are, so everything's in terms of how wide it is, and so because of that, when it creates a div, it does not necessarily know how tall its going to be, when it first creates it, because what happens is that it goes through, and has to render everything inside of it, it would then have to go back out and center everything on the way back out, which is not really performant, anyway. The short answer, the tl;dr, the summary of that is that browsers try to be agnostic to the height, and they just worry mostly about the width.

Audience Questions

Like if I wanted to center the boxes, this is actually not too hard. I'm going to go back to styling in here. If I did, so I have margin on here, so let's, so here I have a 10 pixel on the left, or on the right, and then 30 pixels on the left, if I just change this to auto, and I change this to auto, now it's actually going to center those, right. Auto is just saying that, and this only works on left and right, it doesn't work on up and down, "Please center myself in whatever my parent is, " right. So, like if I have parent-div down here, if I say width only be like 40% of what you can actually be, then let's just say like a border, we'll say like blue, or something like that, I guess. I didn't want that, anyway that kind of makes the point. So, you can see right here the div only takes up 40% of its width. And these are all centered within that. So, the margin auto centers it within its parent, right. So, these ones are all being centered within the parent. So, if I said here margin zero, then that would be pushed to the left up here, right. So, these are all centered in here, not necessarily centered on the page. It's what you would expect, but it's kind of weird to verbalize it, right. That make sense, you follow? Someone commented that flexbox is trying to solve all these problems. Yeah, so flexbox, which we do use a little bit on Reddit, is trying to solve some of these vertical centering woes, because I mean it's been a problem for like 15 years, right. And it just seems kind of silly that we have the same problem that we had 15 years ago. So, there's a new model of displaying

called flexbox, and centering things in flexbox vertically is trivial, like it's really, really easy. You just have to say justify-content: middle, or center, or something like that. But, that's for another day. Is it box-flex property? Flexbox I don't work in enough to actually remember all of the things, but you have to say like. It's part of CSS? It's CSS3, actually it's even beyond CSS3, it's like 3. 1. So, you have to say display: flex, as you can see it just totally up, messed up everything, sorry. Box-flex. But they haven't even agreed on, like they, I say like the browser manufacturers, have not even agreed what that property should be called. So, it's hard to use right now, but CSS-Tricks, if you want to look at it as a really good tutorial on how to use flexbox, if that's something you're interested in learning. I would highly recommend it. That's what I refer to every time I need to use it, cause there's like 30 properties around it that are useful, so. It's difficult though, I'll throw that caveat out, cause you have to worry about like flex direction, and justifying content versus justifying items, and stuff like that. Other questions about what's going on here. Specificity make a little bit of sense to you? Like, the more specific you are, the more like it'll be applied. As a general rule, just something I'd throw out there, is don't necessarily rely on order to solve your specificity, so like you can have like two of them that refer to one class, and then you can just put the one with, that you want applied below it. That's kind of a poor practice because if anyone reorganizes your CSS, which is kind of a common thing to do, it could mess up what you were intending. So, I would just say, play it safe, use two classes instead. Another thing that I catch a lot of like crap for this, so this is a highly contested subject, I never style on ID, ever. Why? It's because if I need to override something with an ID has, it's really hard, right. You can't, cause you can only have one ID on an item, right. So, you can't override based on having two IDs, so you either have to have like an ID in a class, which is really dumb, right. Or you have to use like important, which you shouldn't use, whereas like if it was just a class, and it had two classes, guess what you do, you throw in another class, you have three classes, and you're done with it, like it's just, it requires very little rewrite, if you're doing everything based on classes. The only time that I will style on ID, is like if I can guarantee that like this thing is unique, like I'm never going to want to style this some other place, so let's just throw an ID on it, and be done with it. IDs are a specificity nightmare though, that's why you stick to classes. I don't ever style on tags, or very rarely style on tags, and I... Yeah, very rarely, like I'll style like links, and stuff like that, but I'll very rarely style like divs or something like that, never that. Just stick to classes, you'll have a happy life, if you stick to classes. Okay, other questions? Alright. Let's see, I believe, we went over this, I think. Let me just make sure. We did. One just came in, just kind of talking about like Sass. Versus LESS and stuff like that. Yeah. So, spreading CSS is a bit of a pain. It's kind of archaic. Like there's no idea of nesting in CSS. So, let's say like right here, like if I wanted to do like blog-post, but I wanted to style all the h1s just inside the blog-post. You would think this would make sense,

right, like I could just nest the rule inside of blog-post. Right, so I'd have blog-post, and then inside of blog-post, I'd have h1, right, and I could say like color: red. This doesn't work in CSS, CSS everything has to be flat. If I wanted to do all the h1s inside of the blog-post, I would have to do blog-post h1 color: red. Right, so this idea of nesting doesn't exist. So, some guys, some smart guys got really sick of doing that, and they wrote another language called Sass, there's one called Stylus, there's one called LESS, there's one called, another one. Anyway, those are the three big ones, Sass being the biggest, and then LESS and Stylus are kind of fighting it out for second place. They wrote this language that you write this kind of more concise CSS, and then you compile it, so essentially you run it through another program which then spits out normal CSS. It's very useful, we use it at Reddit, we us Sass. I would recommend as you're starting out, just kind of wrapping your mind around CSS, cause Sass and LESS introduce some really more, like, advance complex ideas, and so it's better to just start out writing vanilla CSS, and then once you kind of like, once you get sick of writing CSS, then it's a good time to jump to Sass. Cause then you're like, "Everything I hate about CSS "has mostly been solved by this, " so. But anyway there's a lot more than just like nesting, but that's one of the things that people really like is nesting. Variables, right, you can have variables in your... Yeah, actually FireFox now allows you to have CSS variables. So, that actually can work now in CSS, which is pretty rad. Because like, for example I say I had this color like, I don't know, say, my favorite one is f06d06. f06d06, it's orange. So, this color right here, say I was using it all over my CSS. Right, like that was like my brand color, and I needed to use it everywhere. It's really annoying to have to repeat that, and then say if like my brand color changes, like I'm Airbnb, and now I'm suddenly red instead of blue, right, you would have to go back to literally everyplace in your CSS and change it, but if you use a variable, right, you change it one place, and tada it's changed everywhere. Which is kind of nice, so that's why variables are useful, they are in Sass, and they just came out in modern browsers, which means we can use them in like five years, when everyone's caught up.

CSS Document Flow

Document flow, everyone's been waiting for this, this is like the big moment, just kidding. You're going to be very disappointed if this was your big moment. Okay, CSS has a concept called float, right. So, right now if you saw all those divs, they're all on different lines, right. But let's say like I have a section on my webpage here, and I want another section on my webpage to be right here, next to it. How do I do that? So, there's a thing called float. You say like, "I want you to take, so I want you to move yourself "as far to the left as you possibly can, " and then you tell the thing below it, "Please move yourself as far to the left as you possibly can, " and so they'll start stacking

each other until they can no longer fit on the same line, and then what they do is is they wrap underneath it, and then they start floating next to each other again. So, how many like Pinterest users in here, right? Pinterest does a really cool thing, it's not quite float, it's actually more advanced than float, it's been notice like they kind of like put everything right next to each other, right. That's kind of what we're talking about here. So, I say as many divs next to each other, as many block level items. So, floating will only work on block level items. So, like you can't float a span, that doesn't really make sense because a span is already kind of like moving next to each other. What does flex, I mean it's almost like that (inaudible) a little bit. Yes, like I see where you're coming from, and like it does do that, but it is so much more complex than that. It doesn't align it, cause when you float it, it actually moves it exactly the way it was in the previous state. Okay, yeah I can see why you're saying that. I hadn't thought of it like that, so. We'll go through some examples, and then go through it some more. So, in fact let's just jump into a CodePen here. Your chat there, kind of, wondering how does Pinterest do what they do, like using grids, or? Pinterest does it with JavaScript. It's actually quite complicated, and people have tried to replicate it, and they did a really bad job. There's like a plugin called Masonry that kind of replicates it, and it's awful. So, they have a bunch of really smart people that got paid to do it. Okay, so, good question though. I believe this is just all the same blog HTML, you can look over it, but it really ends up being the same as what we've been looking at previously. Now, I have blog post one and blog post two being next to each other. Okay, and now I have floated them next to each other. So, like notice, like this one's longer than this one. Yeah, so let's talk about how that happened. CodePen is like doing this weird thing, anyway, whatever. So, here I just, like, changed the font-family, changed the color of the text. Apparently, 333 is a shade of black that's less offensive, it's less stark. So, did that, made the background-color gray, like if you notice the background, or we can change this to be like red, or something like that, right, like whatever. And then, here I've defined it to be a width of 45%, so you can define widths in terms of like percentages, right, it's going to look at its parent container and say, "Like, how wide are you? " And it'll say, "Okay, I want to be 45% "of whatever your maximum is. " So, in this particular case, we're saying like, the width is, parent is the body, so the body is the width of the entire page, "I want to be 45% of that. " And then I said after that, "Please give me 2% margin to the right. " So, like what's cool about this is like if I keep, well I guess, it's kind of, but it kind of re-sizes itself as the page re-sizes. It's kind of responsive design, kind of not really, cause as you can see it messes up right here. But let's actually talk about that for a second. So, if I go too small, what happens? It wraps, that's what this floating thing does, is it tries to fit as much crap as it can on one line, and once it does, it just kind of shoves it down. This can, if your things are not uniform height, this can cause some weird behavior, right, because if I have super long div right here, and I have things wrapping, it's going

to wrap underneath those things, but not always because a floated element cannot be higher than a previous floated element, right. So, if I have thing A, actually hold on, I have a good example of this on my CodePen. This one right here. Luckily, I did this, so I have two fatties right here, and a skinny right here, right. And these three skinnies, notice that these skinnies are not pushing up all the way to the top, right, it's cause this fatty right here is actually higher in the HTML, and so these skinnies can't actually push past the fatty, like that fatty is as far high as they can go. And difference is why they're going to the left and right is like these are being floated to the right, and these are being floated to the left. All of this is to illustrate floats can have some weird behavior. This typically would not be a problem, cause what I would do is, I would just float everything to the left and you typically would not float to the right, and then float to the left, and try and like have that pushing up behavior, but that's kind of what I was getting at there. Okay, so again, like if I had changed this to be like 50%, they're going to wrap, right, because it's trying to have 50% width, and then 2% margin to the right, that's 52%, or 104% if you try and slam them together, do the math, 104 is actually bigger than 100, so it cannot fit on the same line, therefore it wraps. Okay, so, something kind of interesting that we can do, if I say float: right, and move this back to 45, check it out, now blog post one is on the right, whereas if I say float: left, they're going to swap. Kind of sweet, right. So, it just has to do with the order that it's happening, so it's going to take blog post one render it, and then say, "Okay, shove yourself to the right, " and then it's going to take blog post two, render it, and then shove it to the right, and that's kind of how floats work. Questions about that, does that make kind of some sense? In the chat room they want to know how to make them equal heights, so if, I think by just putting that in there. Flexbox makes equal heights really easy. (laughs) Equal height is another pain in the ass. So, well the easiest way is, like, if I say height: like 200px, right, and then say overflow: scroll, or something like that. Okay, now they are equal height because I've actually predefined their heights to be 200 pixels. Now, if you want them to be both responsive, and the same height, that's a trick, I'm hesitant to show you cause it's a little weird, arlight we'll show, like someone might find it interesting, if this blows your mind, let's say just ignore it. Okay, so. This is like, you'll find that like with CSS, like, it's so old, and so broken that you kind of have to hack around it sometimes, so I'm going to say height: 0, so these are all of a sudden going to disappear, right, cause they're going to have no height, then I'm going to say padding-bottom:, and I'm going to say like 50%. So, what happened here? Notice that they actually are the same height now, and they kind of size dynamically as well. What they're doing is they're maintaining aspect ratio, right. Like you think about your TV that's like four by three, or 16 by nine, it's essentially what you're doing it here. Padding-bottom is, despite that fact that it's actually a vertical measurement, it's measured in terms of its width, and so if I'm telling padding-bottom to be 50%, I'm instructing it that you're

supposed to be 50% of whatever your maximum height is, or maximum width, sorry. That doesn't make any sense, it's totally cool, that's like advanced CSS, don't worry too much about it, but it's kind of cool, right. A lot of, like, our website is built on this principle, and it's total hack, right. Okay, well that was a tangent. I have a question. Yeah. So, when you're considering your your margins, how can I say that? Say you have 200 for width, right, and everything you do under, say, under your blog-post, so everything you do in your blog-post, and say that's 200, does everything need to add to that 150, what did I say 200, whatever I said, I'm sorry. That's okay. Does it need to add up to that width size so that you can have enough room to get all your content, images on that page? Okay, so let me rephrase what you're asking, and see if I understand. You have a parent container that's going to be like 200 pixels wide, you're asking me if I want everything to fit inside my container did they need to add up to 200 pixels? Right. The answer to that question is it depends on what you want. There is sometimes that you want your content to flow outside of it, right. So, have any of you seen those like full bleed, like Kickstarter has one, I think at the top, or even redditgifts does, so let's just look at redditgifts. We have this full bleed image, right, so sometimes you might want those images to flow outside of it, so you're only looking at part of your image, right. And that's how they achieve that technique, is they actually have it flow outside of it, but its parent actually can't contain the full width of it. If you want everything to fit on one line, then yeah, it needs to be less than or equal to the parent's width, does that answer your question? Cool.

CSS Float Tips

Let's talk about the great collapse for a second. h1s, let's just say like we do something dumb, and we float our h1s, nope that's actually going to do okay, and then p float: left. Do I have an over, I have an overflow on this, that's why. And I have a height, okay. I'm trying to get something bad to happen in CSS, and then I have the height: 0, padding-bottom. Well, I'll just talk about it for a second because I can't seem to get it to do it real quick, anyway, so, there's something that people often refer to as the great collapse in CSS that when you start floating elements inside of a parent div, the parent div, when you try and style it, collapses. Like it'll have no height, and so like if you have a nice background color, or something like that, it's not styling correctly. This is called the great collapse because you're putting floated elements inside of an element that is itself trying to figure out how tall it is, and so with floats inside of a parent div, what you need to do on the parent div is you need to give it some sort of way to reset itself, it's like you'll see a lot of times something called clearfix, or people will put like overflow: hidden in weird places in the code, that's to get that parent div to reacquire its height, and all it needs to do is just like reset

itself. So, if you notice that you're using floats, and your parent div is collapsing, throw something on like, if you put overflow: hidden it'll adjust itself, or look up like clearfix. If you search for clearfix it'll be like the first thing that'll come up, so. A couple questions here, one they're just kind of questioning what nowrap is and understanding white-space, and how that works. Like the CSS properties? Yeah, so CSS does, or not CSS, but the browser does its best to fit all of your text and everything that it can on one line, sometimes you don't want it to wrap, so you use something like white-space: nowrap, I think is what it is, and then it will like, even if you start going outside of your div, it'll keep, it will not overflow your text, in fact I think if we went back to this and I said like blog-post-p, and I said white-space: nowrap, I gotta remember, I mean, it's not one of the most common ones. Yeah, notice like now I have a super long page because all the text refused to wrap itself. It's typically, not a desirable thing right, but there are occasions like you have like a head, or something, and it's wrapping yourself in a weird way, so you can use white-space: nowrap to prevent this wrapping from happening. And then they want to know about the clearfix. Let's see, I hadn't prepared a talk on it, but let's clearfix. This is the one that I typically use Nicolas Gallagher, super smart guy, so again search for clearfix. And what he does is, alright let's see if I. So again like, you have like a CSS reset, this clearfix is just a little utility class that you can just throw on things to just fix itself. So, let's do, let's just open a new CodePen. See if I can get it to do it. So, I'm going to have div class=, like, "parent" and I'm going to have. child=2 this is the Emmet stuff, maybe not. Alright I guess it doesn't want to do Emmet for me, child, let's see div, div, div. Okay, so we have a parent and two children, we're going to have. child float: left, and we're going to have parent, let's just collapse that one, background-color: red. And here we're going to have height: 20px, width: 20px. Okay, so, and let's just do this background-color: green. And we'll have like padding: 30px. Yeah, that will ruin it, that will actually fix it. Okay, so we have these, now let me save this in, I'll put it in so people can see it. Just add a new slide here. Okay, slide 71 will have that on there now. Okay, so I have these two things, I now no longer have a background of red, right, cause you would expect the parent to have a background red. Maybe before you get into too much more, he just kind of wants to know why do we need clearfix at all, what's the history on it? So, clearfix, I have now demonstrated the effect that I'm trying to demonstrate, is that you would expect this parent to have a background red between these two, these two squares. Let's just make these like 200, so it's more, right, you would expect this background right here to be red, but because of the way floats work, and parent div's heights are determined with floats, it doesn't know what to do about the background, and so it just collapses everything. So, the parent, if I put a border on the parents, it'll be like a little line at the top, border: 1px solid blue, right, so now notice like we have this blue line up here, right, totally weird because we would expect it to just like, "Why don't you just cover everything that the children cover? " Right, and it's

the way the floats are determined, so there's a couple ways to kind of fix this, and the fastest way is you can just say on this parent is overflow: hidden. Overflow: hidden, low and behold it fixes itself, and now it has a background. It's weird, and it has to do with the way that overflow's calculated, I'm not even really sure, like, the mechanics behind it of why that works, just that it works. So, let's talk about clearfix, that's a hack, what I just showed you, but if I throw on here clearfix. So, I just copied some CSS from Nicolas Gallagher's website, we're just going to throw it on here on the bottom. Right there, don't worry about what it actually is, I'm not sure why this works either, but now if I throw this class on here cf, it also will fix it. It's a little bit more clear on what's going on here, like why that exists, like you put overflow: hidden on there, people don't know that you're trying to clear your float, right. In this particular case, you see cf on there, it's like, "Oh okay, he's trying, he's doing a clearfix here, "he's trying to get this float width stuff to work. " That make sense? Cool. It's going to happen to you, and you're going to be super frustrated when it happens, so just kind of refer back to this.

Exercise 3: Styling a Page

Alright, so this is the grand finale for the CSS portion of this class. So, go ahead and open this Pen, and then I put an image in there which I have embedded on the next slide. If you want to see it that way too. So, I've given you a bunch of HTML with, of course, cat pictures, your welcome. And I want you to style it, and you're going to have to use floats, and stuff like that, and I put the box sizing thing on there for you, which I explained earlier. Do you specifically want four and four, or does it depend on how wide your browser? I want four and four. So, I put the solution on there, don't cheat yourself. Try and do it the first time, bang your head against the wall, that's how you learn. And then we will go through it together. So again, the whole exercise is open the HTML, do not touch the HTML, and just use CSS. Questions on it? So, some hints, you're going to have to use text-align, you're going to have to use that margin: auto trick I just showed you, actually no you won't, you won't have to use margin: auto, you will have to use padding. You'll have to use float. Yeah, that's about it. Let's see, I can make this presentation again. I'm sorry, what's the challenge again? So, this CodePen right here, I want you to open it, and I want you to make it look like this, which is this, the one that I've embedded here. So, all those boxes, make them line up left to right. Yup. Without touching what? Don't change the HTML, just change the CSS. Question is: Is it good practice to use text-align on an image? Yes, absolutely. So, yes, you can use, because images are considered an inline type element, like the spans and stuff like that, you can use text-align to center them. Which is pretty cool, and that's awesome. Nothing wrong with that. You also have to use height, height and width for sure. Did you get it, good job. A little extra padding.

That's cool, padding never hurt anybody. My CodePen has the code on the top, like HTML, CSS, and JavaScript, and then all the display on the bottom. Is that like a setting that you customized? You can only do that if you have an account though. Yeah, do you click here? I'm trying to remember, there's a setting for it, it's down here in the bottom right. So, you can change it and flip it so it's on the other side. Right there. Cool. Any questions about like something that's not clear about it? Or anything like that? Alright. I'll give you like five more minutes, something like that. Another little tidbit that my wife taught me about this is like never ever feel guilty about googling anything, right. It's not cheating, like for the first like five years, and probably even longer of your career, you will code by Google, right. It's feels like taking pieces of code that other people have written and just kind of squishing them together. Totally cool, right, don't memorize anything you can find in like five seconds. Yeah, my wife felt it was like a shame, right, to google something, like it was just like cheating on a test, or something like that, but like no one's watching, or anything like that, like it's just get the job done. Which is why I like Stack Overflow, if you're not familiar with that, it was like a huge help. Another hint is probably use percentage widths, as opposed to pixel widths. Right, if you have four in a line, what's 100 divided by four? Right.

Exercise 3: Solution

First of all, this box-sizing: border-box, that we talked a little bit about before, but this makes this so much easier, right. Because the first thing we're going to do is we're going to take these, actually I need this open, don't I? Because I don't remember what anything's called. So, picture-group, right. Or not picture-group, rather, yeah it is picture-group. Because everything's in a picture-group, yup, okay. So, just make this a lot bigger. Come on. There. Okay, so. picture-group, and we're just going to give them a width of 25%. Okay, now they're all like tightly compacted together. We're going to say border: 1px solid black. Okay, and now we're going to float them left, okay, now everything's on the same line. But it looks kind of dumb, right, because they all have these like different heights, right. So, I mean how you want to do height: is kind of up to you, Like, you can just do something like, I don't know, 200px, or something like that. That's kind of an easy way to do it, except that's not nearly high enough, so let's try 400, 300. Maybe like 330. Alright, good enough, right. And let's give them a bit of padding, a little bit more breathing room, so we're going to say like padding: 5px, or something like that. Something you're kind of going to learn as you go forward with this kind of stuff is that you just kind of poke around, see what works, poke around a little bit more, see if it works better. Like, it's just kind of tooling with it, see if it works, it's kind of the hacker attitude. Okay, so now everything's on the same line, we wanted

it to be centered, so we're going to say text-align: center. That's pretty much it, isn't it? Yeah. And then you can make it wider, and notice, like, it all, even when you get super small, which is bad, but whatever, you might want to say overflow: hidden, right, so it doesn't like flow into the other one. But there you go.

Exercise 3: Final Questions

So, with the overflow: hidden, should you be able to scroll down to see the rest? Or is it just like. So, if you did overflow: auto, or scroll, like then when you got down to like really small, it would let you scroll. Okay. Whatever you want. I hate scrolling, so I typically don't like to have scroll bars. Questions about this? Did we, for the most part, get this? Yeah. I just have a question about the form, actually, in the CSS file. Somebody once told me that there is a certain way that the attributes need to be listed in Like the order that they come in? in style, yeah. I mean, so the answer is technically no. Technically, like the browser doesn't care if you say color: green, and then padding: 10px, or padding: 10px, then color: green. But it has more to do with somebody else reading the code. And that's just kind of a preference thing. Okay. That's kind of like a stylistic thing. Like if your team adheres to one, or if you want to adhere to one, then go right ahead. I didn't know if there's a standard. There's not, there's no standard to it. I'm pretty lazy about it, like I just want to, like, write my code as fast as possible and be done with it. I haven't found, like, ordering them a particular way has been helpful at all. I found it took me more time than it was saving me. I guess that's kind of my litmus test, is am I saving more time than I'm spending, right. So like, if it saved me a ton of time down the road, then I'd be all over it, but the fact that I'm spending a ton of time right now, for no savings later. A question in the chat came: How can you use inline-block instead of float? So, we haven't really talked about inline-block, we've talked about, like, block level elements, and we've talked about inline elements, right. We have not talked about inline-block. So, kind of, more going over, like, there's like many ways to skin a cat kind of idea, right, there's many ways to do this, so let's do it without floats. So, now we have these around the same different lines, right, and now we can say, I hate that, we're going to say display: inline-block. And then, we're going to have to say margin: 0 I think. Yeah, so I mean, we're close, probably if I said 24%, it would do it correctly. So, what inline-block is doing is it's going to treat it like an inline-element, right, like an image, or a span, or something like that. You're essentially, you're changing the way that the CSS is going to treat it. Inline-block is kind of a hybrid between block level elements, like divs, and inline elements, like spans, and there's kind of this hybrid approach called inline-block that it's going to treat it like an inline, meaning it's going to try and treat it essentially like text, but it's going to respect width, and height, and padding, and all that

stuff. Because if you're trying to give like spans like a width or a height, it just ignores them, right, it doesn't know what to do with them. But inline-blocks, it will actually try and respect your widths and heights. And I'm sure there is a way to get rid of the spacing here between them, I can't remember off the top of my head, so, floating's easier, I think personally, but it's kind of up to you, whatever you want to do. Good question. So, kind of since we're on the subject, there's kind of another thing, display has a lot of useful things, you can say display: none. It's an easy way to hide stuff, right, I just hid all the picture-groups. Let's get rid of this and say, go back to float: left, get rid of this, put that back in. Okay, and so if I said like, "I no longer want "any images ever to show. " I would say img display: none, okay all images are now going to go away. The browser treats them as if they are not there. Do they get sent to the browser from the server still? And then just not displayed or? Are you talking about the images themselves? Yeah. It will not actually load the images. The browser is smart enough that it has display: none on them, it will not actually load those images. So, as soon as you actually make them visible, then it will actually go out, fetch those images, and bring them back. But yeah, so they will exist in your HTML, but they will not affect document flow. As you notice the text pushed up, right. So, if let's say I wanted the text to actually stay where it was, right. So, there is a property called visibility, and you say hidden, and what that means is, "I want you to still exist, "I want you to still take up space, I just don't want you to actually show anything. " Right, in this case the image disappeared, but it is still actually taking up that same space. Kind of weird, but it occasionally is useful, so. Yeah. What's the order that the HTML and the CSS is loaded, cause isn't it walking the don, while it's loading the CSS? So, HTML is always loaded first, definitely, then your HTML, or sorry your HTML's always loaded first, then in your head element, you should have all of your CSS, like we showed you that head block, right. That's where you stick your CSS, and then that will actually be applied right there, so that you don't get that, what's called flash of unstyled content, then after that, if you put your JavaScript at the bottom of your body tag, like you should, then the JavaScript is then executed. But really the HTML's always loaded first, like that's just a given, and then after that, it's up to you how everything is loaded. You can put your CSS in different places, and have it load different ways, just best practices dictate that you load it at the top. That's your question? Yeah. Because it's parsing it while it's loading, right? You know, I don't remember to tell you exactly. That seems to make sense to me, but I'm not going to say it is, because I don't remember. But that would be a great thing to go read. Other questions? So, how did we do with the exercise. Do okay? Get it for the most part? I'm seeing mostly head nods, which makes me really happy. I think this is like one of the most fun parts of the of web development, is like, kind of like, molding the HTML to, kind of like, fit your vision of what it should be. It's very, for me at least, it's a pretty creative process. So, a couple questions about the choice of techniques. Does it matter the width

of the browser at all for either of these two techniques that we used, the border or the float? So, it certainly does, so imagine like taking this page, and putting it like iPhone size would be like, like this size, right, which is just like a terrible experience. Like, what you would want this to do, and here we're getting back to responsive web design. Responsive design means that, like, between this width and this width of the browser display it like this, and when you're on desktop do this, when you're on, essentially, like a tablet sized browser do this, or, and when, you're on like a mobile sized browser then do this, right. I'll give you like a really basic example of that. We do it on redditgifts /feed, no it's /marketplace/feed there we go. Okay, so this is like a full sized browser width, right. And then I'll use, let's put this on the side. Okay, so full sized like desktop browser, and then once you start getting down to like tablet sizes, then it reorganizes itself and has like three, and then when you get down to like an iPhone size, it has one, and eventually two, right. This is the idea of what we would call like responsive web design, it's like different CSS rules that apply to different sizes. Now, the next question you're going to ask me is like how do you do that? Show us the code for that? I mean it's not actually too bad, it's all in Sass though, so. A little bit different than what we've been talking about, but they're called media queries, and off the top of my head, because I always write them in Sass, I don't actually remember how to write them in raw CSS, but you say, "Between this width and this width apply these rules, " right. Like, these rules only have affect between these widths, these rules only have affect between these widths, right, and then once you get outside of those widths, those rules no longer have affect. With the full frame? Of the browser, right. The viewport? The viewport, precisely. Yeah. So, that's kind of what we would, did I get away from my, what the hell happened? Alright, I guess we went to a new, there we go. Alright, so this is again, like what I would do on this is like I would stack it differently at different widths, right. You would have four on one line, so they would take up 25% on desktop, and then on tablet you would have them take up 33%, and then they would stack three next to each other, and then on mobile they'd stack up like 50%, right, and have two of them on one line. But kind of more of an advanced technique, there's other, like cool, Frontend Masters courses on that kind of stuff, so. Definitely would recommend checking those out. Good question. Other questions? Okay so, in HTML, going back up to HTML, you reference the kit, hold on, when you need to reference something from your own file, is it different, like what's the difference in importing that into HTML? So, if I have an image, in my own, this is my own picture, and I want to import it under Image Source here, how? How do you reference it? Yeah. We'll go more into that tomorrow when we're talking about Node, but the sum of the story is, in fact, you have a whole slide, I think Nina has a slide on relative and absolute paths, don't you? Yeah, I'll let Nina explain it, she's way smarter than me, so. Tomorrow, okay. Yeah I think it's tomorrow. Okay.

CSS Conclusion

Like HTML there's a lot of fun stuff to CSS, it's a very expansive topic. Like, we only scratched the surface of all the different properties you can use. Like, seriously, single digit percentage of all the different properties that are available to you in CSS. Like, there's--yeah go ahead. Opinion on floats versus position? Yeah, I mean it really depends on what you're doing. There's no hard and fast rule, like we have all these tools because they fit different things, and it's kind of like the conundrum of the hammer and the nail, right. Like, if you have a hammer, everything looks like a nail, like you just want to hammer the shit out of everything, right. In reality, like, you have you like a whole toolbox because if you encounter a screw, right, like you need to pull out your screwdriver, and then use that. Like, if you're going to use a hammer on your screw, you're going to have a bad time. So, my opinion is use the one that's correct. There are certainly wide uses for both of those properties. Just personal opinion, like if floats apply, then I would say use floats, right. Like, if it makes sense to use a float, I would use a float over a position. Position, for me, is kind of a last resort, if there's no other way I can reposition something. Good question though. Cool, anymore questions? Yeah. Just curious if you have advice on next steps. Are there websites that kind of guide you through, try to recreate this? You know, cause it's kind of fun, and challenging. For sure, I know front-end masters has additional CSS courses. The big one is responsive web design. Yeah, and that. That's actually a really really good course. That's really fun, yeah. I guess I just meant more like practice exercises. Like someone giving you a comp, and then, like, you like coding it up, like a comp, when I say a comp, like a PSD, like a Photoshop Document, and just like coding it up to look like that? Yeah. I don't have a resource off the top of my head, but they exist, definitely. And even more so, if you want to make a bit of money, there's lots of people that will pay you to do that. But yeah, that's actually a really good practice, it's a really common pattern in web development is here is a image, right, like I designed in Photoshop, please code this up to look similar to it, so. There's a lot of free PSDs, you know, Photoshop templates out there that you can, you know, pull down and then just as a practice exercise, you know, see how close you can make it, you know. Ye=up, for sure, that would be a great way to practice your CSS. And at that point you have, you know, a little bit of portfolio, you say, "Here's the PSD, and then I turned it into this, " and you also have, you know, you can ask better questions to people who are more experienced than you, because you're actually struggling with something real versus, you know, in theory. Yup, totally, good question. That sounds great. Yeah, he just was asking about a real world example where you went with absolute, instead of using the floats. Yeah, let me think about that for a second. So, why absolute positions can be really, really useful is that you need something on top of something else, right. So, I have this, like, image, and I want to

overlay text on the image, right. Because image avoid tags, and you can't actually put text inside of an image, you actually need to use position: relative, or position: absolute to move that that text on top of the image. There are other techniques you can use to accomplish that, but let's for sake of argument say that. So, I was asked a question: Could I elaborate a little bit more of position absolute versus floating? He wanted like a real world example of where you would use that absolute. Yeah, so I just like, while we were away for a second, I coded up a little example right here. So, let's just say like I was making, like a cute, cat picture with like inspirational sayings, or something like that, right. So, I have this image tag, there is no way to like, because images avoid tag you can't put anything inside of it, right. So, I need to take something that's not inside of image, and I need to overlay it on top of the image. This is where position: absolute is useful because you can move things where they would not otherwise be. In this particular case I need to put text over an image. So, what I'm doing here is I, let's just go ahead and delete this stuff. It's kind of a half-baked idea. Okay, so I have this image, and then next to it exists this paragraph, which is, I want that to be over the kitten picture, so what I've done here is I've put position: absolute, which is saying like, "I need you to position this absolute on the entire page, " right. "I'm going to give you coordinates in terms of the entire browser. " So, I'm saying, "From the top of the browser, " so everything is from position right here in the top left, "From the top of the browser, go down 180 pixels, and go from the left go 180, or go 80 pixels, " right. So, sometimes people find those like top, left, right kind of annoying, right. You're not going left 100 pixels, it's from the left you're going 80 pixels, so you're moving something to the right. So, in this particular case, from the top I'm going down 180 pixels, from the left I'm going, so again, understand, like, we're using this to overlay it on something else. This is not the only case when you would do this. There are other cases when you'd use position: absolute, but I would say this is a common use case, when you want to kind of stack elements that otherwise would not stack. That kind of make sense? It's not super common, like I don't use position: absolute like daily, right. Cool. Yeah. Is there a way of positioning text inside of an image that, you know, might be different sizes, or resized on the screen? So, say I just want this text, you know, centered on this image regardless of what size the image. Yeah, so what you kind of get into at this point is you have to go in and do background-images. So, you have to, essentially, allow CSS to govern the image. So, you're going to say background-image:, and you say URL, and inside the URL you're going to put this picture. So, let's just get that, Cut it out, Delete this, right. And now I'm going to have this background-image in here, so and I'll have to make this like width: 100px, height: 100px, or rather 400, sorry. Okay, so now I have this div and it's background is an image, and now I'm free to kind of mess around with it as I like. So, now I can say like p color: white, text-align: center, right. And I'm just going to cheat and say like margin-top:, or actually, you can even do another one, which is line-

height: 400px. And we'll do like text-size:, or font-size: rather 30px, or something like that. So, that's how you would kind of like, and I would probably recommend going more this route, than the other way I showed you, this is easier to kind of wrangle, right. The issue is that sometimes it's difficult to know those URLs ahead of time. Sometimes like you have to wait for your server to know what the image is to stick it there, that's a problem that I constantly struggle with. But, yeah, this'll work. What's the line-height do? So, this one of those hacks I was just telling you about. First of all, this will only work if the text is short enough to fit. So, if I do this, it's going to look weird, I think, maybe not. Maybe it's just not going to wrap. Alright, cool, whatever, anyway. So, line-height, line-height is, like, you know how you like whenever you were in high school you wanted to double space all your stuff? This is the same thing that governs those spaces between. So, if I had like line-height: 2, it says like whatever your normal line-height is going to be, double it, so it would be like double spacing something, right. Where in this particular case you can actually tell it how many pixels high you want the line spacing to be, a hack for vertically centering something is to set the line-height the same as the height of the div that it's in, that it needs to be centered in. In this particular case, I already know that the the height is going to be 400 pixels, thus I know I can set the line-height to be 400 pixels and it will vertically center it. That only works if you know what the height is going to be ahead of time, which I would say most of the time you don't, so. That kind of answer your question? In a very verbose way. Yeah. Cool, more questions about this kind of stuff? Alright. So, going back here. Yeah, like don't worry if this felt like drinking from the fire hose, don't worry if you have not memorized every property that we talked about. It's all super available online, like, not only are you not the first beginner, but there has been like two decades of beginners that have started on you, so the internet is just full of people saying like, "How the hell do I do this? " Right, so every question that you're going to have, guarantee it, you are not the first person that's had this question. So, if you just say like, "How do I vertically center a div? " Or something like that, someone else has asked it, and someone else has already answered it on Stack Overflow. Stack Overflow's going to be one of your best things, I recommend going through CSS-Tricks, Chris, the guy that writes CSS-Tricks puts a lot of research into what he does, and like will give you a very layman's kind of terms for it. And then the MDN, if you want the technical explanation, like what is it supposed to do, what does it do in this edge case, then the MDN is a really good place to go.

JavaScript

An Introduction to Programming

(theme music) We're going to start talking about JavaScript, and this is purely an introduction to programming aimed towards people who haven't really done it at all. So, how is JavaScript different from HTML? Well, HTML is purely presentational, there's nothing dynamic about it, there's nothing that changes. We talked about client-server, server sends back HTML, and there it is, the browser displays it. JavaScript is different, JavaScript is a programming language, and what that means is it's interpreted by your browser. So there are variables, and things change, and there's a line of execution. So, what can we use JavaScript for? And there are a ton of different applications, it's been around for quite a long time. Fun fact, it was created in ten days, which is like that for programming language. So we can use JavaScript for anything from creating something like a popup, have you ever got one of those on an annoying website, to something like a full-fledged video game. The basic composition of JavaScript is a statement, so each line in JavaScript is an instruction, and when the browser reads it, it executes that line of code. So, for the duration of these slides, I'll be showing code in this different font and syntax highlighted, so that'll help you differentiate between what's code and what isn't. So, what's this weird semicolon thing going on at the end of a statement? So, if you think of a JavaScript statement like a sentence, then the semicolon is a period. It means that this line is done, and you'll be seeing a bunch of these. The other thing that you'll see in JavaScript is comments. So, one line comments start with two forward slashes at the end. And comments are not code, but can coexist with code. So, they're not executed by the browser, they're only read by humans, so only meant for yourself or other people, so notes. You can also do multi-line comments, and they are a forward slash and the star, what you want to write, and then they're ended with a star and another forward slash. If you have syntax highlighting in your editor, you'll see that comments are a different color. So, some rule of thumbs about comments, don't try to overuse them in code that you're sharing. So, if you have a really complex piece of code, feel free to comment it, but don't just put comments in willy-nilly if other people are going to read it because they should be able to infer what's going on from your code. If you do go back and update your code and you have a bunch of comments, make sure you update them so that if someone goes out and reads your code, they're not reading the comments and getting the wrong picture of what's going on. So, there are a few ways that we can see results when we write JavaScript. The first way is that super annoying popup, the second way is we can output directly to the HTML page, the third way is that we can display it in that console that we talked about in the beginning.

Exercise 4: Seeing Results

So, I have my JavaScript here, I have three different kinds of statements that will output results, and they're all commented out. So we can uncomment them one by one, see what happens. So when I redden this code, get a super annoying dialogue box. When I uncomment this out, and if you don't want that dialogue box popping up every time, you can put that comment back in front of it. This will write directly to the HTML page. And this will log to the console. As you see, though, nothing happened, and that's because we don't have our console open. So if we go to View, Developer, and the JavaScript Console, we'll see our output right here. Nina, Eduardo is wondering your opinion about having to deal with JavaScript being disabled, and developing a non-JavaScript version? It's always a risk, but my thought on it is, you know, unless you're writing code for like, you know, corporate use, in that case, you know JavaScript is going to be disabled ahead of time, if your code is just out there, it's like, you know, if a person chooses to have JavaScript off, that's their choice, there's really not much you can do about it, there's some like, mobile browsers that might not be able to parse it, and if, you know, those are your users you should try to take care of them. I'll say that at Reddit we do not have a non-JavaScript version of the site, we assume that our users have JavaScript because we've looked at our analytics and much over 90% of them, in fact like, closer to like 97% of our users have JavaScript enabled. Yeah, if you don't, you might be using an ancient computer that might not be able to parse the site, anyway, or you're doing it as a choice, in which case. Some of the browser manufactures are even talking about allowing, or just removing the ability to disable JavaScript. Yeah. So, does everyone see that hello console down here? Okay, for several of the future exercises we're going to be using this console.

JavaScript Variables, Variable Types, and Reserved Words

So the way that we store values in JavaScript is with variables, and you can either declare it and initialize it on one line, which is what you'll usually see, or you can declare it first, and then initialize it somewhere down the line. Until that value is initialized, that variable, it has no value. And really important to consider, don't forget the var, like right here, this var y, don't forget that keyword when you're using a variable for the very first time. If you don't have it in there, and you use that same variable name again, you might accidentally override it, and we'll talk about scope a little bit more in detail later. So Brian mentioned some of this before, how it applies to CSS. So, for CSS, classes, for example, you name them word dash word. In JavaScript, we name our variables in camelCase, so that's lowercase word, and then a word with an uppercase first letter as the next word, and so on and so on. Variables either start with a lowercase letter, a dollar sign, or an underscore, but make sure they don't contain any special symbols like exclamation point or

pound, hashtag, whatever you want to call it, that's not supported and you'll get an error. So, some words are reserved by the JavaScript language, and they can't be used as variable names either. For a beginner, this is kind of a weird error, because you'll get an error, and your code looks right, and you aren't really sure, you know, what's going on. But, these are some pretty common words that you can't use as JavaScript variables, like in or do, this, for, there's a pretty long list of them, and after class you can go through it, there's a link on the bottom of the slide. So, variables can be of different types, and JavaScript is cool because it's dynamic. You don't have to let it know what type of variable is going to be before you use it. So, strings are contained in two quotes, it's any sort of characters, numbers that you won't be using mathematically. Numbers don't have the quotes, and they can be with or without the decimal. So, if we go back into our console here, Sorry, I'm trying to make it bigger, here we go. Can everyone see that font? So, let's try to declare some variables. So, this variable is called name, and it's a string, because it's in quotes. If you press enter, you'll see undefined, which means that running this line of code didn't return anything. Let's make one more variable called age. This one's a number. So, in JavaScript, if you want to see the type of variable, you can use this type of command. And it's a special function, you're not calling it with parentheses, which you'll see later on, so just note that this one isn't going to look like some of the other things you're going to try today. So if we do type of name, we're going to get back a string, type of age is a number, and I'll talk about functions later on, but let's create an empty one. Let's call it myfunc. So, totally empty function, it takes no arguments, if we do type of, myfunc we'll get back a function. So, another type of variable type is a Boolean, and Booleans can have two values, true, which is yes, or false, which is no. There are also special types, if you declare a variable and you don't set, you don't put anything in it, the value of that is undefined, if you want to explicitly empty a variable, you can set it to null, and that'll clear up the value.

Exercise 5: Writing JavaScript

Let's write our first JavaScript, we're going to use one of those annoying popups, and we're going to create a variable, which is our names, and then we're going to popup a message, so let's open this CodePen here. So, here's my stuff, I set a variable to my name, and it's popped up an alert. You can go ahead and comment that out, if you want to keep it at the top for reference. So, I'll give you guys a few minutes to do this exercise. So, add a second string, have it be a variable named name, add a third variable called age, set it to your age, and then create an alert that will pop up your name and your age. They're asking, how do you bring the console up in CodePen? Same, you would go to View and Developer, and JavaScript Colsole. But, if you output your

results using an alert, you won't need to look in the console. Part of the browser, not part of the webpage. What's that? The console's part of the browser, Yes, not the, webpage. Exactly. Thank you CodePen for being helpful. So you can use either a single or a double quote, but to stay consistent, I'm going to use double quotes. It doesn't matter which one you pick, as long as you pick one and stick with it. Are numbers just numbers, or are they integers or floats, or? Depends. Okay, so it does have that capability of distinguishing. No, it doesn't. Everything is a number in JavaScript, Yeah. which is a float, so doing math in JavaScript sucks. Yes. (laughter) Don't use Roscoe for math, but you can declare a different kind of There are libraries that can help (mumbles) yeah, everything's a float. So, Okay, let's do this exercise. So I'm going to use Brian's name. And I'm going to say he's about 105. Don't forget your semicolons like I did. Okay, so we're going to use an alert, and we're going to say name, and use a plus sign to put together strings. We're going to type is in here. Brian's 105 years old.

JavaScript Operators, Expressions, and Comparisons

There are several operators in JavaScript, addition, subtraction, multiplication, division, and modulus, which is the leftovers of a division. Some of the variables can be used with multiple types of variables, such as we just saw, addition, we used a plus to put together two strings. But if those two variables were numbers, we'd actually just get back the result. Let's do that. So, I'm making two variables that are numbers, and if we did x plus y, we would get 8. But, if I did var x is 5 with the quotes, var y is 3 with the quotes, if we did x plus y, we'd get 53, so, something to watch out for. So, expressions, we can also store the results of expressions in variables, which is nice because that means we can reuse them for later. There's two special kinds of operations, and that is a plus plus, and a minus minus. So, the plus plus means increment that variable by one, the minus minus is decrement it by one, so. I can do x equals x plus one, but a shorthand for that is x plus plus. Same with minus, x equals x minus one, the shorthand for that is x minus minus. And adding and subtracting one is such a common operation that you'll see this pretty much all over the place. So, sometimes we want to compare two values, and JavaScript provides these comparisons. There's the double equal, which is equal to, and the triple equal, which means equal in value and type, bang equal is not equal to, bang equal is not equal in value or type, and let me go over that really quick, because it's important, you really won't see triple equals in any other language. So, let's do this, let's say we have a variable called string x, which is five. Make another variable called num x, which is number five. If we do string x is equal to num x, we'll get true, which is like, "Hey, weird, what's going on, they're not really equal. " So, JavaScript has this triple equals, which is like, are the variables equal in both number and type? So, if they were both

strings, would they match? Same with Sorry, same with not equals. So, we can do x not equals to y, which is true, they're not the same. Which is the inverse of x equals y. There's also greater than, less than, greater than or equal to, and less than or equal to, which is useful for mathematical operations. So we can ask JavaScript, "Is three less than five? " it'll say, "Yeah, duh, of course, " right? We can do greater than equal etc, etc. And the results of all these operations are Boolean Values, so it's either true, or it's false. Here's a question back on the operators, Yes. do they do anything on other types of size numbers? So I think like, string said you concatenate them? If you concatenate strings? Yeah, can you use, like, the plus operator? Yeah. What do they do on other types, was the question, I believe? So what they do for numbers is just adding them, what they do for functions, I think is kind of outside of this, the scope of this class. Yeah. Let's find out. Actually, I think it would call two string function, yeah. Actually, I'll do this demo when we talk about functions, I'm getting a little bit ahead of myself. Nothing useful, I think is the answer to that question. I'm not sure what happens if you call less than on a string, it might error out, or not. I'm guessing this might be alphabetical. Yeah, so that checks for order, does that make sense? So the answer is that, depending on what the types are, it may or may not do something useful? Yes, yeah, so, So for example, multiplication of two strings probably doesn't do anything useful, or does it, is it a syntax error I guess that was really. I think it returns not a number, Oh okay, so it is an error. That's not going to stop your program, NaN stands for not a number, which is essentially just that, it's nothing useful. The fun part about JavaScript is like, it does have this loose typing, so you don't have to say if something is a number or a string, which makes things easier, but then the downside is, you know, sometimes when things are out, they can be hard to find, because unlike a compiled language, there's not complaining, JavaScript is just like, "Yeah, whatever I'll do all this stuff you told me to do, " so it's definitely something to be mindful of. So here we have equality comparison, we already went over this.

JavaScript Logical Operators and Arrays

So, here is something that's kind of tricky. JavaScript supports logical operators, not going to do truth tables right now, so I'll just show you guys, I'll show you guys how to use these logical operators in the console. I'm just going to put a comment here so it doesn't execute, this symbol means AND, and AND means that you have two values, an A and a B, for AND to return true, but both of those values have to be true. So if I do true AND true, we're going to get a true. If I do a true AND a false, we're going to get a false. You need two. Oh, thanks. False AND false is false, true AND false is false, false AND true, also false. So there's only one condition where AND will return true. The second guy we have here is the two pipe-looking thingys, and that's an OR

operation. So, OR means that either one of the values is true. And a fun thing about OR is if the first value is true, JavaScript is like, "Okay cool, I'm done, doesn't matter what the second value is. " So you can do like, true AND blah blah blah, oops, true AND Two ands. Oh, thank you. Oops, sorry guys, let's do that guy. Sorry. I'm using the wrong symbol, and that's why it's not working. So, true OR whatever, doesn't matter if it's defined or not, or like not even valid JavaScript. JavaScript is like, "I got a true? Cool, I'm done. " So, true OR false, true, false OR true, also true, the only the only time OR returns false is if both values are false. So NOT is just a negation, NOT true, is false, NOT false, is true. And we can combine these, the AND and the OR and the NOT, in different combinations. I won't go into anything too complicated, but for an example, let's do true AND true, we're going to use parentheses here, and we're going to negate the value of what's inside those parentheses. So true AND true returns true, and then, negating it returns false. Does anyone have questions about this? There's a question about the single logical operators, do they perform bitwise operations? Yeah, Okay. but not something you want to mess with in JavaScript unless you're crazy, pretty much. Even if you're crazy. Yeah, just don't do it. So, here's a combination of a logical operator and a comparison. So, we have x is five and y is three, we check, is x less than four? The value is false, is y less than three? Yeah, it is, so false AND true is false. Same with this operation here at the bottom. X is not less than four, but y is less than four, so if we do that expression and we OR it, we'll get back a true. So, one of the most useful features of JavaScript are arrays. So, arrays are a list of variables, they're written with square brackets. So here is a notation for an array, the values in it are separated with a comma. Arrays have some useful properties to them, one of them being length. So, oops. If we make an array here, we use an open square bracket to start it, we put in some strings, and they're comma delimited, we use closing bracket to close it. Now we have our array. We can call, we use the name of our array, we can call dot length of it, on it. And it'll get back two. So, if you're using the Chrome JavaScript console, just a helpful thing to know is it knows what methods are available on this, so if you start typing, you'll get some helpful hints there. So, if we want to access an item in our array, we use square brackets. So, we call the name of our array open bracket, a number, close bracket. So an important thing to know is that array access is actually zero based, so even though the length of our array is two, we would access the first item by calling fruits zero, and the second item by calling fruits one.

Exercise 6: Arrays

Can you guys read that? It says my favorite fruit is undefined, and that's because we have declared this a variable, but we don't have anything in there, so just a super quick exercise, set the

variable favorite fruit to your favorite item in the array, use that square bracket notation, and then it should update automatically and print out my favorite fruit is, and your favorite fruit. So, let me type along with you guys real quick. My favorite fruit is an orange, so I'm going to do fruits and one. I actually don't need this because I declared that above. Sorry, for this CodePen you're going to have to click on Run one more time. Does everyone have their favorite fruit coming up? Okay, so, what's going to happen if we try to access fruit number 99? We get undefined again, but if we open up our JavaScript console. I'll just copy and paste this here. So try minus one, yeah. There's nothing in that spot, so nothing going on.

Exercise 7: Manipulating Arrays

So there are a few ways that we can change or add items in an array. The first way is that we can reference them by number and just update the value, and that will override it, the second way to do it is to use this method called push that's available on any array. If we push a new value, it'll just add it on to the end, and running this will return the number, the length of the new list. There's also a function called pop, which will pop off the last value in that list, and return that, so let's do some exercises. So we have our fruits array here, if I wanted to change the first item to, let's say apricot. I would access this array by index, and then if I look at my fruits array again, we'll see that the first item has changed to apricot. To add on a new item to the end of the list, we'll do fruits. push, add a new fruit on there, the value that's returned is the length of our new array, which has four values in it. If we want to start removing elements from the end of the list, we'll do fruits. pop. This function takes no values, so we can just go ahead and do that until our list is empty. Is there a way of adding something to the head of the list, or does it always have to go to the unshift? What's that? Unshift. Unshift--- so would you just do this? Uh huh. So let's just add some more values on there, oops. So this is our current state of the array, and let's try unshift again with a different fruit. So if we look at fruits now, we'll see that nectarine has been added to the beginning of the list.

JavaScript If Statements

So far we've been writing a lot of lines of code, and all of them have just been getting executed one by one by one. So an if statement is a way for JavaScript to decide what piece of code it wants to execute, depending on which condition is true. So, here's a super simple example. In the anatomy of an if statement is the word if, followed by open parenthesis, a condition, which must evaluate to a Boolean expression, a closing parenthesis, and then, open bracket, the code that will

be executed only if that condition is met, and a closing bracket. So let's try this out. We're going to make a variable called number of apples, we're going to set it to five, and we're going to write our first IF statement. So, let's say if we have more than three apples, and if you're using the Chrome developer console, and you want to continue on line by line without having it just evaluate when you press enter, press shift and enter, and it'll go to the next line. So, if we have more than three apples, let's console. log a message to ourselves. Say, "I'm apple rich! " So, if we run it, we'll see that I'm really, really apple rich, I have all of five apples. If I go ahead and change this variable, say I make it one, and I rerun my IF statement, you guys can do that by pressing up until you get to the line of code which you want to rerun. Nothing happens, because I'm apple poor. So, there's ways to add on to one IF statement, the first way to do it is an else, so if your condition is not met, you always run the line of code that's in the else statement, there's no other path of execution, so one or the other line of code will get run. There's also something called an else if, and you can have as many of those in between your if and your else as you want, and those are subconditions, so let's do an exercise. I hope you guys like fruits, Brian was tupperware and I'm fruits, so. Here we have this long if, else if, and else statement, let's just go over the anatomy of this really quickly, so the if statement stays in the same format, the else if must be on the line after it, and besides for this adding an else in front, you, the format is the same, the else is a little bit different, there's not condition for it, not parentheses because it will always get run, so we have three conditins here, one for if we have more than zero apples, one if we have less than three apples, and the other one is like, oh, we're out. So, let's try and change this variable and see how our path of execution changes. So, if I have negative 99 apples, everything is the same. If I have two apples, someone tell me which of these conditions will get executed? (comment mumbled) What's that? The first, right, "Eat An Apple, " Yeah, "Eat An Apple, " and, yep, we'll also print "Go to the store, " oops, let's run this, oh, I'm sorry, we won't print "Go to the store, " because we're in an else if, if we change our number to a value that's greater than three, we will print. It'll still be "Eat An Apple, " Yeah, let's change this. So it looks like if you had it set to two, if you had var set to two, then it's going through and reading the conditions, and it just happened to come to "Eat An Apple, " first, Right. even though it met both conditions, right? Yep, so if I go and I change this, let's say at less than five, even though these two things conflict, it'll go to this one first. Does JavaScript have the concept of a switch statement? It does. Does this make sense to everyone? ---apples are greater than five, that makes more sense. Okay. This one, I'm sorry, oops, sorry. It's not greater than five, it's not less than three, so we got no apples left. So, this is a nice, subtle logic bug. And in order to fix this, we would put equals here. So that should fix our, oops. Four is sitting right between three and five, so, Sorry. There you go. This will print, "Go to the store, "

JavaScript For Statements

In conjunction with the if statement, there's a different kind of statement, it's called a for statement, and the anatomy of this one is kind of tricky, so the counter, which is the first expression, is going to be a variable that keeps track of what step you're on. It's separated by a semicolon from a second expression, which is the counting to, and this is the goal, it's how many total steps we're going to take on the statement that's inside of this for loop. The increment counter is how we're going to change our variable, or counter to get to the goal. So, we're going to keep incrementing the counter, and keep running the expression inside of this for loop, that many times until we reach our stop point. So, here's a simple example, we have this variable called text, it's a string, we have a for loop here, so we start at the number zero, we continue until our variable I is less that, I'm sorry, is, goes over five, that's our stopping point, and then, we increment our variable I over and over, and so, if we run this piece of code, what it's doing is appending a blank, a space, and the number I into this text variable, and then it's printing it out, when it prints it out, we'll see that number zero, one, two, three, four, or add it on. Let's do a really simple example, so here's the anatomy of a four, we start with our variable, you might see this a lot, the variable called I is generally used for a counter in programming, we're going to go until I is less than or equal to seven, and the way that we change I to get to that point is by adding one to it with each step. Watch this print out I, You need increment I, not seven. Oh, yes, sorry, nice catch. So that'll print out the numbers zero through seven, when you're writing for loops, be careful that this middle condition is something that can be met, for example, if I change this to I minus minus, this loop would go on forever because there's no stopping point, and there really is no forever in computing, so it probably throw up at some point and give you a big error. So, make sure that that middle condition is something that's achievable, and that your counter is going up or down, or doing something that's going to help you reach your goal. The fun part about for loops is we can use them to iterate over an array, so by calling fruits. length, we know how many items we have in our array, and we can use a counter and go up to the end of that list and do something with the stuff in it, it doesn't matter how long the array is because we're using a dynamic variable. So, let's do this exercise. We got our fruits, we're iterating over them, we're outputting the value to this HTML page here. We're writing BR, which is HTML for new line, and then we're writing what the value of the fruit is at that position in the list. Okay. So let's do an exercise, let's add some items to that array at the top, and run the code again. I'm sorry, let's add some items using fruit. push, and then write a for loop down here, and we'll just write that code again, so fruits. push, whatever you want to put in there, and let's do that a few times and see how that changes our for loop. Does push always just add one thing, or can we get it to like, repeat the whole push

command to get a second thing on there? You can use append, I believe, to another list to your list. Oh, okay. It's concat, on append. Concat, thank you. The syntax for this is always slightly different in whatever programming language you use, so it's easy to mix things up. Okay, so let's put in an extra document dot right here, to see what's in our fruits. So, we got five things in it now, and let's write a for loop. Does anyone have questions about why we're using I in here? No? So, if we run this, we'll just get this list appended at the end of the one that we printed out here because we did, (mumbles) What's that? What's happened to apricot? So just say, fruits equals. Oh, good catch. Here we go, snuck out of there. Right, and the reason we're doing this, is because fruits. concat does not actually modify our fruits list, it returns a new list that contains both lists, so you have to remember to set it back to your original variable here. I couldn't hear, what was that? So, fruits. concat, does not actually modify the original fruits list that we had, so we have to remember to set it back here, because fruits. concat returns a whole new array with the values of both lists combined. So, the array is not immutable, but if you use concat, it almost acts like it is? Not really, so let me open up the console real quick. We have a list here of names. We have a list, let's call it animals, you got kitteh, so if we do names. concat animals, it'll return a new list with both things combined, but our original list is left intact. So we could say that names, There's no assignment in it, then. Right. If I did names equals names. concat, we'll see that names now contains both things. Is everyone done with the iterate over an array exercise? Did you guys all write your for loops? Good.

JavaScript Functions

Functions in JavaScript are a way of kind of grouping a bunch of things together that you want to do over and over into a way where it's easy if repeating them, like the same action that's contained in the function multiple times. So, the anatomy of a function. Here's the variable that we're assigning it to, you can also create function and different syntax, but we're not going to cover it today. We use the keyword function along with parentheses, and inside those parentheses you're going to put what's called arguments to your function, so it can be one thing, or a list of things that are comma separated. And, what these variables are going to do is be accessible inside of your function. So you're passing something into the function, and the function does something to that thing, we have an open bracket, I'm sorry, an open curly brace and closed curly brace, and a semicolon at the end to end the function. And then where this arrow here at the left, everything inside those brackets is the body of the function, that's what's going to get run when it's called. So, when we declare a function, it's different from a statement. So, so far we've had a bunch of lines of code, and they're executed one by one, if there's an if and the

condition isn't met, for example, it's skipped, but when we declare a function this way, we have a variable called printList, and this function is put in it so we can use it later, but nothing inside that statement is actually going to get run, until we explicitly call the function. The way that we call a function is by putting the function name, and then open parenthesis, if the function has no parameters, so this up here is empty, you don't have to pass anything in, otherwise you want to pass in the number of variables that this function is expecting. So, let's do an exercise here. We have our very familiar list of fruits. Down here we have, we have a function called printList, so this function takes a list, it loops over in a for loop, and it'll output a list of values contained in that list. So let's do an exercise, let's add two more lists, we're going to do one of your favorite cars, and one of your favorite ice cream flavors, and let's call printList on your new list. So I'm going to give you guys a few minutes for that one. Probably going to start working along with you guys. There's a question about (mumbles), is there any way for a function to take a specific type only? Not with JavaScript, but what you can do is inside of your function, you can call type of, or do some other sort of check, make sure that what's getting passed into your function is what you expect, if it's not, you can just throw an error, and that's a good way to validate and circumvent something worse going wrong. Does everyone see that same result? So let's go ahead and modify our function. So, outside of this for loop, because we don't want it to happen over and over and over, let's add a new document. write, say "These are a few of my favorite things, " and add one more BR in there, just so we can break up our results. So now, after every list, you should see "These are a few of my favorite things, " Does everyone Could you write like a plus, and then BR, like "These are a few of my favorite things, " plus BR, or do you need a Yeah, you can do that too. (mumbles) Yeah. Does everyone see these results? So let's go up and modify our function again. So, right now I'm conveniently printing stuff out, which, you know, is great because I get to see it right away, but a lot of functions are kind of functional, so you're not using them to print stuff out, you're using them to do something. So, let's clear these things out. Let's do, let's say, a little bit of validation. So, let's use our handy if statement, and say if list. length is less than three. And here we're going to introduce the return statement, so we're going to return "Not enough elements. " So, what return does, is it ends the execution in that function and pops out of it, and whoever called it, is going to get back what you're passing in the return statement. So, we have our validation here. If the list is less than three, nothing after this return will get run, but if our list is greater than three, let's go ahead and return the very first element in it. And yeah, this example is kind of contrived, probably want to do a little bit more to it than just return the first element. So, So now we're storing the result of this function printList into this variable return value, and after we've done all our calculations, now we can just say document. write. So this will go ahead and return that first element in the list. Another thing we can do, is call the function inside of

document. write, which is just another function, so in these parentheses we can just say printList, let's do cars. So, this kind of implicitly calls the printList function on this car's value. Oops, sorry you guys, I messed up here, it should be return list of zero, and not fruits of zero, because then you're just always going to get the same fruit. So, now this is the value that we expect to have. Does anyone have questions about the different ways of calling functions? Why these two things return, do the same thing? If we could go over why printList is called before document. write? Ah, that's a good point. So, in lots of programming languages, things are nested via parentheses, and the thing that's in the innermost parentheses is going to get executed first. So, cars gets executed, doesn't really, nothing really happens because it's just the list, then printList gets executed, and passed into document. write. So, let's say I have another function. I'm going to call it concat values function. Actually, I'm going to call it concat list function. So I'm going to take a list one and a list two, and I'm going to return sorry, that's concat, not contact, so, here we go, so I can nest this even deeper and say document. write, the result of printList, printList gets called on the result of, can call this concatlistfunc, and call it on our fruits and our cars. And wait for CodePen to update. Okay. So, Yeah, so here we're just returning the first item. And we can go ahead and mix these up a little bit. So now that returns Subaru instead of peach. There's a question on like real world usage of anonymous functions, would you recommend naming all of your functions, or using anonymous functions in some place too? I think naming functions and being explicit is a lot better than having anonymous functions because in JavaScript there's this thing called hoisting, and if you're not setting your functions to a variable, things can kind of happen out of order, so you can call a function before it's declared. I consider it a source of mistakes. There are instances where it makes sense to use anonymous functions, and tomorrow morning we're going to go over one of them, which is using jQuery and event listeners. Yeah. Typically, this JavaScript code would sit in the body of the HTML, is that where it would live? Not necessarily, we're going to go over this when we go about, when we talk about paths tomorrow. So, your JavaScript would live in a JavaScript file, and your CSS would live in a CSS file, or multiples of them, and then in your HTML, you would reference those files, and the browser is going to take care of smushing it all together. So, in the functions, obviously you can call functions from within the JavaScript, but how does the JavaScript tie in with events on a page itself, like button clicks, or a mouse movement, things like that? So it depends on the kind of event, but tomorrow we're going to talk about jQuery, and in order to use jQuery, we need to include the jQuery file at the top of our HTML. So what that actually kind of does behind the scenes is takes that giant jQuery file and pastes it into your HTML file, which means that in your HTML, like, or anything that really comes after that jQuery include, you can reference things before it. And that has access then, to the pages? Yeah.

JavaScript Scope

So, the last thing that we're going to talk about today is scope, and scope means where and how you can use the variables you've declared. So, if you define a variable in a function, it's only visible to that function. If you declare a variable outside of the function, and that function is in that same file or space, then you can use that variable in your functions. So, let's talk about what happens when you forget to use var. So, we have two functions here, printFruit, and printVeggie, each declares a variable inside of them, and then, writes that variable. So, if we do printFruit and printVeggie, everything is super happy, we get the results that we expect. So let's do this quick exercise. We're going to copy the document. write that's in fruit into the bottom of your veggie function and just run the code as is. Can someone tell me what the result of that is? So, nada. Yeah. Veggie doesn't know what a fruit is. So, we're going to do something tricky, and remove this var in front of the fruit. So now it's popping up, and the reason is because if you declare a variable in JavaScript without using var, it plops it on the global scope. It's like, "Hey, anyone can use me, read me, change me, do whatever. " And that's a pretty bad thing, it can happen by mistake if you accidentally forget the var, and then reuse a variable name, so don't do it, try to be very careful about using var and not really polluting the global scope with different variables. So even if you have a good use case, you know, maybe you do want a variable used across all sorts of functions, try not to do it, it's a really bad practice. A good rule of thumb about scopes is these little squiggly brackets, if things are nested, then it can probably access the parent variables, but if things are, you know, kind of on the same level like this printFruit and this printVeggie, then they probably can't access each other, and that's functions and classes, but does not apply to if or for statements, for example. This make sense to everyone? This kind of, something that's kind of tricky.

JavaScript Objects and Context

So let's get talking about JavaScript objects. Many of you, I'm sure, are familiar with object orient programming, I'm sure many of you are not, so we're just going to assume that you are not. So, objects in JavaScript are like a collection of properties, it sounds kind of, like really technical and computer sciencey, it's actually not very technical, so we'll look at it here in just a sec. They can contain other numbers, other objects, other strings, functions, anything like that, and they're useful for grouping like properties together. So let's take a look at this car, right? We have this var car, which is an object, right? You can kind of differentiate an object from anything else by the curly braces, right? So, here the car has a make, a model, an acceleration, and then I gave it a function called accelerate, which would just make it go even faster, right? It would increase that

acceleration by ten. We're going to talk about this here literally in the next slide, so if this doesn't make sense to you, about to talk about it so it's good. So, it's kind of a simple pattern, right, but like, just imagine trying to, if you were, like creating cars in your JavaScript, right, then you'd have to keep track of like 30 different cars, and you have to keep track of like, make one, make two, make three, as opposed you could just keep track of like, car one and car two, and they would each have like their own set of properties. It's a really powerful practice because then you can start assuming things about the objects, right, we'll see that here in just a second, and I think Nina actually touched on it yesterday as well, that if you have a person, and a person has a name and an age, right, you can always assume that that person has that name and age, yeah, so let's talk a little bit about this, so I'm sure all of you have seen this sign around, right, like no smoking in this building, or no parking next to this building, right? So what does this mean, in this particular context, right? Well, it means the building that it's next to, right, like, it makes sense next to a building, what happens if I take that like, no smoking in this building, and just stick it in the middle of a field, it just doesn't make sense, right? Like, what is this, what is this building? So this kind of contextual awareness of the word this, applies also to programming, the word this refers to whatever I am next to, whatever I am in, right? So hopefully that makes sense on like a, on a high level, that's actually like, concretely what that means. Yeah, this can be complicated, particularly in JavaScript, so just be careful with it, that's all I'm going to say about that. So let's go ahead and open this. Maybe it'll open, there we go. So let's make this a little bit bigger here since you don't really need to see that. So, here Where was that, what slide was that? That is on slide, one of them, 124. Thank you. You're welcome. So we have a favorite, and then we also have this person here that has a favorite. Okay, so let's first talk about this one. Var person has a favorite car, and it's a Nissan Leaf, and we have this function called getFave, right, this says return this. favorite, so in the context of this object, getFave is the object, right, I think that kind of makes sense, right? Like this, refers to essentially person, right, because it's being called within the context of the person, it's kind of like sticking the no smoking sign in this building on your object, it makes sense that this object is what this refers to, okay? So we'd call document. write person. getFave, okay, and we're calling that function off the object right here, which is what these parentheses mean, right, like it means that I'm calling that function. So it's calling that function, it's returning this. favorite, okay, and this. favorite in this particular case refers to this favorite right here, the Nissan Leaf, okay? Then I'm writing a new line just so, you know, it's easier to read right there, okay? This is kind of a, some black magic, so bare with me for a second. Right here, I'm saying func equals person. getFave without the parentheses. Okay, what does that mean, it means I'm actually not calling the function, I'm just ripping the function out of the object, right? Actually, to be more succinct about it, I'm creating another reference to it, so both of these still exist, right so this

getFave exists, and this func exists, but they essentially refer to the same thing, I mean, it's kind of like if I say var x equals five, and I say var y equals x, right, what is y right now? It's five, and what is x? Also, five, okay, same idea here, we're just doing it with the function. You can treat functions like variables in JavaScript, which is kind of mind-blowing, but it's also extremely powerful, and if you want to see why it's like the most powerful pattern in the world, there's a Frontend Masters course called Hardcore Functional Programming that's just like mind-blowing, like my mind melted when I watched it, it was awesome. Launching next month? Launching next month. Cool, so that is exactly what is happening here, I've now ripped func out, so now func right here refers to this getFave, okay? So, thus far it should make sense, but now I'm running document. write and I'm calling func, so essentially, now what I've done is I've ripped the no smoking in this building sign off the building, and then like I threw it in the middle of the field and like to see what happens, right? So now I'm doing document. write func, it's no longer in the context of this person, it's like a lost puppy (laughs) So what does it do? There is a global object in JavaScript, right, and in the case of the browser, which is what most of what we deal with, it's the window, right, like the window the entire browser object, so I have up here var favorite, which is not in any particular context, it has no scope to it, it's just in the global object, so when I call document. write func, what's it actually looking for? It's looking for this favorite up here, right, because it's now a no smoking in this building sign that's on the entire world, and this favorite is the only one that I can see. Are we okay with that, how do we feel about that? Right, we got Nissan Leaf right here, and we have Tesla Model S right here. That's the explanation, that's what context is. This is mostly to, like this isn't actually a very useful pattern, by the way, this is purely like an academic exercise, you don't actually want to rip functions out, and then typically you want to know exactly what this refers to, and you don't want this to be changing, so. Yeah, it's powerful, it's cool, it's a little confusing. So it's calling both models as a favorite? Uh huh, so I'm calling the function twice right here, person. getFave, that's, I'm calling it within this context of the person, therefore it's returning Nissan Leaf. I'm then calling it again in the context of the global object, and that is referring to this favorite up here, that's why it says Nissan Leaf first, which is this one, right, and then it says Tesla Model S, down after calling this one, which refers to this one. Now, this is actually quite difficult JavaScript, there's actually many front end developers that won't grasp this, I will tell you that because it's one of our interview questions at Reddit, and a lot of people miss it. (audience laughs) So, this is really cool, and I think it's really cool to learn very early on, because once you kind of grasp this, kind of the sky's the limit, there's really cool stuff you can do with contacts.

JavaScript Objects and Context Questions

This may or may not be off topic, I'm not sure, but is there any connection to objects like this and records, like saying or defining a person as Nina or Brian or myself, and, or is it just person as a very specific one set of variables, in this case, Okay, and it could've been just named someone specific because it's not generalized, does that make any sense? Yeah, I think so, Okay. So, we're kind of talking about like strict schemas, we're talking kind of about inheritance, cool, we're talking about that kind of stuff. That concept doesn't really, should I keep going? Yeah Okay. I'm just going to make sure that this doesn't go to sleep. Okay, cool. But you can keep talking. So, in JavaScript, there's no such thing as schema, at least not built into the language. So, objects are kind of whatever you define them to be. That being said, you can actually create what are called constructor functions, there's such thing as like the new keyword, if any of those ring a bell, then great, if not, then don't worry about them because they're not actually really used that much in JavaScript, people tend to just stick to what's called the object literal syntax, which is what I was showing you up here, with like the two curly braces just because it's really short, simple, succinct, but if you're actually talking about wanting to enforce, like every object must have a name, every object must have an age, or something like that. You essentially have to bring in another library to do it, or you have to write what are called constructor functions, which is kind of like, I'm not going to talk about them because I don't think they're that useful. Did I address your question, or did I go in like another direction? (mumbles) Okay. Yeah, just just that, okay, so a person is a specific thing, it's not like you have different people who all have that variable structure, then? Exactly. Okay. It's a single instance of an object. Okay. It's not a general purpose object. Okay. Yeah. And if you wanted a general purpose type object, you would write like a, a constructor function that would return to something like that, cool. From the chat, going back to this example, the question was so it returns the first var it sees after the loop, and then he kind of clarified why does it pick favorites, if you had two vars with global scope, what would happen? So if I had like, down here, and say like, var favorite equals, I don't know, another car of some sort, so it's going to take the last one, right, it's going to take the last time it was declared, right, it's kind of like var x equals five and then var x equals seven, right, like what is x right now? The answer is, first of all, this is a really bad practice, because you don't want to declare your variables twice, like because it's really confusing, you're like, "What is this person actually intending to do? " I don't know, so you'd typically see that and just say like, if you were trying to change the values, but the answer to the simple answer to that question is it's going to take the last one that happened, the last thing that was run. Which I think kind of makes sense, right, it's kind of like math. Does that address the question? One second. Okay. I think it does. (audience talking) So how do we feel about this? This is, again, a contrived example, it's typically not going to be that difficult. Do we want to go one step further and kind of melt your mind real quick? All right, let's do this.

Somebody mentioned that this behaves differently in Chrome and IE9, are you talking about the example? Oh, sorry, sorry. Cool, so, what happens if I want func to to get Nissan Leaf, right, like I ripped out this function, I still want it to call the correct one? There's kind of a fun little thing here in JavaScript that's called bind, so I can say func. bind person, this is kind of recent syntax here, so this is kind of like new frontiers we're exploring here, kind of, okay? Now what happens if we call func again? (indistinct talking) I am typing, no, that's all I typed. Is it, func. bind, no it's not there, I am typing, it's not updating. Maybe while you do that, there was a question that this, this is behaving differently in IE9, I don't know if you have (mumbles) but Chrome would give the same results and IE shows Nissan Leaf and then undefined. Interesting, IE9 is an interesting beast. Without actually delving directly into it I'm not exactly sure what's happening there. Okay. Their JavaScript engine is weird. Okay. That's a good question. So the answer is use Chrome? The answer is I'm sorry if you have to support IE9, that's the real answer. Use Linux, that's the answer, everyone use Linux, just kidding, cool. And another question, what's the difference between bind, call, and apply? That's a good question, sorry, call and apply are very similar, and they have, the only way they differ is one takes an array of parameters to pass to the function, and one takes just like you just list all of your parameters as if they were normal parameters. I get them mixed up, I don't remember which one's call and which one's apply, but one's one and one's the other, so, it's pretty easy to figure out. Bind is a little bit different, bind is actually you take the function and you're permanently binding it's context, which is when this comes back up, I will definitely show you what that means, but essentially, you call bind once, and then every time from thereon out, it will always call with that context, whereas call and apply, both are like a one time thing, so it only changes the context one time. So, and actually I'm going to jump straight into talking about bind right now. So, let's go back to our no smoking sign, because apparently I get stuck on analogies and I can't get off them, (laughter) what was my analogy yesterday? I don't remember, it's okay. Tupperware, yeah, you guys all remember, you're never going to forget that. (laughter) So, bind with talking about our no smoking sign, no smoking in this building, it's kind of like saying like the no smoking sign is like still in the middle of the field, right, but we like set up like a neon sign that says like, "We're in the middle of a field, we know we're in the middle of a field, " but it actually refers to like, that building, way over there, right, like it's just kind of, you can still kind of mess around with it and pull it out, but it's just like, maybe like a mail home to sender address on the sign so everyone knows what the sign is actually, what this still refers to, so in this particular case, I said bind person, which is the name of this object up here, right, so now this function, even though it's ripped out of person, still actually refers still to person. Or, I mean, you could even get a little bit more fancy and say like, var man equals and then you say have he as a favorite, and it's like a BMW i7 or something like that. So this is kind of a fun little thing, you can say like, this refers

to man, okay, hopefully that now runs, whatever, maybe I should just refresh it. I'm apparently having issues with this. Is it supposed to say BMW? It should say BMW. Hmm, I'm not sure, but that's kind of the general idea though, like, that it would say BMW i7, I don't know what happened there, okay. Interesting. person. getFave Not sure. How does it know that man is a person? So, it doesn't, and that's kind of the magic of it, so you can take these functions, you can rip them out of other functions about other objects and apply them to different objects, it's kind of achieving a level of versatility. I think on your func bind, Which line? somebody is saying maybe you need parentheses, you have them, okay. So you're taking the getFave function out of person, and you're binding it to the man object, is that correct? Yeah. Yeah, somebody mentioned that at func 2 you would need parentheses before you call that bind, is that, Oh, that might be it. So, yep, that's it. Anyway, So don't use bind, (laughs) Yeah, no I mean, bind is really powerful, I'm just kind of struggling with, this is kind of a contrived example and I think I've just kind of gotten messed up, so, maybe we'll come back to this but, bind is used for setting context. That's disappointing, anyway. So, let's go ahead and move on to, There's a lot about bind in the advanced JavaScript course, Yeah, It's on the site, Frontend Masters, so, Yeah, take a look at--- and call and apply and all that stuff. Yeah.

Anonymous Functions

I have an add function here, I have a subtract function, pretty simple right? One of them's just taking some numbers and adding them. We have subtract function here, it's taking a function, or taking two numbers and subtracting them from each other, and then here's kind of an interesting one here, we have this math function, right, that takes a number, another number, and then a function, so it's a function accepting another function as a parameter, or function-ception, just kidding. So this operate function is then going to be called within this function. Again, this is kind of a contrived example, but, How does it know that operate is a function? It doesn't, so you have to know that as the programmer. So, if this was not a function, it would just throw an error and die, essentially. Which is scary, but you just have to make sure every time you call math, you have to give it two numbers, and you have to give it a function. Okay, so does that make sense here, this operate is a function that is then being called. So you can pass different functions into here, and they'll be called. So let's take just, let's just look at this very first line here. Math, the name of the function up here, I'm giving it two arbitrary numbers, one and four, for example, and then I gave it the add function, right, because add is a function up here. So, what would you expect these to be, then? I'll give you a hint, it's over here. It's five, right? Does that kind of make sense, that this math function is taking an add right here, or rather, it takes it in right there, and then it's

calling add. Right, so this function is being passed into the function, and then called. Again, this is one of the powerful parts of JavaScript, you can pass functions around like variables. I was just wondering, the slides have the links to the CodePen, but I noticed they're under your account, are they going to stay available, or should we be trying to snag the code? Yeah, (mumbles) they should be there indefinitely. Yep. Okay, so, add one and four, for example, right? That makes sense, so now we're doing subtract five from one, right, that kind of makes sense, okay I'm seeing mostly head nods, which is awesome. Okay, I pulled this one out just to make it a little bit more clear, so, here I'm storing the answer of this math function right here, I have four and five, and this is what we call an anonymous function, right, so this one doesn't, this function does not have a name, it's only actually being used right now in this moment, and then it just like, fizzles away into the ether, right, so essentially right here what I've done is I've written my multiply function, but I wrote it just straight in there, right, like I just threw it straight in this math function, so, in this particular case, it's a function where it's returning the multiplication of the two, it's storing here to answer and it's just writing the answer out right here. So looking at this, knowing how the other ones work, what would you expect the answer to be, should it be 20, right? Are we kind of, does that make sense? Yeah? Question here, can the function be called inside another function without passing function as a parameter? Yeah, the answer is it depends on what context it's in, right? So like, for example, subtract right here, actually is on the global scope, if you're writing your code right, you put very little into the global scope, because it's really bad, it's really easy to override stuff, so I actually could say add right here, and then everything would become add, but that makes this math function lose it's power, right? What was cool about this math function is you could kind of define the behavior of the function from outside the function, which is why we kind of use this operate pattern. Make sense? Cool. So, I think one of the questions you probably might want to ask yourself is "Is this a useful pattern? " right? Like, this seems kind of contrived, like, wouldn't it be, usually being explicit is better than being implicit, because you want to be able to communicate with yourself and other developers, like "This was my intention with this piece of code, " so, what I would say to that, is in certain cases, this makes a lot of sense. We're about to jump into jQuery where we're going to use anonymous functions all over the place. So on line 17 then, when it calls function AB, it's not really doing anything because there's no operate defined, right? Right, so, this is just purely a function definition, so this function is not being called, much like when we defined this function right here, it's not being called, Oh, so it's just an empty definition then, it's just kind of, It's a function definition that is defined, it's simply just not called yet, so I mean, put that in like, the context of like the subtract right here, And it's local to that function down there, so. Yeah, so it's going to be, it's going to exist in the scope of this function right here, and then it's going to like, fizzle away, and go away and never come back. So it's like a

one time. Exactly, so it gets called operate up here, then operate is called, and then it just floats away, never to be seen again. That kind of make sense? So what's the point that you're making by showing this example? The point is that one, that it's possible, and you're going to see it all over the place, and two, that these can be very useful, like we're going to get into click listeners here in just a second, and if you had to define a different function for every click listener that you ever created, it would be a huge pain, this is just shorthand, essentially, it's much shorter to write, and so thus, it's written all over the place. In terms of execution speed, is there a difference between inline function declaration versus having it globally defined? It's going to be positively trivial, the answer is I'm sure there is some implication, but we're not even talking about a millisecond. Or, in other words, don't worry about it, there might be, but it's not worth ever worrying about. Just wondering if you had to iterate through a lot of the same, Like, again, I'm sure you could do millions, and tens of millions, and it would still make trivial amounts of difference. You know, my answer probably is the same. Cool, other questions on this, this is, again, kind of intermediate type JavaScript stuff that we're talking about right here, so if this is hard, it is hard, so. Cool.

Additional JavaScript Topics

Things that we haven't covered here, there's another type of loop called while loops, or do/while loops, if you're coming from another language, this should look really familiar, I use maybe like one a month, or like one every other month, so, the basic jist of it is that you loop until something is true, right? That's kind of the basic jist, it's essentially an if statement that keeps repeating itself until it's true, so, until it's false, rather, anyway. We're not talking about inheritance, like, if you're coming from a language like Java, or from like C++, something that we call like a classical inheritance language, that might be a pattern you're very familiar with, it's used very seldom in JavaScript, it usually causes more heartache than it's worth, so I tend to stray away from it, I write very little inheritance in JavaScript, but it does exist, it is different though. We didn't talk about switch statements, switch statements are essentially if statements that are kind of condensed down, like if you, rather than having if this, else if, else if, else if, you can write a switch statement it's like, if it's this, if it's this, if it's this, it's this, so. We didn't talk, there's so much interacting with the DOM, which is the document object model, we're about to get into that. We did document. write, that's interacting with the DOM, like your modifying what's on the webpage, but there's just so much to it, and it's just like, I'm sure there could be a whole Frontend Masters class on it. We didn't get into newer ES6 syntax, like, let was the question we asked yesterday, it's coming forward, it's not actually available right now so it's not useful for a beginner type class, and just tons more, JavaScript has been around since like, the nineties, so there's a lot to it, so, this is just,

again, scratching the surface, but we have given you the basic tool set to be like a beginning JavaScript developer, like you can go forward with these and do like, awesome stuff.

jQuery

jQuery

Using JavaScript to manipulate the DOM. This is fun stuff, I like this stuff because people start finally seeing, okay, we went through all these academic exercises, I mean, who cares, right? I'm going to show you right now why you care. (laughs) You're going to actually start making changes to your web pages using JavaScript. So what is jQuery? jQuery is JavaScript. jQuery is just a bunch of JavaScript that some really smart guys named John Resig wrote so that your JavaScript is way easier to write. But again, this is just purely JavaScript. This is a JavaScript tool set. Its main purpose is to make common tasks in JavaScript easier, like, for example, changing a class on a DOM node can be kind of difficult for people to grasp, but in jQuery, they just made the syntax really simple, so it's cool. jQuery's also used on some 80% of the top websites in the world. It's like, as far as libraries or tool sets or anything like that, it's far and above the most common. Everyone to some extent in the JavaScript world knows at least that much jQuery. So if you see that dollar sign notation, like dollar sign, then parentheses, like nine times out of ten, that's going to be jQuery, that's kind of like their calling card. If you see dollar sign, it is jQuery. Yeah? Question on the difference between toolkit, tool set and framework. Yeah, the pragmatic answer is not a lot. There's just these words thrown around, toolkit, tool set, which I take those to be synonyms, and then there's another word, framework. That's again some JavaScript someone else wrote, or code that someone else wrote to make your life easier. It's a philosophy difference, and if you actually want to get into it, I talk about it in my other Frontend Masters class, which is the JavaScript Showdown. That would actually be a great reference for that. Two months from now. Two months from now. Cool, so, dollar sign. So if you see a dollar sign like that, that's JavaScript's calling card, or jQuery's calling card. So let's actually start writing some jQuery. I've got a CodePen down here. Let's just collapse this one right here. All right. Let's first look at our HTML real quick. I have HTML, I have a p right here where I have a bunch of Lorem Ipsum text. Down here, this is probably your first time seeing this. I have a script tag, and all the script tag is doing is it's pulling in jQuery. If you want to be able to use jQuery, you have to pull it in first. That's what that does, it says, "Hey, go out to jQuery's website, "grab jQuery and load it. " Now we have all

this JavaScript on our page that we can now use. That make sense? Cool. What you might be asking yourself is, "I see all this Lorem Ipsum text. "It's not showing up over here, why? " That's a great question, I'm glad you asked. (laughter) I have this little snippet of JavaScript right here, caption-text, and then dot text Magic. So, notice over here I have a class called caption-text. Here, remember this syntax, right? This looks like CSS, right? That. caption-text is referring to the class caption-text. They did that on purpose so it should feel like CSS to you. For example, if this had an ID of mycoolid, how would I refer to it? #mycoolid, cool. Or, again, I could say p here as well. This would also work because it would refer to this p right here. You see dollar sign, and then in quotes you're going to have the selector, so what I had before was caption dash text. And then after that, this now essentially that, this contains that jQuery element for you to operate on. Here we're calling the text method, which is the jQuery method. It's not a JavaScript method, It's built into jQuery, and I'm saying I want you to set the text to be Magic. We're going to go over this a bunch, so if this doesn't make sense right now, I'm going to hammer it into your head. Yeah, what's the dollar sign again? That's essentially jQuery. Essentially, it is jQuery. So it's telling you that this line of JavaScript is jQuery. Yes. The dollar sign's really the name of a function. It is. It's just another, it's just a function call. You can actually put jQuery here, it'll still work, but everyone, again, in the world is super lazy, myself included. They just do the dollar sign because it's one letter. Yeah, so it's like jQuery dot text. Yeah. Magic. There you go. Cool, good question, other questions? We good with this? Is there a way to see all the functions attached to that jQuery? There certainly is, but I'd say look at the docs. jQuery has awesome documentation. In fact, let's just go ahead and pull it up real quick. There's not a way in a JavaScript object to get a list of its members, properties, functions, stuff like that? Let's see. Yeah, the answer to that question is not really, not very well. I'm sure there is a way to pull it apart, but it's not going to be very useful. Do some of the IDE tools, like WebStorm? They all have jQuery stuff built into it, particularly WebStorm. But yeah, jQuery has this documentation. You'd just look up here to the API Documentation. Possum, these documentations have been edited and reedited, and reedited. They're really succinct, they're actually really quite useful and intuitive. We have a course called jQuery In Depth. It's really done by the guy who actually made this documentation. Yeah, definitely, so check that out as well. Karl Swedberg. Cool, but yeah, anything you would ever want to know about this. Let's look up the text one, we just did that, so... It just tells you this method does not accept any arguments, or that one doesn't, rather, but we gave it text, so text right here. This is one that we used. I guess that's another good point to bring up, is that these functions will do different things depending on what arguments you give it. If I say dot text with no arguments, what it does is it goes out and grabs the text and gives it back to me. Whereas if I give it text, it's assuming that you want to give it text, so you're actually going to set what it is. In this particular

case, we did Magic, right? Somebody mentioned in chat that dollar sign dot fn actually will list all of the functions. Oh, does it? Object, object. Yeah, I can probably console. log that. Looks like you can console. log it, it worked. There you go, everything, right there. But, again, this is not super useful. I would still recommend sticking to the docs. Well, for jQuery, but there's other APIs, functions, other objects that you might not have documentation. Totally, cool. So there's still the HTML that's ultimately controlling the output in the sense that we don't need a document doc, or some other way to tell it to put that on the screen? It's just HTML, and we're just like, jQuery's just intervening and changing it? Yeah, that's a good way of thinking about it. All jQuery is is it's using raw JavaScript, and just essentially making things that are kind of difficult to do, they essentially just write all the difficult syntax for you so you can have a very clean and easy-to-use API. But there's nothing that jQuery's doing that you can't do by yourself. In fact, there's a lot of people that don't use jQuery because most of the tasks they use for jQuery are somewhat easy to do anyway, but jQuery's a good place to get started. I mean, I've been using it forever, so I recommend it.

jQuery Chaining

Let's talk about chaining a little bit. Again, same text up here, nothing's changed in our HTML. So let's take a look at this. We have text Magic and CSS background-color, and we're setting that to be orange. This effect is called Chaining and you'll see it a lot in JavaScript. I put new lines here, but really what's happening is this. All these periods are next to each other. So dot text and then after that, it's returning an object that then it's calling. css on. And if we wanted to make this more verbose, what we could do is copy that, hit enter there. This is equivalent to what we started out with. So we have this caption-text right here. We're calling the text method on it, the same Magic which you see up here, then we take the caption-text again, we're using the CSS function, which, as you can imagine, lets you modify the CSS on it. We're changing the background color to be orange. What people kind of thought was this is kind of verbose. I want to make a bunch of changes all at once to the same domino, so they eventually made it possible. It's going to blow up. They made it possible to do something like this. You say caption-text right here, and you say, All right, change the text to Magic, and then after you're done doing that change the CSS to be background-color orange. That's kind of what's happening here. Does that make sense? Just chaining methods together, you can simulate this pattern yourself, because all they're doing is returning this. Let's just write a little bit of pseudocode here for a second, like function text, right? Let's say magically you change text here. I'm not going to write out how to change text, and then the last line of it is return this. Because if you think about it, it's returning the object that was

called on, so as soon as you return that object that you called on, then you're free to call more methods on it. Or, to state it differently, caption dash text dot text magic, if I said var x equals that, the answer is like, what is x right here? X is going to be caption-text right here, and then after that we're going to say x. css background-color blue. So we store the variable right there, and then we turn around and we call it. We call CSS on that same thing. What this line up here is doing is cutting out that middleman x variable. I see some blank stares, how do we feel about that? Okay, questions? Which way is the best practice for readability? I kind of like the top, first way. This is fine. As a general rule, I tend to stray away from chaining because I feel like what you're worried about, that it becomes unreadable happens. The exception I make to that is jQuery because it is such a common practice in the jQuery community. This makes sense, right, like I have my caption text, okay I'm going to modify the text a bit, then I'm going to modify the CSS of it. That's a line of thought I can follow. Brian, can you grab the middle bar and just drag it? Oh yeah, sorry about that. So do you ever need to call x? Like, as soon as I put Var=, or Var space x in front of it, it was still doing the Magic. It is implementing it just by creating a variable? Right here? Yeah. Like if you took out the second one, should it just say Magic without the blue? Or would it say Lorem Ipsum because you haven't actually called x yet? So you're talking about if I take this one, right? The next one. No, the next one. Oh. If you define it but don't call it, what should happen? Oh, so you're saying like if I don't store it right here, it'll still work? Is that what... Well no, I was wondering... Okay, so I had deleted the first part in mine, and I just have Var x=, and it's still doing it. Right, the Magic still works? Yeah. So, but you haven't called it yet. You've only defined it, so why is it putting the Magic in there instead of the more-- Okay, I see what you're getting at, and that's a good question. So she's asking, I'm calling this function here, it says text magic and I'm storing it in x, but the effect is still happening, like how is it still happening if I'm storing it, never doing anything with it? So, it's kind of an interesting answer. This dot text function whether or not you store anything, it's going to modify the text. No matter what. That's the point of the text method is to modify it. so even if you delete this, right, it doesn't care if it's being stored or not. So that function is still happening either way. So, or maybe say it in a different way, the return value of this is actually not important to the over function of it, right? So it's returning itself, but that doesn't matter. Does that kind of make sense a little bit? So this is only important if I want to then operate it afterwards. So x dot css, color, red, or something like that. So because it's a call-by reference, when it's called it's actually updating the DOM out? I mean, it has to be executed to update the value of x. Right. So, upon its execution it's updating the DOM. Mhm. So, again here, this and this are equivalent in terms of what's actually happening to the DOM. Here we're just saying it's going to return to you itself, and you're just going to, like I don't care you're giving yourself back to be, you know, you can just go float away and I don't care. Here you're saying like,

I do care what you're returning back to me, I want to keep whatever you're returning back to me. And you're really keeping it, even if you don't assign it to another variable, because it's stored in that global context for chain. Sure. Okay, so you can still reference it if I say later down in the function, like I say this again, and say dot css, color, red, right? So I mean, like it's still available to you if you want to call it again, but you still have to call this function. The reference to what this returns is lost if you don't store it somewhere. But if you take out the semi-colon, you won't have to, the first one. Now the effect is the same 'cause you're really doing another call when you're doing the dot css in the chain. Yes, if I want to chain them together, I can. So somewhere in the global context it is storing the return value of that first function column. I guess. The full answer to that is I'm not exactly sure the mechanics of how JavaScript is doing it. I don't know if they're storing a reference to it, or if they're instantly operating on the return value. That would seem to make sense to me though, that there would be a variable storing it somewhere. But as soon as this, like, so even though I don't have a semi-colon here, right, as soon as this line happens, like I can't come here and say a dot, color, right? It's lost after that.

Responding to Users

What happens if we give the text function we read in an argument? Will it be called, or will jQuery text function be called? Say that one more time? What happens if we give the text function we've written in an argument, will it be called, or will the jQuery text function be called? So I guess if you were to like... Oh, okay. So, he's kind of talking about like overloading a method. It's not really, it doesn't exist in JavaScript. So the answer is, if you called text with no parameters versus one parameter versus two parameters, it's not like Java where it actually will refer to a different method, it all refers to the same method, and it's up to JavaScript, your JavaScript that you write, to deal with those different parameters. Here, now I'm actually going leverage those anonymous functions that we were talking about. Something that happens, oh sorry, go ahead. So now I think they're saying, what if you even just did define your own function called "text. " So object dot text, verses dollar sign dot text, if you created your own object and had a text method on that... Which one would it call? Right. Well, I guess it depends if it's a jQuery object. If it is a jQuery object, you can overwrite their text method, and then call your own. That is a possibility. But if it was a completely separate object, it would be calling your implementation? It would be calling your implementation. So jQuery only modifies its own objects. It doesn't actually go and overwrite other things' objects, or methods or whatever. Does that address it? I'll check here. Okay. Let's talk about pressing buttons. Right, it's kind of a common practice that you'll like press a button on a website and it'll do something, right? So you want to have that ability to react to user inputs. So

in this particular case, we have a button right here, so a button class equals alert button that says "Click me" on it, so now we click it, it's going to say "Hey there! " So let's talk about how we did that. Let's just make this a lot bigger here. Okay, so we grab alert button here. This should look familiar from what we were doing last time. And then there is a thing called a dot click function. That's essentially saying every time my function or my button gets clicked, run this function. So again, I can click it again and again and again and again, right, like it happens every single time. So what's actually happening here in JavaScript, or at least to the DOM, which is the Document Object Model, we talked a bit about that, there is a thing called an event. So every time I click on this, whether or not I'm listening for the event, it happens. Right, JavaScript, like I'm clicking here right now on the body, JavaScript is throwing off all these events for me clicking on the body and just nothing's happening because no one's listening, right? It's kind of those things like, if a tree falls in the forest, kind of problems, right? So in this particular case, and we call this dot click function. The tree is falling in the forest, and we're listening, so it does make a sound. So, yeah. When you call the click function, it's essentially saying, I want to be notified when this event happens, and I want you to run this block of code, right? So here I'm saying alert, "Hey there! " So every time that gets clicked it calls "Hey there! " That kind of make sense? There's lots of events in JavaScripts. Click is just happens to be one of the most commonly used one, right? Because that's usually a direct sign of user interaction with your website. They click on something, they expect something to happen, so you should be listening for it. Questions about this? So that single click event, or the action of clicking on that button actually created a whole string of events, correct? It's just that we're listening for that, the alert button click. Yep. Because there's actually a click in context of the browser window as well. Right. Definitely. I didn't catch all that, but it might have addressed that, they're asking can you explain more on how events are set up in the DOM and how it exactly listens? The question is always how much depth to go into. I'm going to, one, defer to the other jQuery class with Carl, I'm sure he touches on it much more than than I should. But essentially, every time you press a key, you click on something, you move your mouse, your mouse enters something, your mouse leaves something, you push down the mouse button, you let up the mouse button, you click the button, you double-click the button, those are all different events that the browser will fire. Essentially what the browser does is it throws, it says, hey, an event happened here, and it kind of just bubbles up and different things, so you can have multiple things listening for example. If I say. alert-btn, click, I mean, this is contrived, right, you really would just put it up here, but let's just... I can spell. Function. And we'll say like alert-btn. text, changed, right? Oh yeah, period right there. Okay. Okay. So what I'm trying to demonstrate here is these events bubble up and there can be many people listening for these events. Essentially the browser says like hey, this happened in the browser, and all your listeners can say,

oh we don't care. You know, go pound sand or whatever. Or they can say like, oh that happened, I care, I'm going to fire off my code now. So in this particular case, alert button now has two different event listeners that are going to do two different things, based on those events. Just declaring that jQuery function call registers the event listener. There's no sequencing or no positioning dependency variable. It's really that simple. And again, that's why we went over jQuery. It makes thing like this just deadly simple. They're just asking if the on-click and setting that to a function as an attribute in HTML, is that the same thing we're doing here, it's just jQuery is taking care of it? It amounts to be the same thing. That's a really bad way to do it. You do not want to put your JavaScript into your HTML, like it's just, it's the worst thing you can do for yourself. Stick to this. Now if you want to ask me, can you do it in raw JavaScript without jQuery? That's fine. You can totally do it that way and just not use JQuery at all. What if you wanted to reference something that didn't have a tag? Or maybe it does have a tag, but you don't know what it is. Like you wanted to change like the nav bar on browser, or you wanted to do some other device You need some way to get at it, whether that's a class, that's an ID, that's a tag name. You can't get at at something if you don't know what it is. So it can be defined dynamically, right, it could have like var button, and that's equal to this, and I can say right there, button, and here button as well. So that can be filled in later dynamically, but you still have to give it some indication this is what I'm listening for, right? You can even get super annoying and say like "body, " right? So now any time the body is clicked, it's going to get really annoying with me. Don't even have to click on the body anymore. Woops. Damn this page. So again, that can be changed, it can be dynamic, right, but... So it's only things that are in the DOM, not anything that's outside in the browser itself? Okay, so you can create synthetic events, right? Like, that's possible. The browser generates a bunch of really useful events that you're-- I think he's talking about outside the window. Right? Well, outside of the DOM. Okay. So if you wanted to modify, 'cause there's websites you go to that open additional tabs or things like that that are outside the context of the DOM. So your tab is pretty well sandbox. For example, my CodePen tab cannot listen to my Google, that's a huge privacy implication, right? You don't want other people listening to what you're doing. So you are confined to your webpage, whatever that is. There is minor exceptions to it, but I wouldn't worry about it. You can only listen to your own stuff, that's the bottom line. What about things like, if you're writing an app for a global device where you're looking for tilt angle, like an accelerometer or something like that? How would you reference that event? So the short answer to that is a lot of that can't be done in the browser. You have to drop into a mobile app, where you actually have access to those APIs. Isn't there an accelerometer API? I think there is, but I've never used it, so I'm not so familiar with it. There is device orientation, it's called device orientation API. There's GPS. Right, yeah, stuff like GPS. (several people cross-talking)

Geolocation, and basically you have to, the way that the API works... That would exist... You ask, you know, you ask can I have permission to see your location, and then the user clicks Yes and then you get a location address, and it's called a geolocation API. Yeah. Some of it does exist, but yeah. Some of that's discussed in the intro to HTML5 and CSS 3 course on Front End Masters. Good questions though, for sure. So there's a question online. Somebody's asking, so you could define it like this this var dollar sign element with no parentheses equals dollar sign element with parentheses. I'm not sure if that makes sense. Dollar sign var dollar sign element is not really used in jQuery, you know? Yeah. Oh, are you talking about storing a reference to the jQuery object? I think they are. Yeah, so, saying dollar sign button and wrapping that in a JQuery and then calling dollar sign button dot click. Put a dollar sign in front of myBtn, to show it's a JQuery object. Okay, so something like that is also possible. You can store like the reference to these jQuery DOM nodes also totally works. He said it was really one statement. Okay. Yeah, this is a useful pattern as well, so, particular if your'e going to reference it a bunch of times, every time you do this, this dollar sign, it does a look up on the DOM. While it's quite fast, it's not free, so... If you're going to reference a bunch of times, it's a good idea to store it and call those functions like that.

Retrieving Text from an Input

Alright, so let's go ahead and get some text from an input. So here we have an input and we have a button, right? You remember void tabs, right, that's an input right here. And then we have this button right here, the alert button that says "Click me. " And then we're loading the script. That's all the HTML does. Let's go ahead and just make this a lot bigger. Okay, pretty straightforward. And then here we're going to set up an event listener that when I click it, it's going to grab the name out of the name input, which is this one, right? And then we're going to grab the value out of it, which is what the dot val function does. And then after that we're going to alert that name. So if I say "Brian" right here and then "Click me, " it's going to give me back my name, right? Or you can even have it like greet you and say "Hey, " okay? Again we can type "Brian, " and it's going to say "Hey Brian. " So again, just kind of a contrived example, but I'm sure you can see why this would be useful, right? Like if you're trying to log someone in, right, you need to get the username and password out of the form and send it to your server so you can verify it, and be sent back down and say you have good credentials or you have bad credentials. Questions about that? I think, can you show the HTML for this? Yeah. I think there's a question, the button there, it has the type in it. Mhm. They're saying that sometimes they see it with that, sometimes they see it without that. Yeah, I think I've done both, so that's fair, that's a fair accusation. Buttons by default are what are called Submit buttons, so if they're in a form, they're going to try and submit the

form, which, without going into a whole lot of detail about what that actually does, it's actually going to try and reload your page or it's going to try and go to a different page if you don't tell it what type of button it is. In this particular case, I believe I could just remove it. No one would care because it's not actually in a form, right? So it's a good habit to get into. As you can see, I'm not always in said habit, so, but if it was inside of a form it would make a bigger difference. By inside a form, you mean inside a form tag? Literally a form tag, right? Like form, oh wow, I did not want to do that. So inside like a form tag like this, and there would be a closing form tag down here, right? Then this would make a difference. But because there's no form tag, because we're not worrying about it, it's cool the way it is. Good question, though. Other questions about this? Hopefully it's pretty straightforward. Let's just take it a little bit one step further here. Oops. So after we alert the name, we don't want to keep alerting the same name over and over again, so what we want to do is we want to clear out the input. So I'll show you the effect first, and then I'll explain to you what it is. So that's my name, click me, it's going to say "Hey man, " and then it's going to clear out the input. That's what this val, if you give it a string, is going to do. In this particular case, I've given it an empty string, so it's going to set the input to be an empty string. Like if I said hey, and I put "Brian" here, type a bunch of stuff and "Click me, " What's it going to say after I click okay? It's going to say "Brian, " right? So it's, this dot val with a string provided to it, it's going to set the value of the input. But again, we're just going to set it to be blank, because that's kind of a intuitive thing to do right? You want them to say, you know, "Hey Nina, " and it says it right there and you click Okay and then it's gone. So that's a blocking, the alert box is kind of a block type input. What happens if you have multiple functions or multiple listeners, It would block them. Is JavaScript multi-threaded? I mean, is it going to handle those simultaneously, or is it just going to block and the first one, whichever first one it randomly caught first? It's going to block first. So the sum of that story, is don't actually use alerts in production, because the alert will actually pause the execution of JavaScript until the user interacts with it, it's a really bad user experience. Not quite sure what they're referencing, but what if it was a placeholder? Oh yeah, sure, we could show that, too. You can "placeholder" right here. Your name, okay? This is HTML, right? We're going back to HTML, this has nothing to do with jQuery or JavaScript but I'm sure you've seen this before. The grayed-out wording in the background there? It's kind of hard to see right now, but you see how it says "Your name" right there? And then as soon as I start typing it disappears. That's an HTML5 thing, super useful. But yeah. Cool, more questions? This make sense? Like using the values, clearing out the values, that kind of stuff?

Exercise 8: Set Text

So, open this CodePen. Using jQuery, we're going to make it so when the button is clicked, whatever text is in the input, it's going to be set as the text on that p, on that paragraph that's on the page. Again, do not modify the HYML, you can only modify the JavaScript. And I'll give you a hint here, you're going to use text, val and click. Those are the three JavaScript functions that you're going to be using. Okay, let's just take a look at this. Okay, so we have our input right here. I've given you a nice little class to refer to. I've given you a button here, and then I've given you the user text, which is what you're going to replace. Are the instructions clear? Are you understanding what I want you to do? Okay. Just number two. So, essentially, I'm going to type right here, in this thing right here in the top right, I want it to be able so that I can click it, and where it says "Replace me" to be replaced by whatever text was in here. Does that make sense? Cool. What is the URL right in front of the question mark? The LV? The URL in front of the question mark. Oh, the editor's thing right here? Is that what you're asking? Just before the question mark, what are those characters? L? I think that's an L. L-B-W-U-L? Let's see, let's come back here, this might be it. A little bit bigger, clearer, top one. Oops. So my first recommendation for you is you need to set up a click listener on that button. That would probably be the first thing that I would do. Okay, so if you've got a click listener set up, probably my next step would be to grab whatever the val is out of the input. If you don't remember how to do that, the last CodePen that we just did, we did exactly that.

Exercise 8: Solution

Okay, so I'll give you the first little bit right here. You're going to have user dash btn, sorry period user-btn dot click, function. So that'll get your event, or your button listening for that event. Rather, your JavaScript listening for the button event. Most of us getting at least that far? Okay, cool. Question was, dot val and dot text will do basically the same thing when there's text, when it's called in a tag with text? Yeah, val is used for inputs of some sort, whether that be a checkbox, radio button, inputs, val refers to whatever the user has typed in there, or that the user can type in there, for that matter, text on the other hand, is used for paragraphs, divs, things that are containing text that are not editable by the user. But, conceptually what they're doing, which is modifying text of an element, is conceptually similar. All right, so I see most people got the click listener. Are we getting the value out of the input? How's that going? Okay? So let's just go ahead and do that. So I'm just going to put it in word or something like that, maybe even val. That's fine. Dollar sign, what did I call it? User dash input dot val. Most of us get at least this far? So we've now grabbed whatever is in here, so if I type this, whatever is in here, we've now grabbed that out of the input. Now the next thing we have to do is we have to set that as the text of this paragraph

down here. User text. We're going do that using the dot text function, and we're just going to give that text function the input that we got from the user val. Or user input dot val. I think they're asking why you're saving it in a variable. It's certainly possible to do this in one line. I'm doing it, one for clarity, and two so I can do it in multiple steps. An old adage that I learned when I first started is it's better to be clear than clever. Clever code is never a good thing. What is not? Clever. Like doing some crazy operation in like 10 characters that just melts everyone's minds but no one can understand it, right? It's better to take 10 lines and really be verbose and say this is what's happening so that someone can just read it and say, this is what's happening. Now, in this particular case, you could do it in one line and it'd be just fine. But I just recommend, do whatever seems clear to you. All right, so. Now we're going to go ahead and go and say user, sorry ('. user-text'). text(). okay? Then what goes in here? Something needs to go in here. Okay, val goes in there, right? Because here we've, maybe if this isn't as clear, we'll change this to be like, user text, right? And here we'll put user text. Just so you don't get that confused with this val, because they're separate vals. Okay, so user text, user text there. That's it. Four lines of JavaScript. Really it could be done in three, but... All right, so, go ahead. One of the questions is, how would it work doing the one line? Could you maybe just show? Sure. Maybe the difference. So let's just go ahead and comment these out so you can see that. And here you would just do it, dollar sign, user dash text dot text, and then here, Nina talked a little bit nesting functions inside of each other, we're just going to do user dash input dot val. And this should work as well. Yeah. So the question you have to ask yourself, does this look really weird and I don't get it? If you answer yes to that question, then don't do it, do this, right? This is more verbose, but this is, I don't know, I don't want to say concise, but certainly shorter. So let's leave that one commented out. Okay, so do we kind of grasp what's going on here? We're grabbing the value out of the user input, then we're just saving it, and then we're just using it right there.

Other DOM Functions

So other cool functions here that we're not really discussing, we talked a little bit about CSS. That one just allows you to get and set CSS values, right? Like, I'm wondering what color is this div right now, I can ask what is your color and it'll give that back to me. Or I can say like, make your color orange. You would use CSS for that. HTML is very, very similar to text. The only difference is if I give like strong, this is cool, end strong tag, HTML will actually honor that HTML, right? So you can actually feed it HTML strings, and it'll render it like HTML. If you use text, it's actually literally just print out your text for you. Show and hide, it's exactly what it sounds like. If you want to like show something on a page, or if you want to hide it on a page, you just use show and hide. Add

class and remove class. So like for example, if you have a class like this div is now emphasized, you can add like an emphasis div, and then once they change their emphasis to something else, you can remove the emphasis from one, and then add the emphasis to another, right? So you can dynamically add and remove styling and classes and stuff like that. Really useful. And then append. So up until now we've been doing text or dot HTML, which destroys whatever is in there and then sets the new text. So if you actually want to add to the bottom of it, you can use append. Yeah. Back to the show and hide, what does hide actually do? It sets the display to none. So it uses CSS to hide it. There are ways to actually remove things completely, like if that's what you're interested in doing, I would say nine times out of ten that's unnecessary, just doing display none is fine. Cool. And all this stuff is in the jQuery docs as well. Let's go to the next slide. jQuery is just ginormous, right? Like, there's tons and tons and tons of facets to jQuery. It has 300 functions, it's just huge. So if you want to know like how to hide a div in jQuery, just put into Google, jQuery how to hide a div, and you're going to find probably like six pages of valid answers but just because everyone uses jQuery, and everyone's always asking questions about it. So everything that you want to know about it is very well-documented. So yeah, stack overflow is top-notch for jQuery.

Audience Questions

We've been using just part of the jQuery library so far. Like I said the jQuery library is huge, and we're only, the DOM-manipluation part of it, while it is the most prevalent and the most well-used part of it, we will be talking about another part, which is called AJAX here in just one second. But we have some other questions to address. The first one was, why would you use jQuery to add/remove styling classes versus changing CSS directly? That's a great question, and the answer comes in the form of like, it's reusable. So if you're just changing CSS in the JavaScript, it's kind of an anti-pattern, and when I say anti-pattern, it means like it's just not a good idea. Because if I put it into a CSS class and then add and remove a class, I can now use that everywhere in my website, right? For example, if I have an error class, that jQuery adds and removes, what happens if I load a page with an error class, right? Then I kind of have bifrocated my efforts here, that I have to maintain both the jQuery CSS and this other class that mean essentially the same thing. Whereas if I just add and remove classes, it's all available always to be used across our entire website. Laziness, that's my key answer right there. And then the next question was, does the hidden element still take up space? So, we saw this yesterday when we talked about CSS. The answer is no to that question. When you set display to be none, it essentially takes the entire element out of the document flow, so essentially has no bearing on the DOM itself. Good question though. Well

it's still there, right? Because you can just switch that to not be hidden and it'll appear. Exactly. You're not going to go re-download that piece, so it's still there. It is still there, correct, definitely. You just have to say display block, display in line, or whatever it's supposed to be, and it would show back up. Or, in this particular case, you would just say "show. " But yeah, it's still there, it's just not making any manifestation of itself. Cool. Did I miss the explanation of the DOM? Like, in detail, kind of what happens with jQuery pulling stuff out of the DOM? Did you explain that? Not a whole lot, the DOM is a complicated beast. And as opposed to diving in and getting really confusing about it, I think it's better to just kind of take a surface-level, like kind of take it as a black box right now, and say like, this is very complicated, but here's a couple buttons i know if I can press and it'll do certain things for me. There's an article on CSS Tricks called "What is the DOM? " and it kind of explains it in very layman's terms, you know, it's basically like a tree-like structure, and there's these elements. But when jQuery pulls something out of it, does it insert something back into that DOM, that's kind of like a new HTM format or file? Well when you use the jQuery object, you're just wrapping the element in a jQuery object, so it's not like taking it out of the Dom in order to modify it. Okay. It just doing it jQuery's way, right? Yep. It's not really like taking it out. If you say dot remove, that'll take it out of the DOM, but... So, the simple answer to I think what you're confusing here is that we're not, when I say like get the user input, right, it's not actually yanking it out of the DOM, it's actually just pointing at it. It's pointing at where it is right now. So it's just referencing what's already there, it's not pulling it out, messing with it and putting it back, it's just changing it in its place. Okay. That make sense? Okay, yeah. Cool. I think that makes sense.

AJAX

AJAX Introduction

So, Ajax, this is a really powerful pattern, it's hard so just forwarning there. It stands for Asynchronous JavaScript and XML, which is a misnomer. Um, it's really just a name that stuck, everyone calls it AJAX, I would say that most people don't even know that that's actually what AJAX stands for. And so yeah, it's just a buzz word that stuck, and what it really means is like, so imagine when you load a page, your computer reaches out to the server, the server gives you back HTML, CSS, and JavaScript to load on your page, right? That's kind of the pattern that we're used to. There's another pattern that you make an Asynchronous call, or in other words you've

loaded your page and you want your JavaScript to reach out to the server again without refreshing the page, that process is called AJAX. So that I'm on a page, I say like, you know, imagine like an infinite scroll on like, a product page, in fact let's just go ahead and look at one. Going back to Reddit Gifts slash marketplace slash feed, okay? So, here we have this feed of products, right? If you scroll down, it loads more, right? So we're not refreshing the page, it's just, um, this particular page is reaching out to the server again and getting more and more data. I'm going to teach you how to do this technique right now. So, we're going to look at two different APIs over the course of this. We're going to look at Meetups, Meetup, like if you're not familiar with Meetup it's essentially that, they plan meetups and people help people make meetups. They have an API that's just a joy to use, so we're going to use that one. And then of course, we work for Reddit, so we're going to use the Reddit API. It's going to be fun. Uh, let's keep going here. So, AJAX is kind of a pain in the butt, to be totally honest, but AJAX with jQuery is really simple. So we're just going to do it with jQuery, alright. Let's just jump into an example. Alright, so again click me, replace me, HTML's pretty simple here, uh let's just go ahead and expand this one out, okay? So, let's just, I'll show you first of all what it does. It just, you click it and it just like barfs all over the page all of this data, right? That's what's happening here. Uh, let's just refresh the page so you can see why that's happening the way it is. So, button click, right? The button refers to this click me right here, the dot text refers to this replace me right here. And you'll notice as soon as it, the button is clicked, I indicate to the user that we're loading, right? I just change the text to say, "loading", okay? Then I call this dot AJAX method, and dot AJAX takes this, it's called a configuration object, right? Like, we talked about objects that have properties, we're just feeding it an object that has a bunch of values on it. So we're going to do a get, which I'll explain here in just a second. We're going to reach out to this URL, which is api dot meetup dot com, like this is like, again I can just put this in my browser and it'll work. Right? It's just a bunch of like raw data. In fact this is a great time to go over what this actually is. This is called "JSON", JavaScript Object Notation. It should actually look pretty familiar to you, right? It looks a lot like JavaScript right, so we have this, like, results which is an array, right? So that's what those two square brackets mean, and so it's an array of objects which you see these curly braces now, and just all these curly braces have um, different properties on them. Each one of them is representing a city in this particular case, so we have Phoenix, Scottsdale, (is drowned out by sneeze), I guess that's because my VPN's connected, so my VPN actually thinks I'm in Arizona right now, but if I disconnect my VPN, and I refresh this, now it thinks I'm in Minneapolis, so. Minneapolis, Saint Paul, Edin, I assume that's a close city. One thing to note, I think you have like a plugin that makes it kind of pretty print it, so. It did, I do, so if you go to this I imagine a lot of you are seeing this, right? It's just like a blah, throw up of data, um, I have a Chrome extension installed, I can't

remember what it's called but it's like JSON Prettifier or something to that effect, that makes it nice and easy to look at. Great, um, so does this kind of make sense, that's what JSON is it's just it's a way for two disparate entities to communicate with each other. It's essentially computer talk, right? That the client can reach out to the server and say, "I want this data from you". And then like, the server says, "Okay, here's your data". It's a language that the two computers can speak back and forth, and that just, it looks like JavaScript, it looks like normal JavaScript. So, let's go back here to our jQuery, so we're calling dot AJAX and it's doing what's called a "get" request. There are varying HTTP verbs, um, so I reach out to the server and the server says like, "What do you want? " Right, and if you say, "I want to get this from you". It gives you back something, right, if you say, "I want to post to you". It's like, "I'm giving you data to do something with". Which we're going to do here in just a sec, so and then there's like five or six or, actually probably more other verbs, but those are the two that we're going to concern with ourselves today are gets and posts. So, that get is just saying like, "Please give me data back". Okay? We have the URL, I think that makes sense that's where you want to reach out to, and then we have this success, so that's essentially saying, "Once this is successful, please run this". Okay? So, this is called Asynchronous Programming, because this, we'll call this and reaching out to a server is not instant, right, like there's the round trip that has to happen like, I have to say, "Hey, I want this. " Has to go across the great interwebs, and then it has to go to the server the server has to like run its processing and then it has to flow back to you, that is not instant in fact often it can take on the order of several seconds, so we have now introduced what's called asynchrony, right? This is calling, but we don't want our entire program to get blocked, right? We don't want to just have this user sit there and wait while all this is trying to happen, we want to be able to execute more code, we want more things to be able to happen, so essentially what we're saying is run this, and then whenever this happens, whenever like the interwebs gods give us back our data, run this function. So, we've kind of actually already done Asynchronous Programming, right? We, in fact right here is, this click function, we have no idea when this click function's actually going to occur, right? It only happens once the user does something, but it doesn't happen until then. That's exactly what this AJAX function is doing as well, right? It's saying like, "I don't know when the data's going to come back, but once the data "does come back, do this. " Yeah. Um, what's the difference between passing an object or a function method? Uh, it depends on what the function wants slash needs, right? Like some functions expect objects, some functions like, expect variables it depends on what the function is, okay. I think the AJAX function can work multiple ways, I've just found this one to be the most easy to explain, so we don't want that. So, on a high level, like not getting too nitty gritty in the details, does this make sense what's actually happening here, okay? Yeah. So can you have a timeop fail as well as a success? So there is like an error function right,

because there's also the possibility that you reach out to the server and oh no, the internet got destroyed and now this data's not going to come back and you need to do something, right, you can't just like have your whole program fail you need to tell the user, "Oh crap, I can't get your data for you. " Right, I mean we're not going to, say, go into error handling right now, but that's something that does happen, should happen, and you should do we're just not going to, heh, sorry. So, I believe the value is called error, right, I think it's just like, and it's another function that gets run in case the error happens, right? Because you're going to handle them differently, but. Okay, and then this should look familiar to you like I'm just taking the data, the data is whatever's coming back from the server and I'm just setting that to be text, okay? So JSON, we were just talking about that, JavaScript Object Notation, um, basically I'm taking an object the server's giving back to me, I'm saying I just actually want to dump it out, right? If I didn't do this, it would just say object object, so just so you know why I'm doing that. Maybe. Yeah, it says object object because I'm trying to print out an object, an object does not know how to display itself, so in this particular case I just put in JSON a string that probably says, "Make this into a string so that I can display it on the page. " That's all that's doing, okay? I'm sorry, did you explain what the JSON was in that data type, or? So, JSON is like this, right, like it's this information interchange format. And, like, that's what it really looks like, it's just like-- It's just a bunch of data, right? It's a bunch of raw data, it's not really meant for human consumption, it's meant for computer consumption, I have up here this thing that let's me see it all pretty and nice, so that I actually can read it but-- So really, all that JSON is doing is saying, "This is a JSON object", and now you can have all these (background audio drowns out voice) Exactly, in this particular case it's saying like, "Please make this so I can print it out to my page. " You actually would never do this in real life, right like you don't want to show people raw JSON because they're probably going to say the same things you're probably saying right now, like, "What is this? " Right? (laughs) So, yeah don't do that, this is just purely for like debugging, demonstrative practice. And then, just to touch on what jsonp is, um, I don't want to go into depth in it, but suffice to say that this call won't work without jsonp, it's how the meetup API works, um, and it has to do with how the data is loaded back onto your computer, so, don't worry too much about it. It's only useful when it's useful, and that's to say most of the time it's actually not useful, but if you're going to mess with the Meetup API you do have to use their jsonp. And when we get into the Reddit API, we will not be using it, so cool. The API's you're talking about, those are just sources of data? Yeah, that's a good point I should talk about that, API stands for Application Programming Interface, that sounds right, that sounds smart let's go with that. (laughs) So essentially it's saying like, I've written this program, for example, Meetup has written this program that they have all these cool things that their program can do, but they need a way for other computers to talk to their computers, so essentially they

send up like, here is our interface, if you want to do this with our program, like if you want to create a new Meetup, send us this data, if you want to send us... Or if you want to get all of our city data, which is what we're doing right here please call this URL with these parameters, right, it's just kind of this interface for two disparate services to talk to each other, that's what the word API refers to, so in this particular case we're calling this API, which is the city's API on the Meetup servers the same, please give me all of your cities, right, that's kind of the general gist of it. And then here, this data, that's the data that is going to come back from the server.

Displaying Data

Here, we're going to use the same API. Um. Actually, I'll just show you what it does first, okay? So you say click me, and like I took that same API and I displayed its data all nicely, like this is kind of showing you more real world-ish type application, right, like I'm in the city and I want to see the cities around me, right, I could use their API to figure that out, like. Hopkins and Saint Cloud, like I don't know where the hell any of these are so (laughs) Like, there's a city called Wyoming, Minnesota? That seems just really dumb to me, anyway, sorry! I'm just going to rag on your great state here, hold on. Hey we can even get down into Wisconsin. Okay, so that's what's going on here, let's see how we accomplish that. First of all, I'll just show you, this is not what we're actually talking about but I just threw on a little bit of lipstick on it right, like I put a little CSS on it, give it some border, text-align, margins, colors, that's about it, and you're welcome to take a look at that later if you want to. Okay? The HTML literally has not changed from the last page, so don't worry about that. Let's look at what's actually happening here, and it's really not that complicated. So, we do that text loading business again, we're doing API call using AJAX. Don't make yet, in the same URL on point, and the only thing that's really changed here is what's happening in the success function, right? So here we're going a full loop, and we talked a little bit about that yesterday, and we're looping over data dot results dot length, now you might be wondering, "Where did that dot results dot length come from? " So, let's look back here, so results is this part of the data that the API is giving back to me, right? So, I can reference that by data, which is this, right, data is the entire response and I say dot results, right, so. results refers again to this array right here which contains all of the cities. Does this magic connection make sense? Okay, cool. Um, I think this is like the fun part of programming with me, right? This is where you actually get to like, create and do really cool stuff. Okay, dot length, Nina talked about that, like it's how many elements are in the array, in this particular case I think we have 57? Nope, 184. We have 184 and then we're just looping over it, so this is going to get called 184 times, right? Right? I want to see some head nodding, okay there we go. So this is starting at zero? Starting at zero, because arrays

always start at zero, right, always. So if you start at one, you're going to miss the first one, okay? So, place, that's actually the name of the town that we're looking for, the, you know, blanks dot blank, we're going to grab the city, so data dot results i, right so we're going to grab that i place, so for example, the 30th loop through this will actually be 29, right, because it starts at zero, so result number 29 grab the city, okay let's go back here, whoops. Let's look at one of these, so every one has a city, which is right there, and it has a state, which is right there, okay? So we're grabbing the city, and then I'm just putting a comma and space there for nice formatting, and then I'm doing a state, which is right there so we're just doing city comma state, okay? And then all I'm doing right here is we're using this append function that we were discussing before, because if I did text right it would just replace it every time, in fact let's just do it to see exactly, or actually it would be HTML. And it would just be the last one, right it would be Cushing, Wisconson. So that's why we need to use append, because we need to say, "Please preserve what's already there, and please add on additionally to that. " That's it, right? That's pretty cool, isn't it? I think it's pretty cool. It's more data in the API call that you can display like the number of members per city. Sure, you can do it, you can show the zip code, you can show the ID, the country, in this case it's not particularly important, but... Is there some mechanism, that, once you get the results array back that you could figure out what is different, or iterate through the different objects that are in there without knowing what they are ahead of time? So, find out what numbers there are in the object? You certainly could, um, that would be difficult, I would say even borderline very very difficult, um, but I'm not going to say no. The easiest way to code for something like that is to just, like, exactly what I did right here just open it up, see what it looks like and then just code for that. I'm just wondering if you had some data source where the API itself may be dead data. It should be, that'd be a, uh, what's the word I'm looking for? It'd be really difficult to deal with. So, but let's do a thought exercise, why not? So, we have, let's just say var object equals data dot results i, right, so like this is going to go ahead and grab the i object, let's just go ahead and comment these out. Okay. I'm going to show you something called a For in Loop, so you're going to say for var prop in obj, so obj just refers to, or maybe let's just go with like cityObject, right, let's be a little bit more verbose, cityObject, okay? And, then you're going to say, uh, just let's do this actually. Um. We're going to say place plus equals and we're going to say prop. And, we're going to say city object prop and then plus space. This should get really long and really ugly really quick, but let's see how this looks. Um, and then we're going to say, here we're just going to say var place equals (mumbles) Oh, and then you have to say, if cityObject has own property, so I mean this got a little bit more complicated than I was anticipating but um, this is kind of how you would deal with an object that you don't know what's going to be in there. Jeez what happened there? So, if this doesn't make sense to you, it's cool, we got a little

advanced, okay? But notice, we now have the ID, we have the zip, we have the distance, we have everything that's in there, so what this For in Loop does is it loops over an object, and it says, "Give me all of your properties". Right, so in this particular case prop is ID, zip, distance, long number count, right? And then, you have to, because of JavaScript's weird inheritance, you have to ask like "Is this actually yours, or is this your parents? " (laughs) Sound like a child right? Is this really yours, are you 18? Um, so you have to say, "Is this actually yours or does this belong somewhere up higher? " Again, we're not talking about inheritance so if that doesn't make sense, just yeah don't worry too much about it. Then here we're just putting the prop first, which is like the ID, the zip, and then we're putting a dash, and then we're putting, we're asking for, what is the value of that prop right, because these are all what key value pairs, right, so there's country which is the prop, and then this is the city object with the square braces property. Okay, did we just go off a total deep end right there, or does that make some sense to most of you? Cool.

Exercise 9: Displaying Data from Reddit

Alright, so, we've been talking a little bit about AJAX, about doing some asynchronous loading of data, we did that just now with the Meetup API, we're going to do an exercise, this is going to be difficult, right? Like I'm essentially just throwing you to the wolves here. Sink or swim shit right here, okay. So, first of all, let's open this Reddit API call right here. So, first of all, you can look a little bit here, so we're doing a search on, aw, if you're not familiar with Reddit, aw is where they put all the cute pictures, it's the cute picture part of Reddit, and we're all about cute pictures here right? Okay, and then you're saying a question mark, Q equals, so this is the query that we're going to send to Reddit, right, so in this case is we're only looking for puppies, and restrict sr just means like, look only in this subreddit do not look in other subreddits. So, first of all, full disclaimer, I chose and I believe the safest subreddit on Reddit, but it is still on Reddit, stuff happens on Reddit so hopefully nothing is too bad here. Um, but right now, like all it's doing is sending us back a bunch of results from Reddit, like for example let's just go ahead and click on this thumbnail right here, little puppy right there, it's really small but. Puppy hugging what would be some other kitty, or something like that. Anyway. So, what we're going to do here, is we're going to take these results, we're going to loop over them, and we're going to display all the thumbnails from Reddit, it's going to be fun. Okay? If you want to see the API documentation, which is a good practice to get into, I've posted those right there, okay so here's the search, and we're just calling a get on the subreddit and we're getting the JSON feed, um, something I was going to mention before, some of you might be familiar with XML, JSON is a replacement for XML and a much, in my opinion, superior replacement to XML. But you could recross this in XML if you

wanted to. Okay, and then this just shows you all the parameters you can ask for, in this particular case, we don't really care about them. The only one that we're using is restrict sr, which is just like restricting it just to aw and not to other subreddits. Does JSON have any kind of a hierarchy like XML does? Yes, it does, so let's go look here for just a sec right, like we have cont and data, data is nested, has inside of it nested children, and children has inside of it nested data, right so there's this nesting effect. It's just in my opinion much easier to visualize. It's something that JavaScript deals with extremely well. Okay? So, here is the CodePen that I'm going to give you. Go ahead and ignore the CSS, I just wrote a little bit of CSS for that, and then I gave you some notes here using comments, so, first thing you do is you need to attach a button listener to here, the click me, right? On the click, you're going to make an AJAX request for the results of cute puppies. It's going to go to this URL right here, that's probably super hard to see, but, hopefully you can all open it and look at it. Okay, and then for each of the elements in data dot children, append a new image element, right, to text with the source child dot data dot thumbnail, okay? I do have some hints here for you, you're going to need click, append, and AJAX. Whoops. You shouldn't need to touch the HTML or CSS, this should be purely a JavaScript exercise, and this data that you're looking at here is really nested, so like you're going to have to kind of get into some interesting accessing of these properties. So, the data that concerns you is data dot children, like if we go back here, alright so we have data, which is right here, then inside of data there's children, okay, and then children as you notice right here is an array. So, all these different children represent different results of the photos, right? Inside of each child is another data attribute, and inside of there is a thumbnail, right? Do we get like, what we want here to happen? Is anyone confused by the instructions? It's hard, okay? So, like let's just walk by step by step here, um, just write a click listener on the button, so something happens when you click the button. That should be something that hopefully we can do. Let's see, I can also probably show you what it looks like, so let's... So, after you finish with it, I'll show you what it looks like. So you click me, and it should just like throw up a bunch of puppy pictures like that. The CSS is all taken care of so you just have to worry about appending each one of those image elements. So, again, we'll walk through this together, I realize that this is quite difficult.

Exercise 9: Solution

Alright, so how we doing with the click listener part, are we doing okay with that? Alright. So, because we're nice people and we like to let our users know, in fact, I'll just start going along with you, we're going to have, I don't know what did I call that button? "btn" So we're going to have dot btn dot click function, okay? Voila, click listener. Okay, and let's just like, we'll just throw up an

alert real quick just to make sure that actually worked, you'll find that this is a common practice like, did this actually work? Does this actually do what I think it does? So, we click here, hey it actually does like listen for click events, okay? So, awesome, step one done. Okay, let's go ahead and throw up a little loading thing to be nice to our users, so what did I call my little text, I guess that's a misnomer because it's not actually going to be text after we're done with it, but we'll be okay. Text dot, uh yeah go ahead. Um, somebody tried to declare a function before the AJAX call, and then they wanted to call that function where the anoynymous function was declared. Uh, sure, you can do that. But he says it's not working I guess. Okay, so you have like function, uh I guess it'd be like var function equals responder or something like that, okay, or sorry var responder equals function, okay? And we'll just throw like our alert in here right? Alert here, and then here you are passing, rather than putting in a function you would say responder, this should look somewhat familiar to you right because you can pass functions around like variables, okay? And then we should be able to click there and it says here, right? That's also possible, I'm a big proponent of doing an anonymous function and listener, because typically you only want to call that from one place so it makes sense to do it that way. (typing) Okay, so, let's throw up a dot text, dot text, didn't mean to actually do it that way, and I'm going to say loading like that. So now, when we click it says loading, okay, so now our user is on the phone is like, "I have received your input, hold tight, and I'm going to like give you "this thing that you're looking for. " Okay, so, now this is where we actually go out and like reach out to the server, so what jQuery functionality am I going to use? AJAX. AJAX, awesome. So I'll give you a sec to give that a shot again, if you want to. If you want to refer back to the one where we're reaching out to the Meetup API, probably would be a pretty good thing to do. The one thing I will caution you that's different from the previous one, is you know that part where it says jsonp for the Meetup API, just leave that entire line out for here, you don't want to bring that over. Reason being that the Reddit API and the Meetup API work differently that way. Why don't I even go back for just a sec and bring it up here. So you can go back to the earlier one that's just really simple. Okay? Again, this is the Meetup one that you click and it just blahs all over your webpage, okay, you've got text dot loading just like on our other one. We're going to do dollar sign dot AJAX, right, still doing a get request. We're going to a different URL, right, we're going to the Reddit API not the Meetup API, and then we're going to have a success function in there. It should look somewhat like this, in fact if you wanted to just test it out to make sure that's working, just go ahead and throw literally this exact line in there, right, text dot text JSON dot stringify data, and you'll see it should just blah the entire Reddit API response all over your webpage. Like honestly, this is pretty much unchanged other than leave out datatype jsonp, and change the URL and then it should work. Again, like, just as an overarching concept about learning to program, there is no shame there is nothing wrong

with copying what other people are doing and modifying it to your own use case. My only caveat to that is do your absolute best to understand what's going on and why you're doing it, it's a really bad thing to pull code out, put it on a page, and then not understand what's going on. Okay? So, let's go ahead and work on this, uh, API call a little bit. So, as you might imagine dollar dot AJAX, okay, type get, URL, sorry not function, rather, you are just literally copy and pasting this, I guess it's not literally, you're not cutting it out of our monitor, but figuratively copying and pasting, okay? With me so far? And then we just, we give it a success listener, which by the way we were talking about events, um, this success function is just an event listener just like a click function is, like, the AJAX goes out it completes and it fires off an event that says, "Hey, I finished the thing that I was doing. " And this is just like "Oh okay, he finished, I was listening for that, now I'm going to go do this. " Same concept at play here. Okay, and now we're going to say... Some of you might be wondering like where this data comes from, like is it just magical? It's one of those things where you just have to know that the API documentation will tell you that whenever a success function comes back from jQuery it's just going to give you data, like it's just, it is this way because they programmed it that way and you know that by looking at the documentation. So there's not some, like, secret handshake or something that you don't know about that's going on under the, behind the scenes, it's just, you have to read the documentation to figure that out. Okay, dollar sign dot text text, then we're going to do a JSON dot stringify the data. Okay? Lo and behold, now when we click, um, here we're getting the entire response just kind of, you know, vomited out on the page in all of its glory. We follow this so far, this makes sense, this is just more or less reviewing what exactly we did with the meetup one, okay? Alright, so, the question now becomes, we want to display them nicely as images, right, like we don't want it to just throw up on the page we actually want it to do something, okay? So the first thing you do is we need to go back and look at what the data looks like, in this case we have a thing called data, right, and that's where all of our data is being stored, it's inside of data dot children. So, I'm just going to get rid of that one. So, we're going to, um... So let's just get rid of this, okay? So, we need to loop over data dot children, right, because there's many entries in data dot children and you want to get all the images out of each one of them. So, we're going to write a For Loop right, that should sound like something that you would use a For Loop for, okay, so for var i equals zero. I is less than what, so we have data which is the whole object, in fact, I'm going to change the name of this to be response, and you might say like, "You can't do that, right, it's called data! " Well, in reality, you can call whatever you want, you can call this like, "this is not data" (laughs) Or whatever you want to call it right, this is totally up to you to call it like you just have to know that the data's always going to come in that first parameter, so whatever you call that first parameter, that's just from there on out you have to call it that. So I'm going to call it response, just for a little bit of clarity's

sake, so response. Okay, and I'll show you right so it's going to be response dot data. A lot of people when they see data dot data they get really weirded out, so I just wanted to like clarify where each one is coming from. Okay, so in response dot data, which response is now this entire object right, and data is this object right here right? So now we need to go into children, right, which is right here. So, dot children, and then we want to loop over all of children's, um, containing, like all the objects inside of it, so if you remember we can do that if we just say dot length. That'll give us the number of how long children is, okay? That's it, now we just say i plus plus, which means at the end of every loop, increments your counter, and now we have a For Loop that we're going to loop over each one of them, okay? So, let's uh, I don't want to do an alert because that would be really bad, but let's just do like a console dot log of response dot data dot children i. Right, which is going to give us each individual child inside of that array. So, let's just try that, see what happens here. Look at our console and we should see a lot of these objects just being logged out, right, we can look at any one of them and see like, hey look this one's called by author puppy, and it's from imgur, right? So now we're just logging out each individual object that's coming from the API, okay? So now we know that our API call is working, and we're looping over it correctly. We okay with that so far? Okay. Take that as silent compliance. So, awesome. Now we have this data, we have each individual child that we can address now, okay? What do we want to get out of each individual child, well if you notice each child has a thumbnail, right, so this one has a thumbnail, we scroll down here, this one has a thumbnail, scroll down a little bit, this one has thumbnail, that's what actually we care about because that's what we're going to be displaying, okay? So, what we're going to do is we're going to use our jQuery magics and we're going to say, dot text, what method are we going to use here? What happens if we use dot HTML here? We're only going to see the last one because it's going to replace all of them and the last one will be the last one displayed, yeah. Um, somebody's asking about if that success was deprecated, and you should be using done and fail. Yeah, so, yes but no. (laughs) So, there's a thing in JavaScript called promises, we're not talking about them today, it's kind of the newer way of dealing with responses, this is the easier way for people to get started into it, and they accomplish the same thing so we're going to stick with this. If you're interested in learning that one, there's a jQuery course... Deferreds. Yeah, I mean, they're called deferred's, promises, um, but yeah, if you're really interested in learning about that go check it out on, uh either front and masters or search for like JavaScript promises. Okay, so, actually let's just see when we built this before. Whoops, I didn't want to delete that, did I? There we go, I want that. (typing) So, we're going to build an image tag now for each one of these, right? So we're going to say this is like my image right, and we're just going to say img src equals, and then we just want to stick its thumbnail there, right, so we're going to put something here but we're just going to close this out

real quick, okay? In the middle here, it's going to be this response dot data dot children i, right, and then what do we actually want to get out of each child? We want to get the thumbnail, right? So we're just going to say dot thumbnail, so this is going to go into the response object which is going to go into the data object, then we're going to go through the entire children array, for each one of them we're going to grab out the thumbnail. That's what that says. We now have an image tagged, right? So, you might be tempted to do dot text, dot HTML, right, and then say image. Let's try that. Oh we got nothing, so that's not good at all. Unexpected token. Let's see, var image equals image source. Oh, that needs to be, hopefully that doesn't... Okay, so like, we're at least getting something there but no image. Source equals undefined, so what's happening here? You have a single quote and a double quote next to each other. So why, and, that's a good question why do we have that? Right, so this is a string literal, so this is telling JavaScript I literally why you put this there, but if you remember, you're looking at source, right, if I go up here and say, img source equals it needs to have quotes in it, right? So, you actually need to have JavaScript put those quotes in there. It's kind of weird, right? The other thing is if you wanted to do, well, and like the reason I don't do double quotes there right, because otherwise JavaScript's going to get confused right, in fact you can see my syntax messed up here. So, it doesn't care as long as you're matching quotes. Okay, now I'm having a hard time, there you go. So, this is going to be inside this string, and it's required because JavaScript, or rather, HTML needs that quote to parse the image source correctly, does that make sense? Response, data, children. In there, actually do you need a data attribute for the children? Yeah, that's it, thank you. So, what they kindly pointed out to me online is that each child has it's own data attribute, right, so there's data right there, there's also data right there, but notice I was getting undefined, why was I getting undefined? Because I was trying to access a property that didn't actually, that was undefined, right? So, JavaScript says, "You're trying to get something that's not defined, so "screw you, have an undefined. " Or, here you asked for it so you can have it, is probably more akin. I made JavaScript be a cruel mistress. Okay, let's... Let's try that. Hey check it out, you got a little puffy right there, that's nice. Okay, so now we only have one image, why? Because we're using HTML, when in reality we should be using append. Right, because we want to be continuously adding things. Okay, so now if you try doing this, lo and behold, many puppies. Now, we still have this loading thing here, why? Well, when it loops through the first time, it's not going to replace it it's always going to append, and like once you're done loading you actually want to get rid of the loading thing, right? So what we would do is we would just remove it as soon as it came back, so before the For Loop, we're just going to say dollar sign dot text dot HTML, or text, either one would be fine. And just say that, please clear out everything you have, make yourself an empty string I'm about to give you a bunch of more stuff so you can have content. So now if we say click

me, now we actually have real puppies. And some of these thumbnails won't work, um, I'm not really sure why, but that's the case. And you can do all sorts of fun things now, right, I can pick and say cat, and we'll get kitties back, or we can say duck, and we'll get cute ducks back. Look, little ducks. Okay, but kind of cool right? We've now built this app that will give us a bunch of like cute pictures from Reddit. Really, this is kind of the first app we've built, pretty entertaining right? Now, if you wanted to take this a step further, what could you do right? Like you could actually have like an input box that the user would put in something, and then you would change the URL here and you would put whatever the user input was there, and then you would change like what you were searching for based on what the user asked for. Like, I hope you're kind of starting to see like, here I'm giving you a bunch of puzzle pieces that fit together in a bunch of different ways, right or like Legos is probably a better way of putting it, and you can build a spaceship or whatever, right? You can build whatever you're using as tools, whatever you can think of. Does this make sense, what we've done here? Now, if I asked you to go write something really similar to this do you think you could do it now? The answer is like, probably with difficulty, but the answer is with difficulty is better than no. Some of the pictures are coming back as not found, um, is there a way you could capture that in your loop and say okay, don't display those, take them out? So, probably what you would want to do is you would have some like, is this a URL? If this is a URL, then display it, if not don't display it. So if you want to save and try it, if it's displayable then display it, if it's not then don't. Um, I'm not going to do that right now, but that's what you would do. You can also like change the subreddit here, and I do not want to change the subreddit, I trust aw, I do not trust the rest of Reddit. But you can also search for other things on other subreddits as well. Like, that's kind of something you can do. How about this, does this little line right here, the response dot data dot children, dot data thumbnail, does that make sense to people? Like have we talked about that like you're accessing properties inside of an object that just gets really, really, really nested.

Troubleshooting

So this one's from, I'm sorry if I'm saying your name wrong, Siyu? S - I - Y - U. So this is the one that he sent in, um, and this is a really really common problem that you're going to run into all the time as a web developer is like not, in fact you guys saw me do it just like five minutes ago, um not getting all the property names right or the nesting levels correct, that was his problem, he just needed an extra data right here right, because you have to go inside of the entire data object, and into the data object, right? That's how I change this to be response, and as well as this one, like this makes a little bit more clear, right, this is the response object and inside of the response

object I'm going into the data one. He also doesn't need this jasonp, so let's just get rid of that. That wouldn't hurt anything it just wouldn't do anything, so, I think now, yeah his should work because of that. Uh, so we'll leave this page, and this is the one I was looking at for Elizabeth, Elizabeth W. if I recall correctly, I mean she's essentially all there, but if you look here in the console, which is by the way your best friend, it's telling you that there's an unexpected end parentheses somewhere, I haven't found it yet, but it's in there somewhere, and usually they'll give you a nice, uh, in fact I wonder if I can look at it. So, it's right there. So after the text dot append photo it's not expecting this parenthesis right there. So I don't think you need that as well, so let's try that. So that apparently hasn't worked, let's go back to our console. Unexpected end of input, so let's check that out. As you can see, the console is going to be really useful because when you have errors it's going to output your errors to the console. As you can see here though, it's still a syntax error I just haven't quite figured out what it is yet. Okay, so we have the AJAX, I should start rewording some of this. I'm going to talk about data here in just a sec, um, but for this second just bare with me. Something else Elizabeth had is that your search parameters do have to be inside of data if you're going to split them out like she had. Another word of caution we give you is in JSON, they expect commas, so if I had like type get out here and I omit the comma, the whole world ends, actually, so I mean it's like goes down in a fiery blaze of glory. So you need commas to separate different property values. I think there's a missing curly brace in the For. Let's see, let's indent this so I can actually see what's going on, so this one is matched up to that one, right? Success is matched up to this one, and this one is matched up to AJAX, right? Not AJAX, the anonymous function. So, which one is this one matched up to? It's not, so another issue, as you might be seeing here, is like matching parentheses matching curly brackets, matching square brackets. It's very important that you get into good habits or else you run into problems like these, that you feel like are really dumb problems, but in reality like JavaScript can't read your file unless it has correct, um, curly braces, let's put this back on this line. Compact this down a little bit, so that's the AJAX object, the AJAX object ends there, okay and then I might need another curly brace right there to match with the function, and I need one for the click which would be like that. Still unexpected end of input, whoops. Nope, okay. Okay. (clears throat) So, function there, that one closes there, this one closes there, if you don't, if you can't see what I'm doing, you can see that when I click on this curly brace right here, it has an underline right there of the one that the computer thinks it matches up to, that's exactly what I'm doing right now to try to figure out where everything thinks that it should be ending. There's the click button, so those should all be matched now from what I can see. Is there anything above or below? I don't see it. Uh, success, is that, uh I don't know. Alright, so, my next step on a lot of these is just to start commenting out blocks of code and then you can start figuring out what's wrong. Alright

so let's just comment all of that out, and what I've done there is sorry I don't know the Windows one, but it's command forward slash, and that'll comment out whatever you have selected. So, now this code no longer has any effect, so it kind of gives you the ability to figure out what is going wrong. (voice is drowned out) Cool, so that means something in this commented code that is wrong. Did you try to console out the brackets inside of it? Yeah, let's alert here, okay. Oh, I guess it went back to replace what you said. So, now we know for a fact that something is wrong with this block of code, so we're kind of narrowing it down of where the problems could be. So you have to, let's take a look at our For Loop here. Now that you've got this we can just like comment out the For Loop body and see if it's a problem with our For Loop, and say, open back up our console. Okay, so it looks like it's something in our console body. Okay, so it's something here, so we have image source there plus response data children, plus-- Why is there a slash before image? There shouldn't be, at least now we're getting somewhere, now like it's not crashing we're just not getting the actual images. So now I have response dot data dot children and she did the exact same thing I did, she forgot data. Puppies! (laughs) Talk about like the best consolation prize, right? It's like, you fixed the code and now you have puppies. Okay, so hopefully that was an interesting exercise in debugging for you. It's a difficult skill to learn, and it's one that you don't want to develop but you have to develop, so. Well, sometimes it's, majority of your time, when you're working on existing code bases. I'll spend whole days not writing code and just fixing bugs, actually I'd say recently that's been the case a lot. So, hopefully that was good for you guys, hopefully somewhat useful I was going to talk a little bit about data, so you remember how I had like the question mark q equals, um, jQuery is an excellent helper for that. That's weird, okay. Called data, and so they will just like dynamically make that for you so instead of having question mark q equals puppy and sr true, like all you can have to do is give it an object and JavaScript will go ahead or jQuery will go ahead and make that query string for you. This makes it easy to make dynamic, right, like if you wanted to be able to change up the query or whatever, all you'd have to do is feed it a variable and you'd be done. Cool, so I fixed two of them-- How many did you do, two? Two. Did you get the one, Jim S. posted one. I haven't. Did you get that one? Oh. Let's see. (mumbles) So full disclosure, I have not looked at this yet, so. Let's take a look at this, alright so you guys look, button click, well let's just see what's going wrong first. So, he gets to loading, let's take a look at what his console is saying. Failure to load insecure response, oh I bet you he has jasonp down here. Oh no, he has HTTPS, and it doesn't look like he's actually making image tags, right, so first thing, I mean I wouldn't expect many people to recognize this right away, you can't load a secure content onto an insecure page, well you can but not in this particular case. So the fact that he has s here which means it's a secure connection, that caused him some issues, and also I don't think it's served securely anyway, for that matter, so it needs to be served insecurely

because that's what it's expecting. Does it need a query on there as well? Does it not have one? No it has it right here, below the data. Oh okay. Okay and then as far as like this one, he just doesn't have the first part of the image in here as well, so you need to, um, img src equals okay, and then I, surprise puppies. So, interesting problems, any questions about like, how I did that or what I was doing? Like if you're going to do any amount of web development, this is just going to be something you get very familiar with doing. It's also typically easier, outside of CodePen, to debug, but, a lot of the same techniques still apply.

Command Line

Introduction to the Command Line

Okay, so let's talk about the command line. In the beginning, there was the command line and really not much else. If you wanted to do any sort of computing this is what you got before Windows, GUIs, even something as simple as a hardware mouse, everyone used this. And funny enough it's a 30-year-old technology but we're still using it today. It's a little bit different on every platform. So Macintosh and Linux are probably the most similar, although they're not exactly the same. At some point the Macintosh OS was based on Linux but they diverged. So kind of similar but not that much. And then we have Windows which is a whole different beast. So we're still using the command line today because you can just complete repetitive tasks a lot faster. Instead of clicking there, moving that over there and copying and pasting that. You're just running one command that can do all these things for you. And those commands can be automated. So if you guys see the term scripting out there, that's what that means. The other cool thing is that parameters to scripts can be provided which means you can run the script but does something kind of similar, but you're passing in different input. So parameters are a way of customizing how a command is run. And flags kind of do that same thing. We'll see flags later on. So here's what a command prompt looks like on Mac OS. And if you guys want to see that you can go ahead and open up your Terminal app. On Windows that would be command. exe, cmd. exe from the Start, Start and run menu. So here on my Mac is the name of my computer, which is Ninas-MacBook-Pro-2. And then Nina, which is my username, the little tilde before Nina means that I'm in my home directory, and the dollar sign is the end of the prompt. So I can type whatever I want after that dollar sign. So our home directory is on our machine where the most common place where we can write files. So on Mac and Linux, if you have your command prompt open and you type

'cd' it'll take you to your home directory. So on Mac that is / capital U Users / your username. While, on Windows it's usually C, the C drive, and then users and then your username. Something really important to note here is that the slashes face different directions. Do you guys all see that? So big source of confusion for beginners. On Unix-like systems the slash is face-forward. So if you're ever on a server or using someone else's Mac computer this is what their fault structure will look like. While on Windows the slashes are backwards and that's Windows and only Windows. So here's a way to find out what directory you're currently in when you open up your terminal. So if you guys on Mac OS will open up your Terminal and type the command 'pwd' you guys should see what your home directory is. For a Windows users that command is cd which stands for Current Directory. Does everyone have that working? Do you guys all know? No, no. Go ahead. Okay. Just type 'pwd' and then Enter and then that'll print your current directory. Does anyone else have questions about this? No? So here's the command to list our directory contents. On Mac OS it's cd. I'm sorry. Sorry. The title on this is wrong. Let me just change that really quick. Here we go. Sorry guys. So this is how you would go about to move to a different directory. And also a little bit confusing because on Mac the command is 'cd', which is in this case stands for Change Directory. So you would type 'cd' and then a space, followed by the path to the directory that you want to move to. And then you would type in 'ls' and that would list the directory contents. Oh no, sorry. I'm wrong again. That's why it's just the same on Windows also. That you have the dir command on your Windows. Yeah. You have two commands with probably the same title. Sorry. What I wanted to do is that is listing directory contents that we have there. Sorry you guys. I will say this right now that I am not a Windows person and I have not used a Windows machine in about five years, so. That is definitely not my forte. So to list directory contents on a Mac, you would change directory to the folder that you want to be in and then you would type 'ls'. And you'll get a list of items like this. So you'll get your folders and your files. On Windows you would type 'dir' and then the folder that you want to see the listing of.

Navigating with the Command Line

So something really important to note here is that Linux and Mac OS, those files are case sensitive. So what that means is if you have a lower case and you try to refer to maybe an extension that's all upper case those files are not the same. Except Windows doesn't really care. So that's really something to be mindful of. Yeah. Windows honey badger don't care. So to navigate to a different directory we would 'cd' which I showed in the slide before. It stands for Change Directory. On Windows we would use 'chdir'. So a little bit of a more of verbose command. If we wanted to move up one directory, and I could do a quick demo of this for you

guys. So here is Brian's terminal. We can go ahead and use some of those commands that we learned. So pwd will tell us that we're currently in the user's Brian Talks Intro to WebDAV directory. If I wanted to see what the contents of this directory were I could type ls and I could see that we have some files and some folders here. If I wanted to use a flag, this ls command accepts a flag -l. So this is sending in an option to this ls command. So -l means "I want the stuff as a list. " And even though we're running the same commands we're getting the information back in a different format. So to move up one directory, you type in this cd command and you follow it by two spaces. So now if we do our pwd again we'll see that instead of being an intro to WebDAV app, followed by the app folder we've moved up one directory back. So we hopped up to the parent directory. Does this make sense to everyone? Yeah, all right. So this is a really important slide. And relative versus absolute paths is something you'll just come across over and over and over again in web development. So a relative path is a path that starts from the point in the directory structure that you are in. Well, an absolute path means it starts all the way from the beginning. So let's go back to our terminal. And we'll go back up one directory. We'll do our directory listing. So here Brian has a bunch of different folders that I represented in pink. If I wanted to get at something that was in the files folder from this point in our directory structure, which we can see via pwd. So we're in Users Brian Talks we could just say files, whatever, like whatever file is in here we can access it from here. And the important thing to note here is there is no forward slash at the beginning of files. If I wanted to get to this file from anywhere else in the true structure and know that I was always starting from the root you type this forward slash first. And that forward slash is just the root of all the files here on the Mac. So if I navigate to here and list my files we'll see everything. We got our applications folder, the library folders, the users. So from here to access that talks directory we would do Users/Brian/talks and then we would do files and access anything in that files directory. So that's providing the whole path. When you're doing web development a lot of times you'll be trying to access files and directories in a relative way. So in your web app, for example, you'll have a public folder that might have a JS folder with all your JavaScript files or an images folder with all your images. And so you would, "I want image. " And so I would do images then the forward slash, which means the folder, and then access a file in there. Does that makes sense to everyone? Yeah? Cool. Someone once explained to me relative paths versus absolute paths is addresses, right? An absolute address is like she lives on like 200 Mainstreet, right? Like do you know exactly where to go? I can say from anywhere in the city go to 200 Main. It doesn't matter where you are. 200 Main is the same spot. But if I'm giving you directions like go down to the gas station and take a left and then take a right. That is relative to where you are. And like if you get that same direction in another part of the city it's no longer true. So that's relative from where you are and where you're going. Yeah. I really like that analogy.

That's a good one. Yeah? Is anybody else having trouble with the dir command on Windows? Saying command not found? When you mentioned open git, did you have to open git bash on Windows? So just a quick note on that. If they're using git bash, git bash is smart and they can go ahead and use the Linux style commands in git bash. But then I think the Windows ones won't work, right? Right. And I think we might be. So if you're on Windows and you want to use the Windows commands, open the cmd. exe on your Windows computer. Yeah. If you are on Windows and want to use the Unix-style commands you can do those in git bash. So here's a visual kind of directory structure. Someone there is trying to hide something. But if I wanted to do an absolute path to see. html I would do /www/, the name of this bottom folder which is cut off here, /c. html. Now if I was here in this XXX folder and I wanted to access b. html I could just type b. html and I'd be there.

Git and Github

So let's talk about git. Sure all of you guys have learned about Github. So Git and Github are actually two different things. Git's a tool and Github is, it's a platform where they kind of use that tool in a social way. So they host lots of code from lots of different people and it's mostly open and everyone can go ahead and check it out and have a copy of that code. So the advantage of using git is git is a version control system. So you can save snapshots of your files as they were in different stages and if something's going wrong and you need to go back, you can kind of rewind and see what that file look like at a certain point in time. So mostly used for code but version control is also great for even a text project. If you wrote a paragraph and you put it in your version control system then you deleted it and like two days later you're like, "I wonder what that paragraph said. "It actually had a really good idea in it. "I want it back. " You can cruise back through history and take a look at that snapshot. So git works perfectly fine on your local machine, but it's nice having a remote server that you can sync up your changes with. So if anything happens to your machine your changes are still there. Github is a remote server. So you guys all have Github accounts. We're going to go ahead and fork our first project. So I made a test account that I can use with you guys. So I'm going to sign in. I'll give you guys a few minutes to sign in to your Github accounts. Is anybody not in Github? Anybody need more time or should we keep going? That's it, keep going. You guys all following along? Great. So Brian has this project that we're all going to make a copy of, make some changes to and then push them back up to his project. So go to Github. com/btholt and look at Brian's repositories. And click on this one, Pull Requests. So we're going to go ahead and fork this project. And what fork means is take a snapshot of everything that Brian did, copy it over to my machine. I can make changes, I can use

my version control and it doesn't really matter what Brian is doing. But the really powerful feature of forking is Brian's working on his project and I'm working on my project, maybe Brian decides that what I'm doing is really cool and he wants to include what I've been doing into his repository. So he can go ahead and take my fork and copy it over to his code. And so now we've forked, so we're in the prongs, we're doing two separate things but then we joined back together. So let's go ahead and click this fork. And this is going to forward you to a different URL. So now instead of being on Github. com/btholt I'm now on Github. com/ninatest. So this is my own project. So now take this HTTPS clone URL that's right here and copy it. You can do that using this button. We're going to go to our terminal. And go to any directory where you'll be able to find your files again. So on Mac you can do the cd ~/Desktop. That will take you to your desktop. For the Windows users, you guys will want to be using that Git Bash application for this part. So we're going to take that URL that we copied and type in git and clone and then paste it in. What this is going to do is pull down that code from that server, make a new directory for us locally and copy everything that's in there. Does everyone have this step complete? You guys all have your forks checked out? Okay. So now we can go into this directory. It's called pull request one. This is a pretty boring project. There's like really nothing in here. So I want you guys to do an exercise. We're going to liven it up a little bit. So in this directory make a new file, call it your first name. txt. And in that file put a fun fact about yourself. So I'll give you guys a few minutes to do that. What you would call it? Your first name. txt. That's tilde, first name, and then first letter of your last name. Okay, yeah. Because we have 110 people. All right. So likely first names will collide. All right. Let's do a first name and a first last name letter. txt. Okay. How do you want me to create the files? It doesn't matter. You guys just need to make sure this file is in this directory. (indistinct talking) Did you fork the repository? No. Go to Github. com/bholt. Btholt. Btholt. Btholt? Btholt, yeah. Oh yes. (indistinct talking) Make it a new file. Make it like a text file. Yeah. Plain old text file. (indistinct talking) If you are on a Mac and you want to make a new file from the command line or on a Linux machine there is a fun command called "touch". So you can say touch and a file name. And what that does is just creates a totally empty file. So if I do an ls now I'll see a ninaz. txt file in here. If I'm here on Windows just go, we're in the desktop, right? Yeah. (indistinct talking) Okay. The folder on the desktop. So then just drop txt file, open a notepad-- (indistinct talking) Open Notepad or-- Yeah, seriously. Text edit and-- Do you guys get this touch command working? Yeah? Okay. I think on Mac you can just say Open and it'll open up a text edit.

Git Status

So now that you guys all have this file I'm going to introduce a command that we'll be using over and over and over again. And that command is Git Status. So let's go ahead and type that. Can you clear the terminal so you're kind of towards the top? Yeah. So Git Status. So you guys should all see this. There will be a section called untracked files and your file will be in there. Does anyone not see that? Cool. There's a lot of information when we run this command and let's go through it line by line. So that first line on branch master tells us that we're working in the master branch. That's the main branch for most repositories and we'll cover how to create new branches a little bit later on. That second line it says that my branch is up to date with origin master. That means that no one else has-- Yes? I think for those online. I think a couple of people are behind. Okay. One person is getting a syntax error when they're trying to clone it. Maybe you could just show how to do the cloning and then the URL. So the URL is going to be different for all of us. For each person, right. Yeah. But that is-- And then the other thing, they're like wondering what to do after they clone. Maybe could you explain how you created that directory? Yeah. Okay. So that's the Git Clone command up there. And then if I clear that out and I go up one directory and I do an ls I'll see that on my desktop there is now a directory called Pull Request one that was created when I did a git clone. Yeah, so there is this git clone in a URL. All of our URL should be different because we all took a fork from Brian's branch. So you should be git cloning Github. com/ and then whatever your Github username is. Could you just show on the video where you get the clone URL? Yes. I think that's-- Oops! Sorry. Brian's computer is set up all different than mine. Okay. So just again, I went to Github. com /btholt/pull-requests. And I can put this URL in the slide. Get that fork out of the way. Can everyone, who's following along online, see that? So then the step two, create a fork, should go to this URL and then click on this guy right here. There's a helpful pop-up. It says, "It'll fork your own copy of btholt/pull-request "to your own account. " I see that there are only 29 forks, so I'm just going to give you guys-- Refresh the page. We'll see. Yeah, 30. 30. There you go. How many viewers do we have on the live streaming clip? Right now I got 77. Oh, 44 people. I caught you. Who doesn't have a fork? Do you have a Github account? Yes. Okay. No, I got that but I was just-- You don't want to? Okay. Anyone else in here? I see 31. You guys in the room all have your own forks? (indistinct talking) Yeah. (indistinct talking) So Brian's repository is very, very popular. Now refresh the page and see how many people. Oh, that's all right. Okay, I'm going to keep going. So we made a text file, that's our first name and the first letter of our last name. txt in the repository folder. So just to reiterate, right now I'm on my desktop folder, I ran the Git Clone and I put in the URL from my fork. And I'll show you guys where that is again. So if I go to my own Github. Oops, See there's this pull request and the URL is right here. HTTPS clone URL. You could just click this handy icon and it'll copy it your clipboard. So if I do an ls in this directory I see that I have this pull request folder. I'm going to cd, change

directory, into that. If I do an ls here I'll see that I have this txt file. And now I'm going to run a Git Status.

Git Add, Commit, and Push

So Git Status is saying that I have an untracked file. That means this is file a that git knows nothing about, it's never seen it before and you can't do any sort of version control if git doesn't know about it. So the command to add this file to your git repository is pretty straightforward just call it Git Add. There's a question. Did you actually just create that file? I did. He was wondering. Yeah. So you guys create one of your own. So what was it again? Git Add and then the file name? Do you have a file? No, I don't have a file yet. Okay. So go ahead and create that file in the directory called pull requests. Open the folder and then create a file? Or use a text editor. (indistinct talking) Notepad. Yeah. Yeah, open a notepad and then type in. How do you open Explorer-- (indistinct talking) So again git knows that ninaz. txt is not part of the repository. So I'm going to type 'Git' and then 'Add' and the name of the file. I'm in the matrix, you guys. It's all black and green. (indistinct talking) Okay. So have you guys added your name file to the repository? So again we're going to be running this command after almost everything we do. We're going to say, "Git, give me your status. " And it's changed. It's like, "Oh, we have some changes to be committed, " and now it knows that we have a new file. So there are two steps in making sure that all your files are version controlled. The first one is doing this, this Git Add. And what this does is it tells git that there's a new file in the staging area. And the staging area is kind of like a preview. It's like, "What's going to happen when I do the next step? " And if I make a new file right now, let's just say, and I do a Git Status, git will say, "Oh I have a new file but I also have an untracked file. " So when I do the next command, the next action it's called a Commit. It's only going to be taken on the files that git is aware of. So not the untracked files. So the next step is to do a Git Commit which goes ahead and actually puts it in a place where Git knows that it can pull it back if you want to look at it later. And the way that we usually do a Git Commit is by providing a message. So we'll be using the -m flag here and two quotes and we're going to put our message inside. Is there a way you can move that up a little bit? Sure. So when I commit this, git's going to give me a message. It's going to say, "One file is changed and one insertion. " Now when I do my Git Status again I only see my untracked file. And for now, just for example, I'm going to remove that file. I could just do that in the finder. Boom! That's gone. So if I run Git Status again that file is gone and git is happy. But we have a new message here. Git is saying that my branch is ahead of master by one commit. So in this case origin master is Github. That's kind of our grand authority. And git is saying that we have local changes that git, I'm sorry, Github does not know

about. So in order to push up our changes to Github we're going to type one last command and that is Git Push Origin and Master. Does anyone have any questions? I know this is a lot of information to chew on. I would make notice that yours is pull request-1 and everyone else is (inaudible). Oh, yes. Sorry about that guys. I did a little bit of a test before. So it asks for a username? Yes. Put in your Github username and your Github password. Okay. (indistinct talking) Okay. (indistinct talking) I'm totally lost. Colon and then the q. (indistinct talking) Is anyone not at this point? (indistinct talking) So the next step, and this one is super important, is some of you guys have to pat yourselves on the back because this is your first git commit. I'm serious. Okay. So after we did our git push our changes are now synchronized with Github. Yeah? Do we want to wait and kind of let everyone catch up? Yeah, yeah. People online are kind of I think behind. Definitely. I'll go ahead and wait a few minutes. Does anyone have questions about any of this? Yes. So we just updated my branch that I forked on my Github. Yeah. And the reason this is happening is because maybe Brian thinks your changes suck and he doesn't want them to be part of his repository. That means that he's in ultimate control of what code goes into his own repository. You can do whatever you want with your copy, but he doesn't have to accept it back in to his code. And you can go ahead and make your own projects. You can fork something and make it into something new. Yeah. Like actually one project that I created which was a little (mumbles) plug in someone forked it and then made it super popular. So the fork actually has like 500 stars or whatever. Yeah. But my main repository was forked from, his is actually the more active and maintained. So you can fork something and make it your own and then it can get popular from there, too. Yeah. It doesn't ever have to be put back with the branch that it was forked from. Yeah. It kind of goes it's on way.

Cloning Review

I think online I'm trying to check but I think everyone is kind of has a clone locally but then at that point they're kind of, maybe if you could just kind of walk through that again. Okay. And explain like once you clone how to get in to that directory. Absolutely. Is it the Windows folks that are having trouble because-- I think it's a little bit of both. I think they just didn't understand after cloning that had to get in to that directory. Okay. I believe. So I'm going to take a stab of doing this on Windows. Nobody is allowed to laugh at me. And then one quick question. Can you explain the difference between Add, Commit and Push? Yes, yeah. So I'm going to open up my-- Nope, not the Git GUI. I'm going to open up my my Git Bash. And that is not, let's see-- Oh boy. So if we all wanted to put our name files into your project, would that-- It should go over that just the same. Yeah. That's the next step. Next step. All right. So here we got our Git Bash open. And

even though we're doing Windows we can use all of our Linux commands here. Okay. (indistinct talking) I'm on to you. Which one of you was it? Damn Windows people. That's like 80% of the class (laughs). I know. Yeah. We weren't even going to cover Windows originally and then we sent out the survey and we're like, "Oh, okay. " There are still Windows users out there. You guys exist. You're not just a myth. So the very first thing that we want to do is type pwd. That's going to tell us what directory we're in. So when I type pwd I can see that right now I'm in the C Users Brian Holt directory. Let's do a dir here and see what folders we have, an ls here and see what folders we have. We have some local settings, we have some links but here we also have a desktop folder. So I'm going to cd into my desktop folder and say pwd again. And now I can see that I'm in C Users Brian Holt, which is the username for this Windows machine, and Desktop. So now I'm going to grab-- I forked Brian's repository. I have my own copy here. I'm going to grab this clone URL, and go back to my Windows, and say Git Clone and I don't know how to paste on Windows. Let's see. Right click in there. (indistinct talking) Oh you don't? Okay. All right. I'm just going to type in this URL here. You right click on it and then-- (indistinct talking) Right click isn't working. So Github. com/Nina-test /pulll-request-1. git. Let's see if that is it. Oops. Come on Windows magnifier thingy. 1. git. And like Brian said yours will just be named pull-request. git. Now if I type ls, here I have a pull request-1 directory. So I can cd into pull-request-1 and do an ls-- -1. Oops, -1. No. Dot, dot. Cd dot, dot. There you go. And here because I'm using the same account my ninaz. txt is just right here. Create a new one. But let's say it's gone. No more ninaz. txt. So I think with Windows, what I can do, is open up this Explorer thing here and navigate to my, let's see, navigate to my desktop, navigate to this pull-request-1 directory. And let's see. Oops. Sorry guys. It's hard to navigate with this magnification stuff. So I'm going to find my text tool which is probably Notepad. No. All right, Windows people, how do I make a text file on this thing? Just type notepad. (mumbles) Boom. Wow. This is a fine work of technology. We got our Windows Notepad. So I'm going to take my fun fact and I'm going to save it, Save As, go to my desktop, go to my pull request-1 directory and save this file as Nina. txt. So now I'm going to close this guy, Since that file has the same name as the one that was originally up in the master? Are you going to get your conflict now-- It's just Nina, not NinaZ. Oh. Okay. I drag this window in the sites to attach at soft places. It's fun. Okay. So now if I do an ls my Nina. txt file is there now. Okay. Are people online following along? Are there anymore questions? There haven't been anymore questions. So, so far so good. Okay. I can Windows and so can you (laughs). So let's run this Git Status command which will really be running after any git action we take. Okay, it's mad at me. It says that I've deleted NinaZ. txt. "No change is staged for commit. " And I have an untracked file. I'm going to Git Add Nina. txt and do my Git Status again. Cool, I have a green one. These changes are going to be committed. So in order to commit that file we do a git space commit

space -m and we're going to put a message in quotes. So I'm going to say, "I messed up. "Here is another file. " Once I commit this file it's out of the staging area. Git's no longer telling me, "This is what's going to happen "with which files. " It's like, "Okay, I did it. " Right now, your staging area is empty. And this file has moved over into being committed. Oops. So if any of you guys got this message I'm going to go ahead and do exactly what git is telling me. And it's saying it doesn't know who I am. So I'm just going to copy the commands that it printed out for me. So this is saying, "Git doesn't know who you are. " Once you run these commands, the email address and the name that you put in here, are going to be associated with your git commits. So don't put See More button here. I'm going to do this commit again and it went through just fine. If you want to see a history of your commits we're not going to go too much in-depth into this on the command line. But you can type git and then Log. And git shows you all of your commits. Does that commit then also commit the deletion of the other file? No, because it was not in my staging area. The deletion of that file, let me do that again. Git Status. It's deleted but when I do my Git Status it's saying that this change is not staged for commit. You have to actually type-- You have to do git rm and the file name and then that would stage it to be removed from master. Yeah, yeah. Not from master, but from your commit. Well, yes. You can say for master but your changes wouldn't-- Git rm file name and then commit then it would remove it from the master. Yes. So I did a git rm. Until I commit my deletion changes to be committed. Until I commit that file is still, it's not there but git still knows about it. But as soon as you commit that then it actually is gone. And the messages that you'll get using git status are actually super, super helpful as long as you can kind of follow along with the git terminology. So under changes to be committed you'll see this message. "Use git reset head, " which is all in capitals, and then the file to on stage. So let's say I did this git rm and I'm like, "Oh shoot! "That file was really important. "I don't actually want to delete it and then commit it. " I can just copy this exact command that git is telling me to type. And it went ahead and unstaged those changes. So if I do my git status now it knows that I deleted this file but it's not going to push that back up stream. Are all the Windows people caught up? So I did my commit but the one-step that I did not do yet is do a git push origin space master. And this command is going to sync my changes with my Github. com account.

Navigating Github.com

So I'm going to go back to my Github account and we're going to poke around a little bit. You can go ahead and refresh the page. My name would go here, which is what I typed when git asked me what my name was. It says that I changed this three minutes ago. So we can go ahead. It'll give you a link to the latest commit. We can go ahead and click that. Oh, I messed up. Here is another

file. And Github is showing me the exact changes that I made. So if you don't want to fiddle with the command line, which can kind of be a little bit obtuse, you can do a lot of things from the Github. com site. And there's also a UI client available for both Mac and Windows. Okay? Yes? There's a question on-- She's saying every time that she modifies the file should you use Git Add again or is it committed directly? Git add. Right. So here's a funny thing. If you modify a file and you do Git Add on it and then you change it, the first time you called Git Add that was the version of the file that's in your staging environment. If you do any more changes and you don't do Git Add again, when you commit, git will commit the version of that file from the last time that you run git add. So a good git workflow is lots of commits. Anytime you do something they're like, "I'm not sure about this. " Go ahead and add it and commit it. And then you always have a history of it later. If you do commit something and you decide you didn't want it, how do you get back into-- That's kind of outside of the scope of this class. There's a few different ways. But the thing that you're probably going to want to do is create a new branch. So your master, which is like the base branch of any repository, is going to stay clean and working. And then when you want to work on a new feature you would create a new branch. Hack away on that branch. But you always have master exactly where you left it. Does that make sense? What is the difference between a branch and a fork then? I guess I didn't quite understand that. So a fork, you can kind of think of it as a copy of someone else's code. A branch is a copy of your own code. So forking is taking a different repository and making a copy of it that has all the history that that repository has had. So you can see the changes on it. But now you have your own copy. You would make a branch of your own repository. Does that make sense? Fork is a branch. It's just your branch of someone else's code. So we're going to make our first pull request. So we're going to go back to our copy of this pull request, our fork of this pull request project. We're going to scroll over to the right, click on pull requests. And we're going to make a new one. So we can see that Github is being smart and it's like, "Oh, you fork this off btholt. " So I'm going to make a pull request that has all the changes that I made. So you can go and review it. You'll see the changes that you're going to be pushing back up. And click on Create Pull Request. Fun fact about Github comment fields. You can put in all sorts of emoji. So let's make this one colorful. Let's see. Let's put some cats in there. Once you've created a tail and written a message we're going to go ahead and create this pull request. So Github is telling me that I want to merge in two commits back into this original project that we forked from. And those commits are coming from my project. Does anyone need help getting to this point? I think people are wondering how to get to the pull request. So in order to do that, I'm just going to go through it again. I'm going to go to my copy of this repository. And these links up here, this one will take me to my account and this one will take me to this project. So I go to the root of this project. I scroll to the right, click on Pull

Requests, and then New Pull Request. Because I've already done this and I don't have any other changes. Github is smart enough to be like, "Hey, you already did this. "Do you want to look at the one that you already created? "

Pull Requests

One person online said they are at the pull request page, but can you maybe just explain generally more of what pull request is? Sure. I've taken a copy of Brian's code. I've hacked away on it. I did something really awesome and I want to be like, "Hey, Brian, you should use this code "that I wrote in your project. " So by creating a pull request, what I'm doing is making a summary of all the changes that I did. So when Brian goes to look at this, he can see that I made these additions, I made these deletions. And he can decide whether or not he wants to take the code that I made and put it back into his own repository. So he's merging it back in. If he checks out my pull request. If he chooses not to, nothing happens. His code remains exactly the same that it was. So a pull request is really a message to Brian saying, "Take a look at this. "You might want to merge it back in. " Yeah, exactly. So Nina's workflow stops here. She cannot force me to accept your pull request. Later I'll get on and I'll look at her code. I'll either approve it or like I'll give her some messages like, "I like this but change this and this and this first. " I can push it back to her to make more changes or I can say, "Okay, I like it. " And then you merge it in. And so now both of our code base is kind of merged all of their chains together and now only have one code base with both of our changes. I'm actually going to go ahead and use Brian's account and accept all your pull requests so that he can have this super awesome project that you guys are all a part of. So let me open up his account. Let's see if you're on here. Okay. So Brian's Github has all these messages on it. (indistinct talking) Yeah. And all the people are like, "Knock, knock Brian. "Your code was a nice starting ground "but I've made something better with it. "I've changed it. " So if Brian goes to his project his view of the pull requests looks a little bit different. I had one pull request and that was the pull request that I was making. He's got 20 of them. And just a quick note. The emoji work in the comments but not in the tail. You guys are artistic. I like the ghost. It kind of looks like a pirate. It's got one big guy and one little guy. We've got some cats. Oh, someone used my commit message. Someone is telling me that I messed up. So when you put in your pull request the person whose code you branched off of will see all this text. And really instead of the ghost and stuff you should be putting in a reason why Brian should, if this project was real, pull your code back in. Aww! You got to know you did one. Let's see who got some cats and that guy is, it's me. All right. So as Brian, I'm looking at this pull request I see that there are some commits here, there's Mark's text file then he changed it. You can go to these tabs at the

top. I can see the commits and I could see the files that he changed. Awesome. Hack is super fun. So I'm like, all right. This pull request is awesome. I'm going to go and merge it in. So merge this pull request. So now after this merged now my project has Mark. txt in it. So my project and Mark's project both share that file. Two projects were merged together. I'm going to accept some more pull requests. I'm going to go with this ghost one, because ghosts are awesome. Let me show you guys another feature. If I go to these tabs at the top and look at the files changed summary what I can do is actually comment on individual lines of code. There's this little comment icon. It's a little bit off the screen. But if I click on it, I'm going to write "Yeah, it sucks! " Now let's see. Thumbs down. And we'll leave that comment-- You can't put that as me (laughs). You know what? I'm on your watch now (laughs). Wait, wait, wait. I can't or I did? Because I think I did. Sorry, John. Anyhow, when John goes back to his pull request he'll see this comment that Brian made. That Brian made (laughs). All right. So just dropping some knowledge on you guys. This is a lot. It's a new tool, a new website. Does anyone have questions? Is any of this confusing? Now we refresh the page. Do I? Yeah, I think so. No laughing. This is serious business (laughs). Okay. I'm serious though. Does anyone have any questions? Because for the final project we're going to be cloning a real repository of Brian's. So if there are any steps that you missed please let me know. Okay. Let's do one more. Okay. I've been Rick Rolled. Awesome. (mumbles) rocks. So I think this pull request is great. I'm going to merge it in. So now my pull request project has one more text file in it. So if you guys were starting from scratch and you forked this project right now you would get the Erin. txt file on the Mark. txt file along with that original README. It would be a snapshot of what the project looks right now. Okay. We talked about all these stuff. And a lot of this is in the slides. I thought it would be a lot more interesting to do a live demo. But if you need a copy of the commands here they are. We talked about Git Status, we talked about our unstaged changes, we did a Git Add and added that file and it's being helpful in telling us what changes are going to be committed the next time I run commit. We do our commit and we push it back up to origin, which lives on Github. We made a pull request.

Initializing a New Repository

Now we're going to talk about how to make our own git projects. The easiest way of doing this is going back to your Github account. I'm clicking up on this little plus up here. And we say we want a new repository. So we give it a name Github suggest one. So it's like, "How about you call this hairy ninja. " Thanks Github. No spaces or funny characters in here. And we talked about naming conventions, so in HTML you name your classes with dashes. And that same naming convention is usually used for git branches. You can call them whatever you want but this is the most

recognizable way. So I'm going to call this one My First Project. Description is optional. This is important. Public or private. If your code is public, anyone can see it. If you want your code to be private, you have to pay for it. It gets about five bucks a month. But here is the important part. You can have git initialize this repository for you. A. gitignore is a file that tells git which files to ignore. Let's say you have a text file and you have a JavaScript file and the text file has your notes in it and you don't want it cluttering up the repository. You'd create a. gitignore file and say, "Ignore all txt files. " Now git has created a bunch of. gitignores for you already. Here's a huge list of them. For example, Java generates a bunch of compiled files. You usually don't want them in your directory, so you'll pick this. gitignore in Github and just put it there for you. (mumbles) use another one. Yeah. License is something you don't really need to worry about right now but git provides a whole bunch of licenses for you that it will also add back in. So if you already have a git repository, locally, you don't want to do this step. We're going to skip it. But I find it really helpful. If you check this, git will just tell you, "Okay, now Git Clone, " and give you the address of your new project. But we're going to leave this unchecked. And when we create repository Github will tell you, "This is how you either create a new repository "on the command line that will be linked "with this Github repository. " Or if you already have one, which you probably don't,, here is how to sync it up with your Github. So we're just going to go ahead and follow these steps. So we're going to do this Touch README. md and if you Windows people are using Git Bash this should work. So I'm going to make a new directory. I'm just going to do that in the finder. So I'm going to the desktop. I'm calling it Awesome because it's awesome. So now I can change to my directory Awesome. We see it's totally empty. And I'm just going to copy these commands from Github. I'm creating a new file. Now if I do an ls it's in that directory. I'm running git init which will create a hidden. gitdirectory and tell you that, "All right, here you go. "You have an empty git repository. " When we do ls we won't see it. We won't see this. git folder because it's hidden. Okay, we're going to add a README. md and then we're going to do my first commit. Here we see the changes to be committed or this README file. We're going to copy that in. Git is going to tell us that our commit has been done. And this step is the important one. It's kind of confusing, but this step is telling your local repository, "Here's where the server lives, where you're going "to push all your stuff back up. " And the server is hosted by Github. So unless there's an error you won't get any output from that command. And then the last step is git push origin master. And the -u is because we have not pushed from this repository before. So now if we go back to our Github we'll see that file that we pushed from just a few minutes ago. So all this stuff is definitely a lot to chew on. There's a really great free git book out there and it's at git-scm. com. Lots of really good documentation here. (indistinct talking) Yeah. So try git. This is really good. It's interactive. It won't go quite as in-depth as we just went but definitely a good resource. So we're

just quickly going to brush over branches. So like I mentioned before, feature branches are when you want to work on something and not disturb this pristine working copy that you have. They're really not that interesting when it's just you working on a project, but they become really important when multiple people are working on a project and committing to the same repository, the same project. So at work I have my branch and Brian has his branch and we'll go through the pull request process, so that we're not interfering with each other. I don't know what his files are and he doesn't know what my files are and it really doesn't matter. So if we go and make a new branch, and the terminology here is a little bit unintuitive, the command is git checkout and then -b. b is saying, "Git, make a new branch. " And the name is 'my_branch'. Git will just create a new one, switch it right to it. And then if you type git branch you'll see a listing of all your local branches and git will show us our next, to the one you're currently on.

Node.js

Node.js Introduction

One of the really, really fun parts for me, talking about node which is kind of a new and hot thing. So we're going to be running JavaScript which is the language that we already learned today and yesterday but this is not going to be JavaScript run in the browser, this is going to be JavaScript run on the server. So it's the same syntax, it's just there's some slight, not slight, but large differences to it. So essentially what node is is they took the JavaScript engine, which is in Chrome and they just ripped it out of the browser and they stuck it on the server. So what you can expect from Google Chrome's JavaScript engine you can expect from node. It's really cool, it's really fun to program, and if you're using it correctly, it can be extremely fast. So who uses node? I mean as you can see, not a bunch of slackers on this list, this is actually a pretty big group of people doing huge amounts of scale and using node in really cool ways. I was reading an interview with one of the guys at Walmart and they said that their Black Friday last year was boring because node made it so easy to deal with the traffic which is crazy, I can tell you, because our site went down on Black Friday, theirs did not. Let's talk a little about what is it. So it's, again, JavaScript on the server and this is really cool because typically when you've been learning web development you had to learn one language for the browser and then you had to go and learn another language to code on the server. So you had to either learn JavaScript in PHP or JavaScript in Ruby or JavaScript in Python. You had to kind of have these mental context

switches when now you just have to remember like, okay I'm on node so I don't have to use the Window, or I'm on the browser so I have to use the Window. One thing I will give you as a caveat, if you're coming from another language like Ruby, like Python, node is quite different. It's cool but it is different. Let's go and do some Hello World. So there's no CodePen or at least no CodePen that I was satisfied for for node, so we're going to actually have to clone repository which good thing Nina did her thing so now we can go out and clone what we need to do. So pull back up your terminal. I don't think Nina went over this. So say I have a whole bunch of stuff on my screen and I want to clear it out, if you just type clear it's gone. Let's just go to Desktop. So if you're on a Mac that'll work. So here we're on our Desktop, still have some of Nina's folders on here. We're going to go out and clone one of the repositories that I've already written for you so let's go back to GitHub real quick. GitHub, btholt, and it's going to be, you can just go to my user profile. It's not in here so if you go in Repositories, intro-to-webdev-app, that's the one we're looking for. GitHub dot com slash btholt slash into-to-webdev-app. So we're going to come down here to this SSH or click the Copy right here, we're going to go back to our terminal and we're going to say git clone. Now typically, you know how to fork it? In reality, I dont think anyway, you're not going to be pushing code back to my Repository so you can just clone it straight from my Repository and that's fine. The only reason that you would fork is like I want to make changes to this and then I want to commit my changes back to Brian. We're just going to do git clone, we're going to paste that, and we're just going to let it clone onto our computer. So now if you look intro-to-webdev-app is there and we're going to go into that. Anyone have any issues with that? Seems pretty straightforward. So now we notice ReadMe we have an app that's going to be the last part of this workshop but right now we're just going to go into the node exercises. So just in case you wanted to see what this actually looks like in the Finder, this is going to be a Mac only command and I'm sorry, I don't know what it is in Linux or in Windows but you can say open in Mac, and then I want to say space period, which just means this directory, so I'm going to say open period. And now you can actually see what it looks like in the Finder which is pretty cool. It's also on my desktop, so I could've just double clicked on that too, that would've been fine. So we're going to go into node-exercises and we're going to go into basic, that's the first one. Something else I'm not sure Nina talked about but I'll readdress if she did. Notice right there I have basic, I have express, posting, params, cheer and jeer, whatever, right I can type ba and if I hit Tab, the terminal just knows you typed ba, I'm assuming you want to go into something real so it just finishes it for you. It's really fast, right, because we're all lazy, we don't want to type all the words so it's like I will only type two letters. So now we have this app dot js so now we're actually going to start using Sublime which is going to be our text editor, it's right here. So I guess you need to open this in Sublime somehow so you can say Sublime, Project, Add Folder to Project, click that and then

you need to go find wherever it is on your computer that you put this. Mine happens to be on my desktop which is right here and then click on the intro-to-webdev-app and click Open. Did we follow that? Can you say the path again within the Project? So you need to go find wherever you cloned your intro-to-webdev-app and you just need to open that folder. The entire folder, nothing inside of it so click on this and then I would click Open right now. I'm not going to do it because mine actually is already open to my project. You have to create a new project first? You don't have to explicitly do it, if you don't already have a project, Sublime just creates one on the fly. Is that working for people? Did you just do open? So something else you can do, which is along those lines, if I have this open right here I can just take this whole folder and drag it onto Sublime, which I have right here. That'll work too. And if I just do that, it'll open a brand new window with that project opened. Feeling good about this? This is going to really suck if you don't have this open so I want to make sure everyone has it open. I don't have it open. Okay You want to go help her, Nina? Everyone else good? It seems like for the most part people have it open so let's go ahead, let's go ahead and close that out. So we have this nice project open now and we're going to go into node exercises. We have basic right here, that's going to be our first little exercise we're going to work on so open basic and look at app dot js. You should see something akin to this, might be different colors but it should look like that. So again, intro-to-webdev-app, node exercise basic app js. Here we have the most basic of the most basic node app the Hello World. The only thing this is ever going to do is say hello world. Before we get too much into what the actual code is doing let's go ahead and run it and see what happens. Again, I'm here now in the basic web directory or the basic directory. I'm assuming node has already been installed because that was one of the pre-requisites for the class so you're going to type node, and then, or sorry, app dot js. So you're telling node run app dot js. And you should say Server running at that address. Is that happening for most people? Alright, lots of head shaking, I like that. So just grab this http right here, copy that, come back into your web browser, paste that right there, should see Hello World on your browser. Minds being blown right now, you just wrote your first web app which is pretty cool. This is like the basis for everything that we're going to do. So node is actually launching a web sever locally? So you now have a web server running on your computer. By the way, if this number looks annoying, it is annoying, you can type localhost and that works too. That's just letting your computer know please reference yourself. Someone's super excited about node. Then we have this colon 8080 at the end, that just letting the browser know which port to look on. If you don't know necessarily what ports are, it has to do with routing web traffic and stuff like that not a huge deal but it's necessary, it must be there so please include it. So let's dissect what actually happened here. So I'm going to put some space in here so we can just see what's going on. Put that on a new line. So I think this console dot log looks really familiar to you.

Instead of console dot log writing to your browser's console, it's actually going to be printing straight here so we see Server running at this and here we say server running here, that's how that's actually happening. Let's talk about require. This is kind of an interesting one, it's unique to node as in you don't actually do this in the browser, not natively anyway, but we'll just say for right now that you don't do it in the browser. So you're saying to node, node I know you have a library called http, please give it to me. That's what that means. There's quite a few of them, http is going to be the one that we're going to be using today but there's one like FS for File System like if you needed to read files from your system, node includes a bunch of useful things for you to use. So we're saying go out, fetch this http library, give it to me, and then we're just going to use that to create our own little web server. So we're going to say http, please go ahead and create us a web server. And then that takes just a quick little argument and if we give the server a function, it's going to return that on literally every request you give to it. So going back to our little browser, I hear you can put anything after this and it's still going to say Hello World. Any address, doesn't matter. Either that or I coded that really long web address in there secretly but I didn't. It's fun for toy purposes but it's actually not terribly useful for writing a real web page. I do a writeHead. So you remember Nina talking about status codes, 200 is successful. If I did 404, what would that mean? Can't find the page, right? But you want everything to be found so we're giving 200 out and then we're writing this Content-Type. You have to let the browser know what you're giving it because the browser has a bunch of different modes it goes into to render different things, in this case it's just like we're just sending you actual text. Typically we're going to be sending things like HTML or CSS or in our little app we're about to do JSON. So here we're just saying your Content-Type is text slash plain and then there's one called end and that's essentially saying I'm going to end my request, here's what I'm going to give back to the user. So res stands for response, req stands for request, we'll actually get into request later but right now we're just worried about res or response. So we're saying please end this response and tell them Hello World, finished. Once you call end, you can't actually send anymore because it actually wraps up all of your data and sends it back so just keep that in mind, once you call end, it really is the end. We have this listen. We're chaining again, if you remember that. Dot listen on port 8080 which is where that business comes from, This really can be an arbitrary number, just make it over 1, 000, it'll make your life easier. It can be lower but just do that. And this is essentially saying you're going to be on local host which is that number. 127 dot 0 dot 0 dot 1 just refers to yourself if you're a computer, so it refers to this computer right here I'm going to running my server on here. It's a magic number, wouldn't worry too much about what it means or where it comes from, in fact I'm not entirely sure, so that's why I'm not going to tell you about it. And that is the anatomy of the app that we just wrote.

Node Package Manager

Let's talk about npm. Npm stands for node package manager So remember when we used jQuery, we're pulling in code that someone else wrote to make our life a little bit easier. Typically the way we do that, we either have to go out to jQuery's website, download their code, put it on our sever and serve it from a server. We have to either go out to their servers and pull down their code from their server and put it on our webpage and that's all fine, it works. Node kind of took this a step further and they introduced a package manager, I mean they didn't introduce that, they're just leveraging that which is already existent. Now if we want to go out and grab someone else's code, it's really as easy as saying npm install blah and then all of a sudden this code is now on your server and ready to be used. Super cool, right, it's just super convenient. And great news, if you got through that last example you already have it because you have to have npm if you have node. So we're just going to give you a really basic tutorial, well just introductory to it. First of all, to introduce we're going to install nodemon or nodemon, I've heard it called all sorts of weird stuff. I'm going to introduce to you why this is useful first. So my server is still running right now and I want to say Hello Minnesota instead of Hello World, so I change this, I save it and come back to my app and I'm going to refresh it. Notice it's actually still saying Hello World even though I changed it and I saved it and the reason being is your server actually has to restart itself because your server has to reset itself back up before it'll actually recognize those changes. So in order to stop the server, we're going to hit Control and we're going to hit C, the letter C. That should stop your server. That's just your keyboard sending to the server please shut yourself down, I'm done with you. Now I'm going to run it again, I'm going to say node app dot js and if I go back and refresh, what do you imagine you're going to see? Hello World, apparently. Am I changing the right one? Did you save it? Thought I saved it. Is anyone else seeing Hello Minnesota? Yes. That is just pure bizarre. Oh, you know why? I'm changing the wrong one, that's why. I have two of them open. Which one am I changing, that is a good question. App, basic app js Okay, that's right. And where am I? I'm on Desktop. Okay so I was in the wrong one. Okay and I'm going to say node app dot js that should now say Hello Minnesota. Tadah! So that's supper annoying, right, to have to go Control C and then restart your server every single time. Another little trick, if you notice if I hit up on my keyboard, it's going to go back to the last command that was in there so I'll be doing that a bunch too. Anyway, point in case, it's super annoying to have to restart your server. Every time I save a file, I just want my server to restart so I'm going to say npm install dash g which stands for globally, and I'll explain that in a sec, node M-O-N, nodemon. Go ahead and hit Enter, mine's going to error out because I already have it. Maybe not. Maybe it's going to update it, which is probably a good thing. You're going to see npm vomit all over your

screen. What were you typing in? I will type it here again, just a sec. Is that working for most people? Is anyone getting like a permission denied? I'm getting mkdir errors which might be permission. Yeah, okay, so if you're getting that, you're going to need to type S-U-D-O and then you're going to type np, and everything that I say from here on out, you're going to have to type sudo in front of. So you're going to say sudo install dash g nodemon, like that. It's going to ask you for your password, it's the same one that you log in with. Is that working for you now? So it was quiet for a while like thinking. Yeah, it was probably. Plus then you don't want to request all at once it's one package from them. Her server errored. For other people, is this working? Okay. In fact, you don't actually have to put sudo in front this is the only one you'll have to put sudo in front of. Anyway, what sudo means for those that are not familiar with that, it actually stands for switch user and then do which don't actually worry about that. Your computer has lots of guards to make sure that other people aren't doing weird things to your computer like people aren't coming in and stealing all your files and changing your files and so they have this idea of permissions that certain users only have permissions to do certain things. Now as probably the sole proprietor of your computer, you have most of those permissions but your computer really tries to guard even you against like blowing up your computer. And so they make you put this sudo in front of things just like making you're about to do something very powerful and could totally screw yourself up, you type sudo, this is now on you and not on me because Linux doesn't like to be blamed or Unix for that matter. Question? So after taking, oh, never mind. Make sure you get something at the bottom? So if you get something that looks like this with like a little tree or something like that at the bottom, then everything's a-okay. Now let's go ahead and test it out, you should be able to type nodemon and type app dot js and it should give you some sort of output looking like that. Now I should be able to come back here and say Hello Utah, save that. Notice that it restarted due to changes, starting node app dot js, show the refresh, I didn't have to stop my server and restart it, it just restarted itself. What was that command again? Node. Nodemon app dot js. This one? Yeah. Let me clear it and then I'll put it up there. Is that working for most people? Alright, cool. There's lot of utilities like that for node that happens to be a super easy one. Do you think you might be able to show the process on Windows on the chat room? I'm having problems. I can give it a shot. I don't actually even have node installed on my Windows box yet. Anyone here running Windows in Git? Was the process much different? No. Okay. So it's just npm intall dash g and that worked. So really the steps that I just went through should work for Windows but let's just node js. I think this is actually a huge file. Oh no, it's not too bad. Well, I'll just explain it again. So I'm going to say npm install, assuming you have node installed, if you don't have node installed then this isn't going to work, dash g nodemon and if you're on node, that should just work. I don't think there's even the concept of sudo on Git Bash or on the CMD.

Just do that, it should run, it should work. Beyond that, I've actually never worked with node on Windows so it's kind of outside my expertise.

Express

We've now seen how to install a global tool for node. That's what that dash g means, it means I want to be able to use this from anywhere in my computer. That's the last one we're going to install that way, everything else is going to be installed locally. So it's only going to be available in very small parts of your computer. In this particular case it's going to be available in our little apps. We kind of went through that, that dash means it's global. We just built our first and last app using just vanilla node. Why? It's actually really hard to write just plain node as in like it's kind of beyond me even or at least I haven't ever tried. So like jQuery there's something called Express that exists for node. Express is certainly not the only framework that you can use but it's certainly the most common one. And it's just like a design to make writing a node server like a billion times easier. Like I said a billion, maybe a trillion. Okay, so go back to your Sublime. I don't know if you've noticed but I am certainly not prone to hyperbole. So we're going to be in express. Come back to your little command line right here. We're going to get out of here so I'm going to hit cd dot dot. We're back in our node-exercises and we can say okay, and now we need to go to express. Here, we're going to type npm install, very key here do not put dash g, I'm going to repeat that, do not put dash g, express. Just like that, npm install express. Again, much is going to happen. If you get permission errors again, which hopefully at this time you will not, then go ahead and throw sudo before that again but that should look something like that. Look good for people? Can you show that command line again? Yeah, so it's just going to be npm install express. If you're lazy like me, you can actually shorten this down to npm i and that works too. Why wouldn't you install that globally? There's an idea, a philosophy in node, that everything should be totally and completely self contained. So if I have three apps, all of them using express, they're not necessarily going to use the same version of express because express is actually updated pretty regularly and so I don't want to have to update all of my apps just because I won't update one of them so they each can carry their own dependencies. So that means in every project directory you have to expressly install Express? Yup, which is again, it's something you do once and when you start a project and then you work on a project for six months you don't care, right? I'm a huge proponent of this so I think it's a great thing. Okay, cool, we got express going now? Can we use a different folder or is it the same folder? Yeah, we're in the express folder now. One question is when do you use the save? We'll talk about it coming up. We were in basic, we have now moved onto express so I got into the express directory. You should now see this little

node_modules folder. We didn't create that but guess what, surprise, it's there. That's where npm keeps all the code that it brings in. In fact, if you look in node_modules, you'll see that there's a folder called express, if you look in express, you'll see that it has a lot of code. Don't look at it, you don't have to but it's there. Feeling good? Let's go back to our code here. We now are in express, we're looking at the app dot js that's inside of express. So you notice the first thing, rather than requiring http, we're requiring express. So again, to reiterate, excuse me, reiterate how node is going to work with this you're going to say give me express. It's first going to go, like node, personally do I own anything called express, I don't, so I'm going to go look at his externally installed npm type stuff and see if it's out there. It's going to find express in there and it's going to bring that into our project here in this express variable. So now that we've installed npm through npm express, it's now available to us because we installed it. So now we have express right here and here we're going to create an instance of express using this express function. If it looks kind of weird, it is kind of an interesting pattern, but anyway, the app here is now how we're going to operate, like what server. You remember how we were talking about http verbs like GET and POST? Where I say every time that some calls a GET, which is what this get right here means on hello dot txt, this req and res should look really familiar to you, please send them Hello World. This is essentially the same app that we wrote before but now on express. Questions about this? So again, well not again, for the first time, rather, server is an instance of the app that we've created like we've essentially setup the entire app right here and then we get this server running here and say app dot listen and that's when actually the server starts going. If we took this part out right here, we'd essentially configure and server and then they would never run. So this part you're essentially saying I've configured everything I want to now please actually start doing your thing. Cool? Or should we actually see what it does? Nodemon app dot js. Okay, it started running. And then we're going to go check out what it does in the browser. So first of all, we have to go to Hello dot txt because that what we set it up as. Now we see a Hello World here. What would be interesting, try doing it like something else or even just like an empty, like if we do this it's going to say CannotGET. Or if you go to this it's going say CannotGet that. Let's talk about why for just a second. That pattern of giving, essentially throwing up any time someone asks for anything it's like hey, I want this and it's like no, you get this and every request is given that. We actually want to route specific things to specific places like you want your index to be different than your about page than you want about your contacts page. You want to have different routes on your web server and that's what this makes really easy. So we say any time they go to my website slash Hello dot txt, serve them Hello World. We follow, that makes sense? You can change it to be something else, you can be like thing dot txt and then if you saved it, restart, come back over here, thing dot txt, now it's being served there. Makes sense? Just like another way of doing the

same thing. Is the structure then that you have a lot of GET functions for all of the different pages and very last at the end you have the server? Yeah, that's really good. In fact, that's the structure of the whole thing throughout at least our class. We'll actually even get into POSTing which'll be different than a GET.

Exercise 10: Cheer and Jeer

We're going to have an exercise with cheer and jeer. Make an app that has two routes which is essentially we're going to make an app that has two GETs, we're going to have one that's called cheer dot txt and jeer dot txt. Cheer should send something back positive to say and jeer should send something back quite negative. The solution is in cheer and jeer dot and remember you're going to have to reinstall express in this new folder. Let's just get onto it. The whole solution is here in cheer and jeer but we're actually just going to go ahead and create a new folder and call it exercise or whatever you want to call it. Here we're going to have a new file, we're going to save it and it's going to be called app dot js. And then we have to go back here to our get out of here and we're going to go into our exercise folder and we're going to do npm install express just like we did before. There's a question on why you were saving the app server variables. Well you definitely have to save the app variable because using the app variable is how you create the server variable. The server variable, force of habit, I don't know if I necessarily have a good reason for it. That's just kind of common practice, I'm sure it's useful but I don't know why it's useful. So what I did here is I just did npm install express that's all I did. Let's go back to our code here, you'll notice that in our exercise folder we now have the node modules. So here we're going to save our express equals require express and then var app equals express. Pretty straightforward so far. I want to say app dot get and we're going to say slash cheer dot txt, function req, res. If you want to be verbose, request and response we'll do that this time just for fun. And then you're just going to say request or sorry, response dot end You have quite lovely eyes. Can imagine jeer will be quite similar. Response dot end I don't even know, everything that comes to mind is offensive. Your mother was a hamster. Your mother was indeed a hamster. Thank you. After this we're going to save our server. Equals app dot, I can't even remember the syntax off the top of my head so let's just go ahead grab it from express. Yeah, in fact, you are free to just go ahead and copy and paste that because literally, not going to change. Very common practice by the way, it's like I know I've done this before. Just go out, see what you've done and go grab it. Feeling pretty good about that? Apparently not. Any questions about what's going on here? Awfully pretty straightforward. Can you walk through it again? Sure. Bringing in express right here and then creating an instance of express and then we're doing an app dot get which is essentially defining

a new route, so if you go to cheer dot txt or if you go to jeer dot txt they're going to get different responses then we're giving a call back function. Call back function or in this case an anonymous function. And this function is essentially I want you to do this when they reach this route so if they go to cheer dot txt please run this function. Here we're doing response. Response dot end which means please send them back this response if they reach this route and we just did that twice so we define two routes. And then here we just instantiate the server. Make sense? That makes sense. Cool, well let's see what it does. If you want to keep following along, the whole answer is in the cheer and jeer folder. That shirt doesn't look awful on you. I'm a jerk. I'm pretty sure your scent is unpleasant. I don't remember writing that. But something worthwhile of node, right here I'm just saying app dot listen, you actually don't have to have this function right here it's just kind useful. You can see in the command line your server actually did start. But these are essentially equivalent. So we're going to say nodemon app dot js it's listening in port 8080. Okay, and we're going to go to cheer dot txt. You have quite lovely eyes, your browser's admiring you. And jeer, your mother was indeed a hamster. Questions about that? Good stuff, right? So let's just remove that, Don't Save. So this is the one we just wrote. Again, this is like writing server side JavaScript is not easy.

Static Assets

Static Assets. Static assets are interesting because they're essentially things that you're going to give to the user that the server is not going to change. So this is going to be like HTML, CSS, JavaScript, images They're just files that you're going to store on your server and then send to the users. You're not going to look at them, you're not going to modify them, the user's going to say hey, I need an image please give it to me. So we kind of introduce this idea of a static asset or something that's kind like a public directory. Because otherwise, like you know how we're saying app dot get do this, app dot get do this, we could say app dot get smiley face dot png serve this smiley face picture to the user. It's a huge pain, especially if you have a large site with many, many, many images. So what we do is we take a folder and we say this folder is now public so anything that's in this folder can be accessed by anyone. Express is really good about that, you just say hey, express, serve anything in this folder to anyone that asks for it and it just makes new routes for all of your static assets based on what their file name is. Like you know how we said the content type, you have to say those plain text, express is smart enough to figure out what kind of static asset it's serving. So again, it would be extremely painful if this idea of a static asset or a public folder didn't exist. This is one of the things that express adds, it's not part of the normal? So this is an express feature. So I have a folder called, let's clear this out, cd dot dot and we're going to go

to static-assets and we're going to do npm install express. This should feel like a familiar pattern. You should be getting sick of it now and I'm about to show you how you don't have to do that every single time. But hear in a second, all things in good time. So we're in static assets and we're going to look at app dot js in here. Even more simple, here's an express server and then I'm going to use app dot use and then express dot static. So this line may look a little cryptic, it is but express dot static is saying like I'm going to give you this entire directory, please serve it just like we were talking about. Underscore, underscore dirname, that's essentially saying here is the directory. It's essentially the base directory of where your app is it gives essentially an absolute path to find these images. So in this particular case, if we go here, pwd the underscore, underscore dirname is a node variable it's going to be this. Dirname is going to be equivalent to this super long string. because node is aware of where it is, they're just telling you please use your awareness to figure out where this folder is. And then you're going to say on addition to that go into the public directory because you don't want your server side code being public, right, you just want your static assets to be public. And then after that we're just saying app dot listen. So let's give this a shot. We're going to say nodemon app dot js and now all these things in the public directory I just wrote like a little library right here, some CSS, some JavaScript, a little HTML file. You don't have to worry too much about it, you can look at it if you want to. So let's go back here and we're going to go to index dot html. Hopefully you're seeing a large pink button that says press me. And low and behold, of course what did I do, it serves cat pictures. And you can just keep pressing cat pictures and it will keep serving cat pictures. Kind of a one trick pony up here, sorry guys. Let's look at our console real quick and there's this little tab right here that says Network let's go ahead and refresh it, our page and it's going to record everything that comes into the page. it's getting our style dot css, it's getting the html, it's getting the kitties dot js it's doing all of this without us explicitly telling express serve this JavaScript, express serve this CSS, express serve this HTML, it would just drive everyone crazy, it would be impossible. So instead I just put all those files in the public directory and say serve this entire directory. I don't care what's in there. If they know what the name is, they can have it. Make sense? Awesome. I should really trademark this app, it's awesome.

Receiving Parameters

Receiving Parameters. Let's take, for example, kind of like ESPN. com for example. They have a page for each individual NBA basketball team I don't want to have to actually explicitly write a brand new page for every team in the NBA. What I actually want to do is I want to write one page and then have it adapt to whatever teams goes in it. However, that means your page has to be

aware of how it's being reached. So for example, if I want to go to team slash jazz or team slash timberwolves, I want them to go to the same page but with different parameters. This is very possible and very easy with express. So let's go ahead and take a look at that. So we're going to get out of here we're going to go to params npm install express I'll let you do that, mine's already there so I'm not going to keep wasting bandwidth. We're going to go to params app dot js. Close that one. This top by now should look super familiar and we're going to do app dot get and then we have colon name. So this is new, right, we haven't seen this before. That colon means this is going to be some sort of parameter some sort of user generated, or not user generated but dynamic parameter that I want to be able to pass back to my sever so it knows exactly what to serve me. So in this particular case, we're going to set the header it's going to be text dot plain again and we're going to send back You picked name. So it doesn't even really care what the parameter is it's just going to give it back to you. That's how you give it back. This is the first time we're going to use req which stands for request if you don't remember and we're going to look at the params and we're going to get name. There's a question I think when you're moving to examples they were wondering if there's a way they didn't have to keep doing install, but I think you touched that a little bit with installing to each different project. So typically you're not going to move between so many projects so quickly, we're doing a project every three minutes, that's totally and completely atypical. Typically you're going to start on a project, you're going to work on it for X amount of time but you only have to do npm install express once. We're going to talk about another way to manage dependencies, that's a little bit easier, but that's in just one second. How's this code looking for you, does this kind of make sense? So let's take a look at what it actually does. We're going to do nodemon app dot js and now we're going to go back to our app and we're going to slash team slash jazz. You picked jazz. You picked timberwolves, you picked anything, it doesn't actually really care But that just kind of a contrived example of how to get user or client input on the server.

Package.json

Alright, so now we're going to talk about package dot JSON. Hopefully you are all super annoyed by typing npm install express at this point because it is in fact annoying. So let's go ahead and fix that a little bit. So go back to your params directory which is what we're in right now, ls and I want you to go ahead and type this. It's going to be rm, which stands for remove, and then you're going to put rf. Be careful with this command because it's going to recursively remove a bunch of stuff or basically remove everything that you tell it to. If you say remove all of my files, guess what it's going to do, it's going try to remove all of your files, so caution. And then you type

node_modules and essentially what this going to do is remove everything, it's going to remove everything including the folder for node_modules. So now if you look again, all you have is app dot js. So now what I want you all type is npm I-N-I-T, init. If you remember, Nina did GET init, so this is going to be npm init. It's going to ask you a bunch of questions so right here where it says in the parentheses param it's going to try to take your best guest as what you want this answer to be. I want to it to be params so I'm just going to hit enter. I want it to be version zero. I'm going to give a description of this is so cool or this is a cool app. This is going to be a description of your entire app. Entry point app dot js, that's correct. Test command, typically that's going to be npm test don't worry about that, we're not actually going use it. Git repository, we don't have one so just go ahead and Enter. Keywords, we don't have any. Author, me. And license whatever you want to put there, all that's fine. So now we have this npm file that essentially saying this is what our project looks like at a high level. And you're going to say is this ok, yes. So now we have a package dot json If you want to go look at it here, it now exists you can see what's actually in here. Notice that this is just json, this shoud look familiar. So now we're going to say or let's actually just do it npm install dash dash save express. Now it's going to install express again. But notice if you go back over here, it tells it this file depends on express. So now when I'm sharing my project with Nina, Nina can just run npm install without having to know exactly what everything is and it'll actually install all of your dependencies for you. So from here on out I've actually provided these package dot json files for you and so you can just type npm install. Super cool, super easy. So we'll kind of get to that a little bit later. So if you actually want to see this in action, let's go ahead and remove the node_modules again. So if you notice, I just have an app js and a package dot json. I'm going to say npm install and I'm not going to put express just npm install, I'm going to hit Enter and it knows that I want express. Now this doesn't seem probably that mind blowing with just one dependency but when you have 30 dependencies it's pretty mind blowing. You don't have to type npm install for each one of them.

POSTing

So now we're going to talk about POSTing. Often we want to send much more than just a single parameter to the server. Imagine when you're trying to do a form sign up. You want it to accept their username, their password, their email, their address, their phone number. You want to take like an entire form of data and you're going to do this by POSTing. And right now we're going to use jQuery, been using Ajax something that we've all done before. and we're going to POST to the server. So let's go look at node-exercises slash posting. So notice I already gave you package dot json so let's get out of here and we're going to go into posting. Everyone just run npm install.

Mine already installed so it's fine but yours should give a whole bunch of installing going on and you should get both express and there's another one in there called body bodyPaser which I'll show you here in just a sec what that exactly does. So we're going to look at app dot js require express and then we're going to require bodyParser and then we're going to have the app use the bodyParser for json parsing. Now why are we doing this? It's because express itself does not actually understand JSON and the reason for that is express is flexible enough that if you want to have it speak X amount or you want to have it speak another language, it's flexible enough to be able to do that so you just say here is the parser that I want you to use when you're trying to understand what people are trying to tell you. You're going to do app dot use the bodyParser. Here we're doing the static sharing again so that should look familiar and then here, notice we're seeing app dot post instead of app dot get. Key difference there. And then it's going to expect, from the user it's going to expect both a username and a password which come in this body object. And then here we're actually going to do some switching based on if the name is brian and the password is pass which is probably my best password, don't steal my password, then send a status of 200 which means yeah you logged in and you're going to say status success. If they have the incorrect password or anything like that, you're going to send them a 401 which mean you can't log in, that means unauthorized and status failure. You also might have noticed that we're not using end right now, we're using json. As you might guess, this is going to send json back instead of just plain text. And then here we're just saying app dot listen. Here in the public directory, just so you kind of an idea I just have a little button right here that's going to do Ajax, something that we covered earlier and it's just going to send up both the username and the password. And then on success, it's going to log out the data that it sends back. Questions about the code before we move on actually show you and then we can come back to talk more about it. Okay, so I'm going to say nodemon app dot js, started the app. Let's go back here. We're going to go to index dot html which is being served by the static assets so I now have a username and password. Don't worry if you don't see these asterisks, it's my last pass, it's an extension I have installed don't worry about it. So I'm going to send up, first of all, like Nina and wrong or something like that, like some wrong password and it's going to tell me you are unauthorized to reach this API, but if I send up Brian and pass, and I hit Sumbit, I get success. This is obviously not the best way to write your login but kind of a precursor example for how to write some sort of authorization system. Questions about that? Makes sense? Okay, cool. Oh, and also notice right here in app. js I'm logging out whatever username and password so you can see right here it knows the name, it knows the password, etc, etc.

Final Project - Client Side

We have now introduced all the new concepts you would ever need to ever develop any app ever. That's actually not true but we introduced every concept that we're going to introduce in this workshop. So from here on out, this is just going to be a giant review and we're just going to code up essentially a very basic version of Twitter. It's not going to be a concept of users or followers or retweets or anything like that, it's going to be more like Secret, if you've ever heard of the Secret app that people are just going to post little anonymous tweets out there. So we have an entire app just coded for you that you can look at here and the entire thing is already finished and ready to go if you want to look at it. A quick question about request dot body, can you explain where that comes from? Let's go ahead and take a look at that. Request dot body. We refer to it here in posting. So request dot body username request dot body password. So if you look at the posting dot js right here, I'm sending up a username and password kind of like Payload with jQuery. Now the server side needs the ability to access that it does it through the body so if I called this user, I would have to go back and change this to be user. So it's just kind of this contract of how the client and the server are going to talk to each other. Does that make sense? So let's talk about our project real quick, I'm going to go over the client side development of it and then I'm going to hand it off to Nina to finish off the node portion. If you haven't already cloned it, this is the address for that. So here essentially the requirements like imagine the product team coming to you like you're going to build this app they say here we need this and then it's up to you to solve that problem. So we're going to write a server in node. js, the server will serve the HTML, CSS and JS necessary to run our server. It's going to accept a GET of the latest servers meaning like it's going to accept an Ajax request to pull down all the latest tweets. It's going to accept POSTs that we're going to put out there so that we can post new tweets and then for each new tweet, the server it's going to attach the time of when it was posted. So you can see that this was posted yesterday at 5 o'clock or this was posted 10 minutes ago. So those are our basic requirements, then we're going to create a webber. Webber? I'm getting tired. We're going to create a web app that's going to make an Ajax request to get new tweets. It's going to have the ability to accept user inputs via POST so that people can tweet new things and will display in a pretty way those tweets when the server loads as well as after you post a new tweet it's going to add that tweet onto that page. Things that we are not going to do. There's going to be no database to this. So if you restart your server, your tweets magically disappear. So obviously this is not really a complete web app, you need some sort of persistence. And we're not going to have any notion of users, followers, retweets or anything like that, this is just purely anonymous tweets. So I'm going to start here with the web app. Essentially what we're going to

do is we're going to go over here in App so this entire app lives in App. So the first thing you're going to want to do is if you're just here following along with me, you're going to have to do npm install with package dot json. Mine already installed so it's not going to do anything but if you do not have it installed, it's going to spit out its whole log. And now you should have a node_modules which will have body-parser and express. We're going to be working purely in the public directory We have index dot html and it's going to be pure and simple HTML. Don't worry about the charset. We're going to have a link at the top or a title at the top, like on the very top tab so if you're talking about a title like right here. It's going to be called Tweeter. And here we're going load in some Google fonts. I get kind of sick using the default font all the time so there's a thing called Google fonts. If you're not familiar with it, you just search for Google fonts. They're essentially a bunch of free fonts that you can use on your website. We're just going to use this top one right here, Open Sans. We're going to load that up top and that's just loaded by CSS style sheet and then we're going to load style dot css which is a style that we've written right here. Actually haven't really talked about this so far but this is how you load external style sheets. So you put a link tag in the top, there's an href which is essentially pointing to where your style sheet lives. In this particular case, it lives in the same directory so we're going to tell it just load style dot css, it lives in the same directory. That's our head tag, now we're going to go into the body tag In the body tag we're going to have an h1, it's going to say Tweeter. It's going to have a tweet-container. Actually let's go ahead and just look at this and then we'll come back and discuss how it actually is working. We're in app and I'm going to say nodemon app dot js. So now we should just be able to go here. Close this out. This is the basic styling and such and these are the different tweets, essentially. This is the say something part of it, so this is going to be this part right here where you can actually type in something and tweet it out. It's going to have a place, we talked about that, that's that gray text in the background and it's going to have a class of new text input and an id that we'll reference later in the JavaScript. And then a button that's going to allow them to tweet Then we have this div right here which is called tweets we're then going to later fill this tweets with the tweets from the server. Loading jQuery, we're going to load our tweet dot js. Notice that this order actually is important that jQuery comes first and then comes our tweet dot js because out tweet dot js which is js that we've written is going to reference jQuery, therefore jQuery must be available for tweet dot js to work. So let's go into tweet dot js, this is JavaScript that we're going to write or have written. So could you talk about why those JavaScript inputs are at the bottom instead of at the top of the CSS? Definitely, good question. So it may seem a little weird that the CSS goes to the top and the JavaScript goes at the bottom. So the short answer is that it's a best practice, it's just something that you do because it's the best thing to do. Now the reason why it's the best thing to do is that when you load content, you don't

want to load ugly, unstyled content and so you want your CSS to load before your HTML finishes loading so that when your content loads for the first time, it's already pretty. It's not so it loads with no style and then all of a sudden everything just magically looks better. It's a bad user experience. We want all of our pretty styling to load and then we want all of our js to run. So we don't want our page to load slower because the JavaScript has to load first before it can render. And the browser's dumb, it's just going to go down line by line, it's like okay, I'm going to do this and wait I'm going to do this and wait, I'm going to do this and wait so if you have your JavaScript up here, it's like I'm going to load all my JavaScript which doesn't really make much sense anyway because your stuff isn't there to be interacted upon yet. So again, short story is it makes your page load faster if you do it this way. Okay, good question. So the first thing that we do when you load the page is we're going to do a GET on slash ajax which Nina will talk about what that actually is but notice that this is different from the reddit API which we had a full URL for. Slash ajax is actually on the same server as where this index html, it's a relative path. So it's going to load ajax from the same server and then here, we have the success function here that's going to call this appendNewTweet with these tweet data items. And in the tweet data item, there is a tweet dot text and there's a tweet dot time. Haven't really talked about this, a really common way of keeping time in programming is called epoch time. Epoch time, not really sure how you say that. E-P-O-C-H It's seconds since January 1st 1970. Milliseconds. Yes, in this particular case it is milliseconds. Often it is just seconds, though. But make sure you're aware of what we're talking about either seconds or milliseconds. So it's just a big number. I mean it's a huge number right now. Think of how many seconds in the past almost 50 years. So that's what this is doing, this is taking a time in the seconds since 1970 and it's just turning it into if you look over here, a nice little well formatted time that actually makes sense to humans. That's what's happening there. And then we're just wrapping it in a bunch of HTML which we then have styled. I realize that I've actually call this appendNewTweet and then later I called it prepend. I call it prepend which as you imagine, append puts it at the end, prepend puts it at the beginning but I did that because you want the most recent tweet to be at the top, you don't want to have to scroll down to actually see your new tweets. So that's why we use prepend. Questions about that so far? So here, we're calling a click listener on the tweet id so let's go look at the tweet id which is here, id tweet. Okay so every time that button is clicked, it's going to send out an ajax request that's going to post to ajax. Notice that we have a GET on ajax right here and we have a POST on ajax right here, but on the server we've actually separated those two out Nina's going to talk about that. We're going to send json and we're going to tell the server hey server, I'm sending you json, just FYI. And then we're going to send it to JSON dot stringify we've seen that before, we're taking an object and making it a string. And we're going to send a tweet which is going to have

the value of whatever is in here. So we're going to tweet whatever comes out of here That's what that Payload's going to have right here. On success it's going to get back data. Data's actually going to be a new tweet of this tweet that you've sent that you're going to hand it to the server, the server's going to essentially hand it back to you so you can display it and then you're going to call the appendNewTweet and then here you just clear out the value. So just so you see how that works say this is my new tweet, you click Tweet, reaches out to the server, the server then hands it back to you and prepends that new tweet right here. Any questions about how the JavaScript works? So now we have style dot CSS. Talked about box sizing, we're doing border box just to make everything easier. Like I said I put this on every CSS file or every CSS project, rather. Body, we're going to give it a font-family of Open Sans thats the one that we're loading in by Google, and then here we have sans-serif, sometimes people are not able to load external fonts, so we give them kind of a fallback. Everyone has a sans-serif font on their computer guarenteed so you say if you don't Open Sans available, please use a sans-serif font. We're going to set the text color to a nice un-harsh black and if we don't do this, this margin zero, so let's just change that. Or sorry, that doesn't work in CSS, does it so let's just take it out. If we take this out, notice that right now this border goes all the way to the edge, if I refresh the page now it actually won't push up against the border. So you have to have that margin zero or else it won't push up against the border. So again, now it will push up against the border. Leader-h, that's going to be that thing at the top. It has a background color of black and it has a color of, this is really like an un-harsh white So it has a padding on it which essentially gives it this nice black look around it. If it didn't have that padding around it, it would just be really flat and in fact, let's just take a look. It just kind of pushes up, it's kind of ugly, that's not what we're going for. Tweet-container, so that's going to be these things right here. We give them a nice border radius, I don't think we've looked at that before but look at these, they have nice little rounded corners. I Inspect Element on that. So tweet-container. So border-radius right here, if I take that off, notice that has pointy corners now but everyone likes nice, friendly round corners so just going to throw those up there. Has a width of 80 percent of the screen, notice it doesn't push all the way to the edges, it actually just sits there nicely in the middle, the auto centers it. So we do this overflow hidden right here. Remember we talked about the great collapse and how you have to sometimes clear it with clear fixes? So this Tweet button is floating to the right and if you take that overflow hidden off, the containers not actually aware of how that float is supposed to play into the document flow so if you tell it overflow hidden it'll reset and say oh okay, this is actually supposed to fit inside me. That's what that does. Let's look inside a tweet-container now. Tweet-time, give it a smaller font size, make it italic so now it's actually leaning a little bit, color's grey because it's not the focus of it, because this is not design school, I'm not a designer. Margin bottom there, tweet-body just

made the font size bigger essentially. And I think we've essentially gone through pretty much the style. Oh, let's talk about this up here. So I actually made the input a little bit more fancy. Like this blue border is not typical. Where is that? So we have new-tweet-input with the 100% height, border, border-radius, all those things look familiar. Then we have thing called focus it's called a pseudo-class. So right now, it's not focused, it has a gray border soon as I click on it, it is now focused so you can have it actually apply a different CSS when it's focued on. There's lot's of different states for this and I'll show you one here for Tweet as well. As soon as I start hovering over it, it gets lighter as soon as I push it down, it gets darker. That's all done with CSS and let's take a look at how that's happening. So button tweet has all these styles right here none of those should look very intimidating to you but hover, we move from this font color to this font color which is maybe in color green, lighter green and then when it goes active, which just means I'm pushing it down, it goes darker green. Questions about that? Okay and that's essentially it for the client side code. It would be kind of an interesting exercise for you to after you've watched this video or after this, to go back and recreate not looking at the code I've written but instead doing it kind of your own way.

Final Project - Server Side

So we're back and we're going to talk about the server side component of this app and right now we're kind of funny, we're running the client and the server on our own computers. When you guys are going to put something out there, you're probably not going to want to do that. You're going to want to find some web hosting, for example Heroku does a great job with us, they make it really easy for beginners. So your server will be somewhere else and you'll have lots of clients connecting to your server which will be running code like this. So this var express in var app is something that you've already seen. This var bodyParser is something called middleware. For the sake of time, not going to talk about it, it's a little bit outside the scope of this class but it kind of helps us parse some of the input that we're going to be getting. So here's some tweets they we've pre-populated so if you kill your server and start it back up these guys will always be here, these tweets and that's because they're actually in memory. If your create new tweets in your app and you kill your server, those tweets are going to get wiped out. So let's go over the anatomy of these tweets for just a second. So we see our square brackets here, that means it's a list of tweets. These tweets are objects but unlike we've seen before, they don't have names. We don't have to give them names, they just kind of live in this list. So we have a text property and a time property and this new keyword is not something that we've seen before so I'm just going to quickly cover what that does by calling new Date, you're creating a new instance of the date

object. So what this will give back to you is an object with a bunch of properties that knows the time in this very instance, right now. Calling getTime on this object will give you back the number of milliseconds since January 1st 1970 and the reason we're subtracting from this time is because if we go back to our Tweets app, we see this time stamp here and we kind of want people to think that we've had some staggering in between our tweets. So here we see there's just a few seconds difference in each one of these and this minus time is what produces those different time stamps on the front end. So here we're telling express to use the static directory called public and underscore underscore dirname is something that's given to us for free in every node file. Basically, it's telling us what's the directory that this file is running in. There's enough built in is the right word for it but you kind of get it for free is underscore underscore file name so if you ever need to know what the file is called dynamically, you can just use that variable anywhere in your node files and node will be able to tell what that file name is. So let's talk about this app dot get method. Do you guys have any questions about http verbs? So everyone get what GET means? So we're just asking the server just give me what's here. I don't have any input for you, I'm not trying to pass anything back, I have a URL and you have a file I want so just give it back here. Here we're setting the response type. We're telling the server or I'm sorry, the client that the server is sending back JSON and what we're going to be sending back is JSON dot stringify of our list of tweets. So when this page loads, we're calling that GET method, we're getting back the data and displaying it on this page and we should be able to go to localhost 8080 slash ajax and there we just get back our JSON. Does anyone have questions? So this post here is the most interesting part of our whole little app and that's because as a browser, when someone types in here, we're hitting our server and we're hitting that post end point so we get some text, we make a brand new tweet and here in the text we say request dot body dot tweet and that's because what our browser sending us is JSON that looks kind of like this. So that's what the client is sending back to the server that's what the server is working with. So we're saying the text is request dot body dot tweet and the time is just a new Date. So we want that tweet to be saved on our system as getting created right now in this very instance. So we talked about or I'm sorry, array dot push yesterday so what this line of code here is doing tweets dot push newTweet, is adding that tweet on to the end of our tweets list. So this list is getting bigger and bigger every time someone inputs some text on our webpage. Right here we're kind of telling the client hey, we got your information, I'm just going to print out what I saved. Does anyone have questions about what's going on here? So basically this list of tweets is getting bigger and bigger and bigger. And that's the mechanics of our Tweeter. I think Brian and I have a million dollar idea here so you guys better not steal it. Come after you with lawyers. Anyone questions, questions about how this all fits together? You guys all look hungry and tired. How about tweet about some food.

So that's all that Brian and I have unless Brian, do you want to add anything? This is us. We are on Twitter. I'm on Twitter at nnja, Brian is at holtbt and if you guys have any questions, feel free to tweet us. It's probably the fastest way that you'll get back a response. It was really a pleasure working with you all so thank you very much.

Course authors


Brian Holt


Nina Zakharenko

Course info

Level	Beginner
Rating	★★★★★ (663)
My rating	★★★★★
Duration	10h 58m
Released	9 Apr 2015

Share course