# Front-End Web Development Quick Start With HTML5, CSS, and JavaScript

by Shawn Wildermuth

Start Course

Bookmarked Add to Channel Download Course

Table of contents Description **Transcript** Exercise files Discussion Learnin

# Introducing HTML

## Introduction

[Autogenerated] Welcome to the front end Web development. Quick start with HTML five CSS and JavaScript Course. My name is Sean William Youth of Wilder Minds. In this first section, we're gonna introduce the course and also introduce each team. L five. Why, that's important to Web development. You start out by talking about why learn Web development anyway. Then talk about why each male five is important. Dive into HTML markup. Look at the structure of an HTML page. Look what some common tags introduce youto HTML forms and then talk about cross browser hte e mail. Let's get started.

## Why This Course?

[Autogenerated] you're just starting this course, you know that you need to learn some Web development skills, and this course in particular is to help you learn HTML styling with CSS as well as client side code with JavaScript. The purpose of this course isn't to give you a deep breath of knowledge, but to allow you in a couple of hours to really understand the basic concepts of HTML, CSS and JavaScript. Instead of lecturing you with a lot of slides, we're gonna focus most of the course on some practical hands. On examples, we'll be walking through a simple website example until you had a buildup. Ht Mao change the way it looks with CSS and then finally write some code to actually interact with the user and with network requests using JavaScript. If you've been apprehensive about learning Web development, don't be. Lots of developers from different ecosystems have sort of shunned Web development because of the horror stories they hear from Web developers about dealing with multiple browsers. Dealing with the fear around CSS and dealing with JavaScript is a language. Hopefully, in this course, I'll dissolve most of those fears and show you that it's really a pretty good ecosystem these days, and that developing cross browser applications isn't nearly as difficult as it once. Waas and hopefully also dissuade you that JavaScript isn't a first class language that you can build great, softer with that. It's predictable, testable and actually fund a code in Let's get Started with HTML.

## What Is HTML?

[Autogenerated] The first skill you're going to need to learn is HTML and HTML is a markup language, a structure of what an individual Web page is supposed to look like. Hte emails a bit different from some other ecosystems you may be used to working with in that it has been built up through years and years of making concessions. Each team L, when we start to look at it, is really a standard built by the winners early on. A lot of decisions around eight. Cemal We're driven by the browser companies that decided to implement things one way or another. One successful browsers had implemented certain features. It was hard to backtrack as people were coating against those browsers in an act of Web pages that were actually working. HTML doesn't always make the most engineering sense. But having the understanding of how these decisions were made means that you can forgive them a little more, and knowing all the different rules will be enough to help you understand. HTML. Five. In general, we talk about H 05 A lot of technologies get lumped in under this moniker of html five. Start talking about CSS three and features of browsers like index databases and file AP eyes and touch events, as well as the changes to mark up in those sorts of things. Even things like Web storage or Web sequel were lumped under this heading of H M 05 We're not gonna be talking about all these related technologies as you learn Web development, especially client side development. You're gonna learn a lot of these details. This course will cover JavaScript and CSS, but we're not gonna cover the breath of what some technology companies have decided to call each mill five. Instead, we're gonna focus on the markup language itself on the angle bracket language that browsers can interpret and then draw on the screen. So this brings the question of what is HTML itself. HTML stands for hypertext markup language that around for a very long time and essentially is an ankle based structure language that allows you to define different parts of a page, some visible to the users and some not visible to the users. It's derived from the S G M L language, which simply defined general purpose markup language is that it really brought in the ankle bracket structure that we see today in hte e mail, for example, this is a very simple HTML document search and ends with tags. They're enclosed in angle brackets. These tags have a hierarchy structure to them, and it should always start with the top level HTML container. Each team out contains ahead. That contains some metadata about the document and then the body, which is the visual representation of what the user will actually see. And we'll learn what each of these tags mean more in detail when we start to look at actually building these, using a text editor for some developers who have been out there and working in different systems before, they see the angle bracket markup language like hte e mail and assume that's going to file the same conventions that XML does. It is not case sensitive, and it has mixed closing rules. If you're already familiar with XML, this can actually trip you of in ht mouth. For example, the script tag you see here can't be self enclosed, like the image take here. It has to begin with a script tag and then end with a script tag, even if there's nothing in the body of that tag. Conversely, the image source and some other elements can never have a body. They Onley describe a set of properties. You'll see this more concretely as we start to build our own HTML pages. 80 mil relies on something called the document object model. This is basically the hierarchy of elements within an HTML page. Why this hierarchy of elements is how you describe a page at runtime. It also represents an object graph as we look at the markup oven html page. It's really a graph of elements that represent the things that the user can see and interact with, but also a set of objects that use a developer can interact with. So a document contains a head and a body section Head might contain other objects, like the title section. Ah, body might contain headers and section objects that can contain other elements themselves and so on as we start to break this down. So we should really see the document object model as a representation of the objects that are available to the browser and to the client code. Let's see how this is working in actual practice

## Hello Markup

[Autogenerated] slit start by understanding the simplicity of markup. For our examples, I'm using a sublime text editor to just write plain old text files I could be using note pad. I could be using them or E max or any editor, including Visual studio. What we're doing is just creating simple files that within the host as a small website, doesn't matter what tools you're using. I like using sublime in this case because it's really thin and easy to see all the moving pieces. So here we have a really blank slate when we're dealing with mark up were really just talking about creating a hierarchy of elements. So let's just create not even H to mount this point. Just some markup. So some of the basics of Mark up is that you can have an element, and an element is a name of an object, and it's surrounded by two angle breakfasts in less than a greater than And in this case that represents the beginning of this structure to end an element. You're going to start with that same greater than unless then, with the same names we have here. But you're going to start it with a slash in front of it. This is an indication that this is the ending part of this. What is called an element, then everything inside of that is simply the content of that element. In addition, you can have what are called siblings, and that is elements that air at the same level that our sisters or brothers of the other elements. So let's do that with, ah bar element here. So in the case of bar, we might not have any content. And while two perfectly valid to leave the beginning and end part of the element just next to each other, some of the elements support what are called self closing tags, and that is ending an element with a slash greater than this indicates that it's an object, but it's an object that has no content inside of it. It's a simple atomic element. In most cases, Instead of having arbitrary content, mark up will have a higher key relationship or a parent child relationship. So in this case, Fu is the parent of the bar element, and the bar is the child of the few element. So a simple parent child relationship, and this is how you can build hierarchies. One parent has another child, has another child, has another child, and it's perfectly valid to have, Let's say, a couple of these. And so in this case, food as a parent has multiple Children, and these are some of the very basics of markup itself. But we're really talking about. HTML is a markup language, a hypertext markup language. So instead of these arbitrary elements, let's use elements that are actually HTML markup elements to hear. I'm going to change this top level parent to a div. A div is a basic unit in HTML that says, This is a portion of my Web page inside that DIV and innate smell page who might have an image tag as well as, let's say, a paragraph tag. P is for paragraph. So in html we can have these atomic elements, and image is one that is always atomic. It never has content associated with it, and then paragraph tags that may have arbitrary content. This case I have some piece of text that's gonna be shown in a formatting for a paragraph. So this is about the most simple piece of markup we could possibly have in each team out. Even in content scenarios, you can have other parts of the content wrapped in their own elements. For example, if I say hello World and put my name in there, I could surround the world with a B tag or be element, as it's often known, and that is gonna be some indication of how I want Ah, formatting the world. And in this case, that would be bolding of that text. So even in the case of content with HTML were using elements to surround or to tag different parts of the markup so the browser knows what to do with them. Elements themselves can also have what are called attributes, and these are pieces of information name value pairs. Typically that air inside of the actual element tags themselves, so I could define that this def has an idea of top. That image could have an attribute called Source that pointed at some actual source location for an image. Individual elements can have more than one if necessary. Like in the case of image tags, we might have a second attributes that defines the alternative representation of this image for viewers that have images turned off. This is often the case for screen readers for the blind or for the partially blind. If you've come here with some prior knowledge of markup languages like XML, you want to be careful with understanding this mark up, because in the case of XML, there are some rules around XML. X Amount was actually developed after HTML and want to define something that was a little more structure than HTML. It's female hasem specific peculiarities that make it a little bit more difficult to parson period XML. So if you come from the external world, one thing you have to realize is that, unlike XML, the mark appear in HD mouth is not case sensitive, So making a capital I am G for the image tag is perfectly valid. Most people use lower case tag names, but it is not a requirement. There's also this notion of being older, self close, a tag like we have here an image. But not all elements support that there's tags like script in paragraph and give that must end in a full element closing like this. They cannot end in the shortcut version, and there's some elements like the image tag that can't have its own content that never has Children. These peculiarities can throw you because in the Exim aspect they standardize these things, so they were a bit easier to parse. It smell. You're just gonna have to learn about these exceptions and live with them. We're living with them. We can't just change it because we have browsers that go back years and years that have existing websites that are relying on these rules. To be in place is one of the peculiarities of Web development. That can be a little painful, but most of the library's most of the code we're going to be using are going to have ways to sort of back feel older browsers for us, anyway, Let's take this and let's start building our first HTML page now that we've looked at some basics of actual markup

## Structure of an HTML Page

[Autogenerated] next. Let's create a Nate smell page from scratch so you can see the real structure of an HTML page. I'm gonna go ahead and close this temporary file. I'm gonna create a new file. I'm just gonna call index that ht Mao to represent just a simple HTML page each mill pages start with a tag called HTML and that is the parent for the entire HTML document. Inside the HTML document, there is a head section and this is going to contain some elements that air used to interact with the browser, setting the title, bringing in CSS sheets, setting meta tags, things that aren't really about the look and feel of the individual page. For the actual body of the document, we're going to have another tag called Body. The structure is the very skeleton of an individual html page, an HTML parent called HTML that contains two elements ahead and a body.

Before the HTML tag, we start the document out with something called the Dark type. This is a bracket exclamation point. And then the word doc type to indicate what is the actual type of this document. So the stock type is going to indicate what kinds of elements of browsers going to expect in your document. If you include element types it's not familiar with, it's actually going to interpret them. Most browsers do, at least as Div elements. So if you want to have your own named elements, they will be interpreted, for the most part in modern browsers as a DIV element itself. So go ahead and make an actual working HTML page here. Right now we have a validate smell page, but there's nothing at all that's being displayed in the browser. First thing we'll do is start with the head and we're gonna go ahead and give our document a title. Call the Get Hub Hub So simple. What? Paige. It's going to look at the different projects on Get hub dot com and be able to look at information about them, not a super useful Web page for us. It's going to introduce a lot of these concepts in the body itself. Let's start with what's called an H one tag. Each one is a heading is actually six of these daughter defined with HTML H 12345 and six for different heading levels each one is normally the largest of these, and it's used for big visual elements on the page. So let me put that same name here, Get Hub Hub and let's actually go of your HTML page in the browser. So as I go to our index dot html, Page will see a couple of things here. This is that H one we define that is hard in the body. The body is going to exist here in the main part of the browser for one of the other things you should notice is that the title ends up being the title in the tab or in the title box of the browser. Most browsers now are tabbed, but even in the case of non tapped our only individual pages in a browser, you're going to see that title show up right here in the tab. So that title's important. It's not just a piece of record keeping. It's actually something that is displayed to the user, but go ahead and book market. This item, that bookmark is actually going to be the name that was in the title Get Hup! Hup! That title element is important to get and get right. And now back here, let's add a paragraph tag just to indicate some information. This is a sight to search. Get hub for interesting projects. Let's go ahead and create a second paragraph tag that might have a little bit more information. So now we have just structural information River heading and then a couple of paragraphs. If we go back to the browser, we'll see that it's separated these into something that sort of looks like paragraphs we make this smaller will see that the paragraphs will actually text trap, as you would expect, paragraphs would in any sort of textual representation. But we'd like to have some more information here. That might be interesting. Let's go ahead and use another tag around here around the sample site that says, Go ahead and show this in a Tallix And then let's go ahead and do the same here for bold. So these are elements in their own right, but they are also information to the browser to indicate how to format what the user scene in this case use italics in this case. Go ahead and bold those elements, and we can now see that theatre Alec is used here. And then the bold is used here in HTML five. They've sort of gotten away from using the eye and the B tags and instead want to use ones that arm or indicative of what they actually do. So E. M, instead of eye is actually to add emphasis. And instead of B for bold, they actually want to say, We want to make these strong. You can see both of these an example, So I want to make sure you saw them. We go back to the website and refresh. We're going to get the same formatting. So the I tagged as well as the E M tagger. The same instructions to the browser and that is to use a tallix for the font. And B and strong are also both the same thing. Of course, HTML five is a hypertext markup language, and what do we mean by hypertext? One of the basic ways that the Web works is that sites can link to other sites that you can follow this chain of information across from site to site. The way the Internet works today, and this is using a tag called an anchor tag or a tag. So for the get up here, let's give them the opportunity to click on the get hub and open up the gate Hub website so they know what we mean by the gate. Have website. So inside this A or anchor tag, we're gonna include a property called a treff eight. Rift is where should this link? Oh, in this case will just go ahead and say get hub dot com When we refresh this in the browser, The gig. Yeah, but now is an act of link. When we have our over it, we can see it's going to get hub and it displaying differently in the browser so that when we click on it, it will go ahead and take us actually to whatever the Web page, whether it's Internal tour project or an external Web page like we did here with get up, The last thing we'll do is go ahead and replace the contents of the name of our website in the H one here with an actual image. So I'm gonna replace that and just create an image tag itself. And the image take is gonna want a source to some image to display, and in this case I'm going to use a folder structure inside my project. And in this case, I am G slashed logo dot giff. So, of course, our folder here with the index doesn't have any folders. Let's go ahead and change that. You now have an image folder that I just created with sublime that you could use any method you want and I'm going to copy from My resource is folder inside of the samples that are available to the PLO site plus members. You're gonna find a folder called Resource. Is that contain some files, including this logo dot gift. So you can now see when we open this up, there is that little doubt gift. And so this path is saying, Go get and display this logo dot give here. What's interesting about the source here is it's going to be relative to wherever this file is. So by saying image slash local gift, it's going to say, Oh, the sibling of the index in my folder, structure his image and then go find the logo inside of it. We certainly could have also said, Ah, full absolute path to a resource is well But typically, if you're using your own researchers on your own website, you're gonna want these to be relative paths relative to the location you're dealing with. We go to the browser and refresh. We can now see that we have an image that represents the main part of our website. Instead of just some textural information scene with just a very simple set of HTML, elements were able to display images, format text, set the title to our element, so certain to see that the structure this document were creating, that the document is something that the browser is going to render for us, based on the instructions that air in the HTML.

## Tags

[Autogenerated] next, let's talk about some important hags in HD Mt. Earlier briefly introduced the idea of a tag called a Div Negative is just a section of an HTML page. It's sort of a catch, all for structural information. So we take these 1st 2 elements and put them in a div. Were saying that these were going to go together somewhere on our page, and this is later gonna represent the header of our page. Let's do the same here for the body of our page. And let's create 1/3 doof that's going to represent the footer over a page gives themselves. Don't have any you I to them unless you tell them to color or add elements. And so even though we've added the structural pieces thes dibs in place, our page looks exactly the same. We have a dip that surrounds these two elements and another one for the body here and then, actually 1/3 1 that has no content in it yet for the footer, let's go ahead and add that content to the footer. I'm going to introduce you to a way of finding special characters. HTML supports this idea of escaping characters and all these start with the and sign. In our case, we're going to use a simple one called copy, and that's going to show a copyright symbol. And so, as it would be appropriate for a footer, I'm going to show just a short copyright notice. Thes escaped characters are actually what are called entity references in HTML. Thes allow you to create characters that may not be valid purely inside of the HTML. There are other characters, and there's quite a few of these. I'm not gonna cover all of them, obviously, like GT for greater than an lt for Less Than and these air. Interesting because thes represent the symbols of greater than or less than which, of course, would be valid because we need the greater than in less than two represent the DL emitters around each of the elements. You can also represent Unicode characters by just representing them as a number. He's also start with ampersand, but then our immediately followed by a pound symbol and then a number. In our case, if we did, let's say 9 37 That would be the decimal representation which shows this character here or after the number sign. You can use an X to indicate you're going to have a hex, a decimal version and let's do 1 60 for this character here. And so that even if there isn't a special entity reference inside the HTML, you could always drop down into Unicode early in HTML. This def tag represents a rectangular section of the page. That's why, as we're going to use these were headers, footers and sort of the body of our page. The DIV is a good solution for that, but sometimes we need to be able to define an inline section of code, and we can do that with something called a span. It's a hero at a span inside our text that's going to represent just these three pieces of code, and this would allow us later to style or change the look of just the characters that are inside of the span. It represents an inline container, much like the day of represents a rectangular container. Those are the two basic containers you're going to end up using. But even though in a lot of hte e mail, you're going to see Dave used to define those divisions or those containers in each small five. The introduced would have called semantic tags and these air tags that are better named to define what they're actually doing. And some search engines can actually look at this information in order to to discern what data is inside these tags. For us formatting. Some of the tags air format it simply as dibs, but they're supported as these name tags again that have a semantic meaning. Look, having used these here, one of them is tthe e better tag. That says, Here is a section on our page, but I'm going to define it so that everyone knows this is the header of the page. We could do the same down here for Footer again. We're defining a section of the page, but we're also giving it semantic meaning in the mark up. This is so people that air parsing it like search engines, mobile devices, mobile and desktop browsers that they can make some decisions about them, that we are implying that the header is going to look a certain way. The footer is going to look a certain way or the Dev. But you could imagine a search engine as it looks through Mike Oh well, the header information isn't going to be necessarily descriptive of what this site has to offer. Certainly, the footer may not be, but what's in another section on the page might, in fact, be more interesting for indexing for search ability. The last of the semantic tags will show you, and there's a number of them again that could be able to cover. All of them is just called section, and this is to give Maur meaning to the structure of a page. If this were a blawg engine or something like that, we might imagine that this is going to make sense that maybe the body of a block would be here. The header of the block and the footer of the block are here as well, and so that we know that there's gonna be some repeatability. It's again providing the semantic or this really world meaning to these sections on a page. Let's show you a few more tags, including another of those semantic tags that were added in a Channel five, and that's a take called knave. This is a new tag that represents the navigation of a website and so by certain people looking at this, it can know that this is going to be the section in the twin provide navigation across a website, and so inside the now we can define really in any structure we want. A list of links to other pages were actually only going to build this one index page. But let's create a couple of links to go to different pages, even though we might never implement thumb in our simple and small example. So here I might have another anchor that goes to the home page, so I might just start it as index that hte e mail cause that's always going to be the reference to this page itself. Don't create another one for contact, and I'll just pretend we're gonna have a contact page and then, finally, for about every website needs those three right three very standard pages. So much like the other links we've created earlier in the browser. It's just going to show these as clickable links. Obviously, contact and about aren't going to take us anywhere because those pages don't exist. But home will continue to take us back to this originally Team L page for this, I'm gonna actually introduce a new concept. And that's what is called lists. There are two kinds of lists that are pretty common in HD Mount, an ordered list, which is really a numbered list or a kn ordered list, which is really a set of bullet points. So let's start with an ordered list. Search with an ol tag and end with an N tag. Like all of our pieces, and each item in the list is wrapped with a list item tech or an L. I Tech amusing little shortcut here in several I'm last me to quickly just surround and create this structure for us because it's an ordered list. It knows that each of the list items here should be shown with a number sign. In fact, if we go look at it, we can now see that it is just a numbered or ordered list. In our case, we don't really want an order list. We won a nordeste a u L. Because it's gonna be a set of bullet points, and later we might change the style, depending on how we want this to be shown. But we'll use an UN ordered list. This is a pretty common case for navigation pattern. We can see that because it's a kn ordered. Now it knows to just show bullet points by default again in the next module. When we look at CSS, we'll see how to change this toe. Make it look like this exact thing you're really looking for. One thing to notice is that this logo appear we actually have a inside of an H one tag and an H one tag or first heading tag usually has some spacing in it. Same for this paragraph tag we have here later. We're going to design what this header it looks like. Exactly. And so this probably isn't useful to include the H one or the paragraph tag as tags that have some built in formatting to them. And so I'm gonna go ahead and convert both of those two dibs. This won't substantially change the way they look, but will allow us to remove some of that spacing in between them without having especially format that later. The header is really all the items inside This Currently, the section is this whole text that we had created and then finally we have our footer down here. Don't worry too much about this Being a little simple and ugly. We're going to get to changing this design when we get to the second module.

## HTML Forms

[Autogenerated] Now let's talk about forms in HTML So far in the website we've built were on Lee displaying information. We're showing the logo and some information about the site and some texts and a footer. But we're not allowing the user to actually give us any information to collect information from the user. Let's go ahead and do that using HTML forms. We're gonna do this in the section here and right after our paragraph, I'm gonna create a form element. And a form indicates that we're collecting some information and then probably submitting it to somebody, whether that's sending it to the server to some standard service. But we're taking in collecting data from the user so that we can send it back to us someplace on a server and this course we're not gonna cover the programming on the server side. So we might use, um, standard service is that are available out there on the Internet to just see how this all works. An element that's important to forms is one called the input element. We go and look at the page with this input element. You'll see that by default, the input is a text box just a standard old text box. So it's collecting some textural information. Name, password, email, search, string, whatever you want. But it's just a simple input box. We create a 2nd 1 of these, we can tell it what type were actually looking for. Now, some of the controls you're going to be dealing with are going to be input with special types. In our case, I'm going to make the type of type called Submit. And this is going to look like a button. And and this is the button that's going to say, Go where? Start or in our case, search, because we're gonna create a small search form. Here we look at this in the browser. We can now say that we have that input from the first element and then a actual button that is going to be executed when we press the search in order for reform to know where to send the data it's collecting, we need to give it an action attributes on the form itself and for the action where the purposes of our little sample here I'm going to use an action called Echo. So this is, ah, small tool. I have upon as your websites that will take any input from a form and then just show it out on the page so we can see what's being sent. If we run this in the browser and press search, it's actually going to take the contents here and attempt to send them to the action. You can see here sent us to the echo page, but it didn't send any data. The reason it didn't go back is that it needs to know what to call each piece of data. In the case of an input, we need to give it a name like search phrase for an example and by giving it the name of search phrase were telling it to when it sends it to whatever action we wanted to send it to to use that as the name of the value that the user has inputted. But to refresh, you can see what I mean by now. Say hello and press search. It actually adds it here to the query string search phrase equals Hello. It knew that because we had an input called search phrase and then the value was whatever was in that input box Let's use another input here, but in this case will use a type called check box. And I'm gonna give this a name of you stars. And later on, when we implement this against get hubs repositories, we'll see why I'm using a check box here. But unlike the surgeries, I want to give it a default value, so I'm going to say checked equals. True, This is a property on check boxes that says, please make this check box checked by default. I'm also gonna add some text here so that we can see what the check boxes for you stars. Question Mark. And then I'm gonna introduce a new element called a break or be our tech. BR is always self closed in this way, and it's just a single element. But the B R here will cause each of these to be on their own line. This causes a line break inside of the formatting of the HTML. So if we go look at this in the browser, we can now see, we have the text box still there. But now we have a check box by default. And if we go ahead and search well, see we're now getting both of those elements. Search phrase has its own because it's named, and the new stars is now on or true for the check box. The last piece we're gonna add here is going to be a drop down or what's called in a chit amount. A select A select allows you to have multiple options for different options that could be chosen. Let's go to heaven, give it a name and call it lying choice and the option elements inside the select can have an attribute called Selected Now you can say selected equals true. But this is another peculiarity in HTML in that sometimes the attributes existence is enough. And in the case of selected, that's true. Option Selected means that all is going to be the selected item by default. Now the 1st 1 is going to be selected item by default anyway, so we don't really need it there. But we could decide Thio as the default makes C sharp. The selected language. Let's go ahead and put a V R tag here and now Let's go ahead and look at this in the browser. So now we were dropped down and even though it's the middle option, it's still the default selected one. As I change this value, you can see that it's gonna pick whatever is there is well. And if we search because we've named it his Lang choice, it'll show up here as one of the properties as well. Now, the way the http works is, you can send things in a variety of ways. Doing a get is the way that most of the Web works. By typing google dot com, it doesn't get to Google that common, and they return mean html page much like when I do it. Get in the browser for index that HD mail. This page actually is returned in the case of forms. Often you don't want to use the verb of get you want to use the verb of Post Post has the idea of creating new data. It's essentially the insert or the update of data across the Web, much like you might think in standard data access terms. So in the form, we can actually specify which of those verbs air used. We do this by using another attribute called Method, but fault the method is get. But if we change this to a post and just execute our same page again, we're going to see that it post to this echo. You're l just like we saw before. But instead of attaching it as the query string, it's actually posting it as the body. And this becomes important because some types of data aren't able to be part of a query. String the query String semantics After the address here, the question mark query strength implies that we can only have textual information. So if you need to send up an image you chose or some other structural data, it be difficult here. The way it does this is it actually sends it as the body of the request sends it as something called form encoded as a format form does. This by default is just a list of the properties inside the form. Knowing how to change these and knowing that the former object itself was going to handle that is important to know when you're doing Web development. Later on. In the course, when we talk about JavaScript, we will to see a little bit more behind the scenes how the actual networking is happening. But for now, I understand that you can specify the method when you go ahead and to find a form. Let's do one more thing with forms in our form. We just have the input that where we're collecting data. But we don't really have a place for us to show what each of those pieces of data actually represents. In fact, here in the form we have you stars next to the check box. But we don't know that Hello is really the search string or that this is specifying some language. We don't have a way to really describe that. We could literally just say, search there and say language here and get something that approximately looks like what we want. But more interestingly, you can specify a new element called the label. The label is going to allow us to say search phrase like we did a minute ago, but we're going to be able to point it at a specific input. This becomes really obvious when we're doing it or the check box. So let's go down to the check box for a minute and let's around this with a label and we're going to say four equals you stars. Now this label's coming after the check box. How does it know which one? It's four. That's why the four attribute is there. It's to tie the two together as you lay out these pages. The label may not always be before right after some input, and so the four ties them together. Unfortunately, in HTML, the tying together isn't a simple as it looks, because the Four Attributes expects that this name is going to be a new attributes on each of the inputs called I D. The name attributes is used with informs to define what data descend. But I d is used within HT Mao to define the unique identify her for an individual element in the document. And so here, well, actually going to repeat it. And this is something you're going to see very commonly done informs. I wish that they would fix this so we didn't have to do this over and over again. I'm sure someone smarter than I can explain why we have to have these, but we have to have these. Let me go ahead and create a label for the language as well so you can finally see why this is important. We go look at this in the browser. We're gonna have the same look we had a minute ago. But what happens with the fours and actually give some additional functionality to your form? When I click on the label, it's actually going to take me to the control that's important here and for input controls and drop down. So it's not all that interesting, but it's crucial here. This means that when I click on the label for my check box instead of having to feebly go look for just a tiny search area for the check box, it's going to check her uncheck the box for me. But my biggest pet peeves with HTML pages out there is this very simple built in functionality that every channel paint should be doing when they're using. A form simply isn't there, and it makes me a little crazy by adding labels were giving some semantic support for knowing what parts of the page belonged to other parts of the page, and this is really going to eventually once we make our pages look the way we want it will enable us to richly format the pages with CSS and our next module, But that's essentially a core of how forms work. We're gonna come back to forms. When we talk about JavaScript. We'll interact with it a bit more, but at least we can see how we can build up a simple form on a page.

## Cross Browser HTML

[Autogenerated] next, let's wrap up some HTML ideas that are going to allow us to handle Cross Browser each email a little bit better Before we talk about cross browser. Let's talk about one more sort of pseudo tag that's gonna be important for you, and that is the comment structure inside of hte e mail. The common structure starts with a exclamation point or bang if you prefer, and then two dashes and then always ends with two dashes and another intact. Anything inside of this, even multiple lines are going to be comments. It's completely ignored by the par Sirs. This is going to give you a place where you can actually add comments inside your HT mail that are, for the most part, ignored. So sometimes you'll see comments at the top here to define some information again. It's ignored by pretty much everyone, but it could be useful for you to add comments. But one of the places where comments become sort of interesting is a structure that's out there for adding cross browser tests. So in our case, we can actually say if the browser I'm using is E. And it's less than nine. So this is essentially saying, if less than III nine. So I'II 8765 Whatever it is, we want to do something here, and we're using the commenting system so that we can put optional pieces of code here for older browsers. Can the structure is a comment starts and ends, but it also has this if browser version less than you could GT for greater than is well. And then it looks for an end. If afterwards to be sort of the end of this block. And then here we could do whatever we want, we could say. Now this all gets downloaded and consumed by the browser and on Lee, certain browsers that test these things will include whatever is in between these two blocks is something special for these cross browsers. For example, there's a script out there that will handle a team of five tags for older browsers, and this really only effects I'll browsers before nine, which is why we're using that there and so we can add this script. I'll see how script tags work a little later, but essentially we're saying go get this html piece of JavaScript on Lee in the case that were using an older browser. Otherwise, the headers and the sections and the NAVs will all be interpreted correctly if you're using I 9 10 or 11. But using an older version, the script actually goes through and rewrites them to straight out def tags so that the browser knows what to do with them. Otherwise, they might be ignored by older browsers and the magic years. If you are using an older browser that doesn't know about this, let's say you're using an ancient version of the Android browser or, let's say, an old safari browser, because this is just a comment still adheres, the comment rules. This will just be completely ignored, so it's only the browsers thing will actually use it. This is going to allow us to include some code it maybe CSS and maybe JavaScript. It may be some in line properties that are important, their specific to specific versions of browsers. You're gonna see this in a lot of templates and a lot of examples, and this is, in fact, one of the lines of code that goes in everywhere project. I do so ends up just being coming boilerplate background, but understanding what it looks like and what it does. I think it's important for most beginner with developers. Let's wrap up this module.

## Summary

[Autogenerated] So we've been talking about HTML primarily and focusing on some HTML five specific features. But you should really just be thinking of this module as being one about mark up. When we look back at the history of HTML, it's important. Understand that the nature in some cases the peculiarity of hte e mail is then because it's always been written by the people who release browsers, the browser makers and it being the ones that write the specs and the sometimes write them in the reverse order, unlike most computing systems were, you write a speck and then you build the software. Often the software is built on. The suspect is written to match the existing base of users out there. It's easy for new Web developers to get a little scared by everything that encompasses that HTML five moniker. It has really become a catch all for a variety of Web technologies. When we talk about H. Mellor specifically to a five, let's really focus on the ML part of HTML, the markup language part of it, and understanding all the different tank that you may and should re using in your projects. We've only touched that sort of the surface area of them, including forms and structure and header tags and the basic structural information about HTML. But as you build websites, you're gonna find the use in the need for more more tags and you'll be exposed to them as you need them. Remember that eight smell really is just structure. It's just a hierarchical document that displays what the browser should be showing to the user. All that the H E mail becomes is an object graph in memory, So don't get too confused by the nature of HTML. If you think as a developer that it's on Lee going to be an optic graph that we're just creating parent child relationships, a true hierarchy, I think you'll find that's pretty easy to wrap your head around. We spent a lot of time on creating forms in HTML, and this is really the key to accepting input. So whether those are ________ and input boxes and drop downs and radio ________, which we didn't really show you where check boxes or simple textual input, these air all part of the form elements inside of hte e mail. We also showed you a simple technique for bringing in assets like code and CSS or other things that are going to be specific to certain browsers so that you can fix cross browser issues when you need thio. It's becoming rare that you need to, but it's something that you certainly want to keep your head around, and we've shown you some simple techniques for how to do that. This has been the HTML five module. My name is Sean Wildermuth of Wilder Minds. Thanks for joining me.

# Styling Your HTML

## Introduction

[Autogenerated] next, we're going to talk about how to style your HTML. My name is Sean William Youth of Wilder Minds. We'll start by talking about styles in each team now and then. We'll introduce cascading style sheets, walkthrough, CSS rules, CSS selectors sizing in CSS positioning and CSS finally had a debug styling in the browser. Let's get started.

## Styling in HTML

[Autogenerated] Let's start out by talking about styling in HTML. There's a style property on every element inside the HT mouth, and the style property allows you to specify rules for layout. We could certainly use this style property, but in most cases we want to separate the U I and that style. We can do this by creating style tags in the header and including the style information. This will keep it separate from HTML, but this really is only separating it on a per page basis. Instead of that, we're going to use a technology called cascading style sheets or C S s for short. These contain all the rules for styling for the U. I leaves the HTML for structural information. And then this way you can allow for site wide rules, not just page specific rules and overriding for specific cases where you need elements on specific pages to override the default site wide rules. Let's start out by looking at the style attributes and how that works. Over here in our index that HTML that we built in the first module, we're going to see different elements on the page. I come over here to our header element, which, just to remind you looks like this. The header element is the section that contains the logo as well as the site description in the list of knave instructions for the home contact in about pages. Because the header, like every other element, has a style property, we can simply use a set of name value pairs here to change the way this looks. So in this case, I'm going to say background color, which is one of the supported style names, and there's a lot of them will cover a few in the course, but over time you'll learn more and more of these. I'm going to send it to light gray. Now, the way the name value pairs working styling is the name of the property in this case, background color followed by a colon, and then its value. And then a semi colon separates it from the next name value pair set. We save this and show it in the browser. Well, see, now that that header section has a light gray background. When we talk about the style property, this is really what it does is it changes the look in the field of that object, and because the other objects live inside of it, they're also changed by it. In this case, the header take itself has a background of light gray. And because none of the other elements have styles to specify the background color, the light gray shows through or as if the logo or one of the other parts of the page had its own background color. That which you around that part of the element only. But of course, creating these styles on each element individually is really mixing the structural you I information from the way it's displayed on a page, and we want to separate that, and we'll do that next.

## Style Sheets

[Autogenerated] So now let's talk about the style tag that is usually put in the head element appear in the head. We can add a new section, or element called style, and is going to allow us to what styling information that separated from the individual attributes. So in our case, we want to take the existing style information we have here, and we want to put it in the style section. The problem is, we need a way to specify wet element it belongs to. In this case, we can just say Header and then create curly braces around the set of properties we're going to include. By doing this, we're going to get the same behavior we got by including the style element is going to be the instructions for styling all the elements on the page with the name of Header. Of course, we only have one, so it's only going to affect our one style. We refresh it, we can see that it still looks the same. This whole section is called a rule or a styling rule. This is typically going to match one or more elements on the page and then specify how, though, should be formatted. We can add multiple properties here so we can describe all the different ways we want to change the look and feel of that element. In our case, we can go ahead and say, a border property and inside of a rule. Not all of these individual declarations or name value pairs are going to be single valued. We can see the background color has a single value, and that is light gray. But Border has a number of elements to it. We're going to say it's going to be a salad border that is one pixel wide and is going to be black. So in this way it's a compound set of values. We can say all the information about an individual border, and in many cases he's gonna be broken up into individual lines as well. But this usually is a bit more efficient and will make sense more to you as we go along. We look at this now, we'll see that we now have a black border around our entire header section. Let's create another rule and this time it's going to be for a body, and of course body is going to specify for the entire u Y part of our page. Everything that's within the body section one of the properties. We can specify his font family. This is going to be the name of the font that can be used. But of course, in different machines, different fonts are going to be available. So far, family supports fall back. So in our case, we're saying ifs ago. You I exists. Use it otherwise fall back to area that doesn't exist either. Fall back to Helvetica. If none of those exist well, back to any sand serra font you confined. And this way, depending on machine, this may render a little bit differently. Were specifying thought names that we expect to exist on the resulting machines. You can, of course, import and use fonts and use the specific funds that are downloaded. But we're not really gonna cover it. In the simple styling example another property, we can specify ISS font size. In this case, we've said everything inside the body. We're now going to change the standard front family and the standard font size. So this is gonna take our example from this with a lot of Sarah fonts. We're going to take it down to this with a lot of San Serif fonts. Now this machine happened tohave that several U i want on it, so it's actually using that one. But if it couldn't find it, it would again drop down into Aereo or Helvetica her. Any San serif font it could find. If you find a machine without any of those major funds on them, let's go back door header. Let's change the font family here. And let's just specify times New Roman and Serif, So we'll use times New Roman if we could find it. Otherwise, just use a Sara font. Now we specified font family in two different places. And how does it know which wins? Let's look and see what the ugh I tells us about which wins the header won the fight for who is gonna actually use it, And the reason is that CSS and the styling stack on the Web is using a set of rules for cascading these properties because body is specifying the body of the entire page theme or specific tag of Header is gonna win. In this case, there's two rules about cascading. You want to think about one is if a rule occurs later in a list of rules, it will win over an earlier rule. In that more specific rules be less specific rules. In our case, body being the generic for every element on this page loses out to the more specific header rule that specifies what it is for just this section in here. We're going to see some more of this overriding or cascading, as it's known as we go through some of the other examples.

## Introducing CSS

[Autogenerated] So now that you've seen a couple of the pieces in action and working, let's talk about CSS itself. The actual cascading style sheets, the style tags on the top of the page certainly work, but they aren't very reusable. Would have to copy this on the each and every page of my project in order to get them to work across the project. And, of course, maintaining them over time is gonna be a problem. And so, in most cases, instead of including style information anywhere in the HTML page, we're going to separate it out into a new file. I'm gonna create a new file in a new sub directory called CSS, and I'll just call it Site that CSS. Ordinarily, you're going to keep your CSF segregated together so they may have a number of CSS files, some that are specific to the entire site. Some, their specific to certain pages in our case will go ahead and create the site CSS that will use for each page in CSS. It uses the old sea style comment tag. So I would just like to put the name at the top of it. That way we know what file we're looking at. If it's open an editor and all we have to do for the sin taxes used the same syntax we get from the style tag. So I'm gonna go ahead and cut and paste that over into our CSS file. And the trick here is instead of having a style tag here, Rex, you're going to include the style to be downloaded and used on the page using a link tag. So just create a link and their two properties that we need to say one is rail, and that's going to tell it what kind of link. And in this case, it's going to be a style sheet rail. They're a couple of other rail tags that can be used here but central focusing on styling. This is what needs to be in the link forward to download a style sheet. And then there's an H ref tag that represents the actual path to it. Sensor html Page is here. Our site CSS isn't a subdirectory called CSS Weaken, just a CSS stop sightsee assess, and it will download that now that we have both files created and saved. Let's show you that it's gonna show the same data in the browser. So we're still downloading this page. But we're downloading the additional CSS file to style our page for us. We can see that this actually works. But let's come, aunt, out this line that we can see our header go back to the new style San Serif Font, huh? So is pulling in the CSS information directly from our new CSS file that we just created. So let's make some more changes and teach you a little bit more about this in tax. And here let me create a new rule for that footer tag that we're using. We could certainly go ahead and copy this style into the footer and that would work. We can now see the footer is gray and surrounded with black. But this is duplicative. We have the same properties in two places, and instead of doing that, we can actually use a comma and say the's air, the rules that apply to both the header and the footer. In that case, we then no longer need this style and we can just pick which of the items here apply to both the header and the footer so this is now a rule for both. The trick here is that this rule is for everything that applies to both. We can still have a separate rule that only applies to one or the other two have properties that aren't common. So in this case will introduce two more properties padding, which is how much room from the side of the element to the element. So I'm going to give it a little bit more room to put some space between the text and the side of the element and border radius which is going to last around the corners of our border. In this case, I'll put a little circle around the corners. I'm just a footer is well, Azad, this padding, we go back to the website. We can see now there's more space in here and the corners or just a little rounded. So the rules still apply for header in footer for the background and the border. We've just added some other properties, their specific to just the border. And you see that this is not uncommon inside of writing your CSS style sheets, you're going to see that they're gonna be some cases where you can batch together, we're reuse without duplication. One rule as it applies to a number of elements and then have exceptional cases. So it ends up. Happening at the end of the day is that the footer element gets both these properties and these properties. So CSS reads all the rules and then figures out what properties apply to each element by just doing the logic. So now we've talked about named elements like Header Footer, Body, et cetera. Let's show you some new ways to indicate what elements you're going to style.

## ID Selectors

[Autogenerated] Now let's talk about selectors based on the idea of an element. When we look at an individual rule, let's take the body. For instance, the part before the curly braces is called the selector. The selectors in tax is what decides what the rules are going to be applied to. So far, all we've done is used the name of an element. But we can also use a new property I'm gonna introduce called I d. Let's take our section here and say I d equals main. So we're gonna identify this section. Is Maine this way? If we have other section elements, weaken really specify that it's on Lee for this one instance, I de specified to single instance inside of an entire HTML page needs to resolve toe a single element on a page in order to style it. We used the pound and then the name in this case, Maine, to specify the selector for that element. And in our case, we're gonna take that main section and change some of the properties. So first, let's set up a border like we've done before. So be a solid border again one pixel. But instead of picking a named color like black or like grey. Let's go ahead and use a pound Syntex, and when you start a color with a pound, it's going to indicate the red blue green values for the color. This will you have a fuller palette of colors you can choose from after the pound value. There could be either 32 digit Hexi decimal numbers or 31 digit Hexi decimal numbers, and these will specify what the exact color is. In our case, we can say See, see, see again. These values are going to be red, green and blue, and it will pick a color based on that. In this case, because they're Hexi decimal numbers, this is going to be a fairly light gray as well. We're also going to pick a border radius around those corners. Another property we can use is called color colors, going to specify the color of the actual text inside this element that because their background is going to be white, unlike the header in the photo, we probably want a color that's going to show up pretty well by default. So far, it's been black, but I'd like to make things a little lighter than that. So I'm going to specify another pound value in just say, 2020 20 as a dark color. Again, all zeros will result in appear black. All F f f f f F, which is 255 155 155 for each of the red, blue and green values will result in white. So having a red green blue that are all the same is going to give you a color that is some black and some white. In this case, it will be a fairly dark gray. Let's look at this in the browser so we can see we now have a fairly light border around it and that the text over here is just a tiny bit lighter than the black we see appear. It's actually a little hard to see. Let's make a quick chain so you can see it a little better. We change this to F f Oh, that's actually gonna be pure red f f, indicating the hex of decimal number of 2 55 for red, and then no values for green or blue in the same way we could do the same for a mix of green and blue, but in our case will go ahead and return back to that dark gray color is that will look best for our design. Of course, the problem here is that we're stuck next to the header in the footer so you can't really see the border very effectively. So let's go ahead and add a margin. And this is going to be the space between our element, including the border and the edges of the next element. This can take the form of a single number, like five pixels. In that case, it will bring the entire element in five pixels from the top, the bottom of the left and the right. Instead of specifying one value, we can pick individual numbers for each of those values by specifying four numbers. The way this works in CSS, all these numbered values start with the top, go to the right out of the bottom and then go to the left. It's going clockwise around the circle of values and always starts with the top, so we can see now that the top is five, the writers and the bottoms five. And then there's zero at the left hand side in our case, actually want the top and bottom to be about 20. But I want the left and the right to be zero and in fact, zero pixels. We can just say zero. That's about the look. I wanted one, a little space between the header and the footer for us. But because this is duplicative, I can actually also give it to values. And in the two value form, the first value will indicate the top and bottom and the second value will be the left and the right. So that shouldn't change the way it looks because that's in fact, what we're looking for. We're looking for an example where the margin is 20 top and bottom and then zero left and right. And lastly, we want some space between our text and the body of the elements. So let's go ahead and give it a small amount of padding. Patting uses the same rules in that it can be 12 or four values here. In fact, three of supporters well, but we won't really talk about that, but in our case, I just want five pixels to just separate us from the rest of the element. Now we can see this is starting to look a little bit more like what we wanted to.

## Relative Selectors

[Autogenerated] next, let's talk about relative selectors back on our page. We have the main element that would specify the I D. For once, I have a form element that's inside of it, and then inside of it are a number of labels. And we could go ahead and specify ideas for all the labels so that we could then say Search label on the great, another one for laying label, et cetera. But using ideas for everything can get a little laborers, especially. There's some commonality like there is with the forms. So when we we can do this is to use a selector that is positional. So we can say that we want to style the things that remain and then use the greater than to say style things in Maine who are a child called form or also a child called Label. Again, this is mimicking this hierarchy here main form label. Let's go ahead and put fun, Wait, which is another property? We haven't talked about it. Let's go ahead and make it bold so that all the labels are going to now be bold, so that matches it. But it's not actually necessary. Using these greater than symbols can be a bit fragile because, of course, what were to happen in the HT. Mao. If we were to add another layer here, maybe we put this in its own Dev or we moved into another part of the main section.

Having that direct walked down the Dom hierarchy would be a bit fragile. And so the solution there is actually say, Hey, in our case, we're goingto specify that any label anywhere inside of the main tag, we're going to specify with this set of rules Now, this may look similar to what we did here for had earned footer, but this comma is so important. This common indicated the separator between selectors and that the rules have played it all. The selectors in the heading here was down here without that comma we're seeing label that lives somewhere inside this main element. Let's do the same thing for the input elements. Let's do main input on Let's specify its with at 150. This is another property we haven't talked about, but unsurprisingly, probably the you. There's a property called with where we can specify how why these individual elements are, and this will match anything that is an input element. So the search raised the use stars and lane choice. We look now in the browser, we're going to see that this search phrases now this wide but so is the check box, which just looks weird. You don't really want both of them to be that wide. And the select here wasn't included in that with. So how do we address that? We address that by being more specific. So here, like we did earlier, we could include multiple types like, let's go ahead and put our select in here as well. So all of the inputs and all the selects inside of our main will be 150 pixels wide. You know, that's a little better, but we still have this really wide check box. How do we tell it not to use the check box? If you remember, I told you that rules air matched by specificity, how specific it is. And so if we look at our inputs, we have a type declaration here for check box. It's not easy to have a negative match, and in fact, in most cases you're not gonna want to do that. So instead of doing that, let's go ahead and put the type of text here that we can actually match. Input that our text maketh um, this wide as well selects and make them this wide. So we change that. We can say type equals text. And now it will on Lee match for the search phrase and the language. But leave our check box alone. In many cases, we might have a number of these in here like Maine and put type equals password and even main text area because those were all input types that need to usually look in a form to be about the same with. And so we accomplish that by using multiple selectors with one set of properties that to find them. So if we go back to the index at H E mail, the only other one we want to really look at is this type submit, because we've now on. Lee specified the input types were going to change the width of we might want to include a new rule for the submit button itself. In this case will use a number of properties that we've seen already to change the look of the submit button. There's something more pleasing than a square gray box. Gonna have a search button that looks a little more slick. So next let's talk about a selector that can match an arbitrary number of elements.

## Class Selectors

[Autogenerated] So now let's talk about class electors back in her H C mail. We've got an image here that contains our logo. Weaken set a new property on that called class, and the idea behind class is a named matching for some styling rules. It we used for some other things, but they're mostly for styling rules, and this week we can arbitrarily attach a class that's going to match formatting instructions. So let's call this class bordered image. So we want to set up a rule inside of our site, CSS, that says, If an element has the class of bordered image, let's go ahead and give it a border. Let's drop it and appear for no apparent reason bordered image and the way that the selector for classes works is it start to the period and then the name of the class. In this case, I'll say border solid one pixel and let's make it a really dark gray. Let's go ahead and give it a border radius, too, while we're at it. If we look at this in the browser, we can now see that it has a border because we've applied this border to the border image. They were not specifying that this glass has toe on Lee be against the image. We've only named it that we came down here into the index that html for the form and we applied the same class, even though it's called bordered image, probably a bad name in this case. We should now see that same border around the entire form so you can think about this is ways to take simple you. Why decisions like adding a border centering formatting, specifying a farm to those sorts of things as classes of elements. And in this case, we're specifying that a class has a border and we can apply it to which ever elements on the page makes sense to have those such of attributes. It's all additives. So in the case of the form, it'll have a formal here, added additional attributes. This class will be added to the specific properties for the element, so we will get a set of properties that is an an operation against all those properties in the same way we can have. A class in our case will call it simple form that applies to the form so that we can do the formatting for our form elements that are based on the forms class instead of its location inside of the main. So let's assume that in our style she, instead of us wanting all the labels and sizes of the inputs and even the color of the submit button to be based on whether it was inside the main element, we could do the same thing here with simple form. It's still based on the same formula labels inside of the element that's labeled with the class simple form in the text inputs inside of simple form. This admit ________ inside of some performs etcetera. So works in the same way. We're just specifying the container above these inputs and labels and selects and text areas is going to have the class of simple form. This is useful, especially in forms when you have maybe two or three different styles of forms across your Web page. One thing that might not be super obvious, but I'll make it obvious year is that a class could be multi valued. In this case, we have bordered image as a class as well. A simple form is a class. This does not have to be a single valued property. It will take all the classes that it matches and include all of them in the styling for that element in this case, the form. Because we've done this, we can also add a simple form set of attributes for the form itself. So in this case, let's go ahead and add a padding that a specific to the simple form not specific to the bordered image class we used before. It will now see the form has a little space between E and the edge of the form itself. That padding says that all my Children are going to be spaced out a little bit inside of myself. I'm going to give myself, in essence and in your margin if you want to think about it. That way, we will notice that the form here now has a mix. It has the border and rounded corners from the bordered image class, as well as the padding from the simple form class. You can really separate your individual rules based on the functionality they're going to provide to the rendering mechanism

## Sizing and the Box Model

[Autogenerated] So let's talk a little bit about sizing in CSS. We've introduced some different properties like with, and you can probably assume that there's also a height padding and margin and border all make up some part of the sizing story, but I think it's important to understand the CSS box model so that as you're putting these together, you understand where your actual heightened with they're coming from It all starts with your content. You're content is inside of what is called the padding. The padding is inside the border, and then the border is inside the margin. There's also a left, a right, a top and a bottom that can also be specified to move it around the page, and we'll show you some examples of that in a little bit. The height you specify, is really the height of the content alone, not as it relates to adding the border, the padding, the margin and then whether it's top left redder bottom. The with is done the same way. It's for the content alone. There's two more properties, actual with an actual height that can be read at runtime, and we'll see how to do that, a little later, but this represents the actual within the actual height, including the margin, the border and the padding. So there is a disconnect between the way they handle heightened with of the content and actual heighten with which are going to include these other elements. So if you need something to be exactly 200 wide, you have to make sure that you're with plus your padding, Pleasure, border and pleasure margin are going to equal to 100 by specifying the content of 200 in the padding of five, the border of one and the margin of five. You're going to end up with something that is 200 plus five plus one plus five or 211 wide, where your actual with Let's see this in practice. So we go back to our simple form here. Let's go ahead and add of with to it, let's say a with of 300 we can now see that the width of the form is this wide, but the width of the content of the form is going to be 300 wide. But by adding the padding here and the border, Texas is going to be 306 wide. The reason is we have a padding of two in a border of one. So that's 300 plus two, twice once for the left, one for the right, and then the border of one twice, which is another two that makes three or six. And so you're gonna have to take all these different properties margin, border and padding into account. When you're going to specify the size of an element that need to be an exact size, that could be an issue. We'll come back to this exact size in just a minute. Well, let's talk about a common approach of actually wrapping all these pieces so that they aren't the with of our page. You probably noticed that most websites out there are actually centered on the page and let me show you how that works. We come here into the HT male, and I'm gonna collapse these three sections so we can see them all in a page. I'm gonna wrap these in a new div. We can now see that this div was now going to contain all three sections. Good to go ahead and give this debate class old container. Let's create that container class rule container like we've seen before. But in this case, we're going to specify a standard with, and I'm gonna use 989 pixels. This is a pretty common with that works well on all monitor sizes. This really accounts for all of your content to be contained on a monitor 10 24 by 7 68 And then it just centered in all monitors that are larger than that. When you're dealing with mobile science rousers, this is a little different than if you view the floor. Oh, site course on mobile Web development. You see some techniques there for dealing with different screen sizes. But for now, we're just going to go ahead and pick container as 9 89 So far on the course, I've actually had our Web page zoomed in so we could see things a little better. But look to go back to 100% magnification, and we can now see that the whole page is this size, but we'd like to be able to center it in some way. And so there's a little trick here. We could say text a line center, but that's actually not going to center. The debate is going to just send your text. Instead, we're gonna use margin. Market has another value we haven't talked about and that is auto. Remember I said that multi valued margin and padding can both take 12 or four values and our keys. We don't want any margin on the top of the bottom, so we get a zero for the top in the bottom, over the left and the right. We're going to tell it to use a margin of auto. This is a special value that says make the margins the same, but in essence, center them. So when we specify margin of zero and auto here, the container itself and again the container can't be seen. It's just blank at this point, is going to center this. Remember, the container is 9 89 wide, and by pushing a margin of auto, the two sides here get equal sized, so it ends up looking centered. Now we're starting to look a little bit like a lot of sites you've seen that are using the same technique. Next, let's give the container a background color on our case. Let's go ahead and pick dark gray. So when we say dark gray, what we're actually getting is just the container itself is going to be a dark gray until the all the elements here that don't have a specific background color. In that case, our body here are showing through. Of course, the header in the photo, which we specified a color for, are going to show there's on top of that light gray. Instead, we're gonna go ahead and specify a background of white because soon we're going to make the margins or outside of the container a different color instead, lastly, we're going to add a padding here so that there's some space between our container and our other elements. So now let's go back to that body tag and add that color. Let's say blue, just add a little color to our lives that when we see this, will see that there's blue everywhere. It's actually kind of a difficult blue. Let's go ahead and say, Dark green And here in the browser weaken. See, it's that dark green that sort of matches the other green that we're working with underpaid, so it mostly works, but we do have this little space on the top that I don't care for. And so how do we tell it that this container should be against the edges? The reason this is the case is that the HT mouth and the body both have their own margin and padding that aren't zero by default. So we have to add that we can do that with a combined tag to say h melt embody padding zero margin zero and that will, in essence, reset it. So that's now up against the edges where possible. In fact, many cases the starting point for your CSS are going to be those reset for body HTML as well as maybe some other standard TSS that use project on project. So let's talk about position next.

## Positioning in CSS

[Autogenerated] So we've talked about sizing. Let's talk about positioning in CSS. We take a look at her CSS and let's go to that simple form that we have on the page. We can see that there is a property called position and by default this is static and static essentially says, Please draw me in the order of the hierarchy, top to bottom, outer to inner so between, you naturally flow the hierarchy onto the page. By changing this, you can decide to position things in a couple of different ways. Let's change this to fixed. We can now see that the search form is here where we had it before. But it's not taking any Haider whiff because we're telling it that we want it to be fixed on a page in a very specific position. And in this case, it's still going to be drawn in the same place in the hierarchy. Were just telling if it has to be in this exact place, fix simply means fixed in this position, but still relative to the hierarchy. If we change this tow absolute were actually telling it, we want absolute positioning on the page Now. This is something you should use very sparingly. Some of you may be listening to this and go well. I've done Windows development or of Don Mac development, and I can absolutely position things exactly where I want them. This tends to break the Web a little, but there are occasions were absolute positioning, like a floating piece over your website could be pretty useful, and in these cases we need to specify top left, right or bottom to specify that. So if I say top equals five pixels and let's say left five pixels, this will be positioned exactly five from the left and five from the top of the entire window in the same way. If we did bottom and right, we're going to get the same behavior. But for something that may flute on the bottom right of the page again, using this sparingly is useful. But for things that you want to float over the pace that don't flow with the page, this could be pretty useful. Of course, for us. We don't need these all just comment him out so that if you have access to the source code, you can at least see what they were. Let's talk about how things were displayed are laid out on the page next

## Display in CSS

[Autogenerated] Let's talk about the display property now back under HTML. Let's create a new element that's right after this form New Dev and let's call it results. This will be where the results of our search will ultimately end up. We look at these in the browser, this will by default put this below the element before it not going to flow it to the right. Gonna float to the bottom. And let's talk about why that ISS, in the case of forms and dibs there actually buying default, have a property called display of block. So in our case, when we look at the different elements in HD mail, there are some elements that air block and some that are in line in some that are called in line block and there's a few other display properties. But those are the three that I want you to sort of focus on block in line and in line block sonar keys. What we'd like to do is take this whole section here and put it to the right so that our results are to the right of the search pain. We could do that with a little CSS. Now we've already made the width of the simple form a fix. I So let's go ahead and get it right next to it. But make sure it's nearby so we can look at both of them at the same time. Let's go ahead and add a padding of two and let's give it a with of let's say, 600 for now and over in the HTML, I'm just going to give it a class just like we did our form of bordered image again. Having the image on the back of this probably wasn't the greatest idea, and we could always fix that later. So at this point, it should look like this. We have our box, we have a 600 wide and 300 plus 600 should fit here, right? But because these they're both block level elements, they're defined as block and block means that they're going to be top to bottom stacked. Unless we do something different. The way to do something different is actually to change its display and in our case will change it to in line so you can see what the inline display looks like because we used in line it attempted to make it a small as possible, didn't really solve it, made each of the lines still where they were and then dropped this new element next to it just to the left. And that's because in line is more like the way we put in line pieces in text. That's what in line is four for the way that we flow elements within, Let's say, a textual representation, much like this anchor tag here that we have is in line with the text that is an inline displayed element by default on Anchor Tak. Instead, what we want is in line blocks, so we wanted to act like a block, which means mostly adhering to it with treating it like a rectangular element. But we still want them to be displayed in line, if possible. By doing this, both the elements air put left to right, and if we had three that all fit, they would show otherwise they would wrap onto in an individual line. Let me show you that if this was too long, let's say 700 because, of course, 700 plus 300 plus the associated patting on all that is greater than 8 98 pixel with it would just fall into the next line. So in line block is like the way that text works in. That'll allow line rapping in such of elements. And so when you see something like the Netflix queue, we have a bunch of little boxes that are all left to right, and then they wrap on the next line to next lines. Those are most likely in line block elements. But of course, our problem with just the ones that fit here is it isn't a line correctly cause by default. There's also another property called Vertical Line. This aligns things vertically and by default thes two elements are not aligned with their tops, so we have to add that so vile aligning both elements to the top. It now does what we want. Eventually, this will have a lot more information in the search. Pain will always be at the top, aligned with what the results are. Those results may be fairly tall. Once we get into pulling actual data from get hub, there's another way of putting things next to each other that flow in a different way. We'll talk about that next when we talk about float

## Float in CSS

[Autogenerated] So now let's talk about the float property and CSS. We look at our page here. If you remember, we have this navigation element has three different links for the pages. And what we might want to do is actually put these over here on the upper right hand corner of our page. And we can do this with float. Will introduce a couple of other pieces while we go at it. But the primary idea here is to investigate and understand Float. We look at the HT mail for this, we have a knave element that then has an UN ordered list in list items in it, which you probably remember when we built it earlier. So let's start down here. But creating a header knave rule. This way we can have a separate set of rules for the knave element that we might have in the Footer later. We could do this if we wanted to form at all the naps, the same. But our case I'm gonna leave it as editor naff So any knave element inside the header element, I'm just gonna say float right? No, flu has a couple of valid values, right and left are the two that are most likely used. So let's go ahead and say Right in this case, we can now see that the elements are now floated to the right, the entire knave elements that floated to the right. But of course, now that it's being floated, it's no longer taking up space in our header. One of the tricks here with flute is to change the order of the elements. We come here and take our knave element and put it before the others. That way it starts at the top of the head or value. It will now take space over here on the right hand side. But we'd really like these to be displayed next to each other so that if we have three or four or five, they'll just sort of sit on the right upper hand corner of the page, much like you see in a lot of pages. To accomplish this will need a new rule just for the l I or the list items. Now we could certainly have put in u L L I to be very specific, but in our case, we just want to permit all the list items in the same way, much like we saw display being used before. Let's use it again. But say that we want all of these items to be in line. So instead of them stacked block, which is the way that list items normally enlisted, we're gonna tell it to stack them next to each other. So now we're seeing them left to right, floated all the way to the right of the element. And so let's go and add a quick margin right of about five pixels just to keep it off the edge of the page. Now, what's interesting about this is you haven't seen this in tax before and that is margin, dash, right when you just want to override one of the margins or one of the pad ings, you can actually specify the cardinal location in this case, right? As an individual property. So here we're just gonna push it a bit from the right and last we will make it a bit smaller, and then this way, we've used Float to really decide where it's gonna be on a page. One thing you'll notice is that the flow is stacking things either in the left or the right hand side. We have multiple things floated. They're going to stack in that order from left to right or right to left, depending on whether you're using float Writer left. But float doesn't account for space. Float isn't used to compute the size of the container. Do you have to make sure that whatever you're floating is going to exist in that container and be correct? You may even have to size that container to be in large enough to hold the content, because the browsers themselves aren't going to resize the container for anything that has floated. Last thing will do a style. The links Now here these, even though the text is all black, are colored purple for visited pages and then blew for unvisited pages. That's because these anchor tags and even this one and get hub are formatted special, using their own rules. So we come back up to the CSS. We can actually form at ease by saying anchor, which is the A tag. Let's say color equals green. To match our theme, we refresh. It will see that we now have green for all our anchor tanks. Now that we form out of them is green. We've lost the ability to specify whether they've been visited or not, but most like to no longer worry about that. They'll just color the anchor tags to fit the theme of their project, and then they may remove her at the Underline where possible. You do wanna have some indication that these air special in clickable It may not be simply that it's an underlying. It may be a bolding or metallic or, as you have her over, something that may change the look. All those air possible with CSS. But now we have a site that has been style pretty much in the way we want. Let's next talk about how to debug these, using some of the browser tools because there's gonna be times when you think you've got the CSS right. But the browser just isn't displaying it correctly.

## Debugging Styling in the Browser

[Autogenerated] So now let's debug styling inside the browser Come back door page in Firefox. In this case, we have a plug in installed called firebug. All the browsers have their own built in tooling. I like to use fire Fox specifically because the firebug tool, they think is a little bit more powerful than the built in tools on the other browsers. But if you want to get used to using the crone tools, using the eye, any tools or using the Firefox tools, probably a good idea to get a little familiar with all of them were not gonna cover all of them in this course. But they all work in a pretty similar way. In most cases, when you hit F 12 the consul for the tools will show up. In this case, these air, the firebug tools when were debugging CSS. What we really want to look at is what the CSS is for certain elements. So in our case, we can go to the HC male tab, and as we open up the file that we're looking at in hover over elements, you'll actually see them highlighted in the browser. These highlights are showing in blue the content size and then in yellow, the margin size you see that's going to be different for different elements. Let's go down to the header section as well, or the main section, though the main section. You can actually see all three. The blue is the content, the purple is the padding, and then the yellow is those margins we had specified by clicking on one. And let's go ahead and click on our form class. We can see to the right what properties have been specified. So this is showing all the matching styles, the simple form style, the bordered image style, the main style. And then finally the body style. All of these air applied to this element in order to get it to look the way it does. What the tools allow us to do is actually to change this stuff, and we'll see what the result is. This isn't changing the underlying file. This is just changing the live you. So we looked at our form and we go to padding. We could just put in another value like 15 pixels, and we could see it change on the screen. Or, let's say back to zero we can also anywhere there's numbers, use the arrow keys to just sort of go up and down so we can tweak thes to get the look we want. The same is for non numeric value. So if we look at, let's say a vertical line as we push the up or down arrow, it's gonna go through all of the valid values and change them in the browser. Now, in this case, vertical alignment doesn't change any of those values. But if we did it for display, we can see this is what it looks like inline flex block, etcetera. And so we can kind of experiment and learn with the different values for these properties are. Once you've changed these values to the way you want, you can actually right click and copy of the entire rule. And this is a good way to take a rule that your modifying in the browser and go ahead and paste it back into your original source code. Another thing that this is important in seeing and that is the Cascade rules. So I want select an element on the page. I can actually use this inspector and the different tools Call it something different when this is where I can actually go and select with the mouse and individual element I want to look for. You can see it sort of highlights it as I hover over it and I'm gonna go down to the L I items inside of my menu. And if I click on the ally, we can see that this header knave allies what we create in the last section. But because we specified the font size, the font size for the body has been checked out. That means that this more specific rule has overridden this rule, and so we can actually see the cascading nature of CSS here in person. As you're testing these things out, you can actually click on this hidden checkbox. It doesn't show up until you actually hover in order to disable one of the rules. Now, instead of having to delete them, you can just disable them for a minute so you can see the difference between them. And if you're having a problem with an individual style being able to start disabling pieces to see which one was the one that was actually involved and help you sort of narrow down Which one's the problem and noticed that when I disable the font size in list item that this one becomes un selected because this becomes the default value now because the body no longer is overridden. So it really applies the entire set of rules to your CSS in live, and this is really a live view of them. And so using the tools when you're trying to debug CSS, I think, is a great way to not only find and solve problems but also learn how CSS really works. Because in most of the tools, you're even going to get Intel ascent. So if I decide that I want to have border here as a type, B O would start to show border and, as a type even go down to some of the other properties. So it gives me a lot of flexibility about sort of running through and testing May CSS. I suggest you use the tools again. Whatever browser used for development, you can use their tools, but eventually you're gonna probably need to learn the tools for each of the browsers because you may find that some of the problems you're having with CSS, our browser specific. So being able to go in tow i e. And finding out why something doesn't look right. Naive, but looks finding chrome and fire fox or vice versa means you're gonna have to learn the tools for each of the browsers. They're very similar, so it's not as big of a task as you might imagine. Let's wrap up this module.

## Summary

[Autogenerated] So in wrapping up this module learned the HTML itself a structural. You can create the look and feel the rap through the styling stack. You can define the you I the formatting of each element, the formatting of the text, the look and feel of your application strictly through the styling of Europe. Leaving HTML a structural is possible means that you're going to have a separation between what you're dealing with as the structure of each to mouth and how it looks by using CSS or cascading style sheets. You're goingto have a way to support the shared over rideable styles for your Web pages and really your entire website. Each of the rules and CSS has a set of properties that are applying to a certain set of elements. In this way, those set of elements can be defined as a selector in the CSS to tell the browser how to draw your HTML. Understanding the nature of the CSS selectors is key to being able to style your site. So learning the different mechanics of I D. Selectors of class electors of elements, electors of relative selectors are all gonna benefit the CSS. You're going to write. It's also pretty critical that you learn and understand the box model is. We have described it in CSS so that you can really understand how the sizing and layout mechanics of CSS work with HTML. We've also seen that using cascading model of having more specific selectors apply their rules to individual items. On the page is how the cascading allows for this override ability. Thes rules for cascading aren't always intuitive, but certainly we have seen that using the browser tools, we can see how that cascading is actually happening. Ultimately, you will find times when you need to drop down in the bug. Your styling, using the Brower's or tools and assigned from being able to debug these problems will also be a very effective tool for learning CSS itself. This has been the styling your H E mail module. My name is Sean Wildermuth of Wilder Minds. Thanks for joining me

# JavaScript

## Introduction

[Autogenerated] in this next module. Let's talk about JavaScript. My name is Shawn. Will move of wilder minds. We'll start by discussing what is JavaScript. I'll show you how to include JavaScript. Start with some of the basics of JavaScript. Look at variables and types, condition, ALS functions, scopes, closures, objects in a raise and finally, looping. Let's get started.

## What Is JavaScript?

[Autogenerated] So what is JavaScript? The elevator pitch says it's an object during two dynamic language that has become the lingua franca of websites but isn't limited to its use on Web pages. But that really means is it's a very popular language right now, started with humble beginnings, is just a scripting language for websites. But it's certainly matured into a pretty robust language that is used to power websites but also back ends these days. It's also being used to automate a number of different types of systems. JavaScript, one of those languages that you really should have in your arsenal. But I found when talking to new Web developers, JavaScript gets sort of a bad reputation. It does have humble beginnings and some decisions that made early on certainly hurt its reputation. You can do JavaScript and do JavaScript really well, and the trick is to learning its patterns. Don't try to take existing language is that you're working with and try to shoehorn your knowledge of those languages into JavaScript, learn JavaScript for what Javascript really is. So first and foremost, JavaScript is object oriented. Sometimes developers don't want to think about JavaScript as being object oriented because it doesn't have classes in the traditional sense.

It uses prototypical inheritance instead of class based inheritance. It's also a dynamic language, which means it's often thought of as a type lis language. But instead it's a dynamically type language. This means types can change and variables of certain types can change their types. You can opt into some level of type checking as well, or used a higher level language like typescript or coffee script to get the sort of type checking you might want. JavaScript is often admonished for the lack of compilation. If you come from a high level language like C Sharp or Java, you may be used to a compilation step where things air checked like type checking but also package. So you're not delivering pure code. JavaScript is purely considered an interpretive language. But in the case of most instances, in most modern browsers and certainly on the server, JavaScript is just in time compiled like most other high performing languages. In fact, languages like C Sharp in Java when they are compiled their compiled into an intermediate language, that is then, just in time compiled. JavaScript sort of skips that in JavaScript becomes that intermediate language that is then red and then just in time compiled as necessary. So this performance characteristics in certain cases could be Justus high as other high level languages. Enough shop talk, let's see in action on a website.

## Hello JavaScript

[Autogenerated] So let's go ahead and write our first lines of JavaScript on a Web page. We look at the index that HTML from our previous modules. Let's go ahead and create a new tag called Script. Script is normally used to load JavaScript, but it could be used for other languages. So we need to tell script that its type is going to be text slash JavaScript. That's its mind type. And then we can start to write script directly here on the page, a function that is built into the job script language, though you probably should never use in. Production is called alert, and this is just going toe lost a pop up, a dialogue on the screen, and I'll just use it to pop up the phrase Hello, JavaScript. Let's see what this looks like in the browser. We refresh. We can see. Here's the alert dialogue that was popped up with the text we typed into it. We're actually running JavaScript. When we launched this page, let's go back to the page and here, instead of using simple alert, we'll use a keyword in JavaScript called Ivar. Far is allowing us to create a new variable that we want to hold some amount of data. The state. It could be a number a date, a string or more complex types that will look at later. In our case, I'm gonna have a hold of string. So I'm gonna call it a message. And I'm just gonna move this hello, JavaScript into our new variable. And then I can use that variable in a number of places in this case will use for the alert. You go back to the browser. Unsurprisingly, we see the same code, but usually we don't want to use alert instead, if we want to sort of run out debug messages we can use. A object that exists in JavaScript called console console is actually the debug console that's out there. And let's just write out to its log the phrase message back here in the browser. If we use the browser tools which we talked about at the end of last module, we can look at the console. And when we refresh, we can see that message that we spit out. Hello. Javascript here in the console will use the console a little bit in the next few video clips to test some Java script ideas. Let's close that for now. And we look at the HT mail here on the page, we could see that we have several parts of the page that we can interact with down here. There is a day of we created earlier called results and Results just contains some text. We can manipulate that with the HT mouth. We can do that by trying to get a reference to that element. And let's call that the results Div and we can call document which represents the Dom or the hierarchy of elements on the page. And there's a method called Get element by I d. And here we're gonna give it the idea of some object on the page. We want we go down to the bottom again. You remember that the i d for this guy was results. So we come in here and we say results will have gotten a reference to the item on the page and then we can simply use several types of properties that exist to change the contents of that Div. In our case, we're going to say results Div the Inner eight email and here we're gonna just supply the H mouth is going to be part of that page. I'll make a little paragraph and say this is from Java script. We go back here to the browser and refresh. We should see that this should say this is from Java script, but it doesn't. And why is that? The reason isn't all that obvious. The problem is that when we run this code here, remember, we're running this code. Just spit it out to the console and it seems to work fine, but this code isn't running correctly. And why is that? One of the things we're doing here is calling the document. Get element by I d to look at something in the document. But this is actually being loaded and executed as soon as possible and because it is at the top of the page is being executed early. So because of this, we usually take script blocks and put them at the bottom of the page so that they're all the elements on the page have been loaded already. We'll look at some different techniques for dealing with this, But for now, putting them in on the bottom of the page is the right thing to do, so I'm going to cut them from the head and put them just inside the bottom of the body. Usually where I put all my script tags. No, I haven't changed the code here. I've simply just moved it to the bottom of the page. We can now see that it's displaying and running. That could, correctly to show it from the Java script again. That's because all of this stuff is being read by the browser, including the script, and the script is trying to execute as soon as it has been read. But if all the elements on the page haven't been drawn yet or haven't been loaded yet, it's going to beat the browser to looking up within the document before it's had a chance to actually load all those elements. Putting up at the bottom of the page will help us. We'll look at some other techniques in the next module for getting around that for guaranteeing that everything's gonna be loaded. Before we were in some code, we go back to the code for a minute. You'll see that in this case we have some inland code directly in the H E mail. And much like the way that C S s shouldn't be using in line styles, you probably shouldn't be using any inline JavaScript either. So let's do that next.

## Including JavaScript

[Autogenerated] So now let's talk about how to include JavaScript on a Web page without in lining it as you saw in the last clip. Our script code here being in line on the page is interesting, but it makes it a little harder to do things like running tests and to reuse this code and all those sorts of things. And so much like the way we break out CSS we normally break out our JavaScript as well. So in this case, I'm gonna create a new file and folder for JavaScript. And I'm just going to call this our index dot Js though you could call it whatever you want. That Js is the extension for all JavaScript files and depending on whatever you're using for your web development that that J s will guarantee that you get some syntax hollering in that sort of thing. I'm gonna start with a comment and that uses C style comment. So double slash or slash asterisk and ending with the Astra Slash They both work, and at the top, I'm just going to give it the name so we could see pretty clearly when we opened a file that we have the correct file opened, and I'm gonna go grab the contents of this JavaScript as is and move them into the index that Js what we're going to do here instead of having the script in line as we're gonna use a new attribute called Source. This is where we can point it at the file. We want a load. This is how you load external JavaScript. In our case, we're loading some job script, that relative to the page until we can go ahead and include the directory name here and then the file name. But this also could be a fully qualified Web euro. So I could be going out to http colon slash josh cdn dot google dot com and then the name of some common library out there like Jake Warrior angular Js. So you can use this to pull in both first party code, your writing and third party code that you're using on your website. That's all that's needed to bring it in. One thing you may be tempted to do is going ahead and ending it prematurely like that, but because it is HTML, there certain tags that need to have a beginning and ending tag. That script is one of those. This is a mistake you'll invariably make a couple of times until you really get ingrained with making sure the script has an ending tag and not a self closing tag. We look at the same code and just go over to the browser and refresh again. We'll see. The same code is executed. There's our this is from Java script, but in this case it's actually being loaded from an external JavaScript file. We can see this in the browser tools, but going to the script tag, we can see that there's now one script that's actually being loaded for our page and actually look at it. Let's start looking at the language itself so we can see some of the basics of the language.

## Variables and Types

[Autogenerated] Let's now take a quick look at variables in types. We looked at the code we've written here. We could see that we have a couple of different variables that we've already declared. We can actually look at the type of both of these by using a function in the language called Type of. So let's go ahead and write to the console message is and just let's use the ad to contaminate a couple of strings together and look at the type of our message type. Not through the same thing. Whether results steps so we can just start to understand the type system in JavaScript. I'm doing this using counsel that logs will have to look at the council in the browser. So if we refresh the page, we can now say that message is a string. So the message variable is of type string, and the results did is actually a div inside the document. But the types information is only going to tell us it's an object. Therefore, it's not a primitive type. Look to look at a couple more of these. Let's create some new variables and let me create a variable that hasn't been created yet I'll just call it none where I'm not actually assigning anything to it. And I'll do the same thing I did with the other types here. And I'll create a new variable for a number. And let's just say zero and let's look at a 1,000,000,000 type like true false. Now there's a handful of these primitive types as well as complex types out there. So I'm not gonna cover the breath of every type here. But I just want to give you a taste of the different types so you can see that that message is still of type String results Davis still of type object and that a number here is of type number and true false is Boolean all that's pretty much what you would expect. The type that is confusing here is none because it says none is of type undefined. This is actually something we contest for. Undefined is a special key word in JavaScript cheating test to see whether variable has ever been assigned a value, therefore has a type so undefined ends up being one of the types out there. There's something you'll end up needing here and again. You can imagine it tested for things like whether passing variables have an initial ice turn up. One thing to be aware of about JavaScript by default is it doesn't actually need variables to be defined before they're used. That may seem a little odd, but let me give you an example so I could come down here and a sign and new value to a variable with a name that we've never used before. Let's say non existent was a variable name we wanted, and I could just say this shouldn't work now. The problem here, let's go back to the browser and refresh is if we look at the non existent type the console, let's just type on the bottom of screen to write ad hoc JavaScript. You'll see that non existent, the value of it is. This shouldn't work. Most languages force you to have declare variables in our case with the bar before using them. But because of what many think is an oversight with the original design of the JavaScript language, variables don't need to be defined first. The problem this causes is of what if I didn't really want to use a brand new variable, but I simply misspelled it. So what if I said messages equals shouldn't work. Now here's Ah, variable. I think I'm redefining, but I misspelled it. I added a nest iMessage instead of messages, and JavaScript will just go about its way and go. Wow, that's fine. We'll create a new variable called messages, and I'll spend hours debunking this code, trying to figure out why a signing this never gets changed up here and why we have this sort of hanging new variable that was just created by accident. The way to solve this is that newer versions of JavaScript Act um, a three and beyond, which is these days, a pretty old version of JavaScript. Most browsers supported least Atmos script three version of Java script. You can put a little string at the top called Used Strict. And what this tells the compiler is to use some stricter rules when you're parsing this JavaScript. And one of those stricter rules is don't allow on defined variables. Now this used strict is something I'd like to encourage people to use, but it is just a string. Why is it just a string? It's just a string because an older browsers or on devices that don't support later versions of Java script. This is just a string, and it is perfectly valid JavaScript and will be completely ignored. It has no side effects, but a newer versions of the JavaScript compilers. They'll see this and go, Oh, we need to be a little bit more strict in how we interpret this JavaScript. Let's see what happens when we leave this code, in our example, actually get a reference air and an air is thrown by Java script that says Assignment to an undeclared variable messages. And that's because strict is on so we can now go. Oh, this is no longer valid JavaScript. So we need to go ahead and comment that out or delete it whatever you see fit to do in this case. But it's a piece of error code so that we can actually see what's going on and see what's wrong. So adding use script to most of your projects is not a bad idea. Let's move on to condition ALS

## Conditionals

[Autogenerated] So now let's talk about conditional is and how comparison works. The last clip I talked about how this nun variable was of type undefined. This is a pretty common comparison you're going to do so we can use the if constructing JavaScript to test for something. And if whatever is in the parentheses is true, it will execute what is inside of the curly braces. This is C style. It's coming to see sharp as well. And then, of course, you could have else. Or else if with another conditional in it. So these air pretty standard comparison blocks. It's what happens inside. Get rid of the else F. For now, it's what happens on the inside of here that makes JavaScript a little special. So if we go ahead and say if none is equal to undefined consul dot log, none is undefined. We can actually test, and you'll see the undefined us putting a special color here because it is a job script reserved key word. It has a special meaning, and that is that this has never been defined. As we can see here, the very well exists so we can use it, but it's never been assigned a value it's been under defined as a value here. We can see that that conditional, in fact, did what we expected. But JavaScript condition ALS have a special behavior here, and that is all the types are compared together. Let me give you an example, and then we'll come back toe this example. In a minute. Let's say that my a number variable waas the number 10 so it's still a number. And you would expect that if I said if a number equals 10 then we could just right out to the console, right? And you would expect us to run tennis. 10. Well, what happens if we decide to say what if 10. Remember, this is a number not a string is equal to this string of 10 should be equal in JavaScript. It is. And the reason is that when you do comparisons, all the types are coerced to ones that can be compared. So in this case, it's taking this number and saying, Oh, you're trying to compare it to a string. Therefore, I'm going to convert this to a string and then do the test. The double equal sign in a comparison with exclamation equal sign for a negative comparison. They both use this coercion, so you can assume that it's going to try to find types that are actually comparable in this case because they're doing a comparison. They're not doing inequality test. So how do you change that? You do that with 1/3 equal or, in the case of a negative test, a not equal equal. Now, if we do this, what's called an exact comparison we can see the 10 equals 10 does not show up because these are not exactly equal or identically equal his core. Jin has some benefits, though. So if I wanted to say if not none, which is going to end up be the equivalent of saying, If non is equal to undefined, I can actually just use this Boolean expression because it's going to take this undefined value and converted into a Boolean, since it looks like that's what I want to dio. But this is the same as saying none does not equal true right, putting the exclamation point before it is going to allow us to simply say, if that is not a true value, then show it None is undefined. We can see that is still happening in our job script. So this type coercion becomes important as you do tests. So knowing when you want to actually test whether the values are compatible or whether the values are identically equal, you're going to need to be aware of using the double equals or the triple equals. And, of course, the single equals is just for assignment. And so if you had tried to do this, it would work because it would simply assign this to the variable of a number. And because this assignment worked, it would return true to try to avoid single equals inside your if in all cases. Now that we've looked at some basics around types, in comparison, let's look at functions.

## Functions

[Autogenerated] next. Let's look at functions, so function starts with the keyword function and we'll go ahead and give it a name. In this case, it will be showed message. A function has set a parentheses in which you can have function parameters and then a set of curly braces, just like in C Sharp and C plus plus and see that designates the body of the function in our case will just accept one parameter, which will be our message. And then just use the consul dot log to go ahead and dump it into the console. So it's not a very interesting function at this point. Once we've defined it, we can just call show message and then supply some information. The set of parameters in this case we're sending in a string. So if we look at the function, it's a very simple concept. It's some repeatable code, something we're going to call over and over again. You've dealt with other languages. That should be a fairly simple construct. One of the problems that you're gonna haven't function. So is the problem of overloading, and the reason overloading is a problem is that when you think about most languages out there. They have the same concept of having a function that may have one or more parameters, in fact, having different versions of it. Someone give you an example of this so I may have another function with the same name. But let's say I want to take two parameters instead, I'll do what I did before and I put a little plus there so we can tell the difference between them And just in Canton ate all three pieces together, and I'll call it this time with two pieces of information. So you were dealing with a language that wasn't JavaScript. You might think that the first of these was going to call this version of the function with one parameter and that the second recall the other. But if we go over to the browser, we'll see that both versions of show Message used the second copy of the function. The reason for this is that there can only be one function with each name. If we want to support something like additional parameters, there's a few different ways to handle it, but overloading the function, having two versions of the function that except different parameters is not one of them, because in this case, by defining show message again, all we did was overwrite the show message, a common way to handle this as actually to use the test for undefined. So if more was defined or provided, you want to think about it in those terms and we'll go ahead and show the message and include the more. But if one of the cases didn't include a, more like, our first version will just show the message itself. So it's up to you to handle the multiple parameters, and when parameters go missing, we refresh it when concede. We're now getting that second version of show message, but now it's doing the right thing, showing the first message and then the additional second message if one was supplied. So we'll weaken. Define named functions like we have here. We don't always have to great functions in this way. Functions are first class citizens of JavaScript or, in fact, a different and special data type called a function type, so that we can do things like this. So in this case, we're defining a function as just a variable on, of course, toe execute it we can just call Show it. So the difference in Texas here aren't all that important. There's a little different here in the show. Message is going to be available for any code after it's been defined here. The same with the variable here. But with the variable we could more easily or more obviously pass it as a parameter. For instance, Let me give you an example of that. Let's create a new function called Show It. Then, in this case will accept to different parameters a message to show and then something to do after it. And I'll call that something a call back this case. I'm just going to use that, show it and say, show the message and then all back parentheses. So this callback is a function that's being passed into us. This function that's being passed into us then convey executed with or without parameters. In this case, I'm not going to include parameters, but I could certainly send in message or even some hard coded set of parameters as well when calling it a very common pattern in JavaScript is to include a function as an inline parameter so here will just say function again. Notice it's an anonymous function. We don't have a name for it that we could have simply put one of the function names here. But in this case, we want to define that something's going to be done once our message has been shown and we'll just say all back called just to be easy. So this is not defining a function here. This is calling a function dysfunction above it. But we're supplying our call back as a parameter to that function is an important contract because it comes up a lot. Who's JavaScript has a lot in it that ends up being a synchronous things like registering for events, making network requests, et cetera. Those all expect you to be able to start execution of something and then wait for something to happen, like the user click a button or the network requests complete. We look at this in the browser. We'll see that they show it was called. And then once the show, it was called with some data. That callback was then executed. So when we called this, it took this message his first parameter and called Show It in order to spit it out into the console and then executed. This callback on this callback was simply this function that we had defined to say. Here's the thing I wanted to do And this is very much related to the way that things like Lamb does work. Lamb doesn't see sharp and later versions of Java in Ruby, etcetera. A lot of other languages have this contract. You just need to think of this as just a data type that has colorable functional code in it. The next module. When we start to look at how event handling networking is done with J. Query, will see this pattern quite a lot.

## Scopes

[Autogenerated] Now that we have a basic understanding of functions, let's take a look at scoops in the big picture. The idea of a scope is simply where global things are attached to. In the case of all the code we written here, the show it variable the functions called Show it then and show message those air all attached to what's called the global scope because we're not inside of anything else. All the variables we're creating here inside of a scope that everyone shares every job, script, library on the page actually shares this. So you need to be a little careful about using the global scope. So, for instance, I create a new variable called In Global. This variable is created and then attack to the global scope. So I can certainly do things like use it here in the global scope, right? Everything that's part of this global scope is actually attached in a Web page to an object called Window. So I look at window dot in global. This is actually attached to that window object. The window is the parent of the Web page. You're actually sitting on the window, ends up being the collector of everything global. So every function we created will also be here. Show message. Show it, then they're all part of that global scope. Inside of Java script, there's really only one thing that will create smaller scopes, and that is a function in different languages. They have the different idea, but what constitutes a scope? Ah, class a name space, et cetera. But in Javascript, the only thing that creates a scope is a function. So if we create a new function called test me for lack of a better name, we can actually use things that were in our outer scope. What should come as no surprise is that we create a variable inside of test me, and then we try to use it to move this up a little. This will be usable as well, because it's inside the scope where this code is being called. Hopefully, that's not surprising at all. The networks, the idea of a scope being important or the function level scoping important is back here outside the function because if we try to then right, this should fail because some message is not included, were back out in the global scope because it was created inside the function. Therefore, it has no visibility outside the function. Some message not defined. Of course, some message is to find. It's just to find outside the scope of that variable. So this is not supposed to ever work. This idea of scopes and how they work is important, especially when we start to look at the way that things called closures work. Let's do that next.

## Closures

[Autogenerated] closures, Aaron, Important Idea and JavaScript. So I want to make sure that you understand them clearly the idea behind a closure. And we saw that with the global scope in the last clip is that allows outer variables to be visible in inner scopes, and this happens regardless of the lifetime. And that's kind of the key piece here. Let me show you a little piece of JavaScript to describe this. So let's say we have a variable called X. We have some function and dysfunction uses that our variable X and then sometime way later that some function is called. Even if this has gone out of scope for one reason or another, it will be maintained in memory because the function here needs it. So as long as the function lives, this variable has to live for the same amount of time. The JavaScript runtime does this by wrapping it are creating a closure around this variable because it knows because of this function that it is necessary. Let's see how this works in practice. Let's go back and use that show, then message that we did a little earlier. I'll just type with closure, so we know what it was about. And then we're gonna use a callback to go ahead and show yet another message. But this time I'm going to use this some message that's outside of the scope of this new function. Even this little callback function has its own scope. If I call, show it with our some message, we should see that it keeps this temporary variable around until this callback is executed. And this is an interesting point because normally this some messages on Lee going to be a lie for the scope of the one execution of this function. But in our case, the runtime is going to keep the some message around because it knows it's necessary for this additional call back. We can see the call toe with closure. And then when our callback was issued this some message still lived. It had a longer life because that closure kept it available. So in most cases you're not going to need to think logically about closures. But gone are the days where you have to store some global data in order to keep it around for some function or call back that needs it later. The JavaScript runtime will handle that for us. Let's look at objects in a raise next

## Objects and Arrays

[Autogenerated] it's the next topic to look at our objects and raise. I'm gonna clean up some of this boilerplate code we have here and comment out everything below this because we're going to start to build something interesting. We make some space, I'm gonna create a new variable called Result and for the value of result I'm gonna use Currently braces, curly braces are an indicator that what I'm going to create is a new object. The nature of the object oriented system and JavaScript is this idea of objects. Now objects don't have a type necessarily to them. In our case, we're talking about objects that don't have a defined type, necessarily To them, there is a way to create things that look and feel like classes from other languages in JavaScript. What, We're not gonna cover them in this course. In our case, we're just talking about objects. And an object is just a set of name value pairs. So I can say name Jake. Weary language Java script score 4.5. This ends up being an object. This object can now be accessed by using the dot Syntex. So if I want to say console dot log result dot name that will actually work because what it's done is this object notation has defined a set of properties for that object. Now they don't all have to be properties. We could also have a function as a property or even other objects. Now, when we have other objects, is properties they're going to have then your own set of Carly braces and allow you to define its own properties. And this way it really is just a hierarchy. So we think about JavaScript. Being object oriented and really is because we can simply have these objects. Objects also support mutation, so I can very easily just say result. That phone number equals 1234567 And now when I use this, I can actually call phone number, and I'll display that correctly because objects and JavaScript allow you to add new properties at any point. This is one of the places where typical high level language people see sharp JavaScript, C plus, plus etcetera get a little nervous because the type system is very loose and javascript. It's loosely typed in that way, and that's actually kind of a good thing you just have to understand the patterns, your own job script to get comfortable with that. The other kind of object that we're going to talk about here are a raise, and we kind of think of collections in many languages in certain ways and think about different kinds of collections. And a lot of that has supported. But the simple array, the simple collection, a simple list is something that's built directly into the language as well. If I create a variable called Result and put in two square brackets, that's gonna tell it, Hey, you're defining an empty array Honore with new members, and we call it an array. But Array has sort of a fixed meaning. It's more like a collection where we can push items in, pull the mouth, sort them et cetera to fairly mutate herbal array. The way we can add items is by calling push in. Our case will go ahead and add that result as the first item in the collection of results. We could, of course, say push and give it an object notation as well, more normally, if you have the data in code that you want, you're gonna actually use the object notation along with their array notation to create your collection, and I'll do that by just moving this down. I'm gonna copy this into my results and let's put a comma after it and create yet another one and we'll go ahead and use, um, editor and haired and tear tow more quickly build up a new one. Call it you. I changed the score and I'll leave the rest the same. So we now have a collection that has two members in it. And each of those members is an object that represents a bunch of properties that we actually care about and let me come out that a couple of little pieces here that we're not gonna use anymore, like this result. Now that we have an array, there's a few things we can do with it, and I'm not gonna cover most of them. But let's talk about the easiest, which is results. That length. Here is a way you could test to see how large the array is. And then, of course, we could say Console that log results and then just use the brackets in techs to indicate which item in the collection we want. In this case, we're looking at the first item because a raise in JavaScript are zero based. Once we get at this, we could just say something like Name pointed out to the council as well. So the way you're used to dealing with collections and other languages should be pretty comfortable here. Let's take this one step further and talk about looping on her next video.

## Looping

[Autogenerated] Let's see how JavaScript handles looping JavaScript because it came from the world of C and C plus plus still uses the standard C style loop. That's where we can create a variable the beginning. And if we're just numbering through an array, we would start it with zero would then take the X and make sure it's not too long. In our case, that would be results dot length and then the last piece of the four is incremental in the XO that on each loop were incremental in the X and then testing the second piece over and over again. So when the Starts X equals zero, X is still less than the length, and then it runs what is in between the currently races? Once it's done with that increments the X and then re evaluates this. If this continues to be true, they executed again, and so on soon Is it executed? This false? It falls out of the four loop. So in this case, we can get a result which is going to be our results based on which item of the loop were going through, and then we can do something with it for example, just right out to the console, The name of each item. We look at this in the browser we can now see we're seeing each of those items that were in the collection were walking through them one at a time. Looping also allows you to break out of it. So if we want to say if result dot score is less than four break So if it finds an item in the collection that is less than four, which is their second item, it's going to break out of the four loop entirely. It stops the execution of the four loop and falls down to the next statement after the four loop. We can see that here because the first J query is executed. But the second kicker you I skipped because we've broken out of it. We change this to greater than four. We could say continue and continue is another way toe aboard before, But what? It doesn't break us out of the four. It forces it to return back to the top of the four and increment the number again. So in the case will continue everything after it is ignored and it continues back at the beginning of the four statement increments the X and tries again. So in this case, because our scores 4.5 on J Query and less than that on Hickory why we should only see the Jake worry Why in the browser was is exactly what we see. So loops are really simple for walking through any sort of collection. Be able to walk through collections and do something interesting with him. It's something that you're gonna find is usual in your Web development, and we talked a lot about the basics of Java script, and our next module will delve into how Jake weary and make riding your code that you might be using in JavaScript a lot easier and more Web centric. Let's wrap up this module first.

## Summary

[Autogenerated] Let's just wrap up what we've learned about JavaScript. Hopefully, the point I've tried to hone in on is that Java script is a really language. You should treat it as such. Your fear of coming to Web development may have been ________ by people talking down or talking bad about JavaScript. JavaScript has certainly weaknesses that maybe other languages don't. But most languages have pros and cons. I think you'll find what you've worked with Java script a little bit and work with it in the correct way that you find it pretty easy to work with. Using script tags on Web pages is a very easy way to get your job script onto the page and loaded in the correct way. Like most languages, you may already know, Jobs Group includes the standard simple bits like variables, loops, condition, ALS, etcetera, objects and arrays are a little different in Javascript in other languages, but their first class citizens of the language and you should treat them as such. You should get comfortable with looser, typed or malleable or mutable objects in a raise over time, and in many ways, job script was written around the idea that functions are key to the language because you're going to be using functions as sort of the bread and butter off component izing your JavaScript. Understanding the way that scopes enclosures work together is key to doing that correctly. This has been the module on JavaScript. My name is Sean William. Youth of Wilder Mines. Thanks.

# jQuery

## Introduction

[Autogenerated] Welcome to the using Ji Karim, agile of our quick start, and this module will discuss Jake Weary. We'll start off by explaining what Jake Weary is. We'll grab a query from the j query dot com website Show you the basics of Jake Worry talk about event handling and Jake Worry. Jake worried to query the document to change the document to reform networking tasks with J Query and finally will work with some forms with Jake Worry, Let's get started.

## What Is jQuery?

[Autogenerated] The simple answer to what is Jake? Worry is that is a JavaScript library for use on client side Web development. Its purpose is to enable developers to create interactive Web sites that work across different browsers. This includes handling interaction from the user, manipulating the page as well as performing network requests. JavaScript on the Web have been around for a long time, but before Jake Worry came around, it was much more difficult as the different browsers supported different things. Different versions of JavaScript, different versions of the HD mil spec became more difficult to write. One set of code there perform well across these different platforms. The goal of J Query was to hide some of these differences so that developers could focus on adding functionality, not on cross browser issues, but in order to show you a J query is, let me show you it in action

## jQuery Basics

[Autogenerated] So let's take a look at the basics of Jake Weary. So if you started Jake worry dot com, you'll end up on this page is the home of J. Query includes things like the FBI documentation and the ability to download Jake Worry itself. You can get J. Curry in a variety of ways, but we're just going to download the library and use it directly from our own site. Hickory comes in two flavors. Jake Worry one and Jake Worry, too. And the big difference between these two versions is whether it supports Internet Explorer 67 and eight. This is an important difference. The reason they're maintaining two versions is that J. Cory one is larger and in some cases slower than Jake Worry, too. But if you need to support these older versions of Internet Explorer, you're really going to want to use the older version of Jake Worry. They're maintaining a lot of features in both sets of Jake Worry. You're not losing a lot of functionality between them. I typically start with Bakery to unless I find out that our users air actually using these older versions of Internet Explorer 67 or eight So I'm just going to download the compressed version and drop it in our project and hero put it in the same directories. Are other JavaScript back here in the index? HTML. I'm going to add it to our project because it's a base library. I want bakery script to happen before our own scripts. So you used the same script tag. But in this case, I'm going to include Jake Weary as the name of the library course to do the work we want to do inside this page with Jake Worry. Let's go ahead and place the contents of our results. Pain with a new div. That we will call results list results list is going to be the actual listing of the different results on a page over to the index Js. Much like earlier, we had gotten the results object and then used in her html to set it. We're going to do the same thing now that we have J. Curry in the picture. We're gonna go ahead and get the result list from the Dom, and we can do that by using the Jake weary function and equally takes. Is this first parameter a C. S s selector. Syntex. So geek worry is leaning on the CSS electors in tax. You're already learning for CSS to figure out how to find the elements inside your HTML document until we can go pound like the I D and then result list. This is the name or the i d of the element on the page which we see here is a result list. Let's return the result of this query. J query is querying the dom for some elements. Once we have this, we can use several methods for setting different things about this element. In our case, we could use HTML to set some arbitrary HTML much like we did appear with Inter HTML. Or we could even use this more simplified text, which is just gonna change the textural content. We saved that and refresh the browser. We can see it's now setting it from Jake Worry. So an interesting point about this is the Jake Worry object itself that were using to do these queries has an alias and the aliases, the dollar sign and most bakery code and examples you're going to see out there you're going to see them use the dollar signs in Texas because it's shorter In JavaScript, you're allowed to use certain symbols as variable and function names until they've decided that the dollar sign here is going to be an alias for the entire J. Curry, a p I. So that we can just use dollar sign here as an alias for the Jake weary object.

And the rest of this will continue to work just the same as we can see here in the browser. It continues to run. Next. Let's leveraged a quarry to do some event handling for us.

## Working With Events

[Autogenerated] next, let's go ahead and handle. Some events can interact with the users. We come back to our HTML page. Let's go ahead and add a quick button before our results list. And here it will just be a quick button that will allow us to hide and show the result list When we click the button. This would be a really simple interaction with the user. The name of this button is going to be toggle button via an I D tank, so we should be able to just get our toggle button by calling you guessed it pound total button. Once we have it, there's a special function on the returned object called on on allows you to handle events of different kinds. Now, as you mature is a Web developer, you learn a lot of these different event names. There's no real magic to finding all the event names. They're in documentation out on the Web or on W three C dot or GE. You're gonna need to learn these different events, but ________ have a very simple event. If you've ever done any, Windows development should be obvious, and that is an event called Click. We're going to tell it that we wanna handle when this button has been clicked. We're going to do that by supplying the first argument as the name of the event and then the second argument as a callback function. So when click handles on the taco button, let's do something. In our case, let's go ahead and show and hide the results list. Now we already have the result list up here. So like we talked about in the last module, we should be able to use it as a closure into the toggle button here, so we'll save result list dot tuggle tacos, a function inside of Jake Worry that says, if it's shown, hide it. If it's hidden, show it and will actually give it a little lengthen here. One of the parameters of toggle is how long it should take to animate the hide in the show. And so we should be able to see if we're paying attention that the list appears and disappears from view. Let's see if this is working in the browser. We have her butt in here, and when we click it, we should see and 1/2 a second it hides that result list, and if we click it again, it should show it. Hide Show we were first, little bit of event handling directly using J Query and the syntax for doing it is pretty simple what event we wanna handle. And then what is the code that should be executed when it's ready to be handled? Now that we have this Tongling, let's go ahead and toggle the name on the button as well. We can do this by testing. What is the text of the toggle button? So in this case, we're saying, if the text is Hyde, we want to do something about it in a sentence being a traditional sort of property, which you might be used to. Another language is Jake. Query has a function on each of the dumb objects called text, too much like we used. It appeared to set the text. We don't supply any arguments it knows to return the text, and then that way we can compare them. So you used text again to change it to show so majority hide to make it show else I'll set it back to hide. We go over to the browser now we should see it switch back and forth exactly in the way we want. But there's one more problem here that's not all that apparent. We go back to the HTML you remember we put the script tags on the bottom of our page to try to ensure that the whole Dom or hte e mail was loaded before our JavaScript waas. Now, because we're running this all on our local machine, it's pretty much guaranteed to happen in that order, not 100% guaranteed, but it's pretty much guaranteed. The problem we have here is that that little case when they don't load in exactly the same order, could cause some problems in the edge case where the dumb isn't ready before a scripts load, depending on the browser can cause you some problems. And so the way to handle this is actually a handle a special kind of event. In J Query Jake Query allows us to wrap any element that we've created, or that is in our Domine. We have is a raw dom object into a J query object that wraps it, and we can do this by starting with same syntax. But we're simply going to give it that object into. In our case, we want to wrap the document object, INGE, A query Because we want to do something that J. Curry knows about to our entire document. What we want to do is call something called the Ready Function Ready function is there to say, Give me your code that you want to execute once the document is ready. So the syntax is fairly descriptive and what it does when the document is ready, we're going to call a function. No, parameters were actually going to wrap our entire code, even the comments for us. So we wrapped Oliver code inside this document ready function so that none of this gets executed until we know the document and all the elements have unloaded correctly. Once that has happened, we can then do all this work against the Dom like we talked about. We can guarantee that it's gonna happen. Protection event that it's fired from the browser to say the dom is now ready to manipulate the secondary benefit here is it takes all of this code outside of the global scope so that we're not polluting other JavaScript files. We may write later that have the same names in them. It sort of protects them all in this little bubble so that different JavaScript files can't interfere with each other. But the main reason is to make sure that we're ready for this event to be fired once all of the dom is ready for us to manipulate. So far, we've talked about using J Query to query the Dom for elements that have a specific I D specified. Let's talk about acquiring the Dom in general so we can see the real power of a query.

## Querying the Document

[Autogenerated] So now let's dig a little deeper into the way that querying using J query really works. We go back and we look at her index that html you may remember we created this knave section. Miss Nab Section had a number of list items in it that contained the different elements of our menu. Over here in our code, we can use Jake weary again Toe look for those knave elements. Let's remind ourselves again that we used in CSS this header knave ally to look for each of those elements inside of the menu. We did that to display them in line and change their font size. We copy this C S s selector, this entire CSS selector. We can use it inside of Jake Worry. So J query inquiry, anything that is described as a CSS selector. In fact, inquiry add some additional things that aren't allowed in CSS to simplify some of the code that you might write. Well, look at that in a minute. What this returned with this J career function returns is actually something called a wrapped set. Now we returned wraps. That's from all of these queries that we did earlier. But because they were against a single I d. It always returned a collection or a wrapped set of results that only had one member. In this case, we know we're gonna probably get three items here. Instinct as a developer might be to get the collection and then use four tow walk through them. This one of the amazing things about the way that Jake worry works. All these same functions that were used to, like, text and each female that I've shown you already when you execute them against the result of a J query will execute him against all the results in the rap set. This only resulted in affecting one, because the rap set because of this being an i d query, always returned a single result. But we did the same thing here and say, setting the text to the ally items. Well, see that it changed them to all be the same item. So the executed function in the rap set is going to do the same thing thio each element of that rep set. Let's show you one that's a little bit more obvious. There's a function on the rap set called CSS that allow me to go ahead and set CSS arbitrarily this case, I'll do font wait and I'll make them all bold. No, I know what you're saying to yourself. Shouldn't I be doing this in CSS? Yes, you really should. But this will show you the power of what Jake worry can accomplish, imagining that you might need to set something as bold as the user interacts with the page. CSS is gonna apply those rules when the pages loaded, but isn't going to necessarily know what to do with him when a user hovers or cliques or taps on something. So you can now see that all three of these are bold. The work involved to actually execute this query is nontrivial. It's pretty quick in the larger scheme. If you have a very large HTML document, this can be a little slow, so it's often a better idea if you're going to be doing multiple things to go ahead and hold it. So let's call this list items and that will yield the same thing. But that means that if we decide to do something else like sent the size, then this execution is going to be against That result said it will first sent the font wait for each of the items in the wrapped set of the list items and then do the same thing for the font size. But it's often the case where you might want to actually look into the results that, for some more information, do a query against aquarium. Well, certainly we could have changed this and created another query like Header Knave Ally and use a bakery specific extension to the CSS electors in Texas. I'll give you an example of one first, and in this case it will go ahead and on the first list item and set it to

18 points. But remember that by executing this entire list, it could be somewhat inefficient. Because if you have a large HTML document, these queries aren't a trivial effort. We want to be able to save them as much as possible. And one way to do this is to reuse that wrapped set like we talked about. But instead of calling the CSS, we can use a call called Filter to allow us to then do another query against that rap set, and in our case, this will be a much more efficient way to go ahead and set the size of just the first item in er list items. So it looked at some other techniques for clearing the Dom. Now let's look at how to actually manipulate that dumb.

## Modifying the Document

[Autogenerated] Now let's manipulate the document using J Query. Remember back to the last module. We created a collection of results here that we wanted to represent some results from a quarry to get hump. We're going to re use these We had originally gone through in just logged out some information. I'm gonna come at that code out because we're going to do something different. We're gonna actually manipulate the Dom by adding the results to the U I. Well, before we use a simple for each bakery actually comes with a faster version of a loop for us to be able to do this sort of work. So we want to go through the results to execute something for each result to be able to display some ht male in a result list for each of those results. And this improvement to the four loop is exposed on the dollar sign of the J query object as ah named member called each each is just a function and each first takes the collection we want to generate through results and then a call back. This callback expects a variable for the item number, and in that case, I'll call that I and then the actual item in the collection that's being passed in in this case, that's item. It could take some other parameters optionally, but these are the two that are the most common. So at this point, we have as item in this callback each of these items one after the other. So whatever happened in this code is going to be iterated over and over again before we start to fill in the results list. Let's grab that result list collection and just call empty on it. It's another J query function that will allow you to just take all the Children or content of an element and get rid of them because we're then going to add them back here in the for each. So the first thing we want to do in this clause is go ahead and build up some HT Mao, and we'll just use drinking at nation to do it. And we're going to do this by creating a new result where we have a div called Result. An inside of that, there's going to be a title and more just cancan. Captain eight. This by taking that item that represents each of the items in our list and showing the name and then another one showing the language and then another one showing the log in name for the owner. So really simple piece of nested dibs that represent what our new you I should look like once we have that we can just say result list a penned until it to add this new piece of mark up as a child inside the result list, I could go back over to the browser. Well, now see that we have Jake Worry JavaScript, Sean Goldman with Jake or you II. We've built up some markup that represents each of the results that we have mocked up in our code. Just to remind you that was Jake worry JavaScript his language. Sean Wildermuth as the owner, etcetera. To make them a little clear, let's go ahead and add some CSS to format. Those new results divisions that we created and here will just create a margin and a border much like we did in the second module and just set the title section to be bold ID. With this new CSS, we can now see each of our elements a little clear as individual results. Remember back when we wrote the code for hide and show? This continues to work even though the contents are getting bigger because we're hiding the entire element. So let's add a little bit more interaction by using another J query function called Hover. Hover will allow us to do some code when we enter and leave. The element with a mouse, obviously in touch screens is would be total different. But let's focus on the desktop browsers for the moment. Ordinarily, you could think you could do this by saying new results hover, which is Ajay Kori function. And the, however, takes two functions. One function for entering and then another function or call back for leaving. That's what we enter it, what to make it darker and then reverse it when we're done. Make sense. The problem, of course, is that Hover itself is J. Curry extension. A new result is not a ______ quarry object, much like we were able to take the document object that represents a dom element and wrap it in a J Curreri object like this. The same thing works if we wrap text that represents a parcel set of JavaScript. So let's change this new result to be Ajay Kori object. That way we can actually use the hover here. Of course, the thing we're hovering against is this new result. But she quickly, because this is a callback will actually allow us to, in fact, set the this key word as the Dom Element that we hovered over in this case, that this would be that Dom element. But again, we want to use J Query to wrap it so that we can go ahead and set some CSS. So I'll do the same thing yet again, wrapping of this and just seeing. And I'll do the same thing in the other handler, but said it back. Too transparent. Remember, this hover function is just going to set up these two callbacks. So if we've done it right as we hover and leave, I'll go ahead and become transparent and like Gray as you would expect, it's these little niceties that makes J Query really easy to work with, because writing this sort of code becomes trivial. What you're using, Jake worry. So now that we have code that's showing our individual results, and we can hide the results that let's go and look at the way that calling out to get have to get the actual results so we can show him here on the page actually works.

## Networking With jQuery

[Autogenerated] So now let's make some network requests with Jake weary. Another place where Jake, where you can really help if we open up a browser or other tools for looking at network requests, we can simply pay. Send an Earl this case to the gate, huh? B p. I. That describe some sort of quarry we want to execute. In our case, it will be a quarry for the word Jake weary and look for all sorts of projects that have as their language JavaScript and are sorted by stars. It's just a set of curry elements that are added to an A P I directly in a your eye you have exposes. This is a public a p I. And so when we execute it, what we're returning is a list of items Now the syntax where it looks like the beginning of the objects in tax and JavaScript is no mistake. This format is called Jason, her JavaScript object notation. In this case, all of the property names have to be quoted, which is why you see them all quoted here. But essentially, it's close to the object and a race in tax we talked about in the last module for our needs. What we want to focus on is that this is going to return an object that contains the results of some query. It does this by making a network request to get up. What's Grady Variable called? Get Hub Search and we'll just contain the entire your eye. Now we can use J Query to get the results here by calling dollar sign dot get. This is a method inside of Jake Worry that says Go and issue a get against some network resource. Now on, http, there's a series of verbs that you can issue post get put delete patch options. There's a number of them, but when most browsers going get data, the verb is unsurprisingly get. And so what we can do here is just use this gate hub. Search your eye as thesis Oh, RSS of that get and by default we can give it a callback that expects a parameter that contains the results of that get. Now. This pattern of using a callback should be familiar to you with event handling, but it's especially important here because thes network requests are going to be a synchronous you're going to call them here. And then sometime later, when the networking is done, it will call your function with the results. So we come over here and say, Console that log. Are that item stop length? They're warmer, getting our dot items that length. Remember that this are is going to represent the results of this object that is returned and one of the properties of this object are the actual items. This array of items that has returned, so are the items becomes just the objects in tax for that result, and then length is going to tell us how many items air in this collection. We look in the browser. Let's use those tools again to look at the console to refresh the page. We're going to see there's 30 results. In fact, in the console, it'll show you this. Get that it just executed. And if you want to see what was inside of it, you're going to see that the response is that same response we looked at a minute ago. So we're essentially telling Jake, hurry to ask the browser to issue one of these commands for us. Then when it does, we get back in code the results of that command. But instead of just showing the lane, let's do something more interesting. Like calling a new function, call display results and let's pass it the items that are being returned now, we haven't written this display results yet, but we've done most of the work required. This code where we went through an empty the result list and then showed each of the results is in fact, what we want. Our new function display results to call those results as this function. And of course, this results was using these mocked up results not are really results. So let's comment those out because those were just useful for testing and teaching you how that stuff worked before us were actually going to execute this get home search and then used that same could we did a minute ago. But to show you those same results now, why would that continue to work? It continued to work because each of these items actually have the same properties, has a bunch more. But it seemed properties that we had mocked together. So we have a name we haven't owner that has a log in, and we have a language, so we're using some of those same piece of the day that we were before. Obviously, this is returning more data. We could display even more data, but this should continue to work. And let's see if it does. Once we wait a minute, we can see that it returns all of these results. They're all in JavaScript because that's what the quarry was looking for. And they're all related to Jake weary. And one way or another, let's go back and look at the networking code. Just one last time we take this networking code, we have this function in here that is gonna be called with the results. But we don't have a good way to test for whether the results aired out or there was some problem. The network went down, Get have was down. Whatever the case may be, we need to be able to handle those air conditions, and we could simply give ourselves another function here that happened to be our error call back. That would work as in fact, with Jake Worry supports, But he also support another syntax, which I want to introduce to you because it makes more sense to me a little bit easier to read and makes more sense, And that is something called a promise. I could describe promises in depth and show you howto write your own and those sorts of things. But let's just focus on what it's going to give us. In this case, we simply call get with just the your eye. The object that has returned has a number of functions on it, and one of them is success. And just like our code did before, success is going to take a callback function for when it succeeds. But the power of the way the promises work is I can then change this with another call to fail so that I could do something to show or to handle the air condition. I can even add 1/3 1 here, which is done, which will be called in either case to do something at the end, and I'm not gonna actually do anything and done for time's sake. But we can see here that this chain of different event handlers means that it's a lot clear to know which of these callback functions belongs to which case we showed this again in the browser. Once the query executes, we can see the same behavior we did before it helps up working. Suddenly, this code wouldn't run, but the code that rode out to the console that it failed would certainly do that. And of course, you would do something more interesting, like setting some text in the U I to show that it was an air condition. Now we have networking and manipulating the Dominic wearing the Dom. Let's bring it all together by making this form work to actually issue the queries so that the user can actually enter a query in here and see the results.

## Working With Forms

[Autogenerated] Now I've seen Jake. We work in a variety of ways. Let's put them together to make our form actually work. We use the coring pieces we've looked at. We'll look at the manipulating pieces we've looked at, and we use the networking all together to make our form work to start out. We want to handle an event on the actual form. So we go and look at our foreman here. You remember The form is called class bordered image or simple form. But in our case, we're gonna want an I. D. Here because we only want to react to this code for this exact form. So let's call this our get hub search form. You can call it anything you want, but that'll do for now. And because it's an I d. Will start with pound and find our form on the page and use on again to handle an event. An event we're looking for is submit. So when the user submits the form, we want to be able to handle it. Now, if you remember from earlier, we were actually handling it with this action out to his echo tool, I have. So we don't want our form to actually do this execution anymore. One of the things you can do with an event handler is return false from it, and it'll know not to let the event actually fire that you're going to handle the event. But don't let the default behavior actually happen Very common when you're dealing with forms to not actually let the form execute because you may be displaying something you wanna page like we're going to be doing where you may be validating the code on the page and in both those cases, you don't want the form to actually submit and accept a new data from the server and replace your page. You want to be more interactive than that. Now that returning falls, let's do something. In our case, we want to actually execute this query. So when someone searches using the form and presses search, we want this search to happen. So now that it's in here, let's go ahead and look in the browser. Well, see that it doesn't execute this right away. But once we press search, the execution happens. But we want to use this data from our form to compose the query so we're gonna need a couple pieces of information. First, we're going to need the search phrase. We're going to get that by searching for the search phrase on the page, which is this input. And remember, we named it search phrase. So we're just £2 search phrase we're gonna call Method called Vow Battle is going to take the result of that search page and return it. Here is the actual value that the user input it. The user hadn't inputted anything at that point, it will just return us an empty string. We're also looking for whether the check box was checked. And remember this input check box was called you stars. So we'll do the same thing in this case because it's a check box, it'll return. It's a boolean and lastly were interested in the language of choice. And this was a select or a list box, and the idea there was laying choice. So we're going to do the same thing and using the i d and call Val again to get the language that was chosen in our case. We don't want to actually issue the search unless a search phrase was used We just don't want to turn every result out there. So we have to have a sir trays. So first, just test for search phrase. And if it is good, we want all this code in here to run. We want to keep that returned false outside of this if because we wanted to shortcut the forms of mission in all cases, whether we're going to issue a search or not. And if we do want to do a search, we're gonna construct the Earl that we're going to use for the search. But first, let's use our existing result list here. And let's clear it by setting its text to, say, performing search. Remember our network queries gonna take us a minute. So we want to show the use or something while the search is happening. Now let's construct the query. Let's start by re creating this and we're gonna copy all of it up to this. Q equals, because it directly after the Q equals is going to be the actual search phrase we want to enter. And we're just gonna add to this the search raise. If our language choice doesn't equal all that we want to actually include the language choice. Remember, in our list, we have all as an option and then these other languages as other options. If they say all the gate, huh, baby, I says. Don't bother giving us a language. We'll just use all of them. So it's not all we need to include this little piece language and then the name of the language. So get hub search and I'll just use plus equal to upend it to it because of the following this pattern bless language and then the name of the language. Finally, we'll include the use stars. So if they've told us to use the stars, we'll do the same thing. We'll append to it. This last little bit of are you or I. Now when we perform the search will be including the user input into the search. Let's see this working will refresh the page. We're starting from nothing. And if we do a plain old search, nothing's gonna happen. But if we include a search like Newgate shows, performing search and then when it completes now they're using our input. We're getting the results exactly the way it would expect it. We change the search to something like a search engine. We're going to get a whole different set of results, I said. Language C sharp in the language that showing up here it see something's wrong. We look at her browser will see that the search engine is showing language, See, And in fact, search engine has a dollar sign 20 in the middle. And this is on purpose. Actually, when we're constructing your eye, the browsers trying to make sense of all the text we've put in that you or I. So in our code, we were simply taking whatever information was given to us by the user for search and language choice and embedding it in our project. The problem is, certain characters like the pound sign are specially used inside of the Web, and so it doesn't know what to do with them. So we need to use a special function to make sure that these are OK for your eye or your L. We call encode your eye component. When we're adding, it will simply wrapped us or trays around encode your eye component, and we'll do the same for language choice. We'll see here back on the browser when we execute this search, we're now getting language C percent 23 which is the encoding for 23. We're still getting the percent 20 which is the including for a space, because that's the correct thing to do here. The space was fine inner search phrase. But if we search for something with other special characters, we want them to be encoded into the your eye correctly. So we're now getting results that are actually I'll see sharp the way we expected. We change this to JavaScript. We'll get the job script ones. And so we're actually getting results based on what the user actually wants in is interacting with the page and the way you would expect it to. Let's wrap up this module.

## Summary

[Autogenerated] So we've talked about Jake wearing this module and tried to give you a taste of what it's capable of. I have seen that it's simply a job script library for performing some basic tasks in Web development. We've seen that you can include Jake Worry, like any other library might need well on its face. Jake worries about clearing and changing the dom. We can see we can do a lot more and really create rich interactive experiences with it. Because you're leveraging your CSS elector knowledge, you can become better at J. Korean CSS at the same time. Of course, interacting with the user is important, so allowing J query to help you manage events in Web development means you'll be better at those quicker. And, of course, the networking we showed you in Jake Worry is really pretty simple to use. Looking at the way that Jake Weary queries the Dom changes, a Dom can pull data out of the Dom as well as use the networking. You can create really interactive tools using Jake weary and in the Web in pretty simple ways. So in this course we've tried to teach you the very, very basics of Web development just give you a taste of what the different pieces are and how they interact together to create great Web experiences. But of course, we know we've just barely scratched the surface. That's where the other courses that close I can really help you. Jake Query can help you build interactive experiences, but learning a full framework and a data binding framework like angular ember or even knock out can help you create richer, single page applications than you could with just by using J Query. If you knew the Java script, you might want to delve into one of the JavaScript courses in the floor site library to leverage your existing language to learn this new language called Java script. Likewise, there are whole courses on understanding the depths of what CSS is capable of, and I'd encourage you to look at those as well. You and Jake Recon be expended on with other courses in the polar site library. So hopefully this quick start has helped you learn some of these basics and wet your appetite for learning Maur about web development because you've really just only started Thanks for taking this course on jump starting your journey to being a Web developer. My name is Sean Wildermuth of Wilder Minds. Thanks for joining me.

## Course author

Shawn Wildermuth

Shawn Wildermuth has been tinkering with computers and software since he got a Vic-20 back in the early '80s. As a Microsoft MVP since 2002, he's also involved with Microsoft as an ASP.NET...

## Course info

| | |
|---|---|
| Level | Beginner |
| Rating | ★★★★☆ (1620) |
| My rating | ★★★★★ |
| Duration | 3h 4m |
| Released | 22 Aug 2014 |

## Share course