# **JavaScript and AJAX: Integration Techniques**

### **Welcome**

Selecting transcript lines in this section will navigate to timestamp in the video

- [Voiceover] Hey there, this is Ray Vilalobos and welcome to JavaScript and AJAX. In this course, I'm going to show you how to use AJAX with JavaScript and Jquery. I'll start by showing you what AJAX is and how you can use the APIs in JavaScript to build web applications that update your pages, without having to reload them from a server. I'll show you how to read and refresh pages with AJAX, using pure JavaScript. Then, we'll examine how to update and modify the DOM, the Document Object Model structure that browsers use to access elements on a page. We'll also explore reading and passing data in the X amount and JSON format. Plus, I'll show you how to work with the AJAX method in Jquery, to read and write information between the server and the client. Finally, I'll show you how to build a real AJAX project that lets you search through adjacent data file with JavaScript Jquery and CSS transitions. So, let's get started with JavaScript and AJAX.

### **What you should know before watching this course**

Selecting transcript lines in this section will navigate to timestamp in the video

JavaScript and AJAX is not a beginner course, and I'm starting with the assumption that there's a few things you already know before taking this course. So you should be familiar with building online projects and know how HTML works. A really good course to check out is James Williamson's HTML5: Structure, Syntax, and Semantics. You should also be familiar with basic JavaScript concepts. If you need help with that, check out Simon Allardice's JavaScript Essential Training. Now, one of the important things to understand about AJAX is that it is a conversation that happens between your server and your browser. Because of that, it means that these examples are not just going to work from your hard drive; you're going to need to have a server that your browser can talk to. So for that you've got a couple of options. The simplest, and what we'll do on this course, is to work on a live server through FTP. If you need help understanding FTP and working with a hosted server, check out my course on Managing a Hosted Web Site. The other option is to run a local server on your machine. You can do that with software like MAMP on the Mac or WAMP on the PC. If you need help with that, check out Installing Apache, MySQL, and PHP, with David Gassner. Now, you should be familiar and comfortable working with text editors like BBEdit, Sublime Text, or others. In this course I'm going to use an editor called Espresso, but you should use your favorite text editor. If you're on a Mac, you may want to check out TextWrangler from the Mac App Store. It has a cousin called BBEdit that is not free, but is a very capable developer tool. I like another editor that works on Macs, PCs, and Linux workstations called Sublime Text2. Although it's not free, it's quite capable and fun to use. And it really doesn't matter which editor you use, as long as it's something you're comfortable with. Finally, you're also going to need some sort of FTP application. I'll be using Transmit from Panic on this course, but if you're on a Mac or a PC, you may want to check out Cyberduck, which is a free FTP program for Mac and Windows. So JavaScript and AJAX is not necessarily for beginners, but with just a little bit of background experience, you should have no problem with this course.

### **Using the exercise files**

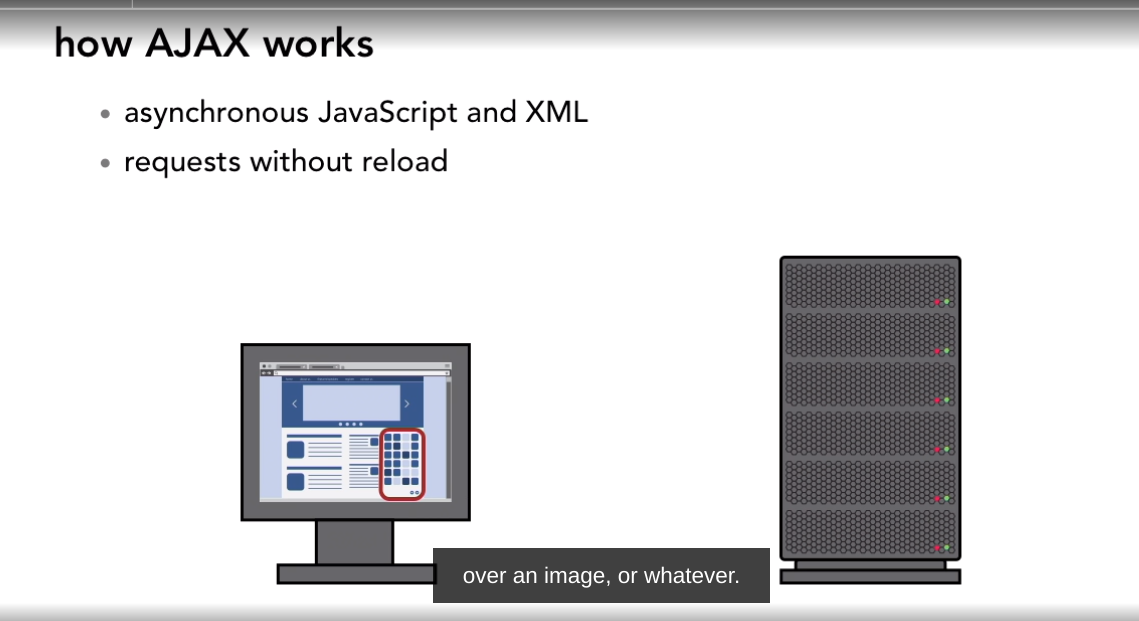
Selecting transcript lines in this section will navigate to timestamp in the video

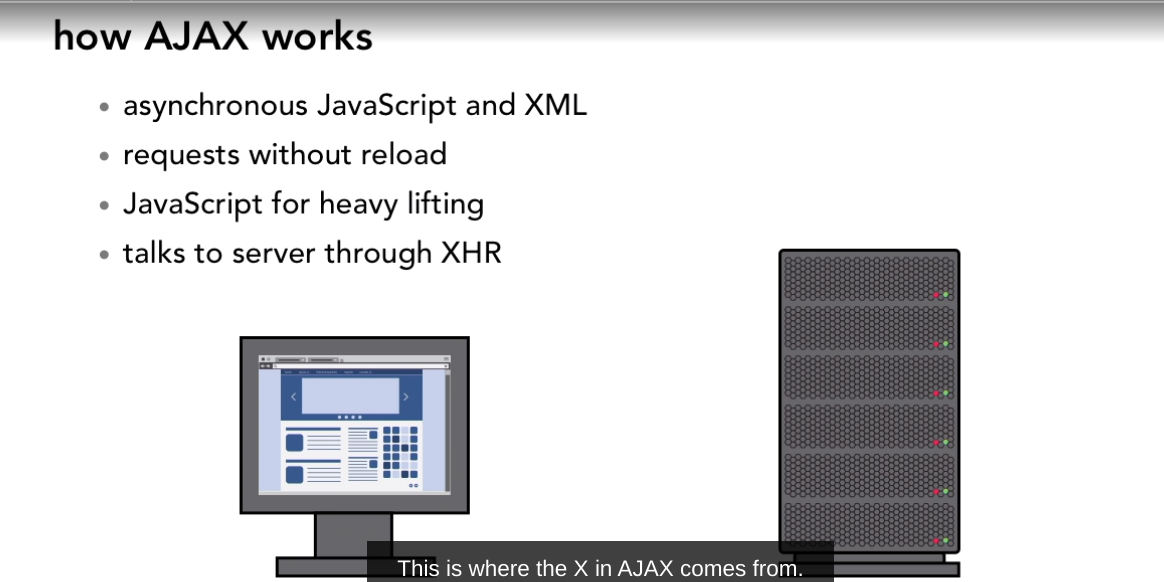
If you're a Premium member of the lynda.com online training library or if you're watching this tutorial on a DVD-ROM, you have access to the exercise files used throughout this title. In the Exercise Files folder you'll find folders for each video in the series. Inside each of those folders you'll find a version called workingfolder that has the website when I began the video, and you'll find a finished folder with a copy of the website when I finished. So I'll usually begin by opening up the copies of the projects from my server and open some of the files in my favorite text editor. If you're a Monthly member or an Annual member of lynda.com, you don't have access to the exercise files, but you can follow along from scratch with your own assets. So let's get busy with JavaScript and AJAX.

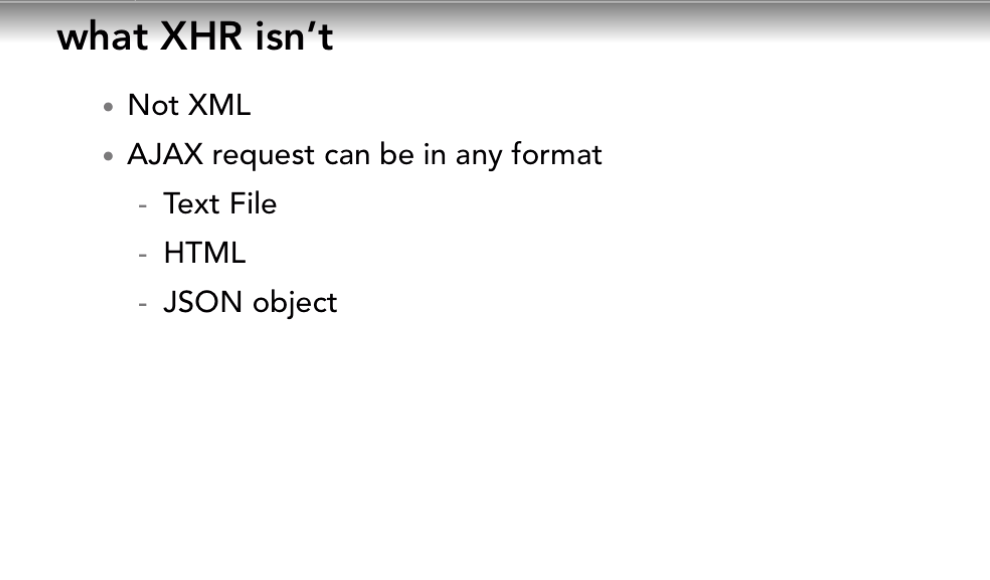
### **What is AJAX?**

Selecting transcript lines in this section will navigate to timestamp in the video

So you may be wondering what AJAX is and why you want to learn more about it, so let's take a look at how you can use AJAX with JavaScript to build dynamic web applications. Before we look at AJAX, let's talk about how a browser gets information to display a page. When a browser requests a page from a website, the browser in your machine is acting as a client. The website is being delivered through a machine that is serving up the information, so we call that a server. When you ask for a page, the client makes a request to the server and the server returns a page. Sometimes when you're looking at a page, you may want to ask for some additional information from the server. Say for example that your page has thumbnails for a photo album and perhaps the album has more thumbnails than you can see in the current view. Without AJAX if you want to see more thumbnails, you'd have to make a request back to the server and it would send you a whole new page with additional thumbnails. That means the server ends up sending a lot of information that you don't really need all over again. With AJAX, the server doesn't have to send you the whole page; it just sends the pieces you need-- in this case new thumbnails. So how does that work? AJAX is not a single technology, but a group of technologies working together to make that happen. Technically, AJAX stands for Asynchronous JavaScript and XML. That's quite a mouthful, so let's break that down piece by piece. Asynchronous means that the client can request new pieces of information from the server at any time. It doesn't have to wait for a page to reload. A new request can be triggered by an event like clicking on a button, hovering over an image, or whatever. The J in AJAX stands for JavaScript. JavaScript is where all this magic takes place. It handles the events that trigger a new request, makes the requests for new data to the server, and takes care of updating only the part of the document that needs to change. JavaScript talks to the server through a set of programming methods called an API and uses what's called an XHR or XML HTTP request. This is where the X in AJAX comes from. The XHR API lets the browser send and request data from a server. This can be a bit confusing because a lot of people think of XML as a language used to describe data that's a lot like HTML. Sometimes people assume that the X in AJAX means that the data from an AJAX request has to be in XML, but the data that gets transferred to and from the server can be in any format, and it's usually either a text file, HTML, or a JavaScript object, like JSON. So AJAX is really just a fancy term for a technology that lets you build pages that update without requiring a page reload. JavaScript does most of the heavy lifting with AJAX, and it uses the XHR API to handle the communication between the client and the server.







### **Using a synchronous XHR request**

Selecting transcript lines in this section will navigate to timestamp in the video

The first step in working with AJAX is to learn about the API browsers provide for sending and retrieving information. The way you access the API is by using a XML HTTP request or XHR object. So let's take a look. I'm going to start off with a super-basic page, and it's actually not going to be much of a page; the only thing this page will do right now is to load a separate script file that's going to do all of the work. So I'm going to go ahead and add the script tag right here. It's going to reference a file called script.js. And I don't need this type =text/ javascript, so I'll take it out. And I'm going to save this, and I'll go ahead and create the separate file. I have Transmit open right here, so I'll right-click and select New File and I'll call it script.js. I'm also going to need another file called data.txt, so I'll create it by right-clicking again, selecting New File, and calling this data.txt. Now, I'm going to open up data.txt. This file is just going to have the words "Hello World" here. I'll save it, close this out, and I'm going to open up my script.js file. I don't really need to do anything with the index.html anymore. I'll go ahead and open this up. And just to make a little bit easier to work with, I'll drag it right next to this index tab and so it will doc it at the same place as this window. So let's start by creating the simplest of XHR requests. We'll need to create a new variable for the object that we're creating. Next, we need to create the request to our XHR object. The open function of this object is going to require three parameters. First, the method, which is going to be either GET or POST. Then the location of the data file. Since it's in the same folder as this document, it will just be the name of the file. I should note here that AJAX requests have what's called a same domain policy, so you cannot request data objects from domains other than the one you're currently on. Now, the last parameter we pass is a Boolean that specifies whether we want the request to be asynchronous or not. To keep things simple, we're making that false, and that makes a request a synchronous request. This makes our browser wait until the request is done before it does anything else, and this is generally a bad idea, but this is going to make our code simpler for now. So we've created this request, but it hasn't gone to the server yet. The send command is going to take care of that. Now, I'm also going to output the results of the request to the JavaScript console. So I'm going to go ahead and save this and take a look at what we get from the server. So I'll load this page up on the server, and right now the page doesn't do anything. What I want to do is right-click. I'm in Chrome right now, so I want to select Inspect Element, and this will take me to the Developer Tools. And then I want to hit on this tab that says Console. So when you click on that tab you'll see that it returned an XMLHttpRequest object. And we can open that up and take a look at what's returned from our request to the server. So one of the important things that we see here is the status of the request. That property gives us a number, which in this case says 200. That means the AJAX call was successful. If any other number was here, like 404 or 500, that would mean the call to the server was unsuccessful. So we can modify our code to check for that property. So I'll save that and then I'll switch over to Chrome, refresh the page, and you'll see we get the same request. Now, you can see that text from the file that was loaded in the response and responseText properties. The responseText property is what you want to use because it always has just the text of our response. So let's go ahead and output that to our document with the writeln command. So let's save that, and we'll switch back into Chrome, and now we can see the text Hello World appear in our browser. So I mentioned before that these requests we're making are synchronous and that means the server is waiting until the requests are done before it continues. Right now you really can't see that as a problem, because we're only making a single request. So I'm going to change the code and make 100 requests to the server to see what would happen. So I'm going to save this, and I'll switch over to my browser, and I'll reload the page, and you'll see all the requests being made because I have the console open right now. **You can see that there's nothing appearing in my browser until it finishes with all the requests. If you take a look at the Network tab in the Developer Tools in Chrome, you can actually see that the browser is executing the requests in sequence and that the page is not even updating until the whole thing is done, and that's not necessarily a good thing.** So use the XMLhttpRequest object to make calls and request data from your server. You start by opening a connection and then sending the data. You can check the status of your requests, as well as access the text returned by your request and do something with that, like writing it to your page.

### **Making requests asynchronous**

**Selecting transcript lines in this section will navigate to timestamp in the video**

**So AJAX requests are supposed to be asynchronous by nature, which means that they should run in the background independently of other events; otherwise they would be called synchronous requests and the language would probably be called SJAX, right? Really, it's the whole reason AJAX is awesome, so let's take a look at making asynchronous requests. So I have a project here open with three files: a data file with the words Hello World, an HTML file whose sole purpose is to call a JavaScript document, as well as this JavaScript document, with a single synchronous request.** I wanted to take a minute to look at this and highlight some of the events that the request triggered. So I'm going to switch over to my browser, and I'm going to load up this page, and then I'm going to right- click and select Inspect Element. Then I'm going to switch over to the console and take a look at this object that's returned from my server. So I'll open up that up, and you'll see that there's a lot of different events right here. They all start with on. you see onabort, onerror, onload. There's an event that we're interested in called onreadystatechange. The browser's AJAX API maintains a property called readyState that has a number indicating how far along the request has progressed. If the Value is 0, for example, then the request hasn't been sent yet. Once the value of that property reaches 4, then the operation of sending and receiving the requests has been completed. So what we want to do is modify our code so that it checks the status of this property. If readyState has the value of 4, then our asynchronous request has been received and we can do something with the data we get back. First, before we send our request, we want to change our open method and take out the false parameter at the end. We could send a Boolean value of true, but this is the default and so we don't really need to pass it. Now we can insert an event handler. Event handlers will run a function when something happens. We want to know when the onreadystatechange event changes. So before we send the request, we'll check that event, and this will be a function literal, so I'll take out the name. And then I'm going to grab all this right here and paste it inside this event handler. I also want to modify our if statement so that it not only checks for the request status, but also for the readyState value. So I'll copy this and I'll paste it twice, and then I'll change this one to say request.readyState. So this should check to see if the value of request.readyState is equal to 4, and also if the request.status is 200. So let me go ahead and save that, and I'll switch over to my browser, and I'll reload the page. And you see that we get the same HTML and we get the same result, but the way we got the result is a little bit different. So, although it doesn't look any different, this is an honest-to-goodness proper AJAX request. It sends a request to our server and then waits for that request to come back at some point; that's called asynchronously. If you want to learn more about the XHR requests, take a look at this document from Mozilla. It's a good resource to see all of the different options available in XHR requests.

### **cripting for backwards compatibility**

Selecting transcript lines in this section will navigate to timestamp in the video

Although usage of early versions of Internet Explorer have dropped dramatically in the last few years, it's still important to know how to run AJAX requests that are compatible with those systems. And actually, the whole concept behind XHR requests was created by Microsoft and implemented first in a browser in IE5. To accomplish this, Microsoft used a technology called ActiveX. Because ActiveX was not available to other browsers like Safari, Firefox, and Chrome, they created a different implementation of the API using the XMLHttpRequest object. Since ActiveX was seen as a proprietary implementation, the XMLHttpRequest object became the standard. So like with a lot of other web technologies, we've ended up with a fragmented implementation when trying to support older browsers. Luckily, all we have to do to support these browsers is to check if the browser API has an XHR object. So first, I'm going to change this line right here to just create the variable. Now I'm going to use an if statement. So if the browser's API has the XMLHttpRequest object, then I'm going to set the request variable to this new object; otherwise, we'll check for the ActiveX object. The rest of our code is not going to change and our AJAX code should execute So let's go ahead and save this and I'll switch over to my browser and I'll in the same manner. just refresh the page. You see that the page looks exactly the same, but my code is now more compatible with older browsers. So even though you might not run into a situation in which you have to support really old browsers, it's good to know that doing it takes just a few lines of code.

### **Updating the DOM with getElementById**

Selecting transcript lines in this section will navigate to timestamp in the video

AJAX uses JavaScript to make changes to the current document through something called the DOM. This is the Document Object Model, and its job is to keep track of the structure of our HTML document. When you work with AJAX it's usually because you want to change something in your page after it has loaded. The easiest way to make a change in the DOM is through a JavaScript method called getElementByID. So the text editor that I use has a really nice DOM preview feature, so I'll turn that on right here, and you can see the DOM for our current HTML document. You can see that there is an HTML tag and inside the HTML tag we have a head section that has a couple of other elements, as well as a body section with just a script tag. If we add another element here--let me add a headline--you will see that that element gets added into the document structure. If I add a paragraph, that also gets added, and if I create an element like an emphasis tag inside my paragraph, that gets added as a children of this paragraph object. Now, if you do add a div with an ID then that ID becomes a way that you can target that particular element. So let me go ahead and save this and I'll switch over to my script.js file and I'll get rid of this console.log method that outputs our content to the console. Now, since we have an ID, we can easily target it in our JavaScript file. First we're going to need to create a variable and then find that element using getElementById. So I am going to replace this document.writeln command and I'll create a variable called modify. I'm going set that variable called modify to be equal to an element in the DOM with an ID of update. Now that I have targeted that div in that variable, I can use that variable and modify the innerHTML, which is the HTML inside the element, to be whatever I want. In my case, I want to use the request and then look for the responseText. Let's go ahead and save this, and I'll switch over to my browser, and I'll load up the page and you could see the document has loaded Hello World into just the ID for update. So let me go ahead and right-click and Inspect Element. We could see the div with the ID of update is right here, and I have inserted the text Hello World from my external document into that div. So getElementById combined with innerHTML gives you a very quick way to update any element in your DOM as long as it has an ID. This is sort of like a laser that lets you target an individual element very quickly.

### **Modifying elements with getElementsByTagName**

Selecting transcript lines in this section will navigate to timestamp in the video

When working with AJAX sometimes you want to access an individual element and sometimes you want to access groups of elements. We looked at how to do individual elements with getElementById in the last video, and in this movie we are going to take a look at accessing groups of elements with getElementByTagName. So you could, for example, ask for the third list element in an unordered list. We are going to need to update our HTML code to do something like that. So, I'll get rid of this div right here and then I'll add an unordered list with some elements. Now I am going to save this and switch over to my script code and modify this getElementById and make that getelements. Remember, there is an S in there. ByTagName, the tag that we want to access is the list item tag. So by doing this, we are going to place an array of elements into the modify variable. Now that we've got the array, we can access an individual element of that array by just putting in the index of the element here. So, if we do modify(2) it will access the third element of the array, because these are 0 index, so the zero element is actually the first element, element number 1 is the second element, and element 2 is the third element. So, if I save this and I switch over to my browser and I refresh the page, you will see that the third element in my list has been updated with the content of our data file. So we grabbed all the list items and we targeted just the third element. Interestingly enough, if you have more than one list on the page, it'll actually let you cycle through all the elements as if it were a single array. Let me show you how that works. Switch back over to my editor. So, in the index.HTML document I am going to get the original list and then make another copy right here. So, let me save that and I'll go over to my script element. I know that there are six items in the unordered list. So if I ask for the eighth element, let's take a look at what happens in our browser. So I'll refresh, and now we have two lists and in the ninth position--that's 6, 7, 8, 9-- it's placed the text of our data file. There are some more objects down here. So there is the other bullets. But it updated the ninth element. Now there is actually a couple of other ways that you can get to the same place, so let's go ahead and examine that. So right here, when I am using modify=document.getElementsByTagName, I am looking for specifically all the list item elements. I can actually do multiple copies of this, so I am going to do a document.getElementsByTagName. And first, I am going to look for the on order list elements on the page--right now there's two of them--and I am going to look for the first unordered list element and inside that element I'll get the ElementsByTagName('li'). That will target our second list, not the first one. And now I will switch the index here to be index number 2. So this is actually doing the same thing in a slightly different way. First we are getting to the second list by using the index (1) here, and then we are getting to the elements inside that list that are of type list item and getting to the third element inside that object. So, if I save this--and I am going to go back into my page and just refresh-- you'll see that nothing is really changing, but the way that we are getting to those elements is different. So what if we wanted to cycle through all the elements in a list? Say that you wanted to update all the elements in our list items with the content from the external document. In that case you can use a loop. So, let's go back into our code. So we went to, after the modify variable gets instantiated, when I create a for loop, and that loop is going to go from 0 to the modify variable's length--and remember that this modify variable is now an array, so any methods or properties that I would use on an array would be valid here. And what I wanted to hear is modify, using each index, the innerHTML of each one of those objects with the response text from the request. And I also want to make sure that I modify my variable here, because it's only targeting the second list element, so I want to modify that to just look for getElementsByTagName, and I'll delete this line right here. So, what this is going to do is look for all the elements in your page that have a TagName of list item. It's going to place all that stuff into the modify variable, and then it's going to cycle through each modify variable, changing the HTML of each one of them to the responseText from our request. So let's save that, switch over to our browser, and we'll refresh, and now you'll see that it's actually updating every one of these bullet items. So, getElementsByTagName is really handy for cycling through a series of DOM elements. Understanding how to handle the array it returns is one of the keys to modifying your DOM with AJAX.

### **Parsing XML using AJAX**

Selecting transcript lines in this section will navigate to timestamp in the video

AJAX was originally designed to read data stored in a format called XML, or Extensible Markup Language. It's very much like HTML, but you get to make your own tags. So let's take a look at an example of what it takes like to read data in this format. We have a new file in our folder called data.xml. It's a file with some data in the XML format. If you know a little HTML this should be super easy to read. You can open an XML file with some browsers to see the structure a little bit better. So we have a speakers object, and inside the speaker objects we have a series of speaker objects. Each one of the speaker objects has different parts to it, so for example we have the <name> tag, the <shortname> tag--which is the same as the <name> tag but without spaces or special characters--as well as the <reknown> tag and a <bio> that has some information about the speaker. Reading this into JavaScript is super simple. As a matter of fact, the XHR object comes with a special attribute called responseXML. It's sort of like the responseText property, except that it converts the data into an object that you can manipulate through JavaScript. So, before we do anything else, let's go back into our HTML file and change this so we're back to a div with an id of update. So, I am going to go ahead and save this, and we are going to switch to the script.js file. And now we need to change the request so that it doesn't request the data.txt document, but instead uses our XML document. I am also going to remove the lines of code that are looping through all of my elements. So now that we are linking to the XML document instead of the text document let's just take a look at the responseXML property of our request. So let me save that and switch over to my browser, and let me hit the Back button to go back into my file, and then I'll refresh this. So, let's go into our console and take a look at what we get back. So, you see that what we get back is essentially a document with some data inside it that's pretty much like what we saw in the XML file. The XML object is really easy to go through. We can use the getElementsByTagName with an HTML or an XML document. So if we wanted to get to, say, the name of the second user, you can do something like this. So let me go ahead and save that, and then I'll switch over to my browser and we will hit Refresh. So what we get back is the entire <name> tag of that XML node. So every element inside an XML structure is considered a node. Elements inside other elements, including the text inside an element, is also considered a node. So, to get to the text of this node we could use this. So let me save that and switch back to my browser and hit Refresh. You can see that we are getting the name. It still has quotes which is not exactly correct, but at least we are not getting the tag anymore. So there is actually a shortcut for the first element in any array, and it's called firstChild. So if we do this, it's the same as writing child nodes zero. Refresh, we get the same results. Back over here, so if you look at what we get, it's the value in quotes, and it's not exactly the text of the node. To get that we would have to use the nodeValue property like this. So I am going to save that, switch over to my browser, refresh, and now we get just the text without the quotes. So, let's see what it will take to make a list of our speakers. We'll have to cycle through the XML file using a for loop. When the request is ready we'll create a variable that looks for the elements with a tag of name. Then we are going to create an output variable with an unordered list. Inside this, we are going to create a loop. And here we'll go ahead and add the value of each result as list items. Now we will send the results of all this to our div with a value of update. So let me save this and preview this in my browser, and you can see the list of all our names. So although you can read data in any format besides XML, the AJAX API was designed with an easy-to-use responseXML property that makes it a trivial matter to read and parse XML files.

### **Reading JSON files**

Selecting transcript lines in this section will navigate to timestamp in the video

Although AJAX was designed to work with files in the XML format, it will read the contents of any text file. So the data can really be in any format; the trick is to know how to parse or translate the data into objects that JavaScript can manipulate. An obvious choice is to use the file in the JSON format. JASON stands for JavaScript Object Notation, and is a way to structure data so that it can be easily converted to a JavaScript object. If you want to learn more about JSON, make sure you check out my course on Building Facebook Applications with HTML and JavaScript. There is a section called Adding JSON Data Feeds, and it has a good introduction to the JSON language. So, let's go back and take a look at what a JSON document looks like. If you watch the movie on parsing XML using AJAX, this is the same data, but in a slightly different format. The good news is that parsing data is super simple--sort of; some browsers support a parse command, so let's go into the JavaScript file and replace the code for processing or XML file. So, I'm going to modify this open command to open the JSON document. Now I'm going to take out the rest of this code right here, and I'll create a variable called items and set it to the JSON object and use the parse command with the responseText of the request. Then I'll output this to the console so we can look at it. So let me save this and I'll switch back to a browser, refresh our page, And if you have the console openm remember that you could just right-click on this page and select Inspect Element if you are using Google Chrome like me. And I can see the series of objects that this returns. If I open that up, I can see each individual object and inside each object, I can see different name and value pairs. So this is a little different from what we would see with XML. It's essentially an array of objects and each object has a label and the data from our file. So here's the bad news: JSON parse is not available in some older browsers. So, if you go to this website, you could see that it's available in most modern browsers, but a lot of the older versions of IE do not support the parse command. So your options are to use EVAL--but that has some security issues--write your own parser, or use an existing library. jQuery for example, takes care of this very well. Let's go back to our code. So we'll modify our code so that it outputs a list of speakers. We'll start by creating the output variable, just like we did with XML, and then in here we'll loop through the valleys of the objects. To do that I am going to use the for in statement. Finally, we'll target the update element with the results of our output variable. So let's save this and switch back to our browser, refresh the page, and you can see that the list appears. So working with JSON files is pretty easy with the parse command. However, it might not be available in all browser versions. So for ultimate compatibility, I recommend that you use a library like jQuery.

### **Using event-driven AJAX**

Selecting transcript lines in this section will navigate to timestamp in the video

You can use AJAX to update items on a page as a result of some type of event. Events in JavaScript can happen as a result of a user action, like a click or a mouse over an element, and some are triggered by other events, like when something has finished loading or when a key is released. You can use JavaScript to capture the event when it happens and trigger a series of commands. So first, we're going to clean up our HTML a little bit. We'll take out this AJAX page h1. It's not really doing very much. So let me go ahead and save that. And then I'm going to take my script and wrap it inside a function. So right now this script is executing whenever it gets to it in the index.html file, which is at the very end. So this page loads and then the script runs right here, so all this is executing right then. So what we'll do is prevent that from happening, by putting everything inside a function. So we'll grab all those code and we'll paste it right there. Let me go ahead and indent that a little bit. And now if I save this and I take a look at my page, you'll notice that nothing is happening because that function is not being called. So one of the ways that we can call a function is by creating a trigger. So I'm going to add a button here and I'll call this button Load, and I'm going to do an onclick event here, and it looks just like an attribute. And then I'll put LoadAJAX, which will call the function we created in JavaScript. So let me save that and I'll switch over to the browser, refresh the page. We see the Load button, and when we click on it, it should load the text from our file. Of course there's more events that you can use. For example, you can see if somebody has placed the mouse over the button. So we can do that with a different kind of event here; sometimes these are called event handlers. So we can do an onmouseover, and I'll save that, refresh our page. Now when I go over this button, it loads the text. So mouse over that button and there's the text. So something interesting about events is that not all elements on a page can trigger the same types of events. So for example, there's an element called onload, so you might be tempted to say something like this: whenever this button finishes loading, then execute the function. So let me save that and see what happens. When I refresh the page, this is not loading, and it really doesn't matter what we do here. That's because the onload event doesn't make sense for buttons; it really only makes sense for a page or a body tag. So it's an attribute that can only go in the body tag. So let me save that and come over here. I'll refresh the page, and now you can see that it's kind of doing what it did before. As soon as the page loads, or as soon as the body of the page has finished loading--and it's going to load the text from our file. That's just one thing to keep in mind, not all elements can trigger any type of event. There's actually a ton of events. This is a really good place to find out about all the different events: when they happen and what triggers them. So events can be triggered either as attributes, like we have done right here, and they can also be triggered directly through JavaScript. So I'm going to take out the onload handler here and I'm going to save this. If I want to trigger this with JavaScript, I'm going to add an ID to my button so I can easily get to it. So I'm going to save this and I'll switch over to the JavaScript file, and what I'm going to do is create a variable that gets to that button using document.getElementById, and the id that I want is the loadbutton that we created in HTML. So that puts the button in the mybutton variable, and then what I can do is check for a click. So I can say mybutton.onclick= and then I can execute the function called loadAJAX. So I'll save that. And notice the different notation here. I don't need the parentheses in here. So I'll save that, come over here, refresh, Load button, and now when I click on the button, it loads my text. So there's a different way to write this, and it doesn't really matter which way you do it, but you'll see it in both ways on the web. So one of the things that you can do to shorten this a little bit is actually create the function as a function literal. So I can get rid of all this right here and just say =function and then it will just execute the function as soon as that mouse is clicked. That's okay if this is the only button that's going to load the list of names for us. If you wanted to use a function that was more generic, you might want to keep it the other way. So let me save this and over here refresh, and we click and there's our text. So events are pretty easy to work with. They can be triggered either through an attribute in the HTML or through an event handler in JavaScript.

### **Understanding and installing jQuery**

Selecting transcript lines in this section will navigate to timestamp in the video

jQuery is a framework that fixes compatibility issues that happen with JavaScript in older browsers. The library also gives you a lot of additional tools for working with the DOM. One of my favorite features of the framework is how easy it makes it to work with AJAX. So let's go ahead and take a look at some of the options you have for installing jQuery. So there's two ways you can include jQuery into your projects. You can download a copy or use it from a CDN. A CDN is a content delivery network. You can see links to those if you click on the Download tab right here and then look over here to the left. Those are links to the most popular CDN libraries that have jQuery. You can also try cdnjs.com; that has copies of a lot of libraries, including jQuery. So the advantage of using a CDN is that if a browser has loaded your library from a CDN when you visited some other website, it may already be cached in your browser when it comes to your site, so your page will load a lot faster. The disadvantage is that CDNs will only work while you have a live connection to the Internet. If you're working with a local copy or maybe flying around on an airplane or coding where your connection is spotty, then the CDN is not going to load at all and none of your JavaScript will work. So you load them both into your code in pretty much the same way. So let's go ahead and go to the front of the jQuery website and click on this Download link. Now, before you do that, see that there's two options here. You have the option of downloading a Production version or a Development version. Now, the Development version has a copy of all the code with comments that's super easy to read, but it's obviously a lot bigger. Whereas the Production is minified and Gzipped, which means that it's compressed, it has no comments, a lot of the variable names have been changed so that they're smaller, and it's really pretty impossible to read. You should use the Development version if you're going to be editing something about the jQuery JavaScript code; otherwise, just use the Production version, and that's what I'm going to do. So I'll click on Download jQuery. Sometimes you see this kind of code right here, which is essentially the JavaScript code for jQuery. I told you the Production version was hard to read, and I wasn't kidding. Now, to save this I'm going to go to the File menu-- I'm using Safari right now--and I'm going to select Save As. And here I'm just going to rename this jquery.js and I'll just save it on the Desktop. I want to make sure that from this Format menu, I choose Page Source. Make sure it's not in Web Archive. So I'll select that, hit Save, and now that saves it into our desktop, so I'm going to switch there. And what I need to do now is copy this into my server, so I'll copy that just by dragging it. I'm using Transmit as my FTP program, so there it is. It's already on my server. And now what I want to do is select the files that I need and open them up in my editor. So I've got all these files opened up in my editor. There's the jquery.js library. I really don't need to see it, so I'm just going to close it up. What I need to do to insert it is go to my index.html file and then load it up just like I would any other script. I want to make sure that I load it up before my script.js, because that's where I'm going to be putting my regular JavaScript and I want to make sure that the library is there before my other scripts load. So I'll do a script tag, and as the source I'll just type in jquery.js. I don't need this part. Now, the only difference between this and loading it from a CDN is that if you were to load this from a content delivery network, you'd want to get the URL of the library and just copy that from here and paste it into your script tag. But I'm not going to do that because I'm just using a local copy. So save it and now jQuery will load into my application, and it's ready for me to use. Loading up jQuery is normally done using one of the CDNs when you're in production, but keep in mind that if you're going to be offline and developing, then downloading a copy is a super-smart idea. One more thing: you should always load jQuery before your regular JS file. That way all the power of the framework is going to be available when you write your scripts.

### **Working with jQuery and AJAX**

Selecting transcript lines in this section will navigate to timestamp in the video

jQuery is a great library and one of its mottos is that you can write less and do more, and one of the places where you can really see that is when working with AJAX. So let's take a look at how we can use AJAX with jQuery. jQuery provides several methods for working with AJAX calls, and you can see all them in this page right here. So for the most part, you end up working with one or two of these. So let's go ahead and load some content onto our page. Our HTML file is a simpler version of what we had in the last chapter, just a div with an id of update, and we're loading the jQuery as well as our script file. The script document right now is empty, and our data file just has the words Hello World. So I'm going to switch over to my script and I'm going to type in some jQuery right here. So I'm going to target the update id element and then load into it our data.txt document. So let me save that and switch over to my browser, refresh my document, and you can see the text Hello World. So there's actually a lot going on in that very simple line of jQuery code. jQuery is taking care of a lot of problems with backwards compatibility in older browsers, and it also creates and sends the XHR request. Not only that, jQuery gives you new ways of selecting elements. Here we're selecting something in the DOM-- in this case a div with an id of update. If you watched the chapter on modifying the DOM, you know that JavaScript has two main methods for getting to elements: getElementById and getElementsByTagName. You may have been wondering why I didn't talk about getElementsByClassName, and that's because although there's some support for this method in a lot of modern browsers, the support is not yet universal, and even if it were, working with complex selectors is much easier to do with jQuery. Let's modify our code so that we can use a class name instead of an ID. So I'm just going to change this ID right here to class, and all I have to do in jQuery is just modify this pound sign right here to select the class of update. I'm going to save that, switch over to my browser, refresh, and the text still comes up. So, classes are supposed to refer to elements that occur multiple times in the DOM. Let's go ahead and go to the HTML file and just copy this update element a few times. So I'm going to save this and I'll go back to my browser and I'll hit refresh, and now you can see that it's loading multiple times. So the load event is just going to take all our content with the same class and replace it with the text from our file. So jQuery is not only going to improve what you can do with JavaScript, but it goes beyond that and gives you totally new selectors with which to access the data. So check this out. I'm going to go into my JavaScript file. So what I'm going to do is change this so that I only select the first of all the update classes. So I'm going to save that and switch over to my browser and I'm going to refresh. And you can't really see what's going on here. So I'm going over to the Elements. I have the Developer tools open. If you don't have that open, just right-click and select Inspect Element. I'm using Google Chrome as my browser. But take a look at the code right here. It's still showing the other classes, but it's only changing the first one. And that's pretty cool. So let's go back and try another one. We can actually tell jQuery to just load up the even classes. And I'll save that, come over here, refresh, and if you take a look at our code, we've got three up here, but it only did every other class right here--and that's a really, really powerful, easy way to do something like zebra striping on tables. Using jQuery is going to take care of a lot of the grunt work necessary to work with AJAX. It also gives you some powerful new ways to select DOM elements. I think it's good to understand the XHR object and how JavaScript handles things. When it comes down to taking care of business, it's hard to argue with the convenience that JQuery provides. If you want to know more about jQuery, make sure you take a look at Joe Marini's excellent course, jQuery Essential Training.

### **Reading data with jQuery**

Selecting transcript lines in this section will navigate to timestamp in the video

In the last movie we used jQuery's load method to load some text from an external source. If you're looking to load structured data, jQuery has a method called getJSON. As the name implies, it's designed to load data in the JSON format. So once again, I'm beginning with a simplified version of our HTML file. It's pretty much just a div with an ID of update, and then they call to jQuery, as well as our script file. Our script file right now is empty, and we're going to need a file called data.json. It has the data that we need to load into our document. So in our script file we'll start off by calling the getJSON method. We'll pass it along the file name that we want to load and then we'll create a function. This would be a function literal so it will have no name. And we want to pass along the data that we get from the getJSON command into this function. So let's go ahead and output the data to our console so we can take a look at it. So let me save this, and I'll switch back to our browser. I'll hit the Refresh key. And I want to make sure I switch over to the Console tag. And that should give you a list or an array of objects. And if we open that up, we can see that each one of those objects has name and value pairs with different parameters here. And this is essentially how a JSON object looks to your browser. So I want to output a list of some of the names of our speakers, so let's go ahead and prepare an unordered list right here. I'll take out this console.log command and I'll create a variable called output. This is going to have an ordered list here. And then at the end of this list, I need to close the ul tag. So instead of using the for-in JavaScript loop, I'm going to use jQuery's built-in Each statement. I'm going to pass along the data and then a function literal. And that function literal is going to have a couple of variables: one of them called key and the other one called value. So this is similar to using a for- in statement, but a little better because jQuery always tries to do a great job of dealing with browser and platform issues. When you have a choice, it's always better to use the built-in functions. So in here I'm going to modify my output variable, and I'll output a list item, and I'm going to use the value variable and pass along the name and close out my list item. So where am I getting this name from? That's actually from the list of objects and the name of value pairs that I get right here. So the name is right here for each one of these objects. So I can call any one of those by just calling this value and then adding .name to it. So to update the HTML, we need to use the jQuery HTML method. So I want to target our update div and then change the HTML to have the output that we created above. So let me save that and I'll switch over to my browser and refresh, and you can see all of the names coming up from the file. So jQuery also provides additional methods to manage your data. Let's say, for example, that instead of just replacing all the HTML inside a tag, we wanted to append some HTML to the end of our list. So let's modify our HTML so that we already have something in this div id update. So I'll just create an h1 tag right here and I'll call it Speakers. I'll close it up. So if we were to run this right now, you would see the Speakers in the text and it would disappear and get replaced with this list. But instead of just replacing the HTML, we can actually do stuff like append the HTML, just with an append method right here. So I'm going to save that and I'll switch over to my browser. And now instead of deleting what was already there, I just placed something after it. Of course if there is append, you would think there is a prepend, and of course there is. So now that puts the text before the content of what was already in there. So with jQuery reading in a JSON file is pretty simple. A lot of times the added functionality of methods like append, prepend, and others makes it a better way to work with JavaScript.

### **Preparing a live search AJAX app**

Selecting transcript lines in this section will navigate to timestamp in the video

So it's time get busy with a real project in AJAX. I'm going to show you how to build a live search application, and this is what the application looks like when it's done. This application lets you type in some text in this input field and shows you only the records in the JSON document that match the search. It's dynamic, in that it updates as you're typing them. We're going to use some of what we have learned about AJAX so far and use some of jQuery's power to handle the heavy lifting. So let me go over the documents I'm going to start with. First off, I've cleared up the JavaScript file so that it has nothing in it. I want to begin with a clean slate. I'm going to get rid of some of the files I no longer need that are still in my server like this data.xml and this data.txt file. So I'm hitting Command+Delete, I'm in Transmit, and then hit this Delete button, gets rid of them. Eventually I'll need to add a style sheet to my HTML file, so I'll go ahead and right-click in Transmit and select New File and type in mystyle.css. You may have also noticed that I uploaded a folder full of images, and these are just photos of the different speakers. So the HTML that I'm starting with is pretty simple. I've changed the title to Live Search, and I just have div with an ID of update and a link to the jQuery and the script.js documents. So the first thing I'm going to need is to add a section for my search area. So I'll create a div, the id will be searcharea, and in here I'll type in an input field. I'll make it of type search. In some browsers you may see the edges have rounded corners. Now, it doesn't really matters what I type for name because this is not going to be part of a normal form, but I do have to add an ID value here because that's what we're going to target JavaScript. I don't need a value, so I'll take that out. I am going to add a placeholder. That's an HTML 5 feature that let's you put in something that the reader can see before they type anything. So if I switch over here and I clear my search, you can see that it says name or info right there, so that's what I'm going to type here: name or info. So this going to need a label. A label is just the text that goes with the input field, and it has an advantage of making the label clickable to the input field. So this needs to match the id, so I'll do an id of search here, and my label is just going to say livesearch. That means that when I am here, if I click on the word livesearch, because that's the label, it will activate my field. That is something really useful for mobile devices. And I'll add a little paragraph here to just describe what the user is supposed to do. So, let me save that. Finally, I need to add a link to my style sheet. And the style is called mystyle.css. I don't need a type here. Let me save that. One thing I like to do is to just put a simple rule in the style sheet to make sure that it's linking properly to the HTML file. So I'll add a body selector here, background, and just give it a background color. So let me save this and I'll switch back to my browser. I need to go to a slightly different URL here. And everything looks like it's working great. So we've done a little bit of work here, and this is pretty much it for our HTML file and our folder structure. It's a really simple page, so now we can move on to start scripting the document.

### **Searching JSON data**

Selecting transcript lines in this section will navigate to timestamp in the video

Previously, we managed to set up our application and load some content from a JSON document. In this video I am going to show you how to handle events from the input field, as well as how to use those events to drive some search functionality into our application. So we can check for an event happening when users interact with a specific element on the page. This technique is called binding the event to the element, and jQuery makes that easy. If you're using straight JavaScript, this is a good list of all the events available in HTML. If you're using jQuery, then you need to check this page for a list of the jQuery version of the same events. So we have events like change(), dblclick(), error(), et cetera, et cetera. The one that we're interested in is the keyup() event, an event that happens when somebody releases the key on their keyboard. So let me come over here. I am going to target the input field with the ID of search. Then I am going to bind a keyup element-- this is the jQuery version--and I'll execute a function literal. That function literal will have everything that we did previously. So all the stuff will go right there. Let me indent that a little bit, and I am going to save this. I will switch over to my page, and I am going to reload the page. Nothing will be showing. As soon as I click right here and start typing something in, then my JSON document will load. This is not really searching anything, but it prevents the display of the JSON data until I type something in the search box. So now it's time to narrow down our results. To do that we're going to need to be able to get the text the user is typing, so we're going to create a variable for that. Here, once again we target the div with an ID of search, and we're getting the value that is in that field. So let me go ahead and output that to the console. Let me refresh the page and as I type something in here, I should be able to click onto my console. So you can see the text that appears when I type something in the input field down here in the console. That's working pretty well. In order to do as search, we're going to use a JavaScript function called search. Search takes a string and looks for another piece of text inside that original string. If it finds it, it returns its position. The string method can search for either some text inside another string or use a regular expression. If you search for some text, it will do a case-sensitive search, which is not what we want. So when somebody types a lowercase b, it wouldn't find Barot, because that user has a name that starts with a capital B. And that's not good. Regular expressions are sort of like search on steroids that let you do case-insensitive and more powerful searches. I am going to need just a very simple regular expression, so let me do that right here. I'll get rid of this console.log, and I'll call this my expression or myExp, and that's going to be a new regular expression. That's going to take in our search field, and it's going to use the case-insensitive search if I type i in quotes right here. Now that we've got these two variables, we can use the search method like this. I am going to wrap this around the list of output items right here, and I will type in if the value of name from our data file and then execute a search passing it a regular expression. The one that we created is called myExp. If that is not equal to -1, it means that it did find that text in the data. So if it does, then I can output all the stuff. So let me go ahead and save this, and I'll go back into my application and I will reload. I will start typing something in here, and you'll see that it is producing the search result. So let me close this out so you could see it a little better. There's all my search results right now. It's pretty much everyone. So if I type in a name, like I will type Ferrar here, it finds that second field. I will just do that a little more. It is dynamic, so it's narrowing things down as I type in more and more text. That's pretty good, but right now it's only finding things in the name field. So I also want to check for the Bio field. I can just take this, and I'll add another set of parentheses here and an or symbol. Paste that other copy of it and change this to check the bio field. So this says look for the regular expressions in the name or in the bio field. I will save that, come over here, we'll refresh, and we'll try something like media, because I know some of these people have the keyword media, or maybe something like art, and now that's working perfectly. So our Live Search functionality is working great, but it doesn't look very hot, so in the next movie I am going to add a little bit of styling.

### **Styling an application**

Selecting transcript lines in this section will navigate to timestamp in the video

So far, we've managed to build a nice live search feature that uses AJAX to search through a JSON data file, but the UI could look a little bit better. So in this video I'm going to show you how to style this application with CSS. Now, I've got the set up in Espresso and this little comment right here essentially ties my live preview with this window. So as I make a change right here, it will update automatically. If you don't have Espresso, you don't really need to type that in, but it's just the way that I have things set up, so that as I type the CSS, you're going to be able to see the changes on the right-hand side. So let's get started with some basic CSS. So first, I'm going to start working on the searcharea, and I'm going to go ahead and center that by setting the margin equal to 0 and auto. I'll text-align this to the center so that the text inside will also be centered, and I'll set up a background color here, and I'll set up a padding of 10 pixels. Plus I'll set up the width to be 30% of the page. That way this is going to be responsive and kind of grow with this page. I'm going to add a bit of a border-radius here, and that will just make my edges round, so I'll use -webkit-border- radius and I'll do 10 pixels. That will take care of Safari and Chrome. And then I'll do -moz-border-radius 10 pixels for Firefox, and then I'll just use the regular rule, which is just border-radius of 10 pixels. I want to make the border just be on the top, left, and right of this box, so I'll modify this -webkit-border- radius to have some different values. So let's style the label. We're giving it some font attributes here. I'll go ahead and make the text- transform be uppercase, so that it displays that text as uppercase always, and I'll add a little bit of padding at the bottom of 5 pixels, just a little bit of breathing room. And I'll set the color to a little bit of a red tint. So now let me do the same thing with the paragraph. I'll grab this and I'll just copy it here, and instead of label, I'm going to target the paragraph. The margin is going to be nothing. Gets rid of all the spacing on the top and bottom. And then I'll do a line-height of 1 em and a padding at the bottom of 5 pixels. Again, just putting in a little more breathing room this time. This looks probably a little bit better when it's wider. So now, let's modify the input field. Once again, searcharea. And I'm going to set the width to be 80% of the size of the window, again, making it responsive. And I'll text-align this to the center. So now this little box is actually going to grow as I change the size of the window. Excellent! So now we need to change the style for the content, which is under the update id. So we'll start with some basic code to change that. I'll do a font-family, and I'm going to use Georgia and as a backup, Times new Roman. And I'll set the width of that to be 70% of the browser window, and I'm going to center it horizontally, so margin will be 0 and auto. And I want to put a border on the top, and I'll make that border dotted. And my border color is going to be CCC. That's just a little gray line at the top of the update div. So let's go ahead and change the unordered list. Right now my lists have bullets in them, so I'll do update unordered list. I'll get rid of the bullets with list-style none, and I'm going to change the margin and the padding to 0. That gets rid of a little bit of space that happens with unordered lists. So now I'll do update ul li, and I'll set the width of this to be 100% of the container. And I'll do a little bit of padding so that it has a little bit of breathing room to each side. I'll set up a background color here. You can actually see the padding better with the background color. So there's without padding, and there's with padding. I'll do a little bit of padding at the bottom, 10 pixels. You can't really see it, but that just gives you a little more breathing room between some of the list items. And I'm going to set the height, because they're a little big, to be 110 pixels. And now that looks kind of weird, so I need to make sure that I do the overflow element here and I set it to be hidden. That hides everything that's bigger than 110 pixels. And we'll do also a little border at the bottom, so that we kind of visually tell the user that there is going to be more information underneath this little border that we placed. All right, so when users hover over the list item elements and when I have a style that kind of expands what I've done, so my background is going to change, the background color. I'll set the height to 250 pixels. That's going to show more of the content behind. So if somebody hovers over one of these elements, it kind of expands this to be bigger right now. It's still sort of funky, but it will work really well when we have the rest of the code in. Let's do the update h2 tags. And I'll do margin. I'll just clear out the margin and padding and then set up a font-size, a little bit of padding at the top and bottom, and we'll change the font here. Now, we'll change the color, kind of a little bit of a red, sort of maybe a coral color, and we'll add a border and a little bit of the margin to give the border a little bit of room right here. So now I'm going to do the image. I'll float the image to the left. That way the text will show to the right of the image. And I'm going to set up the width of the image to just be 80 pixels so that it's not so gargantuan. I'll add a little bit of margin to the right, 10 pixels, just to give it a little bit of breathing room from the photo, and I'm going to make the photos have a round edge, so I'm going to type in -webkit-border-radius, 10 pixels, and I'll do the same thing for Mozilla browsers, and for just the regular version CSS. So let me save that, and that's looking pretty cool, and it's mostly accomplishing what we need here. You can see that this looks really good, especially when it expands. So our application is looking pretty good, and this would probably be good enough for most projects. However, in the next movie, I'm going to show you how to use some CSS3 animations to improve this app.

### **Adding CSS3 animations**

Selecting transcript lines in this section will navigate to timestamp in the video

Our application is looking pretty good and it works well, but I think we could spice it up with some CSS3 animations. So, this won't necessarily work in some older browsers, but I think it adds a little bit of pizzazz to our application and makes it degrade nicely in older browsers. So at the very bottom of our CSS, we're going to add some animation keyframes. We'll need to add them with different browser prefixes. The prefixes are pretty similar, so we'll just do one prefix and copy and paste and then modify it for the other prefixes. So I'll start with webkit, because I'm in a WebKit browser. The way this works is you defined a set of keyframes and then you give them a name, inside this keyframes keyword. So what we want to do here is start our animation at the beginning position, or the 0% position, with the opacity of the object being at 30%. And then at the end of our animation, which is 100% of the keyframe, our opacity will be set up to 1.0, which is full opacity. So what this will do is it will just fade in the elements that I am targeting with this animation. So I have the webkit prefix going. Now, all I need to do is copy that and paste some other copies and modify the prefix for other browsers. So now that we have these keyframes, we can call this animation sequence within our list item. So I'm going to scroll up a little bit and find the place where I do the list items and add some code right here that call in our animation. So, for that, I'll do webkit- animation and then I'll call in my animation (my anim) and make it run for one second, and I'll need to do that with other prefixes. So let me go ahead and save that, and I'll refresh this page, and you can see that these elements right here are fading in as I'm typing in some of these search fields. So, another thing that I want to do is make this expanding animation be little more fluid, so for that, I'm going to use something called transitions. And transitions work sort of like animations, except that they happen when an element changes. So I have got a hover event here, and I want to set a transition that happens when the element, it's in its normal state, and when somebody hovers over the element. Now, I need to just put the transition here in the original version of the element, so I'll do webkit and not animation, but transition. I only want to animate the height element of the transition, so I'll need to tell it how long I want that to last-- that'll be 0.3 seconds--and then an easing-in and easing-out function. I'll just to use ease-out, so that makes it slow at the end of that transition. And let me delete this and just paste this a few times and add the other browser prefixes here. So I'll save that. Let me refresh this, and I'll type in this name here. You can see that when I type in a search that results in multiple fields, you can see that when I roll over some of these elements my transitions, are a lot smoother. If you want to learn more about transitions and transforms, make sure you check out this course from Joseph Lowery in the library.