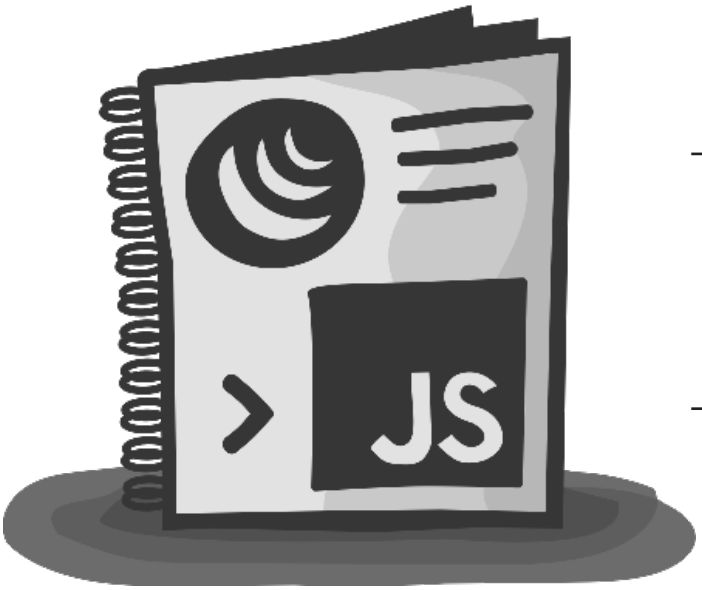


GYMNASIUM

THE JAVASCRIPT & JQUERY SURVIVAL GUIDE

Lesson 3 Handout

DOM Manipulation

ABOUT THIS HANDOUT

This handout includes the following:

- A list of the core concepts covered in this lesson
- The assignment(s) for this lesson
- A list of readings and resources for this lesson including books, articles, and websites mentioned in the videos by the instructor, plus bonus readings and resources hand-picked by the instructor
- A transcript of the lecture videos for this lesson

CORE CONCEPTS

1. JavaScript allows you to change the CSS or style properties of DOM elements, and jQuery makes it quite easy to do this. The easiest way to do this is using the jQuery *css method.*

2. You can create a new embedded style sheet or reference an external style sheet dynamically with jQuery.

3. The jQuery function has the ability to examine the arguments it receives and understand what you want to do. For example if you want to query every image with the class named “logo”, jQuery allows you to do that and then do something else to those images.

4. “Injecting” HTML content into a page simply means that jQuery has the ability to pass HTML elements (such as a button) into your page . The *append*, *appendTo*, *prepend*, and *prependTo* methods are the primary ways to do this.

5. In addiiton to the ability to inject HTML conten into a page, jQuery can also remove an element or elements automatically. Appropriately enough, the *remove* method is primary way to do this.

6. jQuery is used frequently to create animations within a page. The *animate* method is the primary way to do this. In addition, there is a mechanism available called a *completion callback*, which is a way to control what happens once an animation has completed.

7. Understanding how an HTML page loads and the connection between where your scripting code is placed is very important with both jQuery and JavaScript. Code which is placed in the wrong place can sometimes affect the user experience in a negative way. The jQuery *document.ready* method allows you to make sure that your jQuery code, is not executed until the document is ready

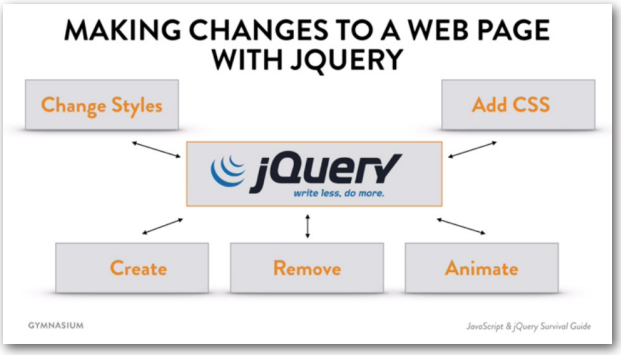
ASSIGNMENTS

- Prepare a final page for production to ensure that your code is placed in the correct location and optimized for users of the page. This process relies on correct usage of the document.ready method as well as putting your code into an external JavaScript file and referencing it in your HTML document. *See pages 24–26 for detailed step-by-step instructions.*

INTRODUCTION

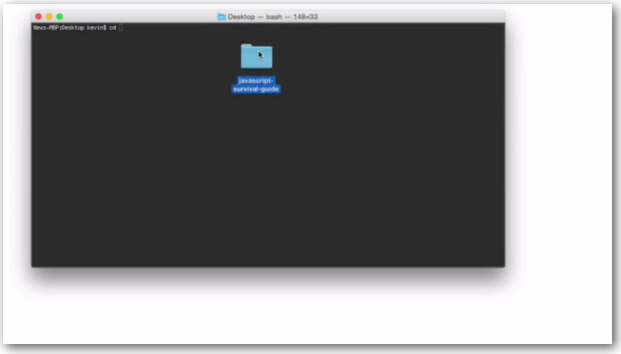
(Note: This is an edited transcript of the The JavaScript & jQuery Survival Guide lecture videos. Some students work better with written material than by watching videos alone, so we're offering this to you as an optional, helpful resource. Some elements of the instruction, like live coding, can't be recreated in a document like this one.)

In this chapter, I'll provide an introduction to making changes to web page elements with JQuery. By the end of this chapter, you'll understand how to manipulate the appearance of a web page by changing styles and adding CSS. You'll also have the skills you need to create DOM elements, remove DOM elements, and add animation to the page.



I'll continue to let you know how to locate each code example in your local development environment so that you can follow along with the videos. For this chapter, we'll be working with the file Making Changes to a Web Page with JQuery.js, which is in the Code Examples folder found in the root of the GitHub repository.

In case you've turn off your computer since the last video, I'm going to restart my local web server and also show you how to access the code examples for this chapter. In my terminal, I'm going to type CD space and then drag my local repository folder over to the terminal, hit Enter, and now I'm in the right folder. Then I'll simply type gulp space serve, and hit Enter. And that'll start my local web server.

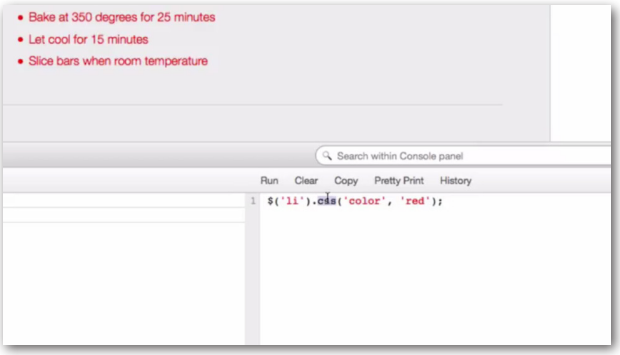


I'm using Sublime Text as my text editor. Your text editor may work differently, but in my case I can simply drag the local code repository folder over here and see all the files in the folder. If you open the Code Examples folder, we're going to be working with a file named Making Changes to a Web Page with JQuery.js. Those are the code examples for this chapter.

Now when you open your browser, if you haven't done so already, simply navigate to local host colon 5000. You'll see our example web page. And then open your Firebug console by clicking the icon in the upper right hand corner.

USING JQUERY TO CONTROL CSS STYLES

When it comes to making changes to a web page, changing the appearance of one or more elements in the page is a fairly common task. And JavaScript allows you to change the CSS or style properties of DOM elements, and jQuery really makes it quite easy to do this. If you look in your Code Examples file, look for the code example labeled “make all list items red.” Copy that code, and paste it in your JavaScript console and run that code. You’ll notice that all the list items turned red. That happened because we said, for all the list items on the page, make the CSS color red.



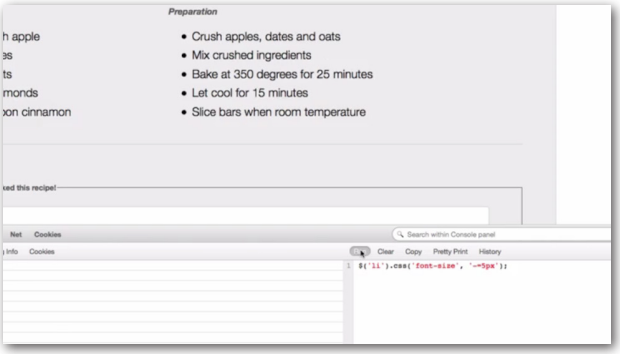
The jQuery CSS method takes two arguments, two strings. The first string is the property that we want to change, and the second string is the new value for that property. Here, I said “red.”

I could have easily said “blue.” I could have said “green.” And I can also use hexadecimal values.

Any CSS property or value that would be valid in a CSS file in most cases will work just fine with the CSS method. Now, we can change other properties besides the color of the text. If you go to your Code Examples file and copy the code example labeled “make all list items 50 pixels font size,” paste that code in JavaScript console, and execute it, you’ll see the code grew to 50 pixels in size. And that’s pretty big and it’s pretty ugly, but I just wanted to illustrate that we can change the font size as well and 50 pixels really drives the point home.

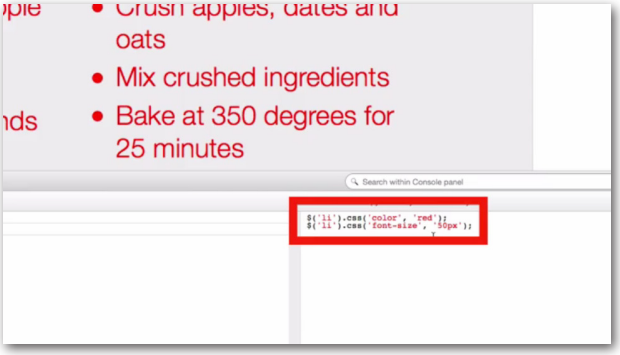
But instead of setting an arbitrary value, we can use relative values. So in your Code Examples file, copy the code example labeled “decrease font size by 5 pixels.” And paste that in here and run that code and watch what happens.

Each time I execute the code, the font size goes down by 5 pixels. That’s because we’re using this negative equals before the pixels. So instead of saying 5 pixels, we’re saying negative equals. So we’re saying, hey, for every list item in the page, make the CSS font size property 5 pixels less than it is now. It’s a relative value.



So I could easily change the negative to a positive and say make it 5 pixels bigger, and it increases the relative value. So it’ll increase. I don’t have to use 5 pixels, so I can say make it smaller by 1 pixel and it goes down really gradually. Or I can say make it bigger, by, let’s say, 20 pixels, and that’s really going to jump a lot with each click. Really getting bigger, but just illustrating that you can use relative values instead of arbitrary values when changing the font size for a DOM element.

Now, back in the Code Examples file, if you copy the code example labeled “change to CSS properties,” let’s refresh the page and get rid of our giant font. Paste this in here. When you execute the code, you can see that we changed two properties. We changed the color property to red and the font size to 50 pixels, and that’s perfectly valid. And it’s OK to do that, but it’s kind of inefficient and I think there’s a better way we can do this.

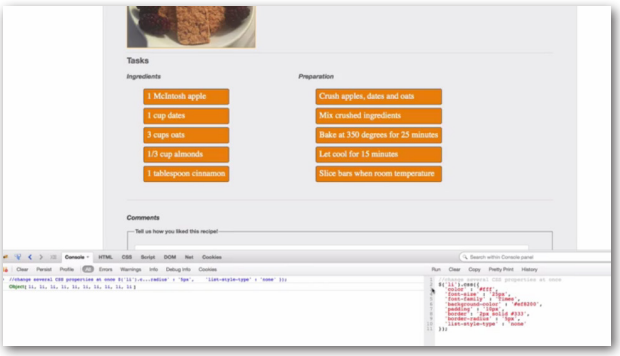


Let’s reset the page and go back to the Code Examples file and copy the code example labeled “use an object to change multiple CSS properties.” So when I paste that code in, what’s happening here is we’re passing an object instead of two strings. So if we look at these examples back-to-back, look at the previous example. And then we look at here, what we did was originally we called a CSS method and passed it two strings as arguments.

Now, we’re passing in an object with multiple properties. The CSS method knows how to look at the argument it received and determine how to process it. So when it receives two strings as arguments, it says, oh, the first one’s a property. The second one’s a value.

I’ll set the color to red. When it sees that it receives a sole object as an argument, it says, oh, for every property of this object, I’m going to change the CSS style of each of these elements—this property that value. So you can pass it an object, which is a little bit more efficient way of doing things.

Now, in this case we changed two properties, but we can certainly change more. If you go back to the Code Examples file and copy the code example labeled “change several CSS properties” and pass that in, you’ll see that we’re changing, I think, about seven or eight properties here. And it really changed a lot. The buttons look completely different.



So we could put as many as we want here. We could put 15 or 20 here. It doesn’t matter. We can put as many as we want, change as many properties as we want.

Now, refresh the page, and go back to one more code example for the CSS method. Copy the one labeled “define an object and then pass it to the CSS method.” So we’ll paste this in, and look what’s happening here.

First, we’re creating an object. It’s a variable that’s an object, and an object is just, once again, an object with properties and values. And then we pass that variable, which represents the object, to the CSS method.

So the end result is exactly the same. I we run the code, just the list items look like orange buttons again. But what we’re doing is creating an object first, and you may have a case where you create—let’s say this is called “orange buttons.” And then you have another object called “blue buttons,” and then you may want to ask the user—do you want orange buttons or blue buttons?

And then depending on their answer, you may pass them orange buttons and you may pass blue buttons to the CSS method. But defining an object outside of the CSS method could set you up to write some little more sophisticated code and do things in kind of a smarter way that allows you to forward your code and have IF condition or whatever. But it's helpful to know that, when using the jQuery CSS method, you do have a few options because that method can take either two strings as arguments or a single object and you can organize your code in different ways to set up how you want to change all the different elements in the page and how they look.

In the next chapter, we'll look at a technique similar to the one which is covered. But in this case, you'll be learning how to dynamically create an embedded style sheet. See you soon.

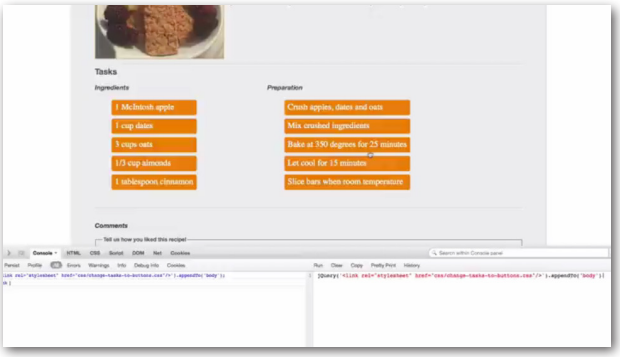
CREATE A CSS STYLESHEET DYNAMICALLY

While JQUERY CSS method makes it very easy to style DOM elements, there may be a couple of scenarios where it's not the best approach.

Let's say there's a case where you want to add hundreds and hundreds of lines of CSS to the page. You don't just want to modify a few elements or a few dozen elements, you want to add literally hundreds of lines of CSS to the page. In that case, you could create a new style sheet or reference an external style sheet dynamically.

In the code examples file, copy the code example labeled Create a Stylesheet Dynamically and paste that in your JavaScript console. Now what I'm doing here is I'm creating a link element which, as you know, is the actual element you use when you reference an external CSS file. I'm creating it and then I'm injecting it into the page.

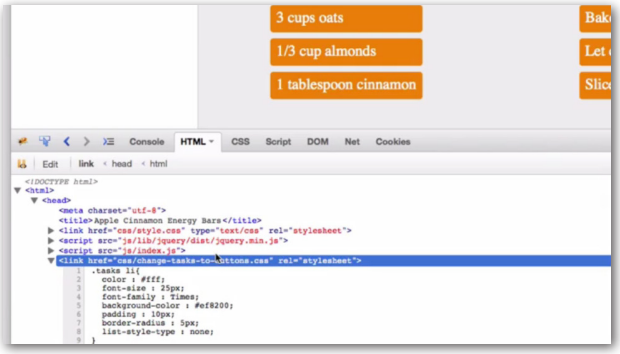
We're going to learn about creating DOM elements in a few minutes, so I don't want to spend too much time on this, but what I just want to illustrate is that I'm going to create a link element which references an external CSS file.



Now the path is a CSS folder and is a file called Change Tasks to Buttons.CSS. So if I look in the CSS folder, www and then CSS, there is in fact a CSS file called Change Task to Buttons.CSS. So we see that the file exists.

So if I look at what's about to happen here, some CSS should be injected into the page, which changes the appearance of something. So when I run this code, we can see that the list items change dramatically. And if I right mouse button click on the page and click Inspect with Firebug, I can see that the body has a link element, and that link element injected a bunch of CSS into the page.

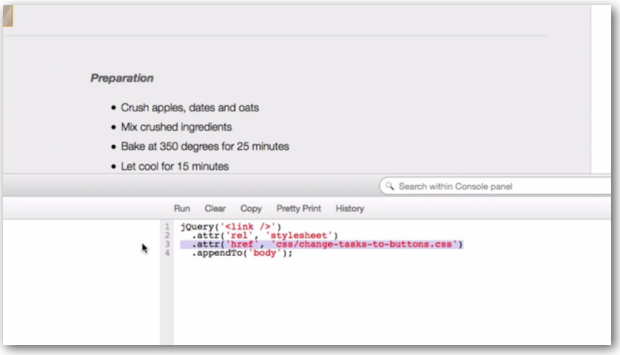
Now technically I should have injected it into the head, not the body, but the point I'm making is you can create a link element on the fly dynamically that references an external stylesheet. In fact, I can refresh the page here and just say append to head. So now when I run this, the CSS changes. When I inspect the page I look in the head and the head has this new link element with the CSS.



So again, this would be a case where you want to add a lot of CSS and maybe you don't want to deal with the CSS method. It would create a lot of JavaScript, and it could actually kind of slow down the page depending on what you're doing. But in this case, it's just one line of code and it injects a tremendous, or could inject a tremendous amount of CSS.

Now if you refresh the page and look at your code examples folder, there's another code example labeled ADD A One at a Time. So if I copy that and come back here, here you'll notice that I'm creating the link element in one shot. I'm saying rel equals style sheet, hf equals, and then create and append it.

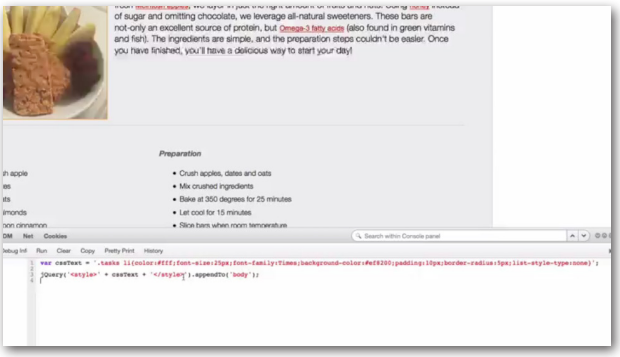
In this example, I create a link, I use the attribute method to add the attributes one at a time, I add the rel attribute and the hf attribute and then append it to body, here I'm going to say head. That's the better way to go.



I want to execute this code. It's the same exact effect. It's just that the code's a little bit cleaner, a little bit more elegant, a little bit easier to read.

But the bottom line is that creating a link element that references an external CSS file is possibly a helpful technique. When you want to introduce a large amount of CSS into the page, you don't want to do it when the page loads. But you want to potentially do it at some point based on a user action or some event—and that this is a much more efficient way of doing it than doing it with the CSS method because you'd have to touch possibly many, many elements which could actually affect the performance of the page. And this would have virtually no effect on the performance of the page.

In this case, we're injecting maybe 10 lines of CSS. But it could be hundreds of lines—so just good to be aware that that technique is available should you run into this situation.



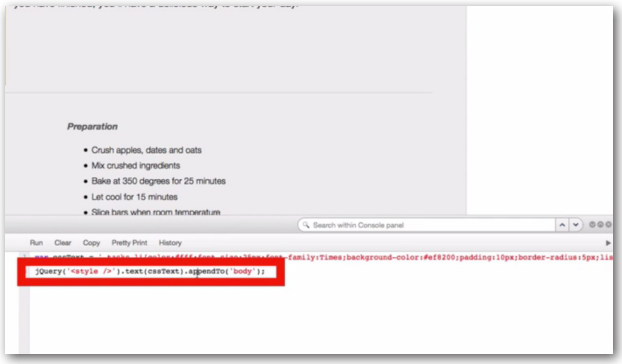
There's another technique that's very similar to the previous one that allows you to inject CSS into the page without referencing an external CSS file. In the code examples file, copy the code example labeled "add an embedded style sheet dynamically."

Paste that here. And things a little bit different here. What we're doing is first taking a whole bunch of CSS—this is just CSS, CSS just like you would see in a CSS file—assigning it to a variable, and then we're creating this style element. And then we're setting the CSS text or this CSS right here as the text property of that style element and then appending it to the body.

So a style element is—creates what's called an embedded style sheet. It's different than changing the properties of the DOM elements directly. And it's different than referencing an external style sheet. We're creating a style tag and injecting it in the body, and that's called an embedded style sheet.

So when we run this, we get the same exact effect. We get the orange button. But the difference here is that we've appended a style element to the body. If you right mouse button click on the body and click Inspect with Firebug, you'll see that on the body, there's a style element. And then here is the CSS text that we injected.

There's a second example in the code examples file labeled "make embedded style sheet more expressive." You copy that, refresh the page, and paste this new code in. And you'll see it's doing the exact same thing. It's just the code's a little bit more readable.



We create the element and then use the text method to set the text property, then append it to the body. And in fact, we can even break these lines down like this to make it a little bit easier to read. Some people say it's more expressive.

We'll run this code. The end result is exactly the same. So this is similar to the previous example in that it's probably something you'd want to do when you want to introduce a large amount of CSS. You want to do it in one shot. You don't want it on page load, but you want to do it based on some situation, like if the user clicks something or some other thing happened.

The difference here is that we don't need to reference an existing external style sheet. We can simply create the style tag, jam a whole bunch of CSS into that tag, and then inject that tag into the body. It's called an embedded style sheet.

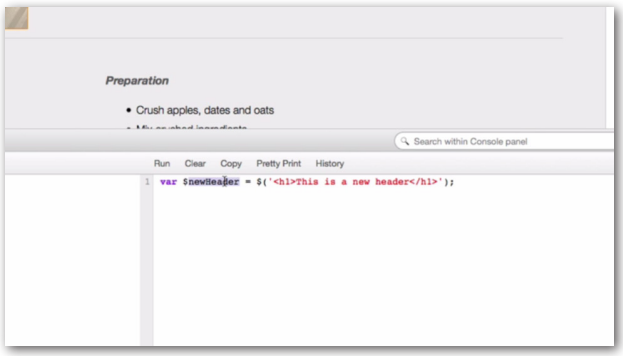
It's probably, again, a scenario that—a situation that—you might run into less than using the CSS method. But it's really good to know that it's possible. I've used it many, many times. And when it's the right technique, it's super helpful.

In the next chapter, we're going to continue on this theme of injecting content into your page using jQuery. But instead of adding CSS, we're going to turn our attention to HTML and learn how to add content such as headings into our page. See you soon.

INJECTING CONTENT: PART ONE

In the next two videos, I'll walk through the stages of how to use jQuery to inject HTML content into your documents. This is Part One, where we'll create and store the content we want to use. And in Part Two, we'll actually inject that content.

In the code example file, copy the code example labeled "create a new header element" and then paste that into your JavaScript console. So let's look at what's happening here. We have a variable called newHeader. And we're passing a string to the jQuery function.



Now, let's line this up with a previous example. A lot of the stuff we were doing was we were doing things like this. We were targeting every list item or every paragraph inside of a list item or every paragraph with a class error.

But all the syntax was very CSS-like syntax that we were passing to the jQuery function. And then we were just doing things with whatever elements that were returned.

Well, here, this example's a bit different because we're passing what looks like HTML. This just looks like HTML. It's a h1 tag with some text inside of it.

So this illustrates one of the most amazing things about jQuery—is that the jQuery function has the ability to examine the arguments it receives and know—understand what you want to do. In the previous example, we—jQuery said, hey, you're passing me what looks like a selector. So you want to query every list item in the page that has a paragraph tag, or you want to query every image with the class logo. Fine. It returns a list of elements. And you can do things to it.

Well, in this case, we're passing it literally markup and jQuery says, you're passing me what looks like HTML. I think you want to create something. I think you want to create, in this case, an h1 tag with the text, "This is a new header."

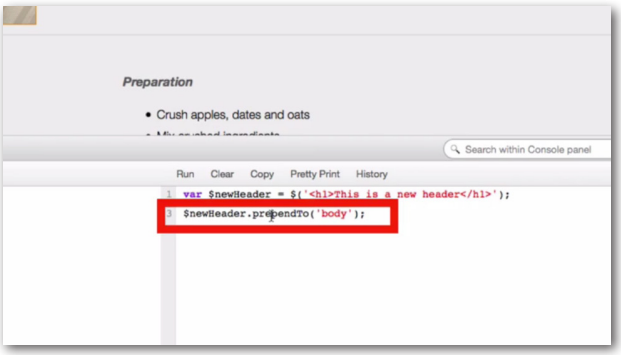
So this variable, newHeader, becomes an object. It's a jQuery object. It's an actual DOM object wrapped in jQuery that we can put in the page. But when I run the code, nothing happens. It doesn't appear to happen. I don't see anything change in the page.

And that's because we created this new DOM element, but we didn't do anything with it. It just exists in memory. We can prove that by using the console.dir statement. If you remember that statement from a few examples ago, we can use that to inspect an object.

So if I run console.dir and pass it this new header variable, you can see that, wow, that's—it really is an object that exists in memory. In fact, it looks familiar to us. Here's the text. "This is a new header."

And it's got other kind of properties that may look familiar to us, like tagName is h1. And we can see the outer HTML here, that's the element that we created. It's just that it exists in memory. It doesn't exist in the page yet. We need to actually put it in the page. So that's what we'll be doing in Part Two. See you in a moment.

INJECTING CONTENT: PART TWO



So refresh the page. And back in the Code Examples file, copy the code example labeled "inject the new header into the body." Paste that code in. And on the first line, it's exactly what we just did—creating a new variable with the new h1 element.

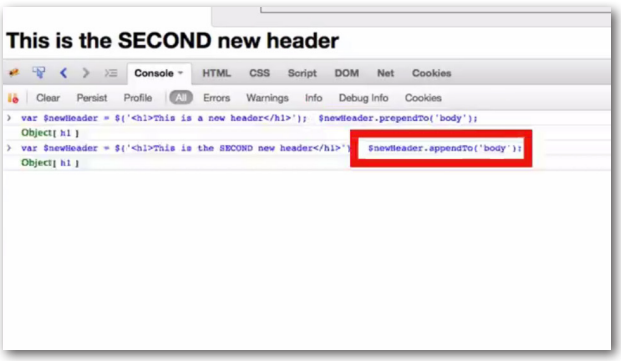
Then we're using a property called "prependTo." That's a property of this object. And what that's saying is make this object the first child of, and then we're passing it body. So we're saying make this new object or this h1 tag make it the first child of the body tag.

When I run the code, I see the new header. It's in the page, so something's working. When I inspect it, you can see that this new h1 tag is, in fact, the first child of the body tag. So the prependTo method allows you to make something the first of or, in this case, the first child of something.

If I were to change the prepend method to appendTo, I'd be saying make this the last child of the body. So if I'm going to make this—let's say this is the second new header. If I run this code, I look at the bottom of the page, there's the new header.

And if I inspect it, I can see that it's the last child of the body. So prependTo allows you to make something the first child of, appendTo makes it the last child of, but either method allows you to actually inject it into the page. We can create the new object or the new element, but we need to actually put it in the page.

So refresh the page. Go back to the Code Examples file, and copy the code example labeled "make new header injection code more expressive." So I paste this code in. And when I run it, the same thing happens. I see the new header.



What's different is, instead of just creating the variable first and then prepending it, I'm just creating it and then prepending it. So it's just the code's a little bit more efficient. It does the exact same thing. But instead of creating a variable and then referencing the variable, I'm just creating the element and injecting it right in there—all in one shot. So that's how to dynamically add a heading into your page.

But what about some other HTML elements? In the next chapter, you'll learn how to add images and then a button. See you shortly.

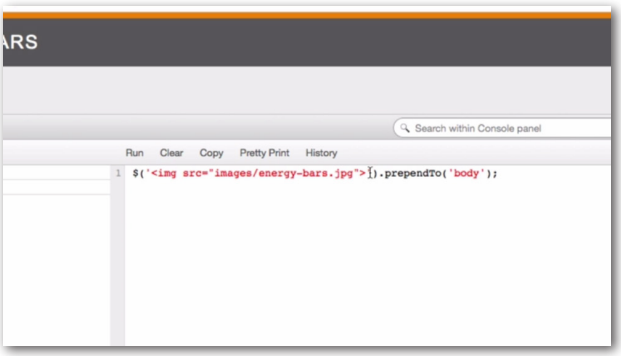
DYNAMICALLY ADDING CONTENT TO A PAGE

In this chapter, you'll take a look at three examples of dynamically adding content using jQuery. First you'll take a look at how to add an image. And then you'll learn how to add a button. In the case of the button, not only will you learn how to add the button element itself, but you'll learn how to make that button control other elements on the page. The third example will demonstrate how to add an entire block of HTML, not just a single element.

So refresh the page. Go to the code example file labeled "inject a new image into the page." Here, if you r`un this, you'll see a giant image. And that's the main recipe image.

It doesn't look too good. We didn't do it in a very smart way. But I'm just illustrating that here I'm creating an image tag. I don't have to create headers. I can create any kind of element I want.

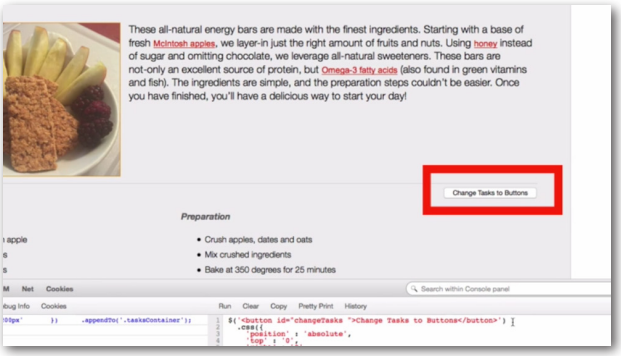
Here's the markup for what would normally be an image tag. In fact, if we look at the actual original image tag, it's no different. Here's the img tag, src attribute, images/energybars.jpg. And if you look at the markup we created, it's virtually identical. In fact, I copied it. So the point is we can create any kind of element that we want.



Let's create a element that's a little bit more useful than just an odd header or a giant image. So back in the code example file, if you go to the code example labeled "add a button," paste that in, and when we run that code, we see that a button gets added to the page.

Well, the button doesn't do anything. And it's not what I want. I want the button in the upper right-hand corner here. So we have a little bit of work to do. The code example labeled "add a button with custom styling" is probably going to help us. So I'm going to copy that code, paste it in the JavaScript console.

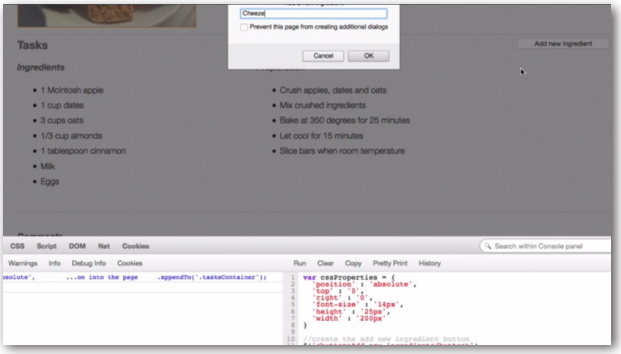
So now when I run the code, I get the button and I get it exactly where I want. And that's because I create the button, I pass it the CSS method with some CSS, and then I append it to the task container.



So I'm just creating the button on the fly, styling it really quick, and jamming it into the page. And I've got the button exactly where I want it. So I'm making progress here.

So back in the code examples file, copy the code example labeled "allow the user to add a new ingredient." So I'm going to copy all this code here and paste it in here.

And so when you run the code, we get the button. And when I click the button, it says, “Add a new ingredient.” I’ll type “Milk.” And all of a sudden, there’s a new ingredient added to the ingredients list. I can type it again and say “Eggs,” and do one more and say “Cheese.”

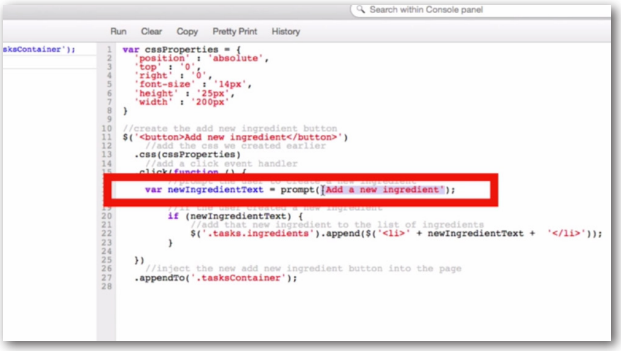


So the point here is that we’re creating a new element on the fly every time. In fact, we created a new element. We created a button. We injected it into the page. And then every time the user creates a new ingredient from the prompt, I create a new list item in the ingredients list.

So let’s look at how we’re doing that. First, we’re creating the CSS properties object. We’ve seen this before. It’s just an object with some CSS keys and values we’re going to use later.

Then we create the new button. We pass it the CSS properties. So now we’re using this object to style the button. And then we’re creating a click event handler for that button.

And whenever the button is clicked, we’re going to create a variable called newIngredient. And that ingredient gets the result of the prompt function. The JavaScript prompt function creates a little pop-up window that asks the user for input. And you can pass it a string that says—a custom message saying, “Tell me your age,” or, “Where do you live”—in this case, “Add a new ingredient.”



And that prompt returns a value. It returns whatever the user entered. So this newIngredient text variable contains the text the user entered. Like when I entered “Milk” and “Cheese” and “Water,” they were applied here.

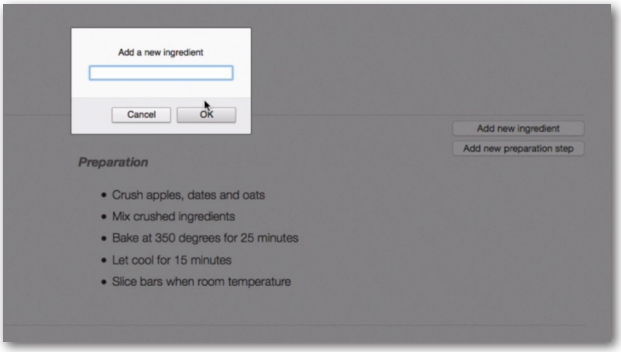
So if the user entered something, then we’re going to create a new list item. So we say here’s some markup we’re passing to the jQuery function. It’s kind of markup. It’s a little bit of a mix because first, it’s an opening list item tag and has a closing list item. And there’s this newIngredient text. That’s the text that I entered, like “Milk” or “Eggs” or “Cheese” or “Water.”

And then we’re appending it to the ingredients list. And then we add that button to the page. So the code is starting to get a little more complicated here. But the point I’m making is that we can create DOM elements on the fly over and over, day and night. It’s trivial. We can do it as much as we want.

And if I keep clicking this button and adding more ingredients, we’re going to keep adding more list items. So creating DOM elements is very easy. And it’s something you can do programmatically as part of your code.

Let’s take things a little bit further with this feature. Copy the code example that says, “allow the user to add a new ingredient or preparation step.” So I’m going to copy all this code and refresh the page and paste this code in and run the code. And let’s see what happens.

So now I've got my new ingredient button. I can click that. And I can type "Milk" and "Eggs" and add new ingredients. And I can also say Add new preparation step. And I can say "Boil Water" or "Cook Eggs."



So I can add ingredients or preparation steps. All we really did is some copy and paste here where this code we just went through—this is the code that adds the Add new ingredient button, adds a click handler that allows you to add a new ingredient.

This code is virtually identical, but it's the Add new preparation step button. But let me point out to you something that just happened here which is kind of cool. We started out by creating this CSSProperties object and then we passed that object or the variable reference to it to the CSS method when creating the new ingredient button.

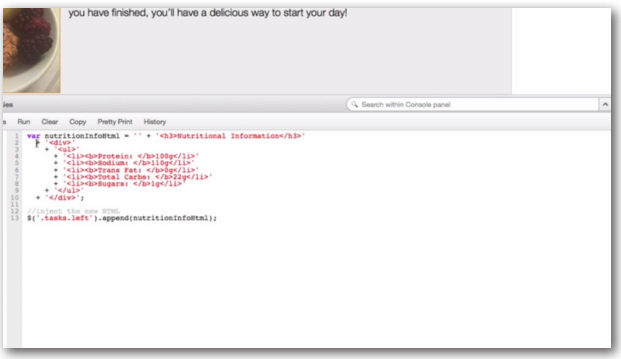
Well, when we created the Add new preparation step button, we did the exact same thing. So we reused this object. It's a perfect example of why that technique is really useful. It's really helpful to know about it because if you're going to be doing a lot of CSS styling in your web page with a CSS method, if you're going to create a lot of the same styles over and over, put those styles in an object and then pass that object to the CSS method multiple times.

So we just reused a bunch of code here. And there's nothing more beautiful than code reuse—not repeating itself. We call the CSS method again on the Add new preparation step because we want to just add a 30-pixel top property to that button so it appears a little bit lower than the Add new ingredient button. We don't want it to overlap it. So we actually call it again.

But 90% of the CSS for that button is identical. It's exactly the same. And we reused that object when doing this. So this is an example of how you can create DOM elements on the fly pretty easily. And you can style them and inject them into the page. And it's the append, appendTo, prepend, and prependTo methods that make this whole thing really just pretty easy.

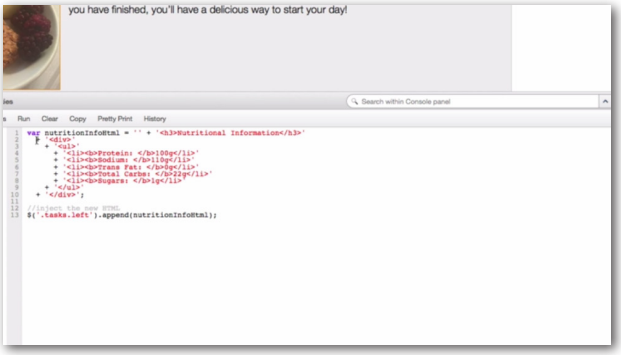
There's one last scenario I want to cover when it comes to creating new DOM elements. In the code example file, the code example labeled "create a large block of HTML"—copy that. Refresh the page. Paste that code in. And take a look at what's happening here.

We have a variable call nutritionalinfohtml. And that is just a string. It's a long string that contains HTML code. And it's just strung together with a bunch of plus signals. We could easily have just gone like this and made it one really, really, really long string. But that's kind of ugly. And it would be hard to read.



So we did it like this because the code is laid out in a way that's really easy to read. It actually looks like markup. It looks like markup that we would write. The only difference is every line is encased in quotes and it's preceded by a plus.

But it's really easy to see that you've got all these list items. They're nested inside of an unordered list, which is nested inside of a div. And it looks just like markup. So we create that markup. And it winds up being a string. And we pass that string to the append method and inject it into the page.



And now when we run the code, we should see a new unordered list right under Ingredients. And that's exactly what we see. So it's pretty cool because the JavaScript code that we're creating kind of looks like HTML. And it's really easy to work with.

If we wanted to add a new list item, we could just copy and paste and say something like that. And if we wanted to double the list, we could just chop out all of that code and paste it in right here.

And it just makes it so much easier to work with the markup you're creating in JavaScript. It's not the only way to go. When you're injecting a large block of HTML into the page, in some cases, it's the better way to go because if you created every single one of these DOM elements individually, you'd injecting each one of them individually, possibly, which would slow down the performance of the page.

But in this case, we're just creating a bunch of HTML and injecting it one time, which from a performance standpoint is probably better. It's really up to you. But it's helpful to know that you do have some options and a few different ways to go when it comes to creating DOM elements and injecting them into the page with jQuery.

So we've spent some good time learning how to add content dynamically. In the next chapter, we'll turn everything around and learn how to remove content dynamically.

REMOVING ELEMENTS WITH JQUERY

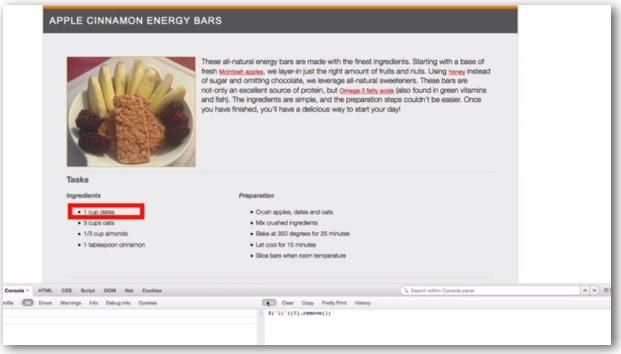
In the previous video, we spent quite a bit of time talking about how to create DOM elements and inject them into the page. But if you think about the opposite end of that scenario, you think about removing elements from the page. Fortunately, JQUERY makes that really, really easy.

If you look at the code examples file, copy the code example labeled remove an element and paste it into your JavaScript console. Now, here we're targeting every list item in the page and we're saying for every list item in the page, remove it. So when I run this code, every list item disappears.

It worked. It was exactly what we intended, maybe it was a little bit of overkill. So let's try to back it up and be more specific about what we want to do here.

So copy the code example labeled remove list items one by one. Paste that code in, and then when I run this code you'll notice that the first of all the list items—not the first item in each list—the first of all of them, which in this case is one Macintosh apple will disappear.

OK, now the first one is one cup dates, and that disappears. Now the first is three cup oats, and that disappears, and so forth. And that's because the JQUERY function always returns a list of elements. There could be no elements, one element, or 1,000 elements, but it returns a list of elements when you query.

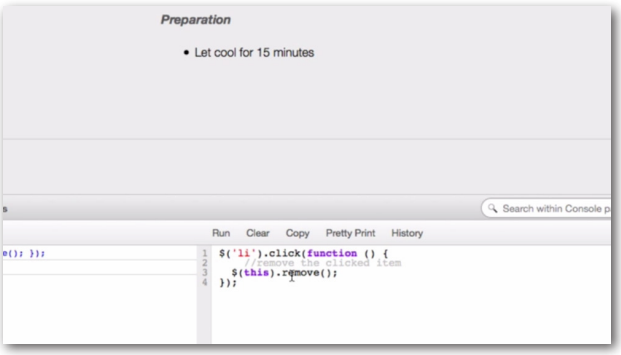


So Here we're are saying of all the list items in the page, we want the first one to be removed. Now, the reason we use zero is because this is a zero-based list. When you have a zero-based list, the zero item is the first item, the one item is the second item, the two item is the third item, and so forth. It's a little confusing sometimes, but just remember that zero is the first number.

So we're saying remove the first one of all the list items in the page. I could easily change this to a three, and that would say remove the fourth one, which I think should be mixed crushed ingredients. So if I click this, mixed crushed ingredients disappeared. That was the fourth item.

If I change this three to a one, that means that the second item should disappear, so that should be one tablespoon of cinnamon. And it did, so that's great. So now I can remove one item at a time, but it's still kind of unrealistic because we're never going to ask the user to execute JQUERY for us [? in ?] the console. We're going to want to remove an item possibly based on the user action. We're going to want to write a click handler that says when something happens, remove an item.

So let's refresh the page and go back at the code examples and copy the code example labeled remove list item on click. Paste that in, now run the code, and click any list item. You'll notice that no matter which item you click, you could just click around, it doesn't matter what the order is. Whatever you click, that one is removed.



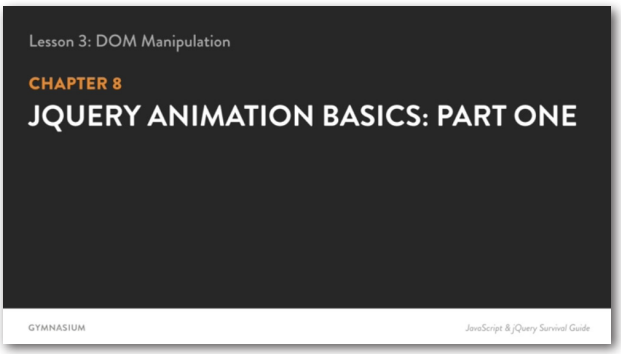
Why? Because we're saying for every item in the page, if it's clicked, this, the element that was clicked, remove. So we're calling the remove method on the element that was clicked. This is probably more along the lines of something you might want to do.

Here, in this case, we're saying hey, there are all these tasks, and if you finish a task, if you have a Macintosh apple, you have one cup of dates, you have sliced the bars [INAUDIBLE] room temperature, just click it and that was from the page. That removes it from your list. That's a very simple example, but it's a little more practical than just saying remove the first one.

But the bottom line is that while creating DOM elements with JQUERY is certainly very easy, removing them is just as easy and the key is the remove event. You want to call the remove event on the element that you want removed from the page.

In the next chapter we're going to switch gears and begin exploring another area that JQUERY is used for, and that's animation. I'll see you in a bit.

JQUERY ANIMATION BASICS: PART ONE



In the next few chapters, we'll take a look at the topic of using jQuery to add animations to your page. Specifically, we'll be exploring the basics, such as how to move and scale an image, as well as introducing a concept called completion callback, which is basically a way to control what happens once animation has completed. In your code examples file, copy the code example labeled slide the image container down 500 pixels. Paste that code in your JavaScript console. And let's just kind of talk about what's happening here.

So first, we're targeting the element with the class of image container. And we're calling the CSS method and passing it an object with some properties, some CSS properties that we want to set. We're setting the position relative and the top to zero. That has no visual effect. The element will still remain where it is. But we'll need that—it's kind of setting us up for what we'll need later.

This line we're going to come back to in a couple seconds. Now, the setTimeout—in the setTimeout, we're saying in 250 milliseconds—or in one quarter of a second—we want to apply another CSS change to the element with the class image container. We're saying, make the top property 500 pixels. Now normally, that would have a pretty jarring effect visually. The image would go from having a top property of zero to having a top property of 500 pixels. And it would happen instantaneously, and it would just kind of appear 500 pixels lower.

But back up here, we're going to make sure this happens gradually. We're going to leverage something called CSS transitions. Now, I don't want to get into an in-depth discussion of CCS transitions. It's a little bit out of the context of this course. But I do want to explain what's happening in this line.

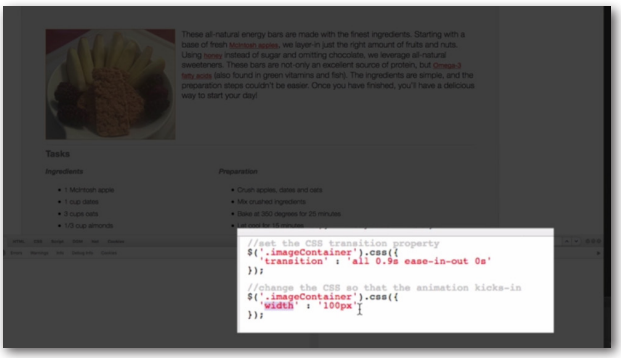
So we're saying the transition property of this element, the CSS transition property, should be set to this value. And this value comes in a few parts. The first part is the word all. We're saying transition all transitional properties. For example, some CSS properties cannot be transitioned. Font family, for example. The font family could be Times, or it could be Arial. But we can't transition from Times to Arial. The font family is either Times or it's Arial.

So we're saying all properties that could be transitioned, we do want to transition it. We want the transition duration to be 9/10 of a second. This could be 10 seconds or 1,000 seconds, but we're saying in this case the duration of the transition should be 9/10 of a second.

We want to ease in and out. And that just means we want to kind of hesitate, apply a transition, and then hesitate. And then this last part of the value says zero seconds. That means don't delay. We could delay 5 seconds or 20 seconds or 100 seconds. But in this case, we're saying don't delay. Apply the transition right away.

So then the end result of this is that the image should slide down 500 pixels. Normally, again, it would just appear 500 pixels lower. But we're using the CSS transition property to say make that change or make the CSS top property change—transition it gradually. So I'll run the code. And we can see the image slid down 500 pixels. And it happens gradually because the CSS transition was applied. And it said make that change take 9/10 of a second

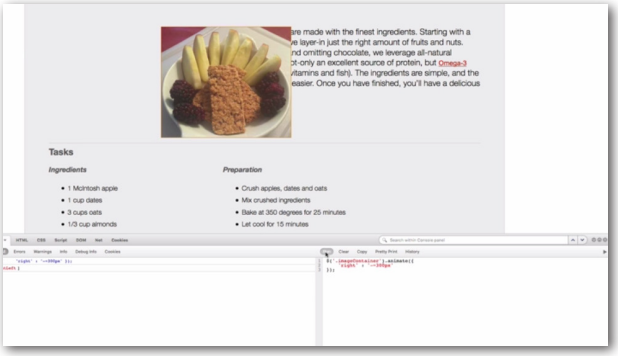
And back in your code examples file, copy the code example labeled change the image container with. Refresh the page. Paste in this code. And here we're doing something similar in that we're changing the CSS transition properties. So that's pretty much like what we did before. But here we're saying change the width of the element with the class of container to be 500 pixels. Change the CSS width property—I'm sorry, to be 100 pixels.



So right now I think it's 400 pixels. We're going to change it to 100 pixels. Again, without a transition, the image would suddenly just appear 100 pixels wide, but we're saying make that change to the CSS width property be gradual. Use a transition, CSS transitions, to make that change gradual. So I'm going to run the code. And the image shrinks down to 100 pixels very gradually by using CSS transitions.

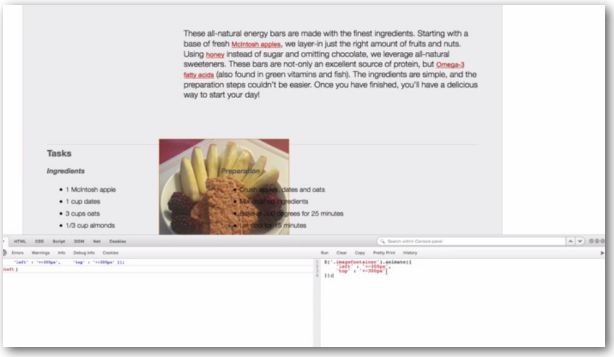
Now, back in the code examples file, copy the code example labeled animation with jQuery's animate method. Refresh your page and paste that code in. In this example, we're doing something different. We're going to use the jQuery animate method. So we're no longer using CSS transitions. We're using the jQuery animate method.

Now, that method takes as its first argument an object. And that object has one or more CSS properties that we want to animate. In this example, we're saying we want the right property of this element to be 300 pixels less than whatever it is. It's a relative change. So whatever it is now, make it 300 pixels or less. When I run this code, we should see the image slide to the right by 300 pixels. That's exactly what we see.



Now, because this change is relative, we're not saying make the right property exactly 300 pixels. We're saying make it 300 pixels less. It's relative. So we're going to keep decreasing the right property by 300 pixels. So I can keep running this code, and it keeps sliding over to the right. And if I change this negative to a positive, we're saying OK now change the right property to be whatever it is plus 300 pixels. So now I should see it sliding back to the left by 300 pixels. So that's a relative value change.

Back in your code examples file, copy the code example labeled multiple animations. Refresh the page and paste that code in. And here we're doing something very similar and pretty much the same thing, but we're doing it to two properties. So I just want to illustrate that when you call the animate method and you pass an object as its first argument, you can pass more than one property.



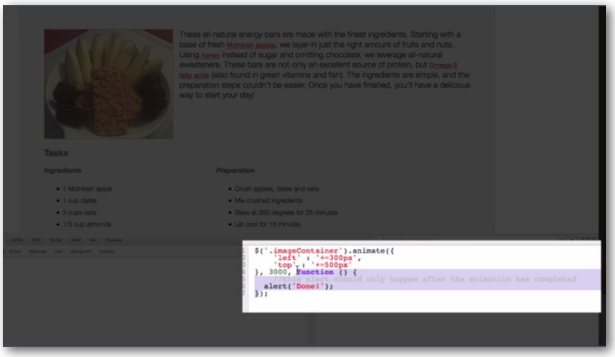
Here I'm going to change the left and top properties. So when I run this code, the image should slide down and over by 300 pixels. That's great. And I can change these positives to negatives. And it should slide back up 300 pixels. And it does. So we can change multiple properties when we call the animate method.

In your code examples file, copy the code example labeled set the animation duration. Refresh the page. Paste this code in. And you can see here I'm passing a second argument to the animate method. The first argument is the object that contains one or more properties that we want to animate.

The second argument is a number. It's the duration of the animation. We're saying, this animation should take 1,000 milliseconds or one second. Duration's measured in milliseconds. We could change this to 5,000, and it would be five seconds, but here we're saying 1 second. So this animation took one second. If you refresh the page and change that 1,000 to 5, you can see the animation now takes five seconds.

It's moving very slowly. So that's five seconds. We can change it to anything we want. If I were to refresh the page and change it—take away one of these zeros—oops—and change that to 500, the animation would be much quicker. It would take 500 milliseconds, or half a second. So we can specify the duration of the animation.

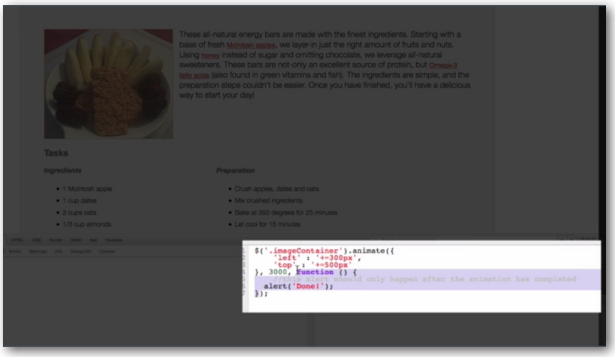
Back in your code examples file, copy the code example labeled add a completion callback. Refresh the page. Paste that code in your JavaScript console. And let's talk about what's happening here.



Now, we're adding a third argument to our call to the animate method. This third argument is a function. And this function is our completion callback. We're saying, when this animation completes, execute this function. So when I run this code, I should see an alert after the animation completes. So the animation completes, and I see the alert.

And this gives us the ability once again to say when an animation completes, I want to take some action. I want to do something. This is our completion callback. It's the third argument we're passing to the animate method. Back in your code examples file, copy the code example labeled add a visual cue in the completion callback. Refresh the page. And then in JavaScript console, paste that code.

Now, something a little bit different is that we're creating a variable here called $ImageContainer. I'm just doing that because I'm going to need to reference this element multiple times. So instead of referencing it twice, I'm going to set a variable and then use that variable multiple times. So the only real change here is that we're in our completion callback.



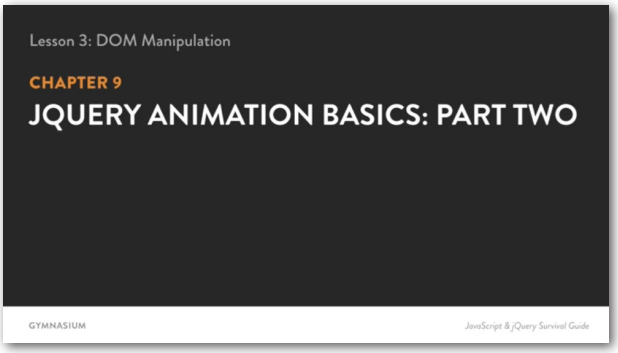
We're going to set a CSS property of the element that was just animated. We're setting the outline to be 10 pixels solid red. So what that means is, when I run this animation, our completion callback will fire once the animation is complete. And in that completion callback, we're changing the CSS property [INAUDIBLE] should see the image have a thick red outline around it.

So once again, first argument is an object containing all the properties we want to animate. Second argument is the duration of the animation. The third argument is the completion callback, or a function that's executed once the animation has completed. So I'm going to run this code.

I see that the element is animated, and then it gets the thick red outline. And that's the completion callback that's being executed once the animation has completed. Now, these completion callback techniques are fairly powerful. So we'll stay on this subject a little bit longer. In the next chapter, we'll explore a more real world example of when you might use a completion callback in conjunction with some user interface elements. See you soon.

JQUERY ANIMATION BASICS:PART TWO

As we continue to explore jQuery animation, let's dive a little deeper into the use of completion callbacks. And before we go into the next code example, I'd like to make a small change to the HTML file.



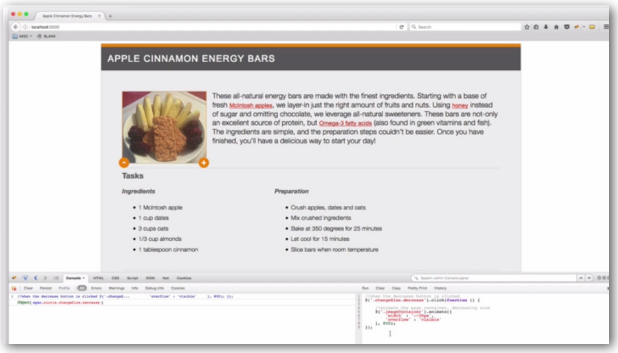
In your code repository folder, open up the file index.html. You should notice that lines 18 through 22 are commented. So remove the opening and closing HTML common tags, save the file, and then refresh the page.

Now you should notice two new little buttons in the lower left and right-hand corner of the image. They don't do too much right now, but they will prove useful to us in a bit.

Back in the code examples file, copy the code example labeled "decrease the image container size on click." Copy that code into your JavaScript console and run the code.

So now you'll notice that when you click the decrease button, the width, this width of the image, decreases by 30 pixels. I've also set overflow visible. And that's just so that these buttons remain visible because they appear outside of their container.

But here, we're simply taking the same approach we've taken previously, which is to set up a click event handler. And then we're using the animate method to change the width of the image.



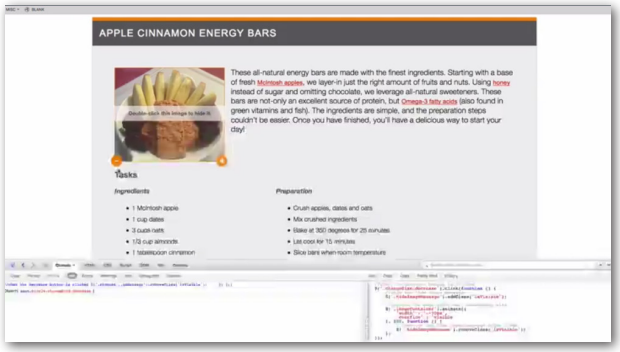
So refresh the page. And back in your code examples file, copy the code example labeled "add a hide image message." Paste that code example in. And let's just look at what's going on before we run the code.

In the click event handler, we're doing something new. We're adding a class. We're adding a CSS class to an element. The addClass method allows us to add a CSS class to a targeted element.

So when I run the code, what happens is now when you click the decrease button, we see this little message that says "double-click this image to hide it." So that's pretty cool. It gives us a little visual cue—tells us something we could do.

But the problem is I don't want it to appear all the time. I want it to appear briefly and then go away. So we need to do a little bit of work here. So back in the code examples file, copy the code example labeled "hide the hide image message after the animation completes." Refresh the page and paste that code example in.

Now when you run this code and then click the decrease button, you'll notice that little message appears, and then disappears. It appears. It fades in. And then it fades out. And the reason that's happening is we're leveraging something we learned a little while ago, which is the completion callback.



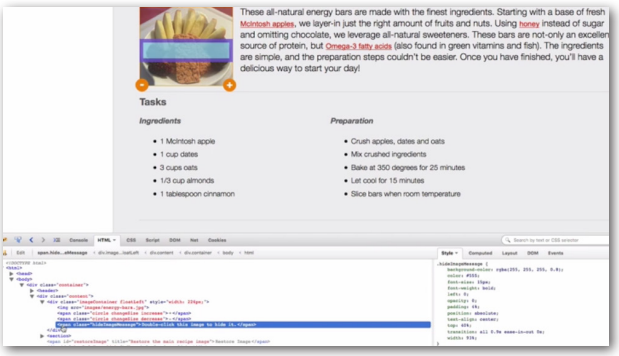
We're passing in a completion callback as a third argument to the animate method. And we're saying the element with the class of hideMessage—remove the class as visible. So here we used addClass to make it visible. Here we're saying removeClass to say we don't want it to be visible.

Let's actually inspect the element. Right mouse button click and click Inspect Element in Firebug. And you'll notice that this element right here—if I click the little decrease button, you'll see it gets this isVisible class for a second and fades away. I'm going to do it again. It has isVisible class, and then the class is taken away.

And the element doesn't just appear and disappear. It fades in and fades out. And that's being handled by CSS transitions. We're using the CSS transition property to say, well, don't just hide it and show it. If we're changing it from visible to not visible, make that transition smooth—kind of animated.

So we get that message for just 800 milliseconds and then it fades away. If you go back to your code examples file and copy the code example labeled “increase the image container size on click,” go back to your console, don’t refresh the page—just paste this new code example in—and now we get that behavior on when we click the plus button and then when we click the negative button.

Once again, if you inspect it, look at this element with the class hideImageMessage. If I click plus, it gets the isVisible class for just 800 milliseconds. If I click decrease, it gets that isVisible class again for just 800 milliseconds.



So in both cases, we’re adding a CSS class to make it visible and removing a CSS class to make it no longer visible. And our CSS transitions are handling the animation of that or making that change gradual.

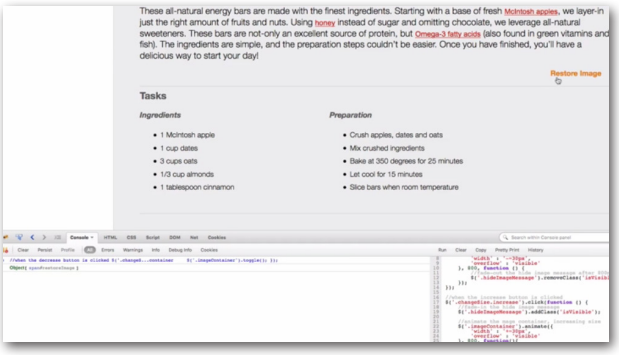
But the one thing is that I still—double-clicking the message tells us that if we double-click the image, we can hide it. But double-clicking the image doesn’t actually do anything.

So back in your code examples file, copy the code example labeled “add the ability to show/hide the main image.” And if I select this and paste that in, now when I double-click the image, it goes away. And when I click Restore Image, it appears. And if I double-click the image, it goes away, and back and forth.

So we kind of have all these features that we’ve been putting together wrapped up in one. So if you refresh the page and select the next code example, which is labeled “full working example for main image animation,” select that code. It’s kind of a lot. It’s all the way down here.

So copy that code and paste it into your JavaScript console. And now run that code. And we should have every feature working. So if I click decrease, the image decreases. If I click increase, the image increases in size.

And in either case, we see the visual cue that says “Double-click this image to hide it.” And if I double-click the image, it’s hidden. And if I click Restore Image, it shows, and back and forth, over and over.



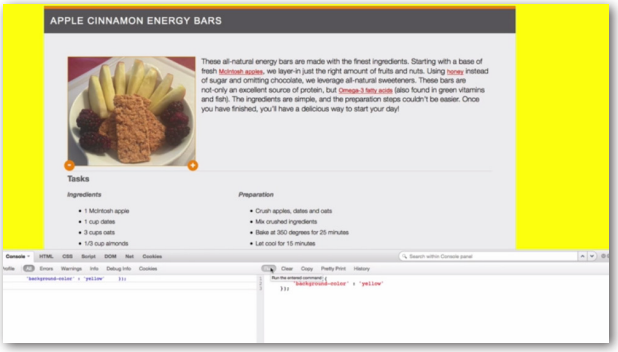
So the purpose of these last few examples was to show how you can give the user little visual cues. But in our case, we just wanted to show the cue at the end of an animation. So we leveraged the completion callback, which we learned about a little while ago, to say, when the animation is finished, I want to do something. In our case, we want to fade the message out. We want to show it on click and then fade it out when the animation has completed.

So you've had a quick tour of how to use animation in a number of different ways. And now we're almost at the end of the course. But before we wrap things up, there are some important concepts that you need to know when it comes to actually using your code in production and not just the JavaScript console. So be sure to pay attention to the next section as we explore the real-world use of your code.

PREPARING FOR PRODUCTION

Throughout this course, we've been executing our JavaScript code examples in the console. I feel that it's an invaluable tool and it's really important to be aware that that's available to you. It's a very helpful way to work out ideas and test your code and try different things without having to save your file and go back to the browser and reload it and then go back to your file. You can just run it in the console. And it's a great way to see your JavaScript run. It's also great for debugging.

But at some point, once you've fleshed out your ideas and you feel pretty good about your code, you're going to want to really put it in a JavaScript file and have it run on the page the normal way. So there's one last jQuery method that I want to make you aware of that's actually really important, and it has to do with timing. It has to do with having control over when your code is executed.



So in the code examples file, copy the code example labeled "change the background color to yellow." And don't copy the whole example. Just first copy these few lines, leaving out the script tags.

Now paste it in the JavaScript console and just run it. And you'll notice the background of the page changes to yellow. It's ugly. It's silly. But it's just to illustrate a point that our code changes the background code to yellow, and it definitely works. We can see that it works. Great.

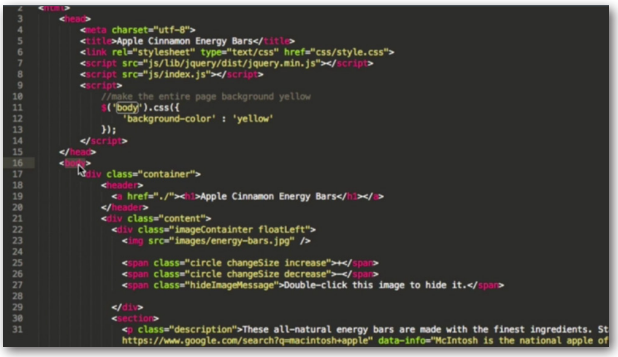
So go back to the code examples file and copy the whole code example, including the script tags. Now go to the index.html page. Open that up. And in the top of the page just one around line 9 under the last script tag in the head, paste it in like that. So we've got our script tag. We've got our code that we just ran. We know that it works. So everything should be fine.

Save your code and reload the page. And nothing happens. The background should change to yellow. And we just tried it here and it worked just fine. I reload the page again. Nothing happens. I run it here. It works.

So why is it working here in the console, but it's not working on page load, floating all the way up here on the page. It's just the page would just be yellow the second it loads.

The reason for this is because we're using jQuery to reference or to query the body tag. But the problem is at that point, the body tag doesn't exist yet. It hasn't been created because the browser parses all of this HTML in order.

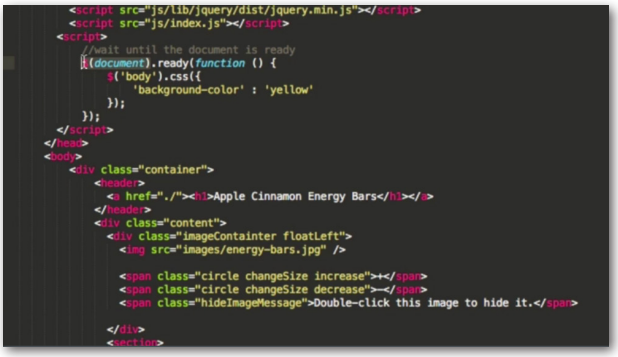
So it says, hey, doctype—opening HTML, opening head, meta, title, link, going to pull in the CSS file. I'm going to download this JavaScript file, which is the jQuery library, and execute it. I'm going to download this JavaScript file, index.js, and execute it. And then it—just one by one.



So by the time it gets to our code, the body tag still doesn't exist. It's about to be created, but it hasn't been created yet. So jQuery, when it runs this code, it says, I don't know what body means. I don't see any elements called body tag. So I'm going to ignore all this code and keep going. It's that simple. So this won't work.

So if we take this script tag, cut it out, go all the way down to the bottom of the page, and just before the closing body tag paste that code in, so now we've just moved our code to the bottom of the page. It's the third to last line. Save your code. Refresh the page. And it turns yellow. Our code works.

So why does it work now? Well, that's because at this point, the body tag does exist. It very much exists. In fact, it was created, like, 100 lines ago. So here when jQuery runs into our code, it says, oh, body tag. Yeah. Absolutely. I know what that is. What do you want? You want to change the background color to yellow? Got it. No problem.



So timing is important. jQuery will query DOM elements, but it can't query elements that don't exist. So we need a way to reliably know that wherever we place our code, jQuery's not going to execute it until the elements that we're querying exist.

So go back to your code examples file. And the example labeled "wait until the document is ready"—copy that one. Now go back to index.html. Go back to the top of the page. And once again, on line 9 under the last script tag in the head, paste this code in. Save it. And then refresh the page.

And now the page turn yellow on page refresh every time. Why is that? That's because we're using a method called jQuery's document.ready. Recall in the jQuery function, we're passing a document object. The document object is a logical representation of the web page. It has a number of properties and methods.

And when we wrap it which jQuery, it has a new method called "ready." So what's happening is we're saying, hey, jQuery, the document—when it's ready, execute this function. It's that simple. So jQuery says, hey, no problem. I will wait until the document is ready. And when it is, I'll execute this code.

So it kind of holds that code. It kind of sticks it off to the side and defers it and then continues going. And it knows that the web page will actually raise an event called “ready.” And when that event is raised or when that event is executed, jQuery says, hey, I got this code I need to execute because the user wanted that when the DOM was ready. So I’ll execute this code.

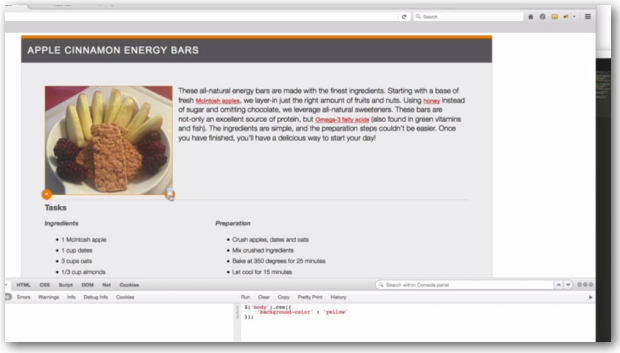
So it keeps this code kind of in memory, holds onto it. When the document’s ready, when all the elements in the page are ready to be manipulated, it then executes our code. So let’s take a look at how we would use this in a little bit more of a real world sense.

So remove that code and just refresh the page to confirm that our code is out of the page. The page is white now. Now, you’ll notice that we’re pulling in an external JavaScript file called index.js. Let’s look at index.js. It’s in a js folder.

When you look at it, it’s empty. There’s nothing in that file. So let’s go back to the code examples file and copy the very last code examples. It’s labeled “used document is ready for main image animation.” So copy all that code and paste it in here.

And then I want you to do something. Select everything inside the document.ready, cut it, and paste it. So move it out of document.ready. So here we’ve got our code. You’ve got document.ready. Nothing’s happening here. There’s nothing happening. But we’ve got our code right here.

So save the file. Refresh the page. And you’ll notice that our event handlers are not working. Our code is not working. Nothing’s happening here. And that’s because yeah, this document.ready call runs, but there’s nothing being passed to it. So jQuery isn’t deferring anything for us.



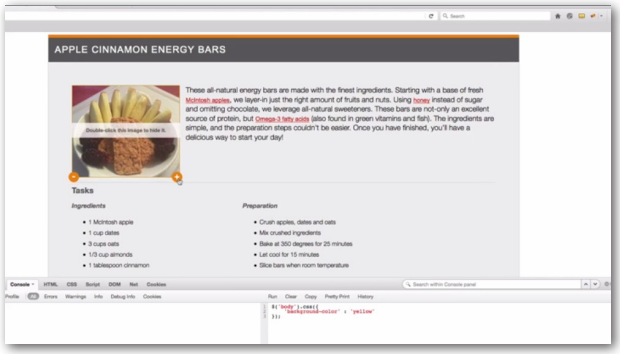
And this code runs immediately. This code happens right here. It happens when this gets pulled in. And the body tag doesn’t exist. In fact, none of these elements exist. None of the elements that we’re querying exist yet. So our code is a complete waste of time. It’s querying a bunch of elements that don’t exist.

So let’s take our code, cut it, paste it back inside the document.ready, save the file, and then reload the page. And now you’ll notice that our event handlers work exactly as advertised. We can increase. We can decrease. We could double-click and restore.

And that’s because up here on the page, very high up in the page before any elements exist, we pull in this file called index.js. And index.js says, hey, jQuery, when the document is ready, execute this function. And jQuery takes this function and holds onto it and keeps it in memory. And then when the web page is ready and all the DOM elements are ready to be manipulated, it executes that code.

So our event handlers are not even executed until the very elements they’re trying to query are available in the web page. And once they are, it executes our code and our handlers work. And we can do everything we expected. And everything works fine.

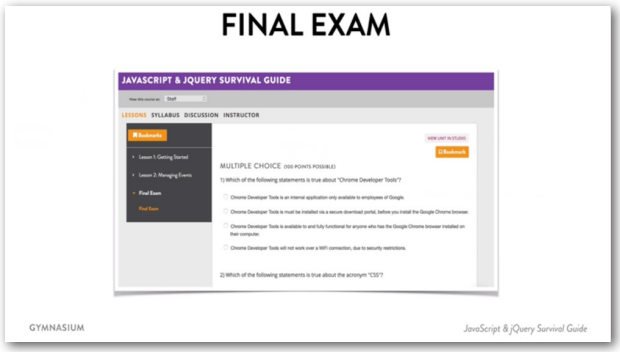
So the main thing I'm trying to emphasize here is that the jQuery document.ready is possibly one of the most important methods you'll ever use in jQuery. And it allows you to make sure that your code, your jQuery code, is not executed until the document or the web page is ready. So you want to always wrap all of your code in a document.ready call to make sure from a timing aspect that the code is executed at the right time and all of the event handlers that you spent so much time working on will work properly every time.



Well, that's it for this exercise, and for this course as well. I hope you found the various exercises useful. It was certainly my pleasure to share them. Don't forget that if you have questions or problems, you can use the classroom forum. Just be sure to post your code so that others can troubleshoot it.

You can always post your files on a site that you're hosting yourself. But an alternative is to use a service such as jsbin.com or codepen.com, both of which will allow you to quickly and easily post HTML, CSS, and JavaScript code for free. Additionally, both services will allow you to reference jQuery in your pages. For instructions on how to do this, there's a permanent post at the top of the class forum that we'll keep up to date.

Also, there is a final exam for this course. If you pass, you'll receive a certificate that you can download, share, or automatically add to your LinkedIn profile. All of the material in the exam is based on concepts covered in the videos. So be sure to review those first.



Thanks again. This is Kevin Chisholm. And you've just completed the JavaScript and jQuery Survival Guide on Aquent Gymnasium.