AQUENT

GYMNASIUM

JQUERY BUILDING BLOCKS:

BUILDING TABS ... TO LAST

LESSON 4

ABOUT THIS DOCUMENT

This handout is an edited transcript of the *jQuery Building Blocks* lecture videos. There's nothing in this handout that isn't also in the videos, and vice versa. Some students work better with written material than by watching videos alone, so we're offering this handout 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. We encourage you to use this handout alongside the videos, rather than as a replacement of them.

INTRODUCTION: BUILDING TABS ... TO LAST

Welcome to jQuery Building Blocks—Five Techniques to Cut Your Web Development Time in Half. This is Lesson 4—Building Tabs... To Last. We'll be covering the common need of building tabs today. I'll show you how to take your tab code and package it up as a plug-in for your co-workers and your future self to use. And we'll be generalizing that code slightly and carefully to allow your future self greater flexibility.

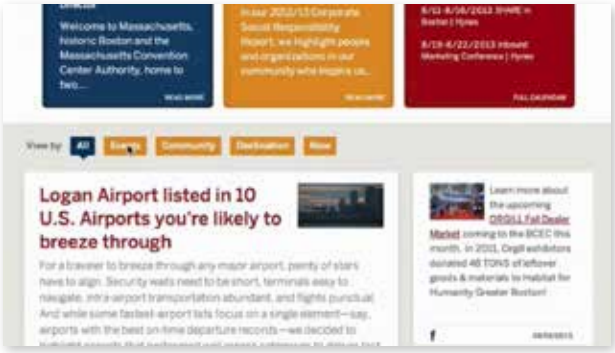
In Chapter 1, I'll briefly introduce you to the idea of tabs. In Chapter 2, we'll build some simple click to show tabs ourselves. In Chapter 3, we're going to wrap that code up in a nice focused plug-in. Then in Chapter 4 we're going to generalize it in a narrow, intelligent way to handle a specific known need at our office. And we will conclude with a conclusion.

Don't forget, you can use the pause button. There's a lot of code in today's lesson. So feel free to give it a pause anytime you need to spend a little more time drilling down in on something; I won't be offended.

CHAPTER 1: THE SPACE-SAVERS

This is in the first space-saving widget that we've taken a look at. Last time we covered carousels. They're flashy. They're very nice when done well. But they're easy to mess up. And very easy to do badly.

Tabs serve a similar function, right? They're space-savers. But they've got a more explicit interface. They're less flashy. The user has got explicit control the whole time. So it's harder to screw up tabs from a user experience perspective. And with this explicit categorization built in, they're useful for clearly organizing content or wrapping up a small amount of content.



Tabs also work great as a replacement for the regular kind of linky navigation bar, right? This case is from the Boston Convention Center website. Now they could've made five links to five different blog topic pages, right? But that would have forced a reload each time. And in this case, a jump back to the top of page, and that would have been jarring and unpleasant.

Instead, they implemented in page tabs. And when it's appropriate, you can even set up each tab to have its own web address, to have its own location up in

the address bar. But we're not going to cover that today.

All right. This lesson is a play in three acts. And you're the star. You're playing your standard role as a developer at our poster store website. It's Friday. 1 p.m. Office is quiet. Most the web team has already gone home. You're headed out of town for the weekend. You want to hit the road by 2:30 to beat the traffic on the turnpike.

Then the boss calls you in. Now, he's an OK boss. He wears very tasteful ties. But he can be a little intense, especially when there's a deadline. And he tells you that he's just found out about an important milestone coming up on Monday. "The VPs," he growls. "We never should have sold out to London."

Now, some VP or other has noticed that the company sidebar is getting a little long. And it's only going to get longer as the company expands. So he wants to know how quickly can turn it from this into this. It doesn't have to be pretty. But it has to work. Because he's giving it to Ethan, the team's designer and CSS guru, to pretty it up over the weekend.

You get up. Head back your desk. And get to work.

Now, your first move is going to be to search the web for tab plug-ins. And there are a bunch, which may or may not be a good sign. Anyway, the first one on the list is going to be jQuery UI. And it's good. JQuery UI is awesome. Lots of people use it. But there's three problems.

First, jQuery UI is a monster. They have this download builder. And you can sort of put the effort into slimming the download down. But that's effort. And you've got a deadline. And it all may not be worth it anyway for one widget.

Secondly, if you're spending your Friday afternoon learning how to bend jQuery UI to your will, then that's all you're learning, right? You're not learning any generally practicable skills. Which is fine if it's going to save you a lot of time or if you're going to use it over and over again. But tabs are easy enough that in this case it's probably not going to. A good rule of thumb, if you think you can implement this in half to 3/4 of an hour, don't bother tracking down a plug-in.

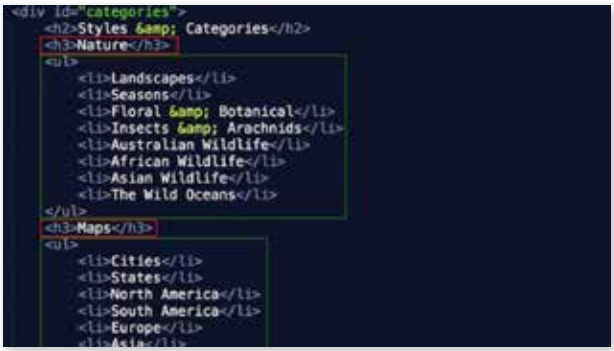
The third issue with this is that—and this is true of all plug-ins—jQuery UIs particular specific tab implementation may not do exactly what you want. And if it doesn't, or if getting it to dance to whatever your particular little tune is is prohibitively difficult, then you're going to end up back at square one with nothing to show for it.

```
18
19 <div id="tabs">
20   <ul>
21     <li><a href="#tabs-1">Nunc tincidunt</a></li>
22     <li><a href="#tabs-2">Proin dolor</a></li>
23     <li><a href="#tabs-3">Aenean lacinia</a></li>
24   </ul>
25   <div id="tabs-1">
26     <p>Proin elit arcu, rutrum commodo, vehicula tempus
27   </div>
28   <div id="tabs-2">
29     <p>Morbi tincidunt, dui sit amet facilisis feugiat,
30   </div>
31   <div id="tabs-3">
32     <p>Mauris eleifend est et turpis. Duis id erat. Sus
33     <p>Duis cursus. Maecenas ligula eros, blandit nec,
34   </div>
35 </div>
36
```

In fact, if you take a look at the jQuery UI tab demo's source code, you notice to your dismay that they've got HTML for the tab separated out from the content. Which is fine, that's a fine way to do it, except that that's not how your page is coded. And the guy who maintains the categories template, guy already went home for the weekend.

You could probably track it down, and modify the template, and push it. But that guy's kind of a jerk. And his code is terrible and fragile. And it would just be better

for everybody if you could avoid touching this HTML—touching this template at all.



So again, our goal is to change this stack of lists into this without changing the existing markup at all. Now remember, Ethan's going to pretty it up over the weekend. So today you're a developer. You're not a prettifier.

Can we do it? Yes. Is it going to be easy? Yeah. It's fairly simple. Fairly straightforward. Especially with a jQuery. Is this story completely contrived with absolutely no basis in my personal experience of how a real development teams works? No. I'm afraid it's not.

CHAPTER 2: SIMPLE TABS

All right. From your lesson materials, please open the index and style files in your ID of choice. And I'm using Sublime Text. Use whatever you like. And please open up the website itself in Chrome, as well.

And here's its initial state. A stack of lists over on the sidebar. Here's its initial HTML. Straightforward stuff. Nothing going on. Just a bunch of h3's and ul's down here in the categories area. This is our initial style.

Now this page includes a lot of stuff we're not going to be looking at. There are some good HTML resets, just gets rid of the default styling, a lot of which we don't want. And then down here here's how our sidebar is set up. It just floats off to the right. Got a fixed width. All very straightforward stuff.

Now, there's a great little trick down here in order to get our custom bullet points in our list. We use a little before pseudo element. And we set the content to whatever we want the bullets to be and give it a little margin. Neat trick.

Before we start any JavaScript, let's set up the initial state of things. Let's separate out the CSS just because while it's in development we don't want to mess too much with the existing stuff. We're going to select the categories by ID. And then this selector says we're going to grab all h3 elements, which are any descendants of the categories.

So our goal here is to float the headers next to each other. And then we're going to hide the content elements, which in this case are the ul's, the unordered lists. So let's just float these left against each other. And then let's select the categories. And then all unordered lists that are a child of them. And we're going to hide them.

Now, let's take a look at how this works. And there they are. It doesn't look good yet. But they're all floating next to each other. And all of the intervening unordered lists have been hidden.

```
196     cursor: pointer; /* ahem fake links */
197     margin-bottom: 4px;
198 }
199
200 #categories ul > li:before {
201     content: "»";
202     margin-right: 4px;
203 }
204
205 /* Tabbification */
206
207 #categories h3 {
208     float: left;
209     padding: 4px;
210     margin-right: 3px;
211     border: 1px solid black;
212     border-bottom-width: 0px;
213 }
```

Now let's pretty these up a little bit. Not terribly much. Again, that's Ethan's job. But let's at least make them look a little bit like tabs. I'm going to separate them out a little bit. Giving it a nice border. And we're going to hide the bottom border. Just because the bottom border, at best, is going to be redundant, and at worst, is going to look terrible.

There we go. Now we've got that. We're going to give ourselves a little background color to them. Just to separate out the unactive ones from the active ones. Give them a little nice light gray. And then let's make them look like links. Let's make that hand cursor light up when we go over them. So there we go.

All right. Now our tabs are ready. We have our unselected tab state is ready. Now just for development purposes, we're going to add a couple things to the HTML. And don't worry, because we're going to roll these changes back in a minute. And we're going to add a class to the very first header called Active. And we're going to do all this with classes.

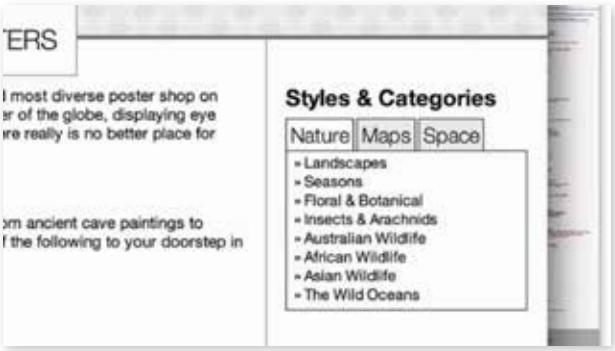
So let's set up our active tab state. We're going to give it a background color. We're going to re-override the background color from light gray to white. There it is. Now let's do the same thing with the Content element, the unordered list. We're going to add ourselves an active class and we're going to show it.

Now, this isn't going to look great yet. It's going to look quite weird, in fact. And the reason is because we've got this sort of in the line of fire, right? This element is still in between things. So we need to figure out a way to position it down below. And the quickest and easiest way is to position it absolutely. We're just going to drop it down like that.

Now that's doing what we want. All we need to do is give it a little margin to knock it down below our level. Now let's say it looks like 34 pixels. Let's see if that's a perfect number that I actually checked out earlier. Looks good.

So there we go. We're going to move this up to always. This is for a very minor

performance bump. The fewer things that change when we activate a tab the better. So let's have the only thing that changes its visibility.



Let's give it a little border. We're also going to need to give it a width. And we're going to hard-code that. That is what it is. That's fine. Ethan can handle that.

So now let's take a look at it. And perfect. Looks great.

So now at this point, we've now built the look of it. This is all of the CSS that we're going to have to do. From here on out, all we're doing is playing jQuery games to mess around with the tabs.

CHAPTER 3: BUILDING FOR THE FUTURE

Let's take those hard-coded initial states. And we're going to turn them into a dynamic initial state, and dynamic ongoing states, with jQuery. Now here is what we're going to need for this initial pass. By default, we want the first tab to be visible.

So we need an easy way to get the first matching element from the list jQuery provides. Since we've got all of our differentiation between active and inactive done in CSS, the only thing that we need to actually be dynamic is classes coming on and off elements. So we need a way to add and remove classes. Again, jQuery provides.

We're going to need to know about it when the user tries to switch tabs by clicking on them. So we're going to use this click method that we've seen before. And, along with the first, we're going to use last in order to add a special class to the last tab for Ethan to use while styling.

Along with the first and last, we need a way to get any arbitrary element out of the array. Now jQuery doesn't provide a nice method for that, surprisingly. But it's still very easy. So if you need to get the second, or third, or fourth element, you can just use the square bracket notation, like this.

Along with taking a selector, the dollar sign function can also take a raw element. And this is something we haven't seen before. So if you have a raw element that's just kicking around, and you want to deal with it in jQuery form, and use all of the convenient things that jQuery gives you, just pass it into the dollar sign function like this. Ergo, in order to get the Nth element out and wrapped back up in jQuery form, just put them together.

All right, let's do the first one. We're going to set up a script element for ourselves to play around in. Don't forget to use your document.ready to make sure that the elements that we're playing around with are there by the time we actually want to run this code. Always remember your semicolons.

Let's get the tabs themselves. We're going to select them. This nomenclature says, I'm going to select all of the immediate children of h3, which are immediate children of the categories ID'd element. Also, keep in mind, just by convention, that you usually are going to be naming your jQuery variables with dollar signs at the front. Often, you're going to be holding onto both a raw element and its jQuery form, and this gives you a very convenient way to have both.

```
        <meta charset="UTF-8">
        <title>jQuery :: Lesson 4 :: Tabs</title>

        <link rel="stylesheet" href="css/style.css"/>

    <script src="script/jquery.1.9.1.js"></script>

    <script>

$(document).ready(function() {
    var $tabs = $('#categories > h3'),
        $contents = $('#categories > ul');

});

    </script>

    </head>

    <body lang="en">
        <div id="container-nav">
```

So we're going to do the same thing for categories. And this is all the ul's which are immediate children of the ID'd category element. We're going to set up our click handler. And we're going to do the thing that we saw before, where we're going to snag the zeroth element out the tabs array, the tabs object.

Now, don't forget—zero, in programming, usually just means the first. They're funny like that. It's called zero base. If you're not familiar with it, you're going to run into it a lot, and it's going cause you of an annoyance. But you'll get used to it.

```
$($tabs[0]).click(function() {

})
```

We call the click function on that first tab. In it, we're going to remove the class active from all of the tabs. And what this does is, instead of trying to find the tab that is currently active and remove the class only from that—which might be a little more effective, might be a little tiny bit more efficient—we're just going to remove it from all of them. It means we have to keep track of fewer things. And simplifying your code is always a good thing.

So we're going to remove the active class from that. Then we're going to get the zeroth element out. And we're going to add to the active class back to it. And then we're going to do exactly the same thing with the contents.

All right. Now we've got the zeroth one done. We are going to update it three times, once for each tab. Let's take a look at how it works. And it works. This is fantastic. This is great.

But you'll notice that I'm committing the cardinal developer sin of copy, paste, edit. And I want to talk about that for a sec. So first, why is it a sin? Why is it problematic to do this? Having code in your code base that repeats but is slightly different each time is a recipe for annoyance next time you have to go and change the behavior in all three places.

```
$(document).ready(function() {
    var $tabs = $('#categories > h3'),
        $contents = $('#categories > ul');
    $($tabs[0]).click(function() {
        $tabs.removeClass('active');
        $($tabs[0]).addClass('active');
        $contents.removeClass('active');
        $($contents[0]).addClass('active');
    });
    $($tabs[1]).click(function() {
        $tabs.removeClass('active');
        $($tabs[1]).addClass('active');
        $contents.removeClass('active');
        $($contents[1]).addClass('active');
    });
    $($tabs[2]).click(function() {
        $tabs.removeClass('active');
        $($tabs[2]).addClass('active');
        $contents.removeClass('active');
        $($contents[2]).addClass('active');
    });
```

But actually, it's much worse than that. You'll notice that these six lines of code have got minor differences every other line. If I make a change to how this works, I can't just re-copy and paste it. I need to choose between copy, paste, edit again, and remembering all of the edits that need to be made, and making them correctly, or making each change in all three places. And there's a very real possibility, you know, that when you go to do this, you're going to be slightly tired or slightly distracted, and you're going to break something in a hard-to-diagnose way.

Another reason this is bad is because, right now, we've got three tabs, 0, 1, and 2. Later on, maybe there'll be more. Or maybe there'll be less. And this is just not flexible code at all. So if this is that bad, then why am I doing it here?

I promise, it's not going to stay that way. This isn't production-ready code. But, you know, we're mid-development right now. And our goal is to quickly get it working, to validate our design and validate our assumptions.

Once we're sure that we're accomplishing what we want to accomplish, we can go back and improve the maintainability of the code. So I promise we'll get to that in a moment.

All right, now we promised that we weren't going to touch the categories template. So let's go back and back out those initial classes. Then, when we refresh, we see exactly what we'd expect, right? Everything is still functional, but there's no initial state. So let's go and get that initial state back.

```
19          $contents.removeClass('active');
20          $($contents[0]).addClass('active');
21      });
22      $($tabs[1]).click(function() {
23          $tabs.removeClass('active');
24          $($tabs[1]).addClass('active');
25          $contents.removeClass('active');
26          $($contents[1]).addClass('active');
27      });
28      $($tabs[2]).click(function() {
29          $tabs.removeClass('active');
30          $($tabs[2]).addClass('active');
31          $contents.removeClass('active');
32          $($contents[2]).addClass('active');
33      });
34      // Set initial state.
35      $tabs.first().addClass('active');
36      $contents.first().addClass('active');
37  });
38
39      </script>
```

Now back at the bottom of our code, we are going to set our initial state with some very simple code. We just need to grab the first tab. And we're going to add that class. Remember, this code is running as soon as the page is loaded and ready. We're going to do exactly the same thing to contents, add that class, take a look at it, and refresh it, and we've got our initial state back. Initial state functions, everything else functions, excellent.

Now as long, as we're in here, let's go ahead and add that first and last states—I'm sorry—classes, to the first and last tabs, just because we assume that Ethan's going to want that. Give it another look, and it works. Good work.

CHAPTER 4: THE TWIST

All right. Now we've had a major milestone here. We have got our minimum deliverable accomplished. And it's not even 1:30 yet. That's a really big deal. Good job.

Now, let's not celebrate too much. We are going to immediately go back and fix all of that code that we copied and pasted. Now our goal here is to make all of this repeat code show up only once but to have it still work, obviously.

Now know that the only thing that changes here is the number, which is great, because it's a strong indication that what we're looking for is an iterator. And an iterator is a way to just do something to every item in the list. You iterate over the list.

Now standard JavaScript iterators are super fast. But they're also cumbersome to write. And they are cumbersome to write safely. So, just for today, we are going to use jQuery's nice convenient little way to do something to every item in a jQuery list, the each method. You pass it a function, and it runs that function once for each item, passing the current item's index, what number item you're at in the list. And it passes the actual element itself second.

All right, so let's do it. Now first comment out the code we're replacing. We want to reference it, so don't delete it quite yet. Now we are adding click handlers to the tabs themselves. So the tabs, that's the list we're going to iterate through.

```
$($tabs[1]).click(function() {
    $tabs.removeClass('active');
    $($tabs[1]).addClass('active');
    $contents.removeClass('active');
    $($contents[1]).addClass('active');
});
$($tabs[2]).click(function() {
    $tabs.removeClass('active');
    $($tabs[2]).addClass('active');
    $contents.removeClass('active');
    $($contents[2]).addClass('active');
});
*/
$tabs.each(function(i, tab) {
    var $tab = $(tab),
        $content = $($contents[i]);

})
// Set initial state.
$tabs.first().addClass('active first');
$tabs.last().addClass('last');
```

Now remember, index first and then actual tab second. Now it's important to note that what jQuery gives you is the raw HTML element, not a convenient jQuery object. So the first thing we're going to do is we're going to wrap it in JQuery.

Now we'll grab the corresponding content. Now remember, the content and the tabs have to be in the same order. And then we're going to go ahead and set up that click handler.

It's going to be basically the same thing as before, except now the elements that it's handling have been determined already by the iterator, which is great. Let's double-check over here, give it a refresh, and everything looks good. Everything is clickable. Wonderful.

A last little note in there that I'll drop in here just for anyone who wants to know why one function can handle all these different clicks: you go to Google for JavaScript closures, and be prepared to lose some sleep.

CHAPTER 5: BUILDING FOR THE FUTURE

It's 1:30. You send a screen shot and some code off to Ethan so he can make sure everything meets his needs. You congratulate yourself for a job well done. And just in your moment of exuberance, a little voice in the back your head whispers, "He always recycles ideas."

And that's true. Your steely-eyed boss is guaranteed to reuse the last good idea he saw at least three times the next two weeks. There are other pages on the company's site that have got similar, but not quite identical, sidebars. And they're basically guaranteed to get the tab treatment.

Now you put the effort into avoiding the cardinal developer sin of copy, paste, edit within this code. So you sure don't want to force your future self in to having to copy, paste, edit the entire tabification code.

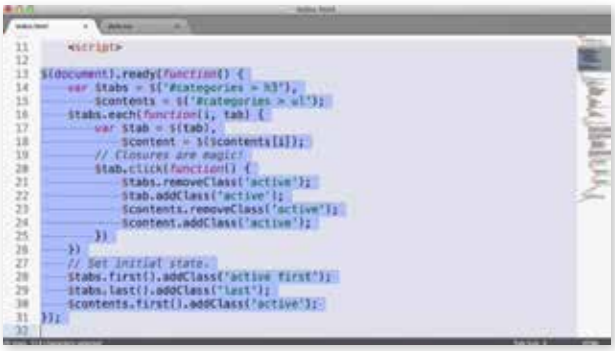
Enter jQuery plug-ins. You can create one to package your code up, so that you—the future you—and other co-workers can easily add it to their pages. So, let's get to work.

In the last lesson, we took a look at downloading and using plug-ins. And remember, they add methods to the jQuery object so that you can call them, like cycle for adding easy carousels. Or raptorize for adding crucial business support capabilities.

Now, plug-ins are fundamentally about abstraction. Their whole purpose is to take a lot of behavior, or a lot of knowledge—basically a lot of code—and wrap it up and abstract it away behind a simple, easy-to-use exterior.

This time we're going to actually create our own plug-in. And for the same reason—we're going to allow the future you to do this stuff more easily. Our goal is to package up something that's going to be narrowly useful to your team, with your team's specific needs.

Three steps. First, we're going to move the plug-in code into its own file. Second, we're going to automate the stylesheet, so it's even easier to use your plug-in. And third, we're going to convert that code from inline code to the plug-in and we're going to use it.



Step one is to move the code into files in its own folder. Let's see. We did this commented code one better, so we don't need it anymore. Let's delete that.

Now, let's just grab all this code and drop it into a new file. Now, by convention, plug-ins live in eponymous folders. And the files themselves are named after the plug-in and the version number. Now that's just to allow for better caching and clearer distinction between upgraded versions of your plug-in. This is 1.0; maybe later on we'll have something better.

Add the script in to pull that code into the page. We're going to do the same thing with the stylesheet. And then just to make sure our extraction went well, we're going to recycle and check. Looks good.

Now, we don't want people to have to bother including both the script tag and the stylesheet, so we can do the fancy little thing and append the stylesheet ourselves inside the JavaScript file. We're going to use the jQuery Append command, which adds the specified HTML or element to the end of whatever is selected.

In this case, we are going to append the stylesheet to the document.head. Now, note that we're doing this outside of the document.ready wrapper. Remember that HTML pages load sequentially, top to bottom, pausing along the way to run any scripts they run into. So although the document hasn't loaded yet, the head exists at this time. And we can target it. We reload. And looks good.

All right. Now we're ready to convert it to a plug-in. Remember, plug-ins add methods to all the jQuery objects on your page. And all the jQuery methods themselves are stored at $.fn. That's where they all live. So all you need to do to add a plug-in is add a function there.

Let's turn this into a math on the plug-in object. Now, a quick note about this. When your plug-in code gets called, it's called in the context of the selected jQuery object. This becomes the jQuery object itself. Now, we're assuming that the developer is going to be calling this on the category's element. So that's going to be what this means. Meaning that our tags and content here are children of this.

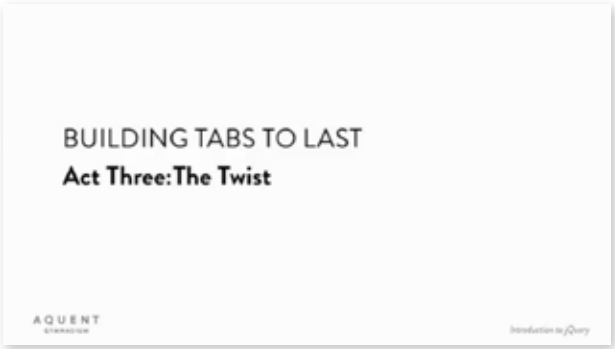
And finally, let's go ahead and call that code from our document. Select the categories and tabify them. Reload. Looks good.

All right. So now we have a plug and play solution for our company's specific needs. And thanks to the magic of closures, which again, go Google that if you want to lose some sleep, it can even theoretically handle multiple sets of tabs per page, which is great.

The CSS isn't generalized yet. But that's Ethan's job. And the CSS is going to be specific to whatever future tabs get created, as well. So that's not something we're going to worry too much about.

CHAPTER 6: CONCLUSION

All right, it's 2 p.m. Getting close. You've pushed your code to the dev branch. You let the boss know. It's a couple more emails, maybe read the Internet for a half an hour, and then you're on the road, right?



Then Ethan comes by. He really likes your tweaks. And he really appreciates the first and last classes that you put on the tabs, by the way. But he wants to know if it's limited to using lists, or if you can use images or other things.

And you squint for a sec and translate his request from designese into programmerese—what he needs is to be able to specify any set of elements for tabs, and any set of elements for content, rather than just the unordered list that we've been using so far.

Now the answer to his question is, no, you can't do that yet, but you've still got half an hour. When you're designing and writing a plug-in, there's a temptation to try and future-code, to sort of predict and handle every single possible need that you could possibly conceive of.

But that's not necessarily a good idea. You end up spending a lot of time working on features that, maybe, nobody's ever going to use. And sometimes, while designing for an unneeded feature, you can paint yourself into a corner that gets in the way of adding an actually needed feature.

So we're not just going to think of every possible need and create it. Instead, Ethan has pointed out a real need. He wants arbitrary control over the tab elements and the content elements. Now that, we can deliver. Now let's take a look at our existing plug-in. It lets you specify the wrapper, and then it makes assumptions about what's going on inside that wrapper.

Ethan has asked us to remove those assumptions and let him make it explicit. He wants us to essentially remove some limitations. The only real fundamental rule for tabs is that the number of tabs has to match the number of content elements. So let's get rid of that limitation.

Now this presents us with a problem. Our current design only takes the one

argument, right? The wrapper. And we now need to specify two things, the tabs and the content. And jQuery isn't really set up to accept two arguments like that. So let's take advantage of the fact that we can pass arguments into the plug-in method call as well. So let's put our tabs and our content elements and path them separately like this.

Now this ought to work great, but it's not super readable. So let's take advantage of the fact that there's a grammatical relationship between the tabs and the content. And let's rename our plug-in something that scans.

Let's call it "makes tabs for." This reads like English. You take these elements and make them the tabs for these other elements. Let's do it. First, let's rename the plug-in. Now remember, inside the plug-in, this is the jQuery object itself. So that can just be the tabs. And the contents is just going to be the selector.

Now, thanks to the was that jQuery works, sort of secretly, this selector can be a string, which is sort of the intention. But it doesn't matter. It can also actually be the actual elements themselves, or it can be an existing jQuery object. And that will just work perfectly fine.

```
To use: Call "tabbify" on a jQuery object containing the parent o
content and tabs. For example,

$('#categories').tabbify();

*/

$(document.head).append('<link rel="stylesheet" href="gymnasium-t

$.fn.makeTabsFor = function(contentSelector) {
    var $tabs = this,
        $contents = $(contentSelector);
    $tabs.each(function(i, tab) {
        var $tab = $(tab),
            $content = $($contents[i]);
        // Closures are magic!
        $tab.click(function() {
            $tabs.removeClass('active');
```

Once we've pointed our existing variables to the correct values, we don't even need to rewrite any of our code. So now let's jump over to the page itself, change up our plug-in call to select first the tabs and then the elements.

Now, let's give it a quick test and looks good. We've removed the artificial restrictions on our tabs. Now remember that the only rule for tabs is that we need the same number of elements in both of the lists. How do you enforce that?

And the answer to that is you use errors. Now, you usually think of errors as bad things. But you know, when you're writing code for another developer to use, it's absolutely appropriate to throw a descriptive error in when something has been done wrong.

They show up in the console like this. So only the developer is ever going to see those issues. And then the developer themselves can fix whatever boneheaded mistake it is they made.

So to throw an error, you literally throw it. You create a new error object passing the text of the error, and then you throw it. And finally, let's update our documentation to correctly document the new API.

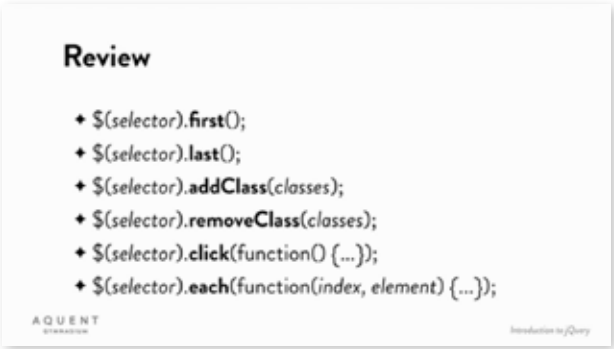
And with that, we're done. That's it. Take a look at the amount of code on the page itself, which is to say, the amount of code that the next developer, or you in the future, is going to have to write. It's gone

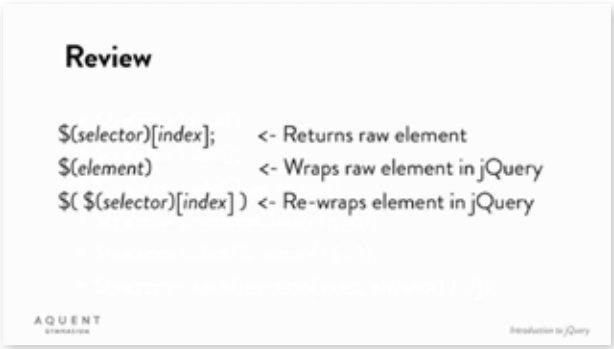
from this, which isn't horrendous, but it's a decent amount, down to this.

And, more importantly for the next guy, you've used abstraction to drastically reduce the amount of knowledge required to pull it off. And with that, we're done. Before we hit the road, let's review everything we used today to implement our tabs plug-in.

First and last get out, predictably, the first and last elements from a list. We used add class and remove class to dynamically handle classes in our project. We used the click function, and we used each to integrate over the items in jQuery list. And again, that's pass in a function which takes the index, or the number that it's at, and the raw HTML element at that place.

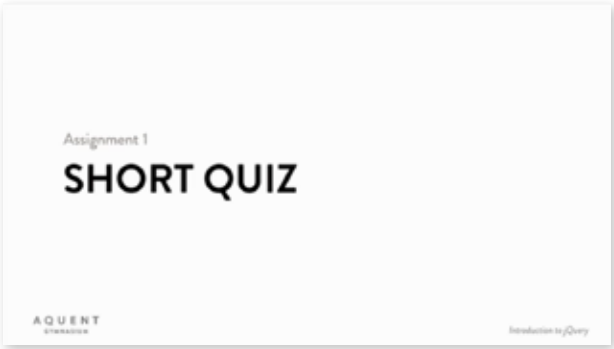
We've done first and last. So in order to get the Nth item out, you can treat it with the square brackets, like any other array. You can also use the dollar sign method to wrap a raw element in jQuery so, ergo, put them together to extract the Nth item from a list and re-wrap it.





Today, you wrapped your code in a plug-in using the standard location for plug-in methods. And you even handled appending the stylesheet using append so that the next guy doesn't even have to worry about doing that. But the next guy can take care of himself because it's 2:30, and you're on the road.

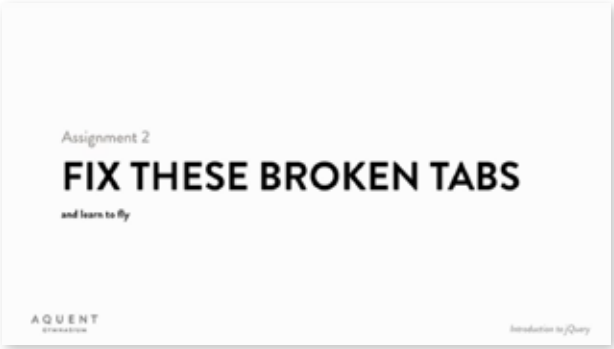
So today we took a look at the common need of implementing tabs. We discovered a situation where the existing plug-ins, probably, aren't going to handle our needs, or aren't going to handle our needs within the time span that we have to dedicate. So we rolled them ourselves. Having done that, we also rolled our own plug-in for the purpose of packaging it up nicely, and passing it along to our co-workers, and looking like a hero.



Now, I have a couple assignments for you. First, if you could just jump over and take a short quiz. Assignment 2, in the broken tabs folder of your lesson materials, I've left some tabs that need fixing. Have a look at those and see if you can get them working again.

And in the carousel folder of your lesson materials,

you'll find last week's carousels. Now, it's only a few lines of code, but there's some specialized knowledge in there. So that's another target for turning into a plug-in for around your office.



So please take care of that. I've started. I've done the first step for you. I've moved the code into separate files for you. If you run into any issues, or if you just want to say hi, please do hop on the forums. Otherwise, we'll see you next time. This has been Lesson 4—Building Tabs… To Last. I'm Dave Porter for Aquent Gymnasium.