Callbacks

The recommended way to attach handlers to dynamic HTML content is by using **callbacks**. A callback is nothing more than a function that we tell some *other* function to run at some specific later time. In order to attach event handlers to dynamic HTML content, I need to run my jQuery code for the dynamic content **after the content is created.** In other words, I must wait until that dynamic content is created before I can attach any event listener to it. One way to do this is to **attach the event handler in the same function that creates the dynamic content.** This is a concept more easily demonstrated than described, so make sure to type along with the video.

Defining a callback

Consider the following portion of a web page:

```
<script>
    $(document).ready( function() {
        $('h3').click( function() { alert('You clicked an H3!'); });
    });
</script>
<body>
    <div>
        <h3>Click me to see a message!</h3>
    </div>
</body>
```

Naturally, this function just triggers an alert any time an *h3* tag is clicked. So what happens if we dynamically generate some new HTML by adding some more jQuery and HTML? When first learning jQuery, the different nested ({ }) marks can be very confusing (and error-prone!), so for clarity, we will indent the following example code a bit differently. Once you are comfortable with jQuery callbacks, you can revert to a more compact style.

```
<script>
    $(document).ready(
        function() {
            $('h3').click(
                function() {
                    alert('You clicked an H3!');
                }
            );
            $('button').click(
                function() {
                    $('div').append('<h3>I am a dynamically generated H3</h3>');
                }
            );
        }
    );
</script>
<body>
    <div>
        <button>Click me to add a new H3 tag!</button>
        <h3>Click me to see a message!</h3>
    </div>
</body>
```

If you try to run this code in your browser, you may be disappointed when you click on the dynamically generated *h3*, because it won't show any message.

The point to observe here is that the jQuery we have written is specifically applied to elements that already exist at the time of **page load!** Even further, try right-clicking and going to **View Page Source**. You shouldn't be able to see any of the *h3* tags you made by clicking the button. This is why dynamically rendered content must be dealt with separately than the statically rendered content -- because the browser doesn't really know that the dynamic content exists. So let's define our callback now:

```
<script>
    $(document).ready( function() {
        $('h3').click( function() {
            alert('You clicked an H3!');
        });
        $('button').click( function() {
            $('div').append('<h3>I am a dynamically generated H3</h3>');
            $('h3').click( function() {
                alert('You clicked an H3!');   // **** here it is!
            });
        });
    });
</script>
<body>
    <div>
        <button>Click me to add a new H3 tag!</button>
        <h3>Click me to see a message!</h3>
    </div>
</body>
```

Notice that all we did was restate the original jQuery code for new content after that new content was created. That's all it takes! There is your first callback. Now even a dynamically generated h3 element should be sending you a message. Why does this work? This works because as soon as we created content, we gave the browser instructions to the code we just declared! This is exactly the same process that happens for the static (page load) content; the handlers get attached as soon as the page content is created!

One last thing: note this repeats the code, which is something developers dislike doing. Following the principle *Don't Repeat Yourself* (DRY), let's DRY out our code. Here is the convention we use to achieve this:

```
<script>
    function attach_h3_handlers() {
        $('h3').click( function() { alert('You clicked an H3!'); });
    }
    $(document).ready( function() {
        attach_h3_handlers();
        $('button').click( function() {
            $('div').append('<h3>I am a dynamically generated H3</h3>');
            attach_h3_handlers();
        });
    });
</script>
<body>
    <div>
        <button>Click me to add a new H3 tag!</button>
        <h3>Click me to see a message!</h3>
    </div>
</body>
```

Notice all we did was just define a function that did exactly what the original code did. This allows us to just call the function instead of rewriting the code, which is a big time-saver! Another thing to note: we defined our function ***outside*** our **$(document).ready()**function but called it ***within*** the **$(document).ready()** function. Take a moment to think about why this worked.

Here's why: by calling the function ***within*** the **.ready()** tags, we told our browser to search **all** of our JavaScript for a function of that name and execute it. Pretty sweet, huh? That is part of the beauty of functions in JavaScript. When the browser/interpreter reads a statement that invokes a function, the **entire** document is scanned for a function of that name, meaning it could be defined on a latter segment of the page and the interpreter/browser will still be able to find it.