JavaScript 2: jQuery and the DOM

These are notes for the second JavaScript workshop at NICAR2016 in Denver. This workshop was intended to follow intro to JavaScript and is best for very beginning programmers. However, you'll get the most of out of this if you have some familiarity with HTML and CSS.

The workshop is in two parts: a short presentation on the DOM and a longer hands-on session on DOM manipulation.

- Slides for presentation on the DOM
- Example page for hands-on

For the workshop, we use an example webpage I've built and we write our code in a separate .js file to establish best practices. To follow along, download this whole repo, open the `example` folder and double-click `index.html` to open the webpage in your browser. The .js file is located in `example/js/script.js`.

Alternatively, these examples work just as well when you enter the code directly into the JavaScript console in your browser. To do that, open the hosted version of the example webpage and then open your developer console. For Google Chrome browsers, use this guide. For Firefox, use this guide. For Internet Explorer, stop using Internet Explorer. Download Google Chrome or Firefox, click yes when asked to make it your default browser, and be a happier programmer.

Javascript and the DOM

What is the DOM?

When we refer to the DOM - or Document Object Model - we mostly just mean HTML. Is it more complicated than that? Well, yes it is. But it is fine if you just think "HTML" when you hear "DOM." Just know that we refer to the DOM when we talk about the ways in which programs (in this case, JavaScript) interacts with HTML.

The rest of the story

When you write HTML the browser reads your `<p>` tags and your `<table>`s and it knows how to make that into a webpage.

How exactly does it do that? I'm not really sure! That's not the important thing for our purposes today. The important thing is that when it creates a web page, the browser doesn't *just* turn tags into elements, but it stores those elements in a Javascript *object* called `document`.

When I say object, I mean an object of the JavaScript kind, the kind which usually looks like this: `{ 'key': 'value', 'another key': 'value' }` This object is stored in a global variable called `document`.

This is actually super cool. Just try it out! In any web page you can open up the console and type 'document'. That will print to the console an object that has a ton of levels. You can explore this object with the developer tools. You might notice that it's hierarchical just like HTML is hierarchical when it's formatted correctly. The `body` tag contains your content. That content might be divided into `div` tags. A `table` tag might contain `tr` tags which contain `td` tags. The browser organizes all these tags into a structure we call a "tree." Really it's just object inside of objects.

What's the big deal here?

Two things: First of all, the browser has neatly organized the HTML(really just text) to a JavaScript object before you've even done any JavaScript coding. Imagine if you had to do that yourself! It sounds pretty hard.

Second of all, the browser attaches all these functions to the `document` object so that you can **manipulate** it. Recall that an object, in JavaScript terms, can have key/value pairs as well as functions. `var myObject = { "stringKey": "string", "numberKey": 9, "functionKey": function(){ console.log("hello!") } }` When you call myObject.functionKey(), "hello" will be printed to the console. In this way, the browser provides useful functions attached to the `document` object. These functions are quite a bit more complicated than a simple console.log and can do things like return specific HTML elements, change them, and remove them. This, in a nutshell, is what we mean by "DOM manipulation".

What does this do for you?

Here are just two things you can do with this: - Provide interactivity to webpages. When a popup appears as you hover over a chart, that's DOM manipulation. When a photo changes as you click through a gallery, that's DOM manipulation. When you submit data through a form, JavaScript can help you read that data in the inputs and send that data to a server. - Create data visualizations. You've heard of D3.js? Fancy DOM manipulation.

So how do we do this DOM manipulation thing?

- Today we're going to use jQuery, which is a JavaScript library that is used mostly for DOM manipulation. We load jQuery into our webpage via a `<script>` tag. I've already done that for you in the example webpage.

You might be wondering: why do we need a library if the browser already gives you all these handy functions for DOM manipulation? Here's why we use jQuery instead: 1. The jQuery syntax is simpler and shorter than the native browser functions. 2. As browsers change over the years, the DOM manipulations change too. If your user has a very old browser and you use the newest methods, your code might break on those browsers. jQuery makes sure your code works on all the different browsers going back for a few years. 3. jQuery is ubiquitous. Learning it helps make you read other people's code and become a better programmer.

A caveat

There are good reasons not to use jQuery in every single app. Often times it's fine to use the native DOM functions, especially as older browsers fall out of favor. But the benefits usually outweigh the cons and you can worry about this nitpicking later when you're more jQuery fluent.

Our webpage

Take a moment to look at the code in our example webpage. We have two paragraphs, two buttons a ordered list. Notice that the first graf has an id of `first`. We'll use that id to first **select** our element, and then **manipulate** it. That's our general pattern when we write jQuery: select, then manipulated. Select, then manipulate. You'll get used to it.

Open `/example/js/index.html` by double clicking the file in your finder. It should open up in your default web browser. Then, write the below code in `/example/js/script.js` inside of the function that I've already written there. The comments will indicate exactly where. `jQuery('#first').hide();` Save `script.js` and then reload your browser. You should see that our first graf has disappeared. We can bring it back with the `show()` method. You can write this directly below the "hide" code you wrote above. It should cancel out the effects of the first statement and the document will look normal again. `jQuery('#first').show();`

Notice that we select the first graf by writing its id in the jQuery function preceded by a `#`. That's how you would select the element in CSS. When we write jQuery we are not always concerned with styling, but our selection syntax generally follows CSS selectors. Classes are written `.className` and ids are written `#IDName.`

Fun stuff

- jQuery's `hide` and `show` functions take one optional argument to regulate the speed of the transition. Try using `fast`, `slow`, or a number in milliseconds.

```
jQuery.hide('slow');
```

Dollar sign

You've probably seen code that uses a dollar sign instead of `jQuery`, as in, `$('element').show()`. That's what most people use, but really it's just shorthand. This used to confuse me, I used to think that the dollar sign was some special syntax that was unlike any other kind of JavaScript. It's not, it's just a function. `$` = `jQuery` and you can use them interchangeably.

```
$('#first').hide();

jQuery('#first').show();
```

Changing styles

Changing CSS on-the-fly is a great use of jQuery. jQuery gives you a CSS-like syntax for changing styles. Why would we do this in jQuery instead of CSS? Because with jQuery we can change styles in reaction to some kind of event like a click, key-press or mouseover.

Let's change the background color of the `body` tag. `$('body').css('background-color', 'maroon');` The `css` method is just another jQuery function like `show`. Only this one has different rules. Remember how the `show` function takes one optional argument? The `css` function takes two *required* arguments. There are tons of jQuery functions. Look at the documentation when you encounter a new one so you can be sure how to use it.

Let's change the css of the 'text' tag so that we can read it better. `$('body').css('color', 'white');` (Bonus info: Why do we select 'body'? Because all the `<p>` s and ` `<li>` s are *children* of body. CSS properties affect all their child elements hierarchically.)

Changing content

So far we've mostly just messed with the CSS. But we can also change the actual content of webpages.

The `html` function takes one argument, a string.

Let's change the header of my webpage. `$('h1').html('My New Title');` Let's rewrite the first paragraph.
`$('#first').html('The old paragraph sucks. This one is much better.');`

Interactivity

This is the fun stuff. You can make your elements interactive with the `on` function. The `on` function takes two arguments: a special string and a special "callback" function. The first argument must correspond to a DOM `event.` There are a lot, but some common ones are `click`, `submit`, or `mouseover`. When an event of that type is triggered, then the callback function (the second argument) fires. That's important to note: the callback function does not fire when it is first defined and read by the browser. It *only* fires when the event is triggered.

Here's the general syntax (don't write this into your `script.js` file): `$('ELEMENT').on('EVENT_NAME', CALLBACK_FUNCTION);` When we write the second argument, a function, we have the option of writing the function directly in the jQuery statement, or we can can define it ahead of time. In the syntax example above, we assume the function has a name, `CALLBACK_FUNCTION`.

However, it's actually pretty common to write the function directly inside of the jQuery statement. It sounds confusing, but it doesn't look too bad if the function is short:

Select the button and give it a 'click' event. `$('#hide').on('click', function(){ alert('Did this work?'); });` The function which contains the `alert()` is called an "anonymous" function because it is not stored in a variable and therefore has no name.

It might be easier to read if you write the handler function outside of the `.on` syntax and store it in a variable. That's fine too!

```
//first define the function
var myAlert = function(){
  alert('Did this work?');
}
//then use it in a jquery statement
$('#hide').on('click', myAlert);
```

Be sure to write the function just like this: `myAlert`, not as `myAlert()` or else you'll actually *call* the function as soon as its loaded, instead of when the click event happens!

Let's combine what we've learned by using some more jQuery inside of our click handler. `$('#hide').on('click', function(){ $('#first').hide('slow'); });` Let's break that down real quick: we selected the button which has an id of `#hide`, we used the `on` function to bind a `click` event to the button. In a callback function, we selected a different element, the element with the id of `#first`. We manipulated that element with the `hide` function and supplied the optional argument `slow` to make sure we could see the hiding happen.

Save this in your `script.js` and then click the "hide" button to watch the magic happen.

If we attach `show()` on the other button and we've got a fully functioning interface! Write this code right below the above click handler.
`$('#show').on('click', function(){ $('#first').hide('fast'); });` Just so we're on the same page, your full JS code will now look like this. The comments are optional ``` $(document).ready(function() { //Attach click handler to hide button $('#hide').on('click', function(){ $('#first').hide('slow'); });

```
//Attach click handler to show button
$('#show').on('click', function(){
  $('#first').hide('fast');
});
```

}); ```

Filtering your selections

So far we've been using IDs to make sure we select the right element. But what if we don't have an ID available? Open up the developer console (option + shift + J on Mac Chrome) and we'll code there so that we can get immediate output

Let's say we need to select the second graf, not the first one. We have a unique ID for the first graf but not the second. How might we select the second graf? `$('p');` This returns an array with ALL of the

tags on the page. There are two, so we just get two. To select the second one, we use a special selector: `last`. `$('p:last');` That returns just one element. Now you can do anything with this element that you can do with normal jQuery. `$('p').hide('slow');` **Bonus info**: If `$('p')` spits out a regular array, why can't I just select from the array like I would any array? `$('p')[1]` The answer is yes, this does return the second paragraph element in the page (remember that arrays start at zero, so `myArray[1]` returns the second element). However, that element will be actual HTML code, not a 'wrapped' jQuery element. It's just text at that point. You can still do jQuery stuff with this element, but you will have to 're-wrap' it again. It's easiest to do that by storing the element in a variable, and then wrapping that element in jQuery.
`//select the second graf and store it in a var var secondElement = $('p')[1]; //do jQuery stuff with this var $(secondElement).hide('slow');`

A more practical selection example

Let's say we want to make the first element of our list bold `$('li:first').css('font-weight', 'bold');` Something crazy: make all the odd lines bold!
`$(li:even).css('font-weight', 'bold');` Why did I write `li:even`? Because JavaScript counting starts at 0! So the first `li` is actually the "zeroth" in JavaScript-speak. The second `li` is the "first" to JavaScript.

Other important tools in the DOM-finding toolkit:

- .find()
- .filter()
- child selectors
- If you are stuck, some terminology might help: a search for 'DOM traversal' or 'jQuery filters' can help. There are lots of ways to find the element you're trying to get and there might be solutions out there you haven't considered.

Resources

- CodeSchool's free jQuery intro course
- Mozilla Developer's Network jQuery tutorial