jQuery

A JavaScript Utility Library

What is a "library"?

- A library is simply a collection of functions and code that someone has written to make a set of tasks easier
- By including the library in your code, you'll have access to its functions and methods
- There are all kinds of libraries for common components or functionality you'd want, so you don't have to write 100% of the code yourself!

Exercise: Understanding Libraries

- Create a new file, `my_library.js`
- Inside of this file, write a function called `libraryFunction` that triggers an alert message
- Include `my_library.js` in an HTML file
- Call the function in an arbitrary `<script>` block after your library has been loaded with a `<script>` tag:

```
<script>
  libraryFunction();
</script>
```

Common library problems

- Load the page with the JavaScript console open (Command + Option + J) to see if any errors are raised when the page and library are loaded
- Sometimes libraries conflict with each other due to similar naming conventions of functions and methods, or your code accidentally overwrites the library's names
- To check to see if a library is loaded, try calling one of its functions or methods. For instance, to make sure the jQuery library is loaded, just try calling `jQuery` in the JavaScript console.

What is "jQuery"?

- jQuery is a self described "write less, do more" library. It's used by many companies, 67.6% of the top 10k sites use it (as of 2017, via BuiltWith.)
- It can be marginally slower than using native JavaScript but exposes common JavaScript functionality in a very developer friendly and cross browser way.
- There are different versions of jQuery that might have slight differences and support more or fewer browsers. Be sure to check your version number when working on a project that uses it.
- Many additional libraries are built on top of jQuery, and will require it.

Including the jQuery library

- There are two ways of including jQuery
- The preferred way is to download it and `src` it locally,

```
<script src="./scripts/jquery.js"></script>
```

- Alternatively you can include it from a CDN, simply search "jQuery CDN" and use a provided link,

```
<script src="https://code.jquery.com/jquery-3.2.1.min.js"></script>
```

Using jQuery in Code

- Once you've included jQuery on a page, or opened a page that has it, you'll be provided a single variable: $
- All methods can be called using this dollar sign, either directly or using dot notation
- You're also actually provided `jQuery`, but `$` and `jQuery` are the same, just aliases

Selecting elements with jQuery

- jQuery can be used to select elements using the same syntax as CSS or `querySelectorAll()` by using `$` as a function

Vanilla JS (document.)	**jQuery ($)**
`getElementById("#id")`	`$("#id")`
`getElementsByClassName(".class")`	`$(".class")`
`querySelectorAll("#id tag.class")`	`$("#id tag.class")`

- Try this first line in the JavaScript console of a page that has jQuery included:

```
$("body")
> [<body></body>]
```

Selecting elements with jQuery

- An array of `HTMLElements` is returned just like the document functions
- However, the array isn't a normal array like the `document` functions, it's now a `jQuery` object
- This object has enhanced functionality and methods that make complex tasks much easier to perform on multiple objects
- All jQuery functions perform the action on **all selected elements** unless otherwise specified

The jQuery Object: Change HTML

- jQuery objects have an `html()` function that gets and sets innerHTML on each element
- Calling it with no argument will get you the html for the **first** element
- Calling it with an argument will set innerHTML for **every** element

```
// Vanilla JS
document.getElementById("header").innerHTML = "New header!";

// jQuery
$("#header").html("New header!");
```

The jQuery Object: Create Elements

- In addition to finding existing elements, jQuery can help you create new elements
- Elements created won't be added to the page automatically, you'll have to place them

```
// Vanilla JS
var successMessage = document.createElement("p");
successMessage.className = "message-success";
successMessage.innerHTML = "Great job!";

// jQuery
var $successMessage = $('<p class="message-success">Great job!</p>');
```

The jQuery Object: Add Elements

- Now that we can find and create elements, we may want to add new ones or move around existing ones

```
// Continued from previous slide
// Vanilla JS
var container = document.getElementById("container");
container.appendChild(successMessage);

// jQuery
$("#container").append(successMessage);
```

The jQuery Object: Remove Elements

- When we no longer want an element, we may want to remove it from the DOM altogether

```
// Continued from previous slide
// Vanilla JS
successMessage.parent.removeChild(successMessage);

// jQuery
$successMessage.remove();
```

The jQuery Object: Classes

- The most common way of re-styling elements, or showing and hiding them is to add or remove classes that have styles defined in CSS

```
// Vanilla JS
var el = document.getElementById("element");
el.className += " add-class";
el.className = el.className.replace("remove-class", "");

// jQuery
var $el = $("#element");
$el.addClass("add-class");
$el.removeClass("remove-class");
```

The jQuery Object: Event Listeners

- jQuery objects have an on( ) function that applies a listener to each element:

```
// Vanilla JS
var buttons = document.getElementsByClassName(".buttons");
for (var i = 0; i < buttons.length; i++) {
    buttons[i].addEventListner("click", onClickFunction);
}

// jQuery
$(".buttons").on("click", onClickFunction);
```

The jQuery Object: Selecting Children

- More often than not, you want `find()` an element inside of another, not select every instance of that element

```
// Show a checkmark when we click on a button
var $buttons = $(".button");
$buttons.on("click", function(ev) {
    var $button = $(ev.currentTarget);
    var $checkmark = $button.find(".button-checkmark");
    $checkmark.addClass("show");
});
```

- If we didn't use `.find()`, we would have shown EVERY button's checkmark.

The jQuery Object: Children & Listeners

- When you have elements dynamically added and removed, you don't want to have to keep re-binding event listeners
- jQuery's on( ) function will also take a selector as a parameter, to only trigger on an element's children
- Bind to the parent that stays around, but select the children that are dynamic

```
var $list = $(".list");
$list.on("click", ".list-item", function(ev) {
    alert(ev.currentTarget.innerHTML);
});

$list.append('<div class="list-item">Hello!</div>');
```

The jQuery Object: Iterating

- If you don't want to do the exact same thing to each element, you'll want to iterate over each( ) element and treat them differently

```
// Show a checkmark when we click on a button
var $buttons = $(".button");
$buttons.each(function(index, element) {
    element.innerHTML = "Button index " + index; // This works, and...
    $(element).html("Button index " + index);    // This works too, but...

    // ERROR, `element` is an HTMLElement, not a jQuery object.
    element.html("Button number " + index);
});
```

The jQuery Object: Chaining

- Unless otherwise specified, all jQuery object functions return themselves when you call them
- This allows us to "chain" functions back to back, calling a function on that return value
- This can save us some keystrokes, but make sure your code is still readable

```
// Longer
var $someElements = $("#container .elements");
$someElements.addClass("new-class");
$someElements.removeClass("old-class");

// Shorter
$("#container .elements").addClass("new-class").removeClass("old-class");

// Same thing, but broken out in to multiple lines
$("#container .elements")
    .addClass("new-class")
    .removeClass("old-class");
```

Waiting for the page to be ready

- As shown before, putting your script tag in the header makes accessing elements impossible since they're not ready yet
- jQuery doesn't solve that automatically, but it provides us a handy snippet for resolving that: The `$(document).ready(callback)` function
- The ready event fires when all of the page's DOM elements are loaded, even if multimedia elements aren't fully loaded, ensuring all elements can be grabbed

Using `$(document).ready()`

- The one and only argument of `ready` is a callback function

```
// No elements found, nothing happens, sad times
$("#my-element").html("Where's my element?!");

// Executed a little later when the document is ready, happy times
$(document).ready(function() {
  $("#my-element").html("Aaaah, there it is.");
});
```

Caveats & Best Practices

Caveat 1: jQuery Method Does Nothing

- **Problem:** You call a jQuery function and nothing happens. What the heck!
- **Solution:** Make sure you used document.ready and spelled it right. When in doubt, console.log()

```
html
<h1 id="my-heading">Ye Olde Textte</h1>


js
var $heading = $("#my-headding");
$headding.html("New text!"); // Nothing happens
console.log($heading) // Prints "[]"
```

Caveat 2: jQuery Conditionals

- **Problem:** We want to do something based on whether or not we found an element with jQuery, but the condition is always true.
- **Solution:** Check `$element.length`, not just `$element`. We want to see if it has elements, not if it exists!

```
// Bad.
if ($myElement) { /* ... */ }

// Good!
if ($myElement.length) { /* ... */ }

// Even better, if you passed in $myElement
if ($myElement && $myElement.length) { /* ... */ }
```

Best Practice 1: Indicate jQuery Vars

- Because jQuery objects have different functionality than element arrays, we want to be clear what we're expecting
- Adding a $ behind the variable declaration is valid JS syntax, better for readers, and makes function arguments clearer

```
var el = $("#el"); // Bad.
var $el = $("#el"); // Good!
```

Best Practice 2: Cache Elements

- Grabbing elements can be very slow on large pages.
- Every time you select an element, you're searching through the DOM again.
- If you need to use an element multiple times, assign it to a variable and reference that.

```
// Bad.
$(".my-elements").addClass("good-class");
$(".my-elements").removeClass("bad-class");

// Good.
var $myEls = $("#my-element");
$myEls.addClass("good-class");
$myEls.removeClass("bad-class");
```

Best Practice 3: Keep Style in CSS

- jQuery comes with a bunch of fun functions for doing display driven things in javascript, e.g. hiding and showing, animating, sliding
- Doing this in Javascript is less performant, mixes functionality with styling, and makes it very hard to undo those styles
- Instead, we should be adding and removing classes to indicate what something should look like

Best Practice 3: Keep Style in CSS

```
// Bad.
$(".form-message").show().style({
    color: "red",
    fontWeight: "bold",
});


/* Good (CSS) */
.form-message {
    display: none;
}
.form-message.show {
    display: block;
}
.form-message.error {
    color: "#F00";
    font-weight: bold;
}


// Good (JS)
$(".form-message").addClass("show error");
```

Exercise: Make This Todo List Work!

- I've provided you all with a half finished todo list project. You're going to finish it.
- Download the project from here.
- You'll find an HTML file with all of the starting elements and necessary files linked up, a CSS file that's styled them, and a JS file with comments that describe what it should do.
- Fill out all of the comments with real code!

Resources

jQuery's Documentation

Learn jQuery (CodeAcademy)

jQuery Basics (Treehouse)