Learn a JS Library: jQuery

So far in this course we've only been writing "plain" a.k.a. "vanilla" JS. All our code interacts with the browser environment (Web APIs) directly and we haven't been using any third-party library or framework.

When a project uses an external library or framework, we say it has "third-party dependencies": the project depends on code that is beyond your control.

Dependencies

What's the difference between a library and a framework?

Options for project dependencies:

- A project can have no external dependencies
- or it can use one or more libraries,
- or it can rely on a framework,
- or it can use some framework along with a bunch of libraries.

Roughly speaking, a library provides objects, functions, methods that you call from your code, but your code determines the overall flow and structure of your project. jQuery is an example of a JS library. There are also libraries that specialize in working with maps (leaflet.js), manipulating unicode strings (punycode), generating shapes (processing.js), visualizing data (d3.js), manipulating the DOM (jQuery), creating user interfaces (React) and so on.

In contrast, when you use a framework, it's the framework that dictates the structure and flow of your project, and you provide code that the framework can call to fill in the gaps and details. Angular and Ember are examples of "model-view-controller" frameworks for JS.

Libraries and frameworks are specialized to simplify certain tasks so you can focus on the business logic of your app instead of lower-level implementation details.

There are hundreds, no, *thousands* of JS libraries and frameworks today, and they fall in and out of fashion. It's common for developers find themselves feeling paralyzed by the available options. *Which framework should I use for my project? Should I learn poop.js? Should I use this library to solve problem x or implement my own solution? What's the best way to write a modern web application? Are other developers going to think I've fallen behind if I don't use the latest popular framework? What should I learn to improve my resume? Is this framework going to become widely adopted or it is just hype?*[1]

The importance of JavaScript fundamentals

It's a common view that developers reach too soon for a library or framework, forgetting that many things are simple to implement using plain JavaScript without any dependencies.[2]

Regardless of what frameworks and libraries you happen to use, it's a great advantage to know vanilla JavaScript well because...

- You are able to implement custom behaviour that isn't necessarily provided by any library. You could write your own libraries!
- You can evaluate whether it's worth implementing a solution yourself versus using an existing

library.
- When you do use 3rd-party code, you are able to extend or customize its behaviour to suit your needs.
- You understand the fundamentals of the JavaScript language and the browser environment, which makes it easier to debug all code, whether it uses a 3rd-party library or not.
- Knowledge of plain JavaScript is a long-term, transferable skill that is useful in all JS projects, past and future.
- Your experience with plain JS makes it easier to learn new frameworks/libraries, understand how they work, evaluate their pros and cons.

Nevertheless, gaining experience with libraries and frameworks is essential for working on most large projects. In future courses, you will start to see how a large project is held together and set up, how we manage complexity, what we can do to make a project easier to maintain, how dependencies interact, how they are managed and configured, and more.

Pros and cons

Using third-party code implies some trade-offs.

- You can time save time but not having to implement a solution yourself. You can focus on the overall logic of your project, instead of having to deal with low-level details yourself.
- Your code might look simpler, easier to read, because the library abstracts away implementation details.
- The library likely provides a more robust solution than you could easily write because its authors specialize in that area and have been working on it for a long time and have tested it extensively. For example, a huge benefit of using jQuery is its exhaustive cross-browser compatibility.
- Your own project code ends-up being cleaner, easier to read and maintain because the domain-specific code is hidden in the library you're using.

On the other hand, there are drawbacks, too.

- When you include a library in your project, all the library code has to be downloaded by the client when your app loads. (Code bloat.) If you're only using the library for one thing, it may be better to just implement that one thing yourself.
- The library itself may have bugs and these bugs may be harder to identify.
- Debugging can be harder; you have to understand how the library handles errors.
- You have to learn how to use the API provided by the library, which can take as much time as just implementing the solution yourself.
- The library may be easy to use for simple cases, but if you need solve an unusual problem, it can be hard to get the library to do what you need.

Aside from all that, when deciding whether to use a library/framework, also ask yourself:

- Is the library well-supported? Is it in active development or is it stagnant? Does it have good documentation? Does it have a community of users who can answer questions?
- Is the library's license compatible with your own project's license? For example, if you're building a proprietary project you can't have any dependencies that have a "copyleft" style of license.
- Does the library provide good cross-browser and accessibility support?
- How flexible is it? Can you adapt it to your needs, or extend it?
- It is compatible with any other libaries that you're using?

Respecting software licenses

All software has some sort of copyright license. Most libraries and frameworks are licensed as open source software, which allows you to use them for your projects, but different licenses have different restrictions.

When your project uses external software, check the license of each dependency you have and follow its instructions. At the very least you will have to include a copy of each license in your source code to clearly mark which code is not yours and to provide attribution to its original authors.

Resource: guide to common open source licenses

jQuery basics

Here's a simple example without jQuery; it only works on modern browsers:

```
document.addEventListener("DOMContentLoaded", function() {
    var a = document.getElementsByTagName("a")[0];
    a.addEventListener("click", function( event ) {
        console.log("Clicking this link doesn't navigate.");
        event.preventDefault();
    });
});
```

And the same example with jQuery; it will work across all browsers (if you use jQuery 1.12):

```
$(function() {
    $("a").click(function(event) {
        console.log("Clicking this link doesn't navigate.");
        event.preventDefault();
    });
});
```

jQuery is a DOM manipulation library first created in 2006 to make it easier to write cross-browser-compatible client-side code. Its popularity probably peaked in around 2012. Today, jQuery isn't "trendy", but it's used in 70% of web apps [3]. It simplifies the creation of visual effects while taking care of cross-browser considerations for you.

Try out this introductory tutorial to review jQuery basics.

Including jQuery in your project

- To work with jQuery locally and debug your code, download the jquery js file in uncompressed form
    - If you download the compressed/minified js file, debugging any errors will be impossible because the lines of code will be unreadable. The compressed js file is for deloyed apps only.
- When actually publishing your app, your script tag should link to jQuery that is hosted on a CDN to provide quick loading of the library.
    - jQuery is so common, your user's browser has likely downloaded a copy from the CDN (Content Delivery Network) before, which means it probably already has a cached copy and doesn't have to re-download.
    - If a new copy has to be downloaded, the CDN will make sure your user's browser downloads from the nearest available server.

Let's say the file you downloaded is called `jquery.js`. To use it, include it in a `<script>` tag in the document `<head>` before any other `<script>` tags that might want to refer to the jQuery namespace.

Which version to use?

- The shiny new jQuery 3 doesn't support Internet Explorer 6-8, but it takes better advantage of modern Web APIs and is required for some new jQuery plugins.
- If your project needs to support Internet Explorer 6-8, then use jQuery 1.12. This "branch" of jQuery isn't going to get any new features, but the core functionality is mostly the same as 3, and it will continue to receive support and bug fixes for a while.

`$` is the jQuery function

```
$(function() {
    $("a").click(function(event) {
        console.log("Clicking this link doesn't navigate.");
        event.preventDefault();
    });
});
```

In the example at the top, we start by invoking a function named `$`.

- `$` is the property that jquery.js adds to the `window` object; it's how you access all jQuery functionality.
- `$` is an alias for `jQuery`, so `$(blah)` is the same as `jQuery(blah)`.
- `$` behaves differently depending on the *type* of the argument you provide.

document-ready

Same example as before:

```
$(function() {
    $("a").on("click", function(event) {
        console.log("Clicking this link doesn't navigate.");
        event.preventDefault();
    });
});
```

In the above example, we start by passing a function to $, and `$` causes that function to be triggered when the DOM is loaded.

So `$(someFunction)` has the same role as `document.addEventListener("DOMContentLoaded", someFunction)`. An older way to handle DOMContentLoaded in jQuery is with the ready method, which looks like `$(document).ready(someFunction)`, but this is deprecated.

There are many more ways to use the jQuery function.

Retrieving elements

`$(selector).action(...)` is a common pattern when using jQuery. You can see it in previous examples and here:

```
// Find all h1 elements in 'important' CSS class and make them red
$("h1 .important").css("color", "red");
```

When you pass in a selector expression to the `$` function, it returns a jQuery object. The selector is similar to CSS selectors: `p`, `.someClass`, `#someID`, etc.

Instead of a selector, you can also pass in one or more Element objects to the `$` function to "wrap" them in the jQuery object:

```
// the two following lines are mostly* equivalent
var x = $(document.getElementsByTagName("p"));
var y = $("p");
```

This is useful if you are already working with an Element or maybe an Array or NodeList of elements, and you want to convert that to a jQuery object.

Whether we do it in jQuery or vanilla JS, retrieving elements with a selector is a search through the DOM, which has a performance cost. For the sake of efficiency, it's best to cache elements that will be used later rather than searching for the same elements over and over.

The jQuery object

The jQuery object represents a collection of zero or more DOM elements and provides convenient methods to interact with those elements. It's the return value of the jQuery function, `$`.

If the selector you use doesn't match any elements, the resulting jQuery object will have `length` 0. Example: `$("#doesNotExist").length == 0`

`$("h1")` returns a jQuery object that represents all the h1 elements in the DOM.

- To get the first h1 DOM element, write `var h = $("h1").get(0)`. Here, `h` is of type `Element`, which is part of the standard Web API, not jQuery, so we can use properties like `h.parent`, `h.appendChild`, `h.innerHTML`.
- To get a jQuery object that *wraps* the first h1 element, write `var fancyH = $("h1").eq(0)`. `fancyH` represents the first h1 element, but it's not an `Element` object. So `fancyH.innerHTML = "hello"` won't work, but we can use jQuery methods like `fancyH.html("hello").`

Be careful to distinguish jQuery objects from standard Web API data types like Element and HTMLCollection -- they do not have the same properties or behaviour!

If you have an Element object and want to manipulate it with jQuery methods, you can pass it to the ` `$` function to get a jQuery object for it. Example:
`fancyChild = $(document.body.firstElementChild)`

A common naming convention when working with jQuery objects is to use variables that start with `$`, like `$heading`, just to remind ourselves that it's a jQuery object.

```
var heading = $("h1").get(0);
var $heading = $("h1").eq(0);
```

Note that if you call `$` twice on the same arguments, it will return two independent jQuery objects.

```
// first and second are two different jQuery objects that refer to
// the same DOM Element instance
first = $("#intro");
second = $("#intro");
```

Moreover, jQuery objects are **static**, so if you add or remove elements to the DOM, the jQuery objects don't get updated.

Modifying elements

One you've retrieved DOM elements wrapped in a jQuery objects, you can modify the elements with methods provided by jQuery. Note that these methods modify *all* the selected elements.

Change element attributes with attr:

```
// set src attribute of specific image element
$("#myImage").attr("src", "photo.jpg");
// set "checked" attribute to true for all checkboxes on the page
$("input[type='checkbox']").attr("checked", true);
// you can also set several attributes at once by passing in
// an object literal
$("#myImage").attr({
  src: "photo.jpg",
  alt: "Self-portrait",
});
```

Change inline CSS styles with css:

```
$("#myImage").css("border", "solid 1px");
$("div").css({
  "background-color": "blue",
  "margin": "5px"
});
// remove an inline style
$("div").css("background-color", "");
```

Add and remove CSS classes applied to elements with addClass and removeClass. This amounts to adding/removing `class` attributes for the elements:

```
$("#main").addClass("important");
// Add icon class and important class to all image elements
$("img").addClass("icon important");
```

Set the text content with text:

```
$("#main").text("This replaces all the text in the #main element");
```

Set the inner HTML with html

```
// replace the content of each div element with the following HTML
$("div").html("<p>This HTML will get <em>parsed</em> and inserted "  +
    "into the DOM under #main, replacing its children.</p>");
```

You might be wondering, why not just access the `src` property directly on the element, like `myImageElement.src = "photo.jpg"`? If you're already using jQuery in your app, there are two reasons use methods like `attr()` instead of the native properties:

1. `attr("src", "photo.jpg")` and other similar methods return a jQuery object representing the elements you have previously selected, so you can chain many method calls together. See the section on "Chaining" for a detailed explanation.
2. jQuery methods ensure cross-browser stability, since some native attributes can have different names across browsers.

Getter versus Setters

Many jQuery methods behave differently depending on the number and kinds of arguments you pass in. In all the examples in the previous section, the methods set properties on all selected elements. Let's look at some of the same methods when there are invoked as getters instead.

Look up value of given attribute in *first* selected element with attr:

```
// get src attribute of first img element found in document
// same as document.getElementsByTagName("img")[0].src
var s = $("img").attr("src");
// get value of "checked" attribute for first checkbox input found in document
var c = $("input[type='checkbox']").attr("checked");
```

Look up *computed* style for given property is *first* selected element css:

```
// get computed background-color of first div element
$("div").css("background-color");
```

Computed style refers to the overall combination of all external, internal and inline styles applied to the element.

Get all the text content of *all* selected elements and their children with text:

```
// If the #main element is
// <section id="main"><h1>Welcome</h1><p>Hello there!</p></section>
// then "Welcome Hello there!" will be assigned to t.
var t = $("#main").text();
```

Get the inner HTML of the first selected element with html

```
// If the first section is
// <section id="main"><h1>Welcome</h1><p>Hello there!</p></section>
// then "<h1>Welcome</h1><p>Hello there!</p>" will be assigned to h.
var h = $("section").html();
```

Method Chaining

The following line of code shows how jQuery methods can be chained together.

```
$("#registration input").addClass("big").css("width", "300px").attr("required", true);
```

To make it more readable, it can be broken into many lines, like this:

```
$("#registration input")
    .addClass("big")
    .css("width", "300px")
    .attr("required", true);
```

It's equivalent to writing:

```
var inputEls = $("#registration input");
inputEls = inputEls.addClass("big");
inputEls = inputEls.css("width", "300px");
inputEls = inputEls.attr("required", true);
```

Let's describe step-by-step what is happening.

1. `$("#registration input")` calls the `$` function and returns a jQuery object that represents all input elements that are under an element whose id is `registration`.
2. `addClass` is a method on that jQuery object and it reeqturns another jQuery object that represents the same input elements as before. Now these input elements all belong to the CSS class called big.
3. `css` is a method on the new jQuery object, it adds a style attribute to all the input elements in the jQuery object, and then returns another jQuery object that represents the same input elements.
4. Finally, the `attr` method adds a required attributed to all the input elements in the new, new jQuery object and returns yet another jQuery object that represents those same elements.

In summary, setter methods on The jQuery Object usually return a jQuery Object, so you can continue calling more methods like this in a "chain."

When you use method chaining, keep careful track of the return value of each method to make sure you are operating on the elements you actually want to change.

For example, what does this do?

```
$("p").css("color", "red").eq(2).css("color", "blue").addClass("different");
```

1. Get all paragraphs in the document.
2. Change the `style` attribute of each paragraph to be "color:red" and return all paragraphs.
3. Get the third paragraph.
4. Change the `style` attribute of the third paragraph to be "color:blue" and return the third paragraph.
5. Append "different" to the `class` attribute of the third paragraph and return the third paragraph.

Effects

There are jQuery methods to create animations and effects on elements.

The show and hide methods change elements' `display` CSS property. When `display` is `none`, the element occupies no space on the page. If you specify an speed or interval like `$("#cat").hide("slow")`, jQuery gradually changes the width, height and opacity CSS properties of the element asynchronously over the specified time interval.

The fadeIn and fadeOut are similar.

There's also a handy jQuery.fx.off property: when set to true, the duration of all effects is changed to 0. This can be a nice option to offer to any users who find animations annoying, or need your app to be accessible, or a running on older browsers or platforms.

DOM manipulation

To create one or more new elements, just write a string of HTML code and pass it to the `$` function:

```
// create one section elements
var s = $("<section/>");
// create a whole subtree of elements
var t = $("<section><h1 class="foo">Welcome</h1><p>Blah blah</p></section>");
// create an element with some attributes
var a = $("<img/>", {src: something, alt: description});
```

The remaining examples in this section will all refer to the following HTML code:

```
<section id="main">
  <h1>Introduction</h1>
  <p>First paragraph</p>
  <p>Second paragraph</p>
</section>
<footer>Author info</footer>
```

The following code adds a paragraph after "Second paragraph": get all paragraphs, pick out the last one, create a new paragraph that contains "Third", add it to the DOM after the last one. The return value of after here is the second paragraph.

```
$("p").last().after($("<p>Third</p>"))
```

If instead we called `after` on the jQuery object representing all paragraphs (`$("p").after("<p>New</p>")`), we would create and insert two new paragraphs: one after "First paragraph" and another copy after "Second paragraph." The before method is similar.

`last()` is an example of a filtering method.

To insert a new element as the last child of another, use append. To insert it as the first child, use prepend instead.

```
// append a paragraph as last child of section
$("section").eq(0).append($("<p>Another</p>", {id: "par4"}));
```

And of course, you can remove elements as well.

```
// remove all paragraphs and their child content from the main section.
$("#main").remove($("p"));
// A different way to call `remove`: remove all section elements
$("section").remove();
```

When you remove an element, you remove all events and data bound to the element and its children. An alternative is detach, which just removes the elements and children from the DOM while keeping all events. Use detach if you want to put the element(s) back into the DOM at a different location.

> The .detach() method is extremely valuable if you are doing heavy manipulation on an element. In that case, it's beneficial to .detach() the element from the page, work on it in your code, then restore it to the page when you're done. This limits expensive "DOM touches" while maintaining the element's data and events.

Concepts/properties/methods like parents, children, next sibling also correspond to properties on jQuery objects. The find is also useful for searching a subtree of the DOM.

Event Handling

To attach an event handler, select the element(s) you want to attach to, call the on method:

```
// like document.getElementById("button").addEventListener("click", doSomething);
$("#button").on("click", doSomething);
```

You can also attach many event handlers in one call to `on` by passing in an object literal:

```
$("#button").on({
  mouseover: doSomething,
  mouseout: function (e) {
    // something else
    }
  });
```

You can remove event handlers with off, as long as they were attached with jQuery in the first place. Example `$("body").off("click", "#button");` -- there are many other ways to call `off`.

What we've learned so far about event delegation, bubbling, capture and the Event object in vanilla JS is pretty much the same in jQuery, except that some cross-browser issues are simplified. One nice thing is that the `on` method passes a jQuery Event object to the event handler.

```
function doSomething(e) {
  // because doSomething was attached via jQuery,
  // e is a jQuery event object and it's guaranteed
  // to be there and it's guaranteed to have a target
  // property
  // get a jQuery object for the target element:
  var target = $(e.target);
  target.css("color", "red");
  // to access the Web API event object (from vanilla JS)
  var original = e.originalEvent;
}
```

You may still want to interact with the original Web API event to access some properties that the jQuery event hasn't copied over to itself.

The jQuery event is guaranteed to have the following:

- `target`
- `preventDefault()`
- `stopProgagation()`
- `type`
- and more depending on the type of event.

You can also pass your own data to jQuery event objects when first attaching the event handler.

Ajax

jQuery provides an $.ajax function, and other convenience methods like `$.get()`, `$.getJSON()`, `$.post()`, and `$().load()`. These wrap the properties of the standard XMLHttpRequest object. (I'll only describe the $.ajax function because it's best-practice to use just that, rather than the convenience methods.)

As in plain JS, responses are handled asynchronously with callback functions. A big difference between $.ajax() and XMLHttpRequest is that there is no `onreadystatechange` handler. There is also no distinction between "open" and "send": to send a request you just call the $.ajax() function.

`$.ajax()` accepts an object literal as a parameter. This object configures the request. It's called the "options" or "settings" or "configuration" object.

```
var request = $.ajax({
  url: "https://represent.opennorth.ca/representatives/",
  data: {gender: "F"},
});

request.done(function (data, status, r) {
  // do something with the json data (could be other data type)
  // r is the request object
});

request.fail(function (r, status) {
  // deal with failure (with request or with server)
  // r is the request object
});
```

In the example above, we could have set the `type` of the request to `"GET"`, but that is the default setting. Since it's a GET request, the name-value pair in the `data` property will be converted to a URL query string. jQuery takes care of encoding the data as well, so we don't need to call `encodeURIComponent` ourselves.

The data type of `request` is a jQuery-XMLHttpRequest, which has most of the same properties as XMLHttpRequest, and more. The `done` and `fail` functions don't distinguish between errors with the request or with the server: either we got the expected data back from the server (done) or something went wrong (fail).

So you don't (and can't) check the readyState of the request, or the status code of the server. The jQuery ajax function abstracts away all that.

As an alternative, you can register `success` and `error` callbacks directly in the options object:

```
var request = $.ajax({
  url: "https://represent.opennorth.ca/representatives/",
  data: {gender: "F"},
  type: "GET",
  success: function (data, status, r) {
    // do something with the data
  },
  error: function (r, errorType) {
    // deal with the error
  }
});
```

The main difference between `success` and `done` is that there can only be one `success` callback, but you can register many callback functions with `done`. Same with `fail` versus `error`.

jQuery Plugins

Conclusion

We've examined the trade-offs to consider when choosing library or framework, and looked at jQuery as our main example.

jQuery Core is a well-designed library that makes lots of common tasks more convenient by reducing how much we have to type and hiding away some details. The ongoing work on developing jQuery has resulted in a treasure-trove of cross-browser wisdom. The project has great documentation and community resources.

All that being said, we know plain JavaScript and many parts of the modern Web API, which means we know that jQuery doesn't do anything magical for us. We understand what jQuery methods are doing and we can definitely implement some approximation of all these behaviours ourselves. We don't need jQuery but we can make a thoughtful decision about whether to use it or not. The same applies to any other library or framework you might learn next.

---

1. This sarcastic article illustrates what is overwhelming about the JavaScript landscape today.↩
2. Do you get this joke?↩
3. https://w3techs.com/technologies/details/js-jquery/all/all↩

Last update: 2017-05-02 00:32:19

Source: _javascript_423/lectures/12-14_jquery.md