Introduction to jQuery

Download Demo Code <../jquery-demo.zip>

Goals

- Develop a conceptual understanding of jQuery and its methods
- Explain why you would or would not use a library like jQuery
- Compare and contrast jQuery with vanilla JavaScript

jQuery

What is jQuery?

It's a library for:

- Manipulating the DOM
- Adding Event Listeners
- Animating Elements
- Making HTTP Requests (AJAX)

Why should you still learn jQuery?

- Brevity and clarity
- Cross-Browser Support
- AJAX
- 77% of the top 1,000,000 most visited pages use it

Including jQuery and Selecting Elements

Including jQuery

- https://code.jquery.com/ <https://code.jquery.com/>
- Once you include jQuery script, you have access to global ***$***

> **Note: $ in the browser console**
>
> Just because ***$*** has a value in your browser's console, this does *not* mean that the page you're on necessarily uses jQuery. Most browsers reserve ***$*** as a sort of shorthand for

> ***document.querySelector***, unless some library overrides this behavior.
>
> If you see something like `ƒ $(selector, [startNode]) { [Command Line API] }` as the value for ***$***, this means that jQuery is *not* installed. On the other hand, if you see something like `ƒ (e,t){return new he.fn.init(e,t)}`, this means that a (minified) version of jQuery has been installed.
>
> ***$*** is just a shorter alias for a global ***jQuery*** object when jQuery is loaded, so another test is just to check in the console whether there's a global variable called ***jQuery***.

Selecting elements

It's as easy as using CSS selectors! (except you need to remember your CSS selectors)

```
$("ul")
// like document.querySelectorAll,
// this will select ALL uls

$("#todo-container")

$(".carousel-image")
// like document.querySelectorAll,
// this will select ALL the elements with that class
```

What does this give you?

A jQuery object

jQuery objects are NOT the same as DOM elements

To access an element, use the ***get*** function:

```
let $listItems = $("li");
$listItems; // a jQuery object

$listItems.get();
// an array of HTMLLIElements

$listItems.get(0);
// the first HTMLLIElement
```

Storing jQuery Objects in variables

It's a common convention to store jQuery objects in variable names that begin with ***$***.

This isn't necessary, but clearly communicates your intent.

```
let x = $(".class1");
let $class2Elements = $(".class2");

// 200 lines later...

console.log(x);
// wtf is this

console.log($class2Elements);
// nice, this is probably a jQuery object!
```

jQuery Methods

Common jQuery Methods

A great way to learn these is to compare them to vanilla JS methods!

- ***.val()***
- ***.text()***
- ***.attr()***
- ***.html()***
- ***.css()***
- ***.addClass()*** / ***.removeClass()*** / ***.toggleClass()***
- ***.empty()*** / ***.remove()***
- ***.append()*** / ***.prepend()***
- ***.find()*** / ***.closest()*** / ***.parent()*** / ***.next()*** / ***.prev()***

jQuery getter / setter pattern

- Vanilla JS: `.getAttribute(attrName)` and `.setAttribute(attrName, newValue)`
- jQuery: `.attr(attrName, newValue)` *(second param is optional)*
- This is common with jQuery methods

Chaining with jQuery

Almost all jQuery methods return a jQuery object, which allows for *method chaining.*

Instead of performing DOM operations line-by-line, we can chain method calls together on a single jQuery object.

Instead of:

```
let todoContainer = document.querySelector("#todo-container");
todoContainer.style.color = "red";
todoContainer.innerText = "look at this!";
todoContainer.addEventListener(
    "click", function(evt) { console.log("clicked!") });
```

We can have

```
$("#todo-container")
  .css("color", "red")
  .text("look at this!")
  .on("click", function(evt) { console.log("clicked!") });
```

Creating elements

Instead of using `document.createElement("li")` we can simply create an element using `$("<li>")`

- `$("<li>")` Create a new ***li***
- `$("li")` Select existing ` <index.html#id2>li`s

Waiting for the DOM to load

With vanilla JS we have ***DOMContentLoaded*** and ***window.onload***, with jQuery we have:

```
// waits for the DOM to load
$(function() {

});
```

You may see this version:

```
// waits for the DOM to load
$(document).ready(function() {

});
```

Events and Delegation with jQuery

jQuery events

jQuery's ***on()*** works similarly to ***addEventListener***. It lets you specify the type of event to listen for.

```
//prints when item with id "submit" is clicked
```

```
$("#submit").on("click", function() {
  console.log("Another click");
});
```

```
//alerts when ANY button is clicked
$("button").on("click", function() {
  console.log("button clicked!");
});
```

Why Use on()?

In most cases, ***click()*** and ***on("click")*** will both get the job done. HOWEVER, there is one key difference:

- ***.click(callback)*** is a shorthand for ***.on("click", callback)***
- ***on()*** accepts optional argument between type of event and callback
- This flexibility allows us to leverage *event delegation*.

Event Delegation

Event delegation allows us to attach an event listener to a parent element, but only invoke the callback if the event target matches a certain selector.

This will work *even if elements matching the selector don't exist yet!*

```
$("#meme-container").on("click", ".meme", function(evt) {
  deleteMeme(evt.target);
});
```

- Less code
- More performant

Event Delegation: Vanilla JS vs. jQuery

Vanilla JS

```
// deletes a meme when it is clicked
// even if it doesn't exist on page load

document.getElementById("meme-container")
  .addEventListener("click", function(evt) {
    let target = evt.target;

    // checking for "meme" class on target
    // this logic would need to change a bit
    // if we were searching by something
    // else (eg tag name)

    if (target.classList.contains("meme")) {
      deleteMeme(target);
    }
  });
```

jQuery

```
// deletes a meme when it is clicked
// even if it doesn't exist on page load

$("#meme-container")
  .on("click", ".meme", function(evt) {
    deleteMeme(evt.target);
  });
```

Wrap Up

Why might you not use jQuery?

- The DOM API is much more standardized
- It doesn't do anything you can't do on your own
- It's an unnecessary dependency

> **Note: You might not need jQuery**
>
> If you're ever on the fence about whether you should include jQuery or not, here's a website that shows you how to implement a lot of jQuery functionality with vanilla JavaScript: You Might Not Need jQuery <http://youmightnotneedjquery.com/>.
>
> Their general philosophy is that if you want to use jQuery because it makes building your app better, great! Go for it. But if you're building a library, it's worth asking whether you *need* a dependency like jQuery.

Your turn!

jQuery has some of the best documentation out there