on to jQuery

After using Javascript for a while, you'll get sick of writing out `document.getElementById('id')` every time you need to select a DOM element. That's where jQuery comes in.

jQuery is the most prolific Javascript library. It provides concise macros and cross-browser functionality for common JS operations, making JS less painful and more readable. See http://jquery.com/download/ to get the source files – you can either download a local copy or link to the copy hosted by Google's CDN.

jQuery() function; shorthand: **$()**; http://api.jquery.com/jQuery/

You'll see this often used as a DOM selector.
`$('#id')` is shorthand for `document.getElementById('id')`, and
`$('.class')` is shorthand for `document.getElementsByClassName('class')`.

You can also use it to create DOM elements on the fly. `$('<div>hello!</div>')`

Chaining functions
$() actually creates a jQuery object from your selected DOM elements, which can then have jQuery methods applied to it. In turn, jQuery methods return the jQuery object they operated on.

This means that you can chain them together instead of maintaining a mess of variables to pass jQuery objects around with. Let's change the attributes and the html of some elements in just one line:

```
$('.class').attr('title', 'Hello!').html('hello there world');
```

Notice that $() returns a single jQuery object referencing *all* the elements that match your selector. When a jQuery method operates on a jQuery object, it operates on *all* of those DOM elements.

Selectors
You can use the $() function to select DOM elements any way you would use a CSS selector.

Nesting: `$('.one .two')` returns an array of all elements with class 'two' that are descendants of elements with class 'one'.

Filters: `$('div:first-child')` returns all divs that are the first child of their respective parents.

Full documentation: http://api.jquery.com/category/selectors/

DOM Manipulation / Traversal

jQ has many utility functions to help you manipulate and traverse the DOM. Here are some good starting points to have in your back pocket:

.addClass(), .removeClass() – self explanatory

.find() – filter the descendants of a DOM element by a selector. This may not seem useful since selectors can nest, but it is extremely useful in functions scoped to a DOM element, such as event handlers, which we'll discuss soon.

.html() – get or set the HTML nested within a DOM element. Given no parameters, it returns the current HTML content; given a string, it sets that as the new content, for example `$('#hello').html('Hello World')`

.attr() – get or set attributes. `$('img').attr('title', 'This is an image.')`

Full documentation: DOM manipulation, DOM traversal

Event handling

jQuery makes it easy to attach event handlers to objects.

Example: suppose we want to handle clicks on a div with id '`button`'. We can use a jQ selector to find that element in the DOM, and then call the $.click function with an anonymous function as a parameter. It will then bind that function to the 'click' event.

```
1 $('#button').click(
2   function(){ //this function is the event handler
3     //in this scope, this refers to the clicked button
4     $(this).find('.label').html('Clicked!');
5   }
6 );
```

You can attach multiple event handlers at once using function chaining:
`$('#button').click(function() {...}).hover(function() {...});`

optional detour if you care to understand how this works instead of just copying examples

Javascript scoping
The *this* keyword in a function refers to whatever object the function is being called on. In the *click* example, the event handler's *this* is a bare DOM element. That's why we wrap it in $() on line 4 before chaining jQuery functions to it.

A nested function has access to its parent's scope, with the exception of its *this*.

It's common to declare `var self = this;` in the parent whenever you need to reference the parent's *this*.

Anonymous functions
An anonymous function is declared without a named identifier to refer to it.
Normal function: `function hello() { alert("hello"); }`
Anonymous function: `var anon = function() { alert("anon"); };`

The most common use for an anonymous function is as an argument to another function. This is termed a *closure*, and it's especially common to use them in event handling. The event handler starting on line 2 is an example of a closure.

end detour

jQ provides an extremely useful event handling function, **.ready()**, which runs once the entire DOM is fully loaded. This makes it great for setting up event handlers and other functionality that relies on selectors. You will likely have to use .ready() on every page that uses Javascript you've written.

If you are dynamically loading content, you may get into a situation where the elements you're trying to bind to don't exist on page load, or are constantly being reloaded! For example, a photo gallery that loads new photos whenever the user enters a filter. In this case, you might need to delegate events using the .on() function.

Here's a great tutorial on making sure your events are bound correctly:
http://docs.jquery.com/Tutorials:AJAX_and_Events

More: mouse events, form events, browser events

Multiple versions of jQuery
At some point, you'll probably be juggling a bunch of useful jQ plugins – underscore, fancybox, jcarousel, jQuery UI – only to discover that they're in various states of maintenance and built on top of different versions of jQuery. Fortunately, jQ has something called no conflict mode that lets multiple versions of jQuery coexist on the same page.
http://docs.jquery.com/Using_jQuery_with_Other_Libraries

(A good practice is to include the version number, e.g. `var jQuery_1_4_1 = $.noConflict()`, instead of making up random names as this article suggests)

More materials & references
official jQuery tutorials
  general jQuery walkthrough
  interactive tutorial!
jQuery for beginners

jQuery API

CoffeeScript – a cleaner, prettier way to use Javascript, if you're brave

AJAX (Asynchronous Javascript and XML)

AJAX is a blanket term for the collection of technologies used to create asynchronous web applications. By asynchronously exchanging data with a server, we can create interactive web applications that don't need to be refreshed.

Why?
Suppose we need to implement an autocomplete feature that updates whenever the user types into a search box. Without AJAX, there would be no way to do this without refreshing the page.

The 'infinite scroll' common to sites like Facebook and Pinterest is another example of a page that could not be implemented without AJAX. It's obviously impractical to load all of the content in existence at once; instead, a new chunk of data is requested once the user scrolls to the bottom.

Basically, AJAX is used whenever we need to talk to the server without freezing control flow, and without refreshing the page.

What does jQuery have to do with this?
Before jQuery, developers had to implement AJAX differently for every browser. jQ gives us cross-browser AJAX support in the form of the $.ajax function, to which we'll pass parameters such as the request type, url, and data we're sending to the server.

We then attach callback functions (.done(), .fail()), which constitute the asynchronous component of AJAX. The request is fired off to the server. When the client hears a response, it tells the appropriate callback function, which then executes.

jQ also defines some convenience functions which are all you'll need most of the time – you rarely need the full-featured $.ajax function over these.
$.get – perform a GET http request
$.getJSON – GET request where we expect the response to be formatted as JSON
$.post – POST http request

(JSON is a data format that's basically a serialized Javascript object.)

Cross-origin resource sharing
For security purposes, browsers employ the same origin policy, which prevents scripts from accessing methods and properties on pages from different sites.

JSONP requests are not subject to the same origin policy. A brief example:
http://stage.learn.jquery.com/ajax/working-with-jsonp/

Useful tutorials/references

A quick net.tuts+ tutorial covering several AJAX use cases
A thorough tutorial / overview of jQ's AJAX capabilities