Learning jQuery

- Hour 1: Intro to jQuery
  - What is the deal with jQuery
    - Wikipedia defines jQuery as:
      - "…A cross-browser JavaScript library designed to simplify the client-side scripting of HTML.
      - "It was released in January 2006 at BarCamp NYC by John Resig.
      - "Used by over 52% of the 10,000 most visited websites, jQuery is the most popular JavaScript library in use today."
    - What jQuery is for
      - Cross-platform standardization
      - Simplification of complex JS interactions
      - Animation!
    - Why it's become so popular
      - Simple, familiar syntax
        - Use of CSS selectors leverages what HTML devs already know
        - Plain-English, common-sense methods are easy to follow
        - Simple tasks can be accomplished simply.
      - Compactness
        - Syntax and chaining enable a lot of functionality per line of code.
        - Fewer lines of code means smaller file sizes/downloads.
        - Short functions are easier to debug.
      - Community - huge user base; lots of support/tutorials
        - Huge user base means tons of support for your questions.
        - Established, popular framework means a cornucopia of available plugins
      - Ongoing development
        - Heavily optimized JS core engine = fast and stable
        - Evolving core can incorporate up-to-date technologies and design patterns
    - How you might use it on your own projects
      - Ironing out cross-platform bugginess
        - Did you know that IE6 doesn't properly handle floats, margins, positioning, selectors...?
        - Did you know that IE7 interprets comments as part of the DOM?
        - Did you know that Firefox uses a totally different rendering engine than Safari/Chrome?
        - Did you know that Android's webkit is, like, 2 years behind iOS's webkit?
      - Adding interactivity and animated transitions to pages
        - Client-side form validation
        - Accordions to reveal extra content
        - Tabbed subsections
        - Overly complicated navbars
      - Easy AJAX
        - Pull the contents of external HTML files into your page
        - Submit forms without refreshing
  - The Query object
    - You create a jQuery object by placing a CSS selector into the jQuery syntax: $('').
      - You can use advanced CSS3 selectors even in browsers that don't support them.
      - In some cases you can put HTML into the jQuery object, for DOM insertion.
      - jQuery will also convert $(this) into a jQuery object, inside a function.
    - The jQuery object is an array-like JS object
      - Note that the jQuery object is a list of nodes that match your query.
      - Even if there's only 1 match, it's still a list. It's not the node itself.
  - Setting up your jQuery
    - $(document).ready(function() {});
    - Understanding what is and isn't part of the DOM
    - Trying it out
- Hour 2: Utility functions in depth
  - Utility functions: what they are
    - Overview
    - Simple examples
  - Some perform DOM transformations:
    - .css()
    - .before()
    - .wrapInner()
    - .addClass()
  - Some serve to modify the contents of a jQuery object:

Learning jQuery

        - .filter()
        - .has()
        - .find()
        - .closest()
        - .next()
    - And some do animation:
        - .animate()
        - .fadeOut()
        - .slideUp()
    - There are also a few boolean utilities that make conditionals easier:
        - .is()
        - .hasClass()
        - .contains()
- Hour 3: Chaining
    - Intro to jQuery chaining
        - Chaining performs multiple sequential operations on a single jQuery object.
        - When combined with filtering and DOM traversal, it's incredibly efficient.
        - It also teaches you to think in a hierarchical, OO way, which is good for more advanced stuff.
    - Make sure you refine/filter existing selectors rather than performing new queries.
        - $(".slideShow.active").removeClass("running").addClass("stopped").find("li.slide.active).removeClass("active").append("<p class="resumeLink">Resume slideshow</p>);
    - Save your jQuery object to a variable so you don't keep re-crawling the DOM each time.

        -
          ```
          var cart = $("#shoppingCart");
          var cartItems = cart.find("li.purchase");
          if (cartItems.length > cartLimit) {
              cartItems.last().addClass("finalItem")
              cart.addClass("cartFull").find(".cartNav").hide();
          }
          ```
    - You can use end() to keep working with your original jQuery object after you've modified it.
        - $("#carousel li.active").addClass("inactive").find(".pagination").fadeOut().end().next("li").addClass("active").find("pagination").fadeIn();
    - Why we bother optimizing these things when we're talking about milliseconds, here
        - Sure, a desktop browser is fast, but what about handheld devices? Embedded browsers?
        - Those milliseconds add up fast, especially when interactive widgets start competing for cycles.
        - As your skills as a developer increase, the complexity of your projects will increase
        - In web-based applications, the DOM is in constant flux, so you might as well build good habits now.
- Hour 4: Extending jQuery with JavaScript
    - Loops

        -
          ```
          $("#main form .validate").each(function () {
              var myInput = $(this);
              if (myInput.validate() === false) { myInput.addClass("error"); }
          });
          ```
        - Caveat: Loops are easy to understand, but they're huge performance hogs, and can often be avoided if you know what you're doing.
    - Conditionals
        - if ($("#sidebar").hasClass("open")) { do something } else { do something else }
        - if ($(this).is(button[type="submit"])) { disable button during ajax }
        - Caveat: many JS conditionals can be avoided by judicious combinations of selectors.

            -
              ```
              if ($(this).hasClass("active")) {
                  $(this).removeClass("active"); }
              else {
                  $(this)addClass("active");
              } could be rewritten as:
              ```
            - $(this).toggleClass("active");

            -
              ```
              $(".newsItems > li").each(function(){
                  var me = $(this);
                  if (me.has("ul.bullets")){ me.addClass("bulletList"); }
                  else if (me.has("button.launchSlides")) { me.addClass("slideShow") }
                  else if (me.has("textarea") { me.addClass("contactForm")} )
              }); could be rewritten as:
              ```

      - $(".newsItems > li").has("ul.bullets").addClass("bulletList").end().has("button.launchSlides").addClass("slideShow").end().has("textarea").addClass("contactForm");
  - Binding functions to listeners

    -
      ```
      $("#tabs > li").one("click", function() {
          $(this).addClass("selected").siblings().removeClass("selected");
      });
      ```
    - Caveat: Overuse of "this" can cause complicated scope problems down the road. Use with care!
    - Caveat: too many simultaneous listeners will slow down page responsiveness, so make sure you remove listeners when they're not relevant.
  - Returning functions as arguments

    -
      ```
      $("#boxes > div").removeClass("open closed hidden").addClass(function() {
          return $(this).attr('data-defaultstate');
      });
      ```
    - Caveat: If you find yourself using functions as arguments often, you'll probably be better off solving the problem with a custom function, rather than by passing a bunch of mini-functions to jQuery.
- Hour 5: Dynamic and asynchronous functions
  - Preset time loops
    - A single delayed event is set up using setTimeout();
      - Using a named function: setTimeout(hideNav, 3000);
        - This only works when the function is not getting passed any arguments.
        - Use functionName, not functionName(). You're binding a function for future use, not invoking it.
      - Using an anonymous function:
        ```
        setTimeout(function () {
                $("#main .navbar").hide();
            }, 3000);
        ```
      - You can cancel a setTimeout if you've previously saved it to a variable.

        -
          ```
          var barHider =  setTimeout(hideNav, 3000);
          clearTimeout(barHider);
          ```
    - Similarly, a recurring looping delay is set up using setInterval() and canceled using clearInterval();
      - setInterval(updateLog, 3000);

      -
        ```
        setInterval(function () {
            var counterBox = $("#main .counterBox");
            var currentValue = parseInt(counterBox.text(),10);
            currentValue = currentValue + 5;
            counterBox.text(currentValue.toString());
        }, 3000);
        ```

      -
        ```
        var myCounter =  setTimeout(countMe, 3000);
        clearInterval(myCounter);
        ```
  - Server-triggered responses
    - A server response is not an event, but you can still attach a callback function to it.
    - Remember to handle both success and failure responses!

      -
        ```
        $.ajax({
                url: 'search.twitter.com/search.json?q=blue%20angels&rpp=5&include_entities=true&result_type=mixed',
                dataType:'jsonp',
                success:function(response){
                    localStorage.twitterSearch = JSON.stringify(response);
                    console.log('I think that worked.');
                },
                error: function(){
                    alert('Error, error: does not compute.');
                }
            })
        ```
  - Framework-triggered responses
    - These are typically callbacks that run once an animation has completed.
      - .fadeIn(1000, function(){ alert("I'm here!"); })
      - .slideOut(500, function() { $(this).find("li").on("click", selectThisSection; })
  - User-triggered responses
    - Most user inputs trigger JavaScript events, which you can tell your code to listen for.
    - In jQuery, listeners are added via:
      $("someSelector").on("click", functionName);
    - Remember to remove event listeners as soon as they're no longer needed:
      $("someSelector").off("click", functionName);

Learning jQuery

- jQuery also offers a self-removing one-time event listener: $("someSelector").one("click", functionName);
- Some typical user interaction event listeners
  - click
  - focus/blur
  - mouseover
  - keydown