Client Server and Web development

Chapter 4 : Introduction to Jquery and Ajax

I. Introduction

JavaScript, jQuery, where to begin? This is a classic chicken and egg problem when learning programming. You need to know a little JavaScript in order to use jQuery, but jQuery can save you a ton of energy doing things that'd take forever in ordinary JavaScript. In this course, we'll consider the two concepts in parallel from a beginner level.

JavaScript is a **programming language** invented by Brendan Eich in 1995, and **jQuery** is an open-source **JavaScript library** invented by John Resig in 2006. Writing pure JavaScript (the programming language) in your web projects can change how your pages behave. It complements the HTML and CSS of your site, which determine your content and your page appearance, by adding a layer of interactivity to the page.

jQuery (the open-source JavaScript library) allows you to harness the power of JavaScript to accomplish a myriad of awesome things on your webpage. With jQuery, you might:

- Add, delete, or modify HTML elements within your page.
- Change the styles of elements on the page by modifying their associated CSS.
- Animate elements on your page.
- Send and receive data from a server via AJAX (asynchronous JavaScript and XML) so your page doesn't have to be reloaded after submitting a form.
- And more!

At the same time, using jQuery will also allow you to enjoy increased cross-browser functionality. Pure JavaScript is known for being finicky across different browsers like Internet Explorer, Chrome, Safari, etc, that could all potentially manifest your JavaScript code differently. jQuery saves you this headache because it's designed with maximum compatibility in mind.

This compatibility happens via jQuery's use of **CSS selectors**. Selecting elements becomes faster and more direct than selecting elements in classic JavaScript, and since most web developers already understand CSS selectors, this knowledge lets them use jQuery easily.

What do JavaScript and jQuery look like?

Here's a sample chunk of JavaScript that adds a "starred" class to each list item.

```
var listItems = document.querySelectorAll('li');

var i;
```

```
for (i = 0; i < listItems.length; i++) {
   listItems[i].className = 'starred';
}
```

Here's a sample chunk of jQuery that does the same thing:

```
$("li").addClass("starred");
```

You'll notice a lot of **semicolons** and **brackets** in JavaScript, and many **$ (dollar signs)** in jQuery. For all of these wild symbols, you can accomplish the same things with both JavaScript and jQuery. However, you can often do the same things with less effort and code using the latter.

Great, but how do I use either JavaScript or jQuery?

For all of its broad features, **jQuery is simply a JavaScript file**! To use jQuery in your projects, you must include it in the web page, which you will see how to do in part 1.1. Including the jQuery file in your project will let you:

1. Select elements using selectors
2. Do things to the selected elements via methods

These two basic actions allow you to accomplish all the possible use jQuery use cases mentioned above. You'll then organize your JavaScript files the same way you'd organize CSS or other languages in your projects; either as external files or directly within your HTML.

1.1 Including jQuery in your project

Did all that JavaScript seem a little too complicated for not much gain? Good! We'll take all those concepts and make them much easier to use with **jQuery.**

First, you need to **include** jQuery in your web projects. There are two different formats in which jQuery is available: **development** and **production** formats.

The **development** file is larger because it's annotated in a way that makes it easier for developers to understand what's going on in the code. There are code comments, useful whitespace to separate sections, and more. These annotations interspersed among the code make it easier to integrate jQuery smartly, like having tips directly within the code. The file containing the jQuery development format is simply jquery-3.4.1.js (the numbers may be different if you use a different jQuery version).

The **production** format is smaller and therefore faster to use on your live web pages. Most of the annotations you'd find in the development format have been removed, since the end "user" will be a web page and not a human being who would benefit from additional remarks about the code. The file containing the jQuery production format is jquery-3.4.1.min.js . The only difference in the file name is .min meaning the contents is compressed.

In sum:

- Use the development format when you're working on your project so you see the helpful comments and annotations.
- Use the production format once you make your project live on the web so it performs better.

Including the files

You also have two options for including jQuery files (whether development or production) within your code. The first is a classic scenario: you **download** a copy of either the development or production format and include it with your project files. Again, you download the files here: https://jquery.com/download/

Reference it in your HTML as follows:

```
<script src="folder_name/jquery-3.6.4.js"></script>
```

A boilerplate HTML document ends up looking like this:

```
<!DOCTYPE html>
<html>
  <head>
    <title>Page title</title>
    <link rel="stylesheet" href="css/styles.css" />
  </head>
  <body>
    <p>Page content here</p>
    <script src="js/jquery-3.6.4.min.js"></script>
    <div id="divTest1"></div>
    <script type="text/javascript">
    $("#divTest1").text("Hello, world!");
    </script>
  </body>
</html>
```

1.3 Selecting elements

Part of jQuery's joy is how easy it is to select elements from a page. Think of selecting an element as identifying it. You might need to identify an individual elements, multiple elements, elements that meet a set of criteria, etc, in order to modify the element. jQuery lets you identify and select elements using a concept you may already know if you've built web pages before: CSS selectors.

To recap CSS selectors, here are some basic ways to select elements in CSS. Once you select elements, you can then apply style rules to them, setting their font sizes, colors, etc:

Select by element type: h1 , h2 , p , img , etc.
Select by class: .starred , .featured , etc.
Select by ID: #about , #contact , etc.

The good news is that you can select elements in jQuery the same way you select elements in CSS, including selecting by element type, class, and ID. The list is far greater than that though. Let's jump in.

Selecting in jQuery

In CSS, you select elements by simply writing them, like applying styles to h1 elements:

```
h1 {

/* style rules here */

}
```

With jQuery, you must take an extra step. Enter the famous jQuery dollar sign! $() is shorthand for a function called jQuery() that finds elements in a page and creates jQuery objects that reference the found elements.

$('p') : selects all p (paragraph) elements
jQuery('p') : same thing, but takes more time to type. Use the dollar sign!

jQuery cannot be used on elements until those elements are **jQuery objects**, which gives them access to the whole jQuery world. Enrobing them in the $() method turns them into **jQuery objects** on which you can call **jQuery methods**. Before that, they're simply elements in the DOM and won't respond to jQuery methods.

In the following explanations, we'll look at specific ways to select elements, though you shouldn't memorize anything. When you know you'll need to modify elements with jQuery, just reference the jQuery documentation to see how to go about selecting them. It'll be easy to find with this base knowledge of selection.

Select by relationship

Elements that are "related" are often referred to by family vocabulary. You can have **parent, child,** and **sibling elements** within markup, and determining which elements are which family

member determines entirely on your markup structure. It's often useful to select elements based on their relationships to other elements both in CSS and jQuery, so let's explore how to do so (or brush up on your knowledge if you're already familiar).

Descendants

$("ancestor descendent") : Select all elements that are underneath and within an ancestor element

Example: select all li s that are within ol in the markup.

jQuery:

```
$("ol li")
```

HTML:

```
<ol>
   <li>First item</li> <!-- selected -->
   <li>Second item</li> <!-- selected -->
   <ul>
      <li>Unordered item</li> <!-- selected -->
   </ul>
</ol>
```

Parents and children

$("parent > child") : Select all children that are immediate descendants of a parent element (also called a child combinator selector).

Example: select all li s that are direct descendants of ol.

jQuery:

```
$("ol > li")
```

HTML:

```
<ol>
   <li>First item</li> <!-- selected -->
   <li>Second item</li> <!-- selected -->
   <ul>
```

```
    <li>Unordered item</li>
  </ul>
</ol>
```

Siblings

$(element ~ siblings) : Select any siblings of the first specified element.

Example: select the li siblings of the first li .

jQuery:

```
$("li#first ~ li")
```

HTML:

```
<ul>
  <li id="first">One</li>
  <li id="second">Two</li> <!-- selected -->
  <li id="third">Three</li> <!-- selected -->
</ul>
```

$(element + sibling) : Select the sibling element that immediately follow the first specified element.

Example: select the first sibling of a specified li .

jQuery:

```
$("li#first + li")
```

HTML:

```
<ul>
  <li id="first">One</li>
  <li id="second">Two</li> <!-- selected -->
  <li id="third">Three</li>
</ul>
```

Filters

CSS selectors are cool and all, but jQuery even provides extra selectors! Let's look at **filters** now. Sometimes you don't care about the relationships between your elements; you just care about what they **are** or **aren't.**

Examples

:first : selects the first of an element

jQuery

```
$("p:first")
```

HTML

```
<p>First text</p> <!-- selected -->

<p>Second text</p>

<p>Third text</p>
```

:last : selects the last of an element

```
$("p:last")
```

HTML

```
<p>First text</p>

<p>Second text</p>

<p>Third text</p> <!-- selected -->
```

:eq(index) : selects the element at a specified index number (this is where arrays come in handy!)

jQuery

```
$("li:eq(1)")
```

HTML

```
<ul>

   <li>First (so index = 0)</li>

   <li>Second (so index = 1)</li> <!-- selected -->

   <li>Third (so index = 2)</li>

</ul>
```

:gt(index): selects element(s) at a greater index number than the specified number

jQuery

```
$("li:gt(0)")
```

HTML

```
<ul>
```

```
    <li>First (so index = 0)</li>

    <li>Second (so index = 1)</li> <!-- selected -->

    <li>Third (so index = 2)</li> <!-- selected -->

</ul>
```

:lt(index) : same concept but for elements at a lesser index number than the specified number
jQuery

```
$("li:lt(2)")
```

HTML

```
<ul>

    <li>First (so index = 0)</li> <!-- selected -->

    <li>Second (so index = 1)</li> <!-- selected -->

    <li>Third (so index = 2)</li>

</ul>
```

:not(selector) : selects elements that aren't, well, the selector!

jQuery

```
$("li:not('.vegetable')")
```

HTML

```
<ul id="groceries">

    <li class="vegetable">Eggplant</li>

    <li class="vegetable">Carrot</li>

    <li class="fruit">Apple</li> <!-- selected -->

</ul>
```

Much like quoting in English, when using multiple sets of quotation marks in one statement, your interior quotation marks must be different than the exterior ones (see the single vs. double quotes in $("li:not('.vegetable')") , similar to "He said, 'Hello.'").

You can learn more about basic filters in the jQuery documentation.

Additional filters

As you begin to get a sense of how to use filters (colon, filter name, parameters if necessary), know that the list of possibilities is quite long! There are visibility filters, content filters, attribute filters, and even form filters.

:hidden: selects hidden elements (elements that have a CSS display value of none, are type="hidden" , have width and height values of 0, or have hidden ancestor elements)
:visible : selects visible elements
:contains("text") : elements that contain the specified text
:has("element") : elements that contain the specified element
[attribute] : elements that have the specified attribute, ex. $("[align]")
[attribute="value"] : elements that have the specified attribute and value, ex. $("[align=center]")
[attribute!="value"] : elements that do not have the specified attribute and value, ex. $("[align!=center]")

You even have a set of selectors for form elements, which is very useful if you use jQuery to validate forms before users submit them (common). These filters are shorthand for $("[type=XXXXXXX]") . For example, $(":checkbox") is the same thing as $("[type=checkbox]") , but the filter shorthand saves you extra typing!

:input : selects input elements
:password : selects password inputs
:text : selects text inputs
:checkbox : selects checkboxes
:radio : selects radio buttons
:submit : selects submit buttons

Remember, **don't worry about memorizing selectors**. When you know you'll need to use jQuery on certain elements, reference the jQuery documentation to find the best way to select them, now that you understand generally how selecting elements occurs!

1.4 Understand jQuery methods

Once you've selected your elements in jQuery, you want to bring them to life, right?! You'll do so using jQuery methods. Let's jump into methods with a theoretical approach. If you've only worked with HTML and CSS before this course, methods are a new concept. However, they resemble JavaScript functions seen earlier in this course.

You "**call**" a method on a set of objects, thereby applying the method's actions. Here's your standard jQuery statement format:

1. Element selection turned into a jQuery object with $()
2. Period
3. Method name
4. Parentheses, either empty or containing parameters

You've got #1 and #2 down by now, and #3 will be a predefined method from jQuery itself. I've consistently thought that parameters, which you pass to the method, are one of the hardest concepts to teach within programming though. Parameters are **information** you pass to a method so it can do a specific thing. In real life, imagine I'm hiring someone to paint my office. I give them an action to do -- paint -- but if I don't give more information, I won't get the outcome I want. Information I'd pass would include the paint color, the pattern, which walls to paint, etc. This information is the equivalent of a parameter in code.

Within parentheses next to the jQuery method name, you can pass parameters if you need any. These parameters could include a color name, a time duration for an effect, text, or more. It depends on the method. Otherwise, leave the parentheses empty.

You'll **apply** methods in the next chapter! Let's focus on methods themselves for now.

Effects

We'll discuss the most fun methods first. With animation and effect-based jQuery, you can take selected elements and make cool things happen to them visually, like making elements:

- appear once a button is clicked.
- appear slowly on a page.
- slide up or down a page.
- perform your own custom animations.

.show() : displays the selected element(s).
.hide() : hides the selected element(s).
.toggle() : displays or hides element(s) depending on the element's current state.
.fadeIn() : fades in element(s).
.fadeOut() : fades out element(s).
.fadeTo() : fade element(s) to a certain opacity.
.slideUp() : hide element(s) with a sliding up motion.
.slideDown() : hide element(s) with a sliding down motion.
.slideToggle() : hide or show element(s) with a sliding motion depending on the element's current state.

For all of the above elements, you can pass several optional parameters. You'll commonly pass the **effect duration in milliseconds**. For example, .fadeIn(1000) will cause an element to fade in over the course of 1000 milliseconds (1 second). The higher the number, the slower the effect. Alternatively, the strings 'fast' and 'slow' can be passed instead of numbers. 'slow' represents a duration of 600 milliseconds, and 'fast' represents a duration of 200 milliseconds.

Without any parameters, the effects happen over the default duration of 400 milliseconds.

Content manipulation

With content-focused jQuery methods, you can change the page text, elements, and attributes. You'd do this when changing content upon a button click, when a user performs a particular action, if something else happens on the page, etc.

.html() : replace page HTML content
.text() : replace page text
.replaceWith() : replace element(s) entirely, not just its text or HTML
.remove() : remove page elements
.before() : insert content before element(s)
.after() : insert content after element(s)
.prepend() : insert content inside the selected element(s) (after the opening HTML tag)
.append() : insert content inside the selected element(s) (before the closing HTML tag)
.attr() : set an attribute and its value or simply get its attribute
.removeAttr() : remove an attribute, RIP
.addClass() : add a new class to element(s) (without replacing its current classes)
.removeClass() : remove a class from element(s)
.css() : personal fave. Get or set an element's CSS properties, even multiple properties at a time.

DOM traversing

Sometimes you'll need to modify elements relative to other elements within the DOM. By identifying elements in this way, you can modify elements that are **only** within other elements, elements that descend from other elements, etc. You have a few methods available to do this:

.find() : find element(s) **within** current selection that match parameter
.parent() : access direct parent of element(s) or parents if .parents()
.children() : access children of element(s)

Size and positioning

Dimension and position-focused jQuery methods let you adjust the sizing and layout of elements.

.height() : box height without margin, padding, or border
.width () : box width without margin, padding or border

If you want to go truly wild with box dimensions and element sizing, there are jQuery methods that consider border, padding, and margin together or separately.

.innerHeight() : height including padding
.innerWidth() : width including padding
.outerHeight() : height including padding and border
.outerWidth() : width including padding and border
.outerHeight(true) : the same method as above, but passing the parameter true includes the

margin too.
.outerWidth(true) : the same method as above, but passing the parameter true includes the margin too.

Element positioning can be handled with two methods:

.offset() : set element coordinates relative to the very top-left of the document object
.position() : set element coordinates relative to its offset parent, useful for positioning elements within the same DOM element. You'll probably use .offset more.

1.5 Event-based actions

Making things happen on a page when user's interact with it is one of the most satisfying things about jQuery. User clicks on a thing? Boom! Page changes color. User moves their mouse? Boom! Everything fades out except one quote. The combinations are endless. Let's see how to use events in jQuery to do awesome things with your selected elements and methods.

The jQuery .on() method is your key to working with events. You pass the event in question (for example 'click') as a parameter to the .on method, followed by a second parameter containing what's called a "handler" function. Within the function, you write your usual jQuery code that will be executed upon the event's happening.

Using .on() creates what's called an **event listener**, which means the code is waiting for the certain event to happen. JavaScript itself also features event listeners, which make JavaScript and jQuery the tools of choice for building user-interactive pages. jQuery just makes it way easier than pure JS!

Here's a list of common events you might pass to the .on method, their functionalities easily discerned by their names:

```
.on('click', function() { … }

.on('scroll', function() { … }

.on('hover', function() { … }

.on('mouseover', function() { … }

.on('mouseenter', function() { … }

.on('mouse leave', function() { … }

.on('keydown', function() { … }

.on('keyup', function() { … }

.on('keypress', function() { … }

.on('focus', function() { … }
```

```
.on('blur', function() { … }

.on('resize', function() { … }
```

Previously, jQuery had specific methods for each event instead of having you pass event names as parameters. You may see these methods in certain codebases; technically, you can still use them, but using .on( ) plus parameters is a better choice:

```
.click()

.scroll()

.hover()

.mouseover()

.mouseout()

.mouseenter()

.mouseleave()

.keydown()

.keyup()

.keypress()

.focus()

.blur()

.resize()
```

Let’s compare the two in practice. For example, to provoke an alert when someone clicks on a paragraph element, you’d write:

```
$(‘p’).on(‘click’, function() {

   alert( “Someone clicked on a paragraph!” )

});
```

In shorthand, this would be:

```
$(“p”).click(function() {

   alert( “Someone clicked on a paragraph!” )

});
```

Using .on() reads much better from left to right and is the newer practice. This is the route you should take!

Event object

Sometimes you'll need to have information about the event itself in order to accomplish what you need. For example, maybe you want the time to appear next to an element when the event occurs.

The event object has several property types. Of course, you have the type property, which describe which type of event occurred. You also have the event's target property (which element kicked off the event, such as 'click'), pageX and pageY (mouse positions from the left and top of what the user sees of the page), timeStamp (how many milliseconds it's been since 1/1/1970, from which you can calculate exact dates and times), and more.

To interact with the event object in your function, you pass the event as a parameter therein. Let's put this into practice. Write a paragraph in HTML that, when users click on it, turns into text that describes when they clicked.

1. Declare a variable that references all paragraphs
2. Pass the event as a parameter to the function
3. Create a date variable that grabs the event's timestamp
4. Replace the paragraphs with the text, "You clicked on [date]"

```
var $p = $('p');

$p.on('click', function(event) {

  var date = new Date(event.timeStamp);

    $p.text("You clicked on: " + date)

});
```

A simple application, yes, but passing the event object can be very useful. If you have a bunch of items on a page, and you want users to move certain items to a different section (ex. a favorites list) by clicking on them, you'll need to know the exact items they clicked. You could get this with the target property, which you'll see in the next chapter's example.

1.6 Putting it all together

Take a moment to process the following HTML and JavaScript (and to some extent, CSS). Think about how it might work together. There's a link to preview the code at the end of the page.

Here's our HTML:

```
<html>
  <head>
    <title>Course example</title>
    <link rel="stylesheet" href="css/styles.css" />
  </head>
  <body>
    <div id="list_section">
      <h1 id="header">To-do list</h1>
      <ul>
        <li class="item">Buy movie tickets</li>
      </ul>
      <div id="newItemButton"><button href="#">new item</button></div>
      <form id="newItemForm">
        <input type="text" id="itemField" placeholder="Item" />
        <input type="submit" id="add" value="add" />
      </form>
    </div>
    <script src="js/jquery-3.4.1.js"   integrity="sha256-
1XMpEtA4eKXNNpXcJ1pmMPs8JV+nwLdEqwiJeCQEkyc="   crossorigin="anonymous"></script>
    <script src="js/your_js_file.js"></script>
  </body>
</html>
```

Here's our JS:

```
$(function() {
  var $list, $newItemForm;
  $list = $('ul');
```

```
  $newItemForm = $('#newItemForm');
  $newItemForm.on('submit', function(e) {
    e.preventDefault();
    var text = $('input:text').val();
    $list.append('<li>' + text + '</li>');
    $('input:text').val('');
  });
  $list.on('click', 'li', function() {
    var $this = $(this);
    $this.remove();
  });
});
```

Here's our CSS:

```
body {
  background-color: #85cff7;
  font-family: 'Helvetica', 'Arial', sans-serif;
}
#list_section {
  text-align: center;
}
h1 {
  color: white;
  margin: 0px auto;
  padding: 20px;}
ul {
  padding: 0;
```

```
}
li {
  background-color: #fff;
  color: #000;
  font-size: 20px;
  list-style-type: none;
  width: 20%;
  margin: 0px auto;
  border-radius: 3px;
  border: 1px solid #000;
  padding: 10px;
  margin-bottom: 10px;
}
p {
  background-color: #fff;
  color: #666;
  padding: 10px;
  display: inline-block;
  margin: 20px auto 20px auto;
  width: 80%;
  border-radius: 5px;
  text-align: center;}
.item {
  background-color: #fff;
  color: #000;
}
```

```
input[type='text'] {
  font-size: 12px;
  padding: 6px;
  border: 1px solid #000;
  border-radius: 3px;
}
input[type='submit'] {
  background-color: #fff;
  color: #000;
  border-radius: 8px;
  border: none;
  padding: 5px;
}
#newItemButton {
  display: none;
}
#itemField {
  margin-top: 60px;
  width: 10%;
}
```

II. Make JavaScript easier with jQuery

This part will go over animations in jQuery, probably one of the most satisfying ways of using this JavaScript library. Let's go!

Hiding and showing

The hide and show methods do exactly what you'd expect! They hide and show elements by working with the height, width and opacity of the objects concerned. You can pass a numerical value to these methods to indicate how long you want the animation to take (in milliseconds) or use words like fast or slow to indicate the same thing.

```
<button id="show">Make even-numbered lines appear</button>

<button id="hide">Make even-numbered lines disappear</button><br />

<table border>

  <tr><td>a</td><td>b</td><td>c</td></tr>

  <tr><td>d</td><td>e</td><td>f</td></tr>

  <tr><td>g</td><td>h</td><td>i</td></tr>

  <tr><td>j</td><td>k</td><td>l</td></tr>

  <tr><td>m</td><td>n</td><td>o</td></tr>

</table>

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

<script>

  $(function() {

    $('tr:even').css('background','yellow');

    $('td').css('width','200px');

    $('td').css('text-align','center');

    $('#show').on('click', function() {

      $('tr:even').show('slow');

    } );

    $('#hide').click(function() {

      $('tr:even').hide(1000);

    });

  });
```

```
</script>
```

In the above HTML, you find a table made up of five rows and three columns. The formatting of the table is performed by jQuery. A yellow background color is assigned to the even rows of the table, then the width of all the table cells is set to 200 pixels. Finally, the cell contents are centered!

The accompanying jQuery controls that when the user clicks on the second button, the even-numbered lines disappear and reappear slowly if the first button is clicked.

You can pass words like fast or slow to set the animation, or you can use a time in milliseconds. To make an element appear over the course of 2 seconds, you would write: $('element').show(2000)

You can use time parameters with show() , hide() , fadeIn() , fadeOut() , fadeTo() , slideDown() , slideUp() , slideToggle() and animate() , which we'll see later.

Chained effects

You can use the next method to chain a bunch of effects together.

Let's try this out by controlling the appearance of some images.

You'll be reusing this example in the activity at the end of the course. Try to break it down as best as you can on your own right now!

```
<button id="show">Make the images appear</button>

<button id="hide">Hide the images</button><br />

<img src="https://pixabay.com/static/uploads/photo/2014/03/05/19/23/dog-280332_960_720.jpg">

<img src="https://pixabay.com/static/uploads/photo/2016/02/18/18/37/puppy-1207816__340.jpg">

<img src="https://pixabay.com/static/uploads/photo/2015/02/02/17/06/chihuahua-621112__340.jpg">

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

<script>

  $(function() {

    $('#show').on('click', function() {

      $('img').first().show('slow', function showNextOne() {

        $(this).next('img').show('slow', showNextOne);

      });

    });
```

```
    $('#hide').on('click', function() {
      $('img').first().hide('slow', function hideNextOne() {
        $(this).next('img').hide('slow', hideNextOne);
      });
    });
  });
</script>
```

Fading

Fade methods act similarly to hiding and showing, though they show and hide things by manipulating the element's opacity. In their most basic form, they don't need any options passed to them.

```
$('selection').fadeIn();

$('selection').fadeOut();
```

You can also pass time values to them though.

```
$('selection').fadeIn('fast');

$('selection').fadeOut('slow');
```

Check out this example!

```
<img src="https://pixabay.com/static/uploads/photo/2014/03/05/19/23/dog-280332_960_720.jpg" id="puppy">

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

<script>
  $('#puppy').hide().fadeIn(2000, function() {
    alert('The puppy is here!')
  });
</script>
```

Go further

Set a delay before animation

Sometimes you don't want an animation to execute instantly, leaving a few seconds before kicking it off. In that case, you'll want to set a **delay!**

```
<style>

  #message { display: none; background-color: yellow; }

</style>



<span id="message">This text will be shown for 2 seconds</span><br /><br />

<button id="showMessage">Show the message</button>



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

<script>

 $(function() {

   $('#showMessage').click(function() {

     $('#message').fadeIn('slow').delay(2000).fadeOut('slow');

   });

 });

</script>
```

2.2 timers

Set up a timer

Once you've worked with animations, you'll probably want to be able to set timers on the animations you create. The setInterval function comes from JavaScript but is easily integrable in jQuery. You'll use functions like setInterval and setTimeout if you're programming games or other more dynamic animations on the web!

setInterval() calls a function at specified intervals that you'll pass a parameter in milliseconds. The function will keep being called at these regular intervals until its opposite function -- clearInterval() -- is called in the code.

Here's the general syntax of setInterval .

```
function name() {

  // Add JavaScript or jQuery here

}

setInterval(name, time);
```

- name is the name of the function to be performed periodically;
- time is the amount of time between two consecutive executions of the code.

Basic clock

In this first example, we will create a basic digital clock using the JavaScript setInterval() function. The time will be updated every second and displayed in an HTML span using jQuery instructions. Check it out:

```
<span id="time"></span>

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

<script>

$(function() {

  function Clock() {

    var theDate = new Date();

    var t = theDate.getHours() + ":" + theDate.getMinutes() + ":" + theDate.getSeconds();

    $('#time').text(t);

  }

  setInterval(Clock, 1000);

});

</script>
```

When the DOM is ready, the Clock() function is defined. After creating a date variable, the date's hours, minutes and seconds are retrieved via the getHours() , getMinutes() and getSeconds() functions and stored in the variable t :

```
function Clock() {

var theDate = new Date();

var t = theDate.getHours() + ":" + theDate.getMinutes() + ":" + theDate.getSeconds();
```

The content of the #time span is then updated by posting the value stored in the variable t :

```
$('#time').text(t);
```

The Clock() function executes every second using the setInterval() function. 1000 milliseconds -- passed as the second argument -- represent 1 second.

```
setInterval(Clock, 1000);
```

Even if calling Clock looks weird -- no parentheses -- that's the way it works.

Repeated animation

Clocks are all well and good, but let's create something more dynamic. The code below represents a rolling ball animation. The ball is shown and then rolls around slowly in 4 directions (150 px of movement each time)!

```
<style type="text/css">

 #ball {

   width: 10px;

   height: 10px;

   background-color: red;

   border: black 2px solid;

   border-radius : 10px;

   position: relative;

 }

</style>

<body>

 <div id="ball"></div>

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

 <script>

   $(function() {

     function roll() {

       $('#ball').animate({left: '+=150'}, 'slow')

                 .animate({top: '+=150'}, 'slow')
```

```
            .animate({left: '-=150'}, 'slow')
            .animate({top: '-=150'}, 'slow');
    };
    setInterval(roll, 2400);
  });
 </script>
</body>
```

The document's HTML only contains a div with an ID of "ball." Once the DOM is ready, a JavaScript function gets defined (called roll). Within this function, there are four calls that make the ball roll in 4 directions.

At the end, you can call setInterval(), pass the roll function as your first parameter, and the interval time (in milliseconds) as the second parameter.

```
setInterval(roll, 2400);
```

Why 2400 milliseconds?

The slow value in jQuery takes 600 milliseconds. Multiply it by 4, and you have your answer!

III. Understand AJAX

3.1 Introduction

AJAX, which stands for **Asynchronous JavaScript and XML,** is a nifty web technology that lets you load data without entirely refreshing web pages each time a user requests something. Think of the way you use Facebook; you can leave comments, like something, and interact with the page without it reloading with every action. This happens via AJAX!

Let's quickly see a few tools that govern requests and responses over the web.

HTTP

HTTP stands for **Hyptertext Transfer Protocol.** You're using HTTP all the time on the web. Take a look up at your browser's address bar, where you see openclassrooms.com. See how it's preceded by https? This means HTTPS (the secure version of HTTP) is the protocol you're currently using!

Protocols are a defined system for sending information. HTTP works on a client/server model, meaning you have a client on one side making a **request** and a server that sends back a **response.**

There's plenty to say about HTTP, but let's leave it at that for now. For more information about HTTP, check out this writeup from the Mozilla Developer Network.

JSON

JSON stands for **JavaScript Object Notation.** It's a data structure!

Code will often receive data in JSON, process it (also called **parsing)**, and use it somehow, whether it displays the data or does something with it behind-the-scenes.

Here's an example of a candy list written in JSON that we'll use later in the course.

```
[
  {
    "name": "Gummies",
    "brand": "Haribo",
    "quantity": 5
  },
  {
    "name": "Chocolate",
    "brand": "Hershey's",
    "quantity": 3
  },
  {
    "name": "Licorice",
    "brand": "Twizzlers",
    "quantity": 4
  },
  {
    "name": "Truffles",
    "brand": "Godiva",
    "quantity": 0
  }
]
```

Despite its name, JSON is not specific to JavaScript. Many programming languages can interact with data formatted in JSON.

AJAX

Again, AJAX stands for **Asynchronous JavaScript and XML**. In traditional HTTP situations, your browser loads a page that it receives from a server. For example, when you submit a form, your page usually refreshes to show a new page that confirms the information was submitted.

With AJAX, however, you can update a page *without* reloading it entirely.

AJAX and jQuery

If you've studied JavaScript before, you know its syntax is riddled with keywords, punctuation, and more. It's possible to make AJAX requests with pure JavaScript, but in classic jQuery tradition, you can do the same thing with jQuery instead *much more easily.*

Here are the very simple examples we'll be recreating with jQuery. Give them a quick readover (no need to actually do the examples):

1. Birthday surprise: https://openclassrooms.com/courses/use-javascript-on-the-web/make-your-first-ajax-request
2. Candy inventory: https://openclassrooms.com/courses/use-javascript-on-the-web/extend-your-ajax-request

Setup

You have two options for following along, depending on how comfortable you feel setting up a server on your computer.

Easy

Use CodePen. Test the examples by linking to HTML or JSON content in other CodePens. Read more about CodePen and AJAX here: https://blog.codepen.io/2013/09/23/ajax-codepen/

Less easy (but probably necessary someday)

AJAX will not work when you're opening local files on your computer! For security reasons, it only works when the code is running on a web server. This is to make sure you're not accessing data from websites you're not supposed to.

To test out the following examples, you have several options to set up the requisite web server. Since I already have the programming language Ruby on my machine, I can run the command:

ruby -run -e httpd . -p 5000

from Terminal when I'm already within the same directory as your HTML. I navigate to http://localhost:5000 to see your page.

Alternatively, you can get a server up and running quickly with MAMP (Mac) or WAMP (Windows). Installation is very fast, and you'll navigate to http://localhost plus an optional port number, depending where you set it up.

Setting up a web server from A to Z is outside the scope of this course. Feel free to follow along on CodePen if you don't know how to set up a server yet.

Setting up a web server can be discouraging, but don't worry; for the sake of this course, you'll be able to do everything on CodePen if you're not yet comfortable with server stuff. In order to properly advance, you should take the time to learn how to set up a web server though.

3.2 Load page portions via AJAX

If you checked out the JavaScript-based examples in the last chapter, you saw the first AJAX example we'll be tackling with jQuery is a birthday surprise page. It involves clicking a button and, upon clicking, loading the HTML from a **separate HTML page**. AJAX is great for loading external HTML!

Here's the first view the user will see:

Upon clicking the button, a birthday greeting loads. This birthday greeting is housed in a different HTML document (in this case, a separate pen on CodePen, but the idea is the same if you're working with two HTML documents on your computer).

Notice the key word I used while describing what happens: **load**. jQuery has a method called just that! Let's check out the jQuery load method to see how it'll help us build this birthday example.

Loading HTML

AJAX and jQuery can handle all sorts of data from other sources, but one of the most common uses involves loading HTML from a separate HTML file **into** an existing HTML file.

Take the example of a navigation menu; maybe you have the sidebar's HTML in a file like navigation.html, and you want to load that file into your main index.html page if someone clicks a button to see it.

The jQuery load method is very simple.

1. Select the HTML element that'll contain the new HTML.

2. Call .load() .
3. Pass the URL of the new HTML.
4. Pass a specific selection of the new HTML if you only want that content to load (optional).

Here's an example of loading external HTML from a file called navigation.html -- specifically the elements with the class "menu" -- into a div with the ID "content".

```
$('#content').load('navigation.html #menu')
```

load() is the simplest jQuery method to pull in content via AJAX, but it's not the only one! We'll see more soon.

Let's tie this back into our birthday greeting example. Since the example is hosted on CodePen, the separate HTML we'll be loading is in the form of a separate linked CodePen document. **If you were using two HTML files in your web project, it would be the same format -- minus the CodePen URL!**

Here's some basic HTML that contains only a button and an empty div . We'll load our external HTML into the empty div and hide the button once clicked.

```
<body>

  <h1>Today's your special day!</h1>

  <button id="reveal">Why's that?</button>

  <div id="ajax-content">

  </div>

</body>
```

Here's the contents of the **second** HTML file, the one that you'll load via AJAX into the previous HTML. It'll fit in the div with the ID "ajax-content".

```
<h1 id="birthday-greeting">It's your birthday!</h1>
```

Before writing the JavaScript, look at the list of steps again.

1. Select the HTML element that'll contain the new HTML.
2. Call .load() .
3. Pass the URL of the new HTML.
4. Pass a specific selection of the new HTML if you only want that content to load (optional).

We already know the answer to #1. It's the empty div described above.

As for #3, the URL of the HTML, will be a CodePen link in this case (but in a local project file, it'd be the path to your other HTML file. Here's the direct link to the CodePen HTML: http://codepen.io/eclairereese/pen/BzQBzR.html

Luckily, we can avoid #4! We want to load the **whole** external HTML document, not just a particular part.

Together, these criteria fit together as follows:

```
$('#ajax-content').load('http://codepen.io/eclairereese/pen/BzQBzR.html');
```

Add event handler

We need to add additional code to hide the button once it's clicked, so only the birthday greeting is shown.

You can wrap the above AJAX loading in a click event handler! Within the function linked to the click event, you'll load the AJAX **and** hide the button.

Here's the final jQuery code.

```
$('#reveal')

  $('#ajax-content').load('http://codepen.io/eclairereese/pen/BzQBzR.html');

 $('button').hide();

})
```

Test it!

With one simple line of jQuery, you can load in external HTML via AJAX! There are other methods just like load that you'll see in the next part

3.3 Loading other data via AJAX

In the last part you saw jQuery's load method, which can load external HTML into a document via AJAX. There are other methods for retrieving other kinds of data though!

Let's take a look:

- $.get() uses the HTTP GET method to load data that you request.
- $.getJSON() uses the HTTP GET method to load JSON data that you request.
- $.post() uses the HTTP POST method to sends data **to** a server.

The dollar sign, period, and method name may seem like wonky syntax, but that's the correct way it's written!

To all of these methods, you can pass a url, data, callback , and type in the parentheses as parameters.

The URL specifies where you want the data from. Data represents additional information sent to the server. Callback represents the function that should kick off when the data comes back. Type indicates the type of data expected from the server.

In this last course example, we'll look at the $.getJSON() method, because JSON is likely to be the most common data format you receive via AJAX (*maybe* behind HTML)!

Many API responses are in JSON, which stands for JavaScript Object Notation. It's a data structure, so if you ping the Instagram API for photo data, you'll receive a bunch of JSON describing the properties of the photo you requested (its ID, where it was taken, etc).

Here's an example of JSON, listing out some candies, brands, and quantity available.

```
[
  {
    "name": "Gummies",
    "brand": "Haribo",
    "quantity": 5
  },
  {
    "name": "Chocolate",
    "brand": "Hershey's",
    "quantity": 3
  },
  {
    "name": "Licorice",
    "brand": "Twizzlers",
    "quantity": 4
  },
  {
    "name": "Truffles",
```

```
    "brand": "Godiva",
    "quantity": 0
  }
]
```

JSON is very readable and allows objects to have properties associated with them that you define with strings. Ultimately, this JSON file is one giant string!

JSON with jQuery and AJAX

Let's say this JSON is stored somewhere else (an external website, somewhere else in your data files, etc), and you want to load it via AJAX. You can use jQuery to do this!

Here's what we want to accomplish:

1. Transform the JSON (basically a giant string) into a jQuery object
2. Create a list to hold the candies
3. Create a list item for each candy.
3a. If the candy has a quantity greater than 0, mark it as available.
3b. If the candy has a quantity less than 0, mark it as sold out.
4. Add a non-functional "Buy" button just for appearances.
5. Insert that new HTML into our page.

1. Transform JSON into a jQuery object

I've set up a pen with the above JSON in the JavaScript portion of the pen, so we can pretend *that's* the JSON we're trying to access in another HTML document.

Here's the above JSON on CodePen: http://codepen.io/eclairereese/pen/OXRjWV?editors=0010

As mentioned above, the $.getJSON() method allows you to retrieve JSON via an HTTP GET request. You'll pass the URL of the resource you desire. For us, it's the JSON CodePen! My call therefore looks like this:

```
$.getJSON('http://codepen.io/eclairereese/pen/OXRjWV.js')
```

You may be wondering why I didn't pass a function as a parameter. You could do that directly! However, jQuery objects returned by an AJAX call also have access to three methods: done(), fail() , and always(). Each takes a function argument that is called when the request terminates. For more control over what happens with your response, you can chain on one of these methods and pass your function there instead.

fHere's the added call to done() (I chose done() because it's used for successful responses):

```
$(document).ready(function () {
  $.getJSON('http://codepen.io/eclairereese/pen/OXRjWV.js')
```

```
  .done( function(response) {

    // function details

  });

});
```

2. Create a list to hold the candies

This is as simple as writing HTML! Create a <ul> with a class of candies and assign it to a variable called candyList , making it easy to reference in your jQuery.

```
var candies = JSON.parse(req.responseText);

var candyList = '<ul class="candies">';
```

3. Create a list item for each candy and add class depending on item's quantity

We'll use the $.each method to grab each item in the list and create a new candy entry for it. $.each will run once for each item in the candies array starting with an index of 0 (the first candy). For each candy, an <li> will be created with a class of item . If the item has some stock left (quantity > 0), it'll also have the available class. If not, it has a sold-out class. This lets you set different visual styles for sold out items.

Each candy's name and brand will be printed within its new li . Each li is added to the ul via candyList += .

```
$.each(response, function(index, candy) {

  if (candy.quantity > 0) {

    candyList += '<li class="item available">';

  } else {

    candyList += '<li class="item sold-out">'

  }

candyList += candy.name + '<br>' + "Brand: " + candy.brand;

// ...
```

4. Add a non-functional Buy button just for appearances.

More easy HTML! Since you're still "within" the li tag, you'll add an HTML button for each candy to our growing candyList variable (at this point, a giant chunk of HTML).

```
candyList += '<button>Buy</button>'
```

Now that you've written everything you want shown for each candy, you can close up that the $.each method as well as the li tag.

```
    candyList += '</li>';

}
```

Once you're outside of $.each you can also close the ul tag. This means the loop will have run as many times as necessary, and all the lis were created.

```
candyList += '</ul>';
```

5. Insert that new HTML into your page

Now you have a big chunk of HTML you generated line by line with jQuery. To insert it into your actual page HTML, you'll select the candyListing element from the original HTML. It's a special div we created at the bottom of index.html (originally empty but soon to be manually filled with our new HTML) via the html() method.

```
$('#candyListing').html(candyList);
```

Candy totality

Phew! All those steps got us here:

```
$(document).ready(function () {

  $.getJSON('http://codepen.io/eclairereese/pen/OXRjWV.js')

    .done( function(response) {

    var candyList = '<ul class="candies">';

    $.each(response, function(index, candy) {

      if (candy.quantity > 0) {

        candyList += '<li class="item available">';

      } else {

        candyList += '<li class="item sold-out">'

      }

      candyList += candy.name + '<br>' + "Brand: " + candy.brand;

      candyList += '<button class="btn btn-default">Buy</button>';

      candyList += '</li>';
```

```
    });

    candyLis

    $('#candyListing').html(candyList);

  })

})
```

Here's our result!

Test it!

Congrats, you've made it through the course! Hopefully you feel ready to take jQuery into your web projects and beyond.