Lesson 6

jQuery

In this Lesson

6

JAVASCRIPT

SHARE

In part of being a web designer or front end developer you will commonly run into JavaScript, often referred to as JS, and jQuery. Within the top 10,000 websites JavaScript is used within over 92% of them, and jQuery is used within in over 63% of them. Needless to say, they are fairly popular. You may even aspire to write JavaScript or jQuery to build your own behaviors at one point or another.

If you are asking what exactly are JavaScript and jQuery fear not, this lesson gives a brief overview of JavaScript and then takes a look at jQuery.

JavaScript Intro

JavaScript provides the ability to add interactivity to a website, and help enrich the user experience. HTML provides a page with **structure** and CSS provides a page with **appearance**, JavaScript provide a page with **behavior**.

Like CSS, JavaScript should be saved in an external file with the `.js` file extension, and then referenced within an HTML document using the `script` element. Where the JavaScript reference is placed with HTML depends on when it should be executed. Generally speaking, the best place to reference JavaScript files is right before the closing `</body>` tag so that the JavaScript file is loaded after all of the HTML has been parsed. However, at times, JavaScript is needed to help render HTML and determine it's behavior, thus may be referenced within a documents `head`.

```
1    <script src="script.js"></script>
```

Values & Variables

Part of the fundamentals of JavaScript include values and variables. Values, in general, are the different types of values that JavaScript will recognize, while variables are used

to store and share these values.

Values may include strings of text, true or false Booleans, numbers, `undefined`, `null`, or other values such as functions or objects.

One popular way variables are defined is with the `var` keyword, followed by the variable name, an equal sign (`=`), then the value, ending with a semicolon (`;`). The variable name must begin with a letter, underscore (`_`), or dollar sign (`$`). Variables cannot begin with numbers, although they may be used subsequently, and they cannot use hyphens whatsoever. Additionally, JavaScript is case sensitive so letters include `a` through `z` in both lower and uppercase.

The common convention around naming variables is to use camelCase, without the use of any dashes or underscores. camelCase consist of combining words while removing spaces, capitalizing the beginning of each new word except for the initial word. For example, `shay_is_awesome` would more commonly named `shayIsAwesome`.

```
var theStarterLeague = 125;
var food_truck = 'Coffee';
var mixtape01 = true;
var vinyl = ['Miles Davis', 'Frank Sinatra', 'Ray Charles'];
```

Statements

As a whole, JavaScript is a set of statements, of which are executed by the browser in the sequence they are written. These statements provide commands which determine the different behaviors to be taken. Statements come in all different shapes and sizes, with multiple statements separated with semicolons, `;`. New statements should begin on a new line, and indentation should be used when nesting statements for better legibility, but is not required.

```
log(polaroid);
return('bicycle lane');
alert('Congratulations, you ' + outcome);
```

Functions

Adding to the fundamentals of JavaScript, it is important to take a look at functions. Functions provide a way to perform a set of scripted behaviors now, or saved for later, and depending on the function they may even accept different arguments.

A function is defined by using the `function` keyword followed by the function name, a list of commas separated arguments wrapped in parentheses, if necessary, and then the JavaScript statement, or statements, that defines the function enclosed in curly braces, `{}`.

```
function sayHello(name) {
  return('Hello ' + name);
}
```

Arrays

As you may have recognized, some values may be returned as an array. Arrays include a way to store a list of items, or values. Arrays are helpful for many reasons, one being the ability to be traversed with different methods and operators. Additionally, depending on the situation, arrays can be used to store, and return, a variety of different values.

Generally speaking arrays are identified within square brackets, `[]`, with comma separated items. The items start at `0` and increase from there. When identifying the third item in a list it is actually identified as `[2]`.

Objects

JavaScript is also built on the foundation of objects, which are a collection of key and value pairs. For example, there may be an object named `school` which includes the keys, also known as properties, `name`, `location`, `students`, and `teachers`, and their values.

In the example below the variable `school` is set up as an object to hold multiple properties. Each property has a key and value. The entire object is wrapped inside of curly braces, `{}`, with comma separated properties, each having a key followed by a colon and value.

```
// Object
var school = {
  name: 'The Starter League',
  location: 'Merchandise Mart',
  students: 120,
  teachers: ['Jeff', 'Raghu', 'Carolyn', 'Shay']
};

// Array
var school = ['Austin', 'Chicago', 'Portland'];
```

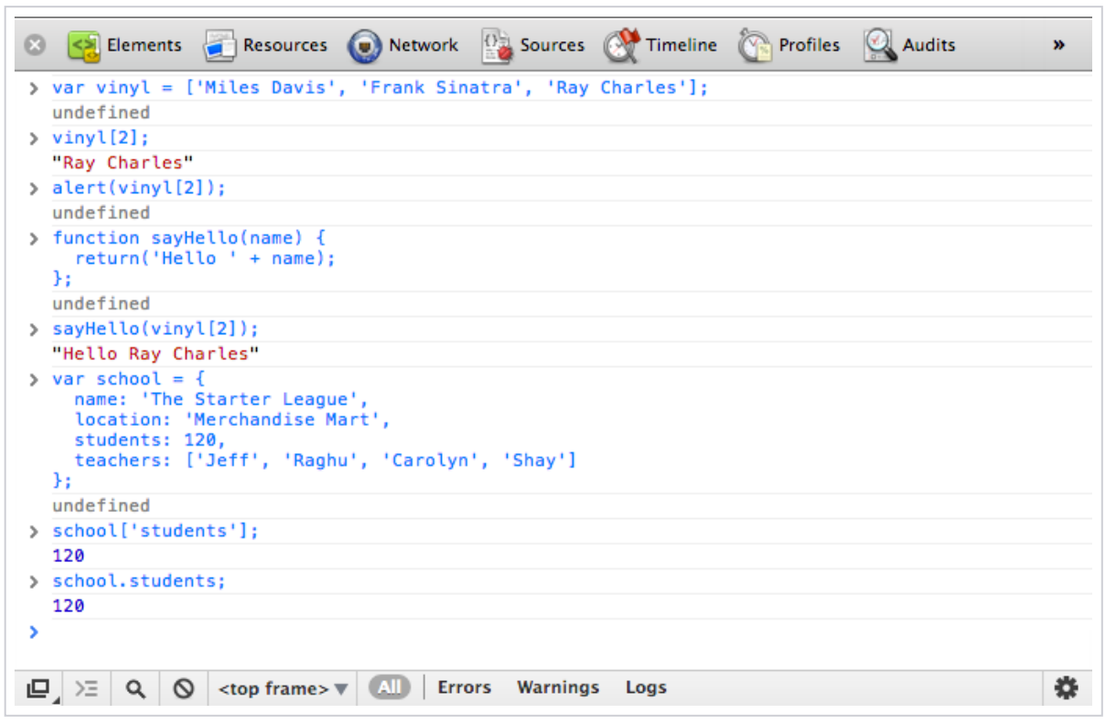

Fig. 6.01

Using the developer tools built into the Chrome web browser, JavaScript may be run from within the console.

jQuery Intro

With a basic understanding of JavaScript and some of it's foundations, it is time to take a look at jQuery. jQuery is an open source JavaScript library written by John Resig that simplifies the interaction between HTML, CSS, and JavaScript. Since 2006, when jQuery was released, it has taken off, being used by websites and companies large and small.

What has made jQuery so popular is it's ease of use, with selections resembling CSS and a comprehensible separation of behavior. The benefits of jQuery are massive, however for our purpose we will only be considered about the ability to find elements and perform actions with them.

Getting Started with jQuery

The first step to using jQuery is to reference it from within a HTML document. As previously mentioned with JavaScript, this is done using the `script` element just before the closing `</body>` tag. Since jQuery is it's own library it is best to keep it separate from all the other JavaScript being written.

When referencing jQuery there are a few options, specifically as whether to use the minified or uncompressed version, and as whether to use a content delivery network, CDN, such as Google hosted libraries. If the code being written is for a live, production environment it is encouraged to use the minified version for better loading times. Additionally, using a CDN like Google also helps with loading time, and potential caching benefits.

```
1    <script src="//ajax.googleapis.com/ajax/libs/jquery/1.9.0/jquery.min.js"></s
2    <script src="script.js"></script>
```

In the code sample above, notice the second `script` element referencing a second JavaScript file. All of the custom, handwritten JavaScript and jQuery should be written in this file. Additionally, this file is specifically placed after the jQuery file so that it may reference jQuery functions already defined.

Where is the leading http?

You may have noticed that there isn't a leading `http` within Google CDN reference example above. The `http` has been omitted intentionally to allow for both `http` and `https` connections. When working locally, without the benefit of a web server, the leading `http` will need to be included to prevent attempting to locate the file on the systems local disk drive.

jQuery Object

jQuery comes with it's own object, the dollar sign, `$`, also known as `jQuery`. The `$` object is specifically made for selecting an element and then returning that element node to perform an action on it. These selections and actions should be written in a new file, referenced outside of the actual jQuery library.

```
$();
jQuery();
```

Document Ready

Before trigging any jQuery to traverse and manipulate a page it is best to wait until the DOM is finished loading. Fortunately jQuery has a ready event, `.ready()`, which can be called when the HTML document is ready to be altered. By placing all of our other custom written jQuery inside of this function we can guarantee that it will not be executed until the page has loaded and the DOM is ready.

```
$(document).ready(function(event){
  // jQuery code
});
```

Selectors

As previously mentioned, one of the core concepts of jQuery is to select elements and perform an action. jQuery has done a great job of making the task of selecting an element, or elements, extremely easy by mimicking that of CSS. On top of the general CSS selectors, jQuery has support for all of the unique CSS3 selectors, which work regardless of which browser is being used.

Invoking the jQuery object, `$()`, containing a selector will return that DOM node to manipulate it. The selector falls within the parentheses, `('...')`, and may select elements just like that of CSS.

```
$('.feature');           // Class selector
$('li strong');          // Descendant selector
$('em, i');              // Multiple selector
$('a[target="_blank"]'); // Attribute selector
$('p:nth-child(2)');     // Pseudo-class selector
```

This Selection Keyword

When working inside of a jQuery function you may want to select the element in which was referenced inside of the original selector. In this event the `this` keyword may be used to refer to the element selected in the current handler.

```
$('div').click(function(event){
  $(this);
});
```

jQuery Selection Filters

Should CSS selectors not be enough there are also custom filters built into jQuery to help out. These filters are an extension to CSS3 and provide more

control over selecting an element or its relatives.

```
$('div:has(strong)');
```

As they stand these filters may be used inside of the selector, however not being native to the DOM they are a bit slow. The best results with using these filters is accomplished by using the `:filter()` method, which is part of the traversing feature in jQuery.

Traversing

At times the general CSS selectors alone don't cut it and a little more detailed control is desired. Fortunately jQuery provides a handful of methods for traversing up and down the DOM tree, filtering and selecting elements as necessary.

To get started with filtering elements inside the DOM a general selection needs to be made, from which will be traversed from relatively. In the example below the original selection finds all of the `div` elements in the DOM, which are then filtered using the `.not()` method. With this specific method all of the `div` elements without a class of `type` or `collection` will be selected.

```
$('div').not('.type, .collection');
```

Chaining Methods

For even more control as to which elements are selected different traversing methods may be chained together simply by using a dot in-between them.

The code sample below uses both the `.not()` method and the `.parent()` method. Combined together this will only select the parent elements of `div` elements without a class of `type` or `collection`.

```
$('div').not('.type, .collection').parent();
```

Traversing Methods

jQuery has quite a few traversing methods available to use. In general, they all fall into three categories, filtering, miscellaneous traversing, and DOM tree traversing. The specific methods within each category may be seen below.

Filtering

- `.eq()`
- `.filter()`
- `.first()`
- `.has()`
- `.is()`
- `.last()`
- `.map()`

- `.not()`
- `.slice()`

Miscellaneous Traversing

- `.add()`
- `.andSelf()`
- `.contents()`
- `.end()`

DOM Tree Traversal

- `.children()`
- `.closest()`
- `.find()`
- `.next()`
- `.nextAll()`
- `.nextUntil()`
- `.offsetParent()`
- `.parent()`
- `.parents()`
- `.parentsUntil()`
- `.prev()`
- `.prevAll()`
- `.prevUntil()`
- `.siblings()`

Manipulation

Selecting and traversing elements in the DOM is only part of what jQuery offers, one other major part is what is possible with those elements once found. One possibility is to manipulate these elements, by either reading, adding, or changing attributes or styles. Additionally, elements may be altered in the DOM, changing their placement, removing them, adding new elements, and so forth. Overall the options to manipulate elements are fairly vast.

Getting & Setting

The manipulation methods to follow are most commonly used in one of two directives, that being *getting* or *setting* information. Getting information revolves around using a selector in addition with a method to determine what piece of information is to be retrieved. Additionally, the same selector and method may also be used to set a piece of information.

```
1  // Gets the value of the alt attribute
2  $('img').attr('alt');
3
4  // Sets the value of the alt attribute
5  $('img').attr('alt', 'Wild kangaroo');
```

In the examples and snippets to follow methods will primarily be used in a setting mode, however they may also be able to be used in a getting mode as well.

Attribute Manipulation

One part of elements able to be inspected and manipulated are attributes. A few options include the ability to add, remove, or change an attribute or its value. In the examples below the `.addClass()` method is used to add a class to all even list items, the `.removeClass()` method is used to remove all classes from any paragraphs, and lastly the `.attr()` method is used to find the value of the `title` attribute of any `abbr` element and set it to `Hello World`.

```
$('li:even').addClass('even-item');
$('p').removeClass();
$('abbr').attr('title', 'Hello World');
```

Attribute Manipulation Methods

- `.addClass()`
- `.attr()`
- `.hasClass()`
- `.prop()`
- `.removeAttr()`
- `.removeClass()`
- `.removeProp()`
- `.toggleClass()`
- `.val()`

Style Manipulation

On top of manipulating attributes, the style of an element may also be manipulated using a variety of methods. When reading or setting the height, width, or position of an element there are a handful of special methods available, and for all other style manipulations the `.css()` method can handle any CSS alterations.

The `.css()` method in particular may be used to set one property, or many, and the syntax for each varies. To set one property, the property name and value should each be in quotations and comma separated. To set multiple properties, the properties should be nested inside of curly brackets with the property name in camel case, removing any hyphens where necessary, followed by a colon and then the quoted value. Each of the property and value pairs need to be comma separated.

The height, width, or position methods all default to using pixel values, however other units of measurement may be used. As seen below, to change the unit of measurement identify the value then use a plus sign followed by the quoted unit of measurement.

```
$('h1 span').css('font-size', 'normal');
$('div').css({
  fontSize: '13px',
  background: '#f60'
});
$('header').height(200);
$('.extend').height(30 + 'em');
```

Style Manipulation Methods

- `.css()`
- `.height()`

- `.innerHeight()`
- `.innerWidth()`
- `.offset()`
- `.outerHeight()`
- `.outerWidth()`
- `.position()`
- `.scrollLeft()`
- `.scrollTop()`
- `.width()`

DOM Manipulation

Lastly, we are able to inspect and manipulate the DOM, changing the placement of elements, adding and removing elements, as well as flat out altering elements. The options here are deep and varied, allowing for any potential changes to be made inside the DOM.

Each individual DOM manipulation method has it's own syntax but a few of them are outlined below. The `.prepend()` method is adding a new `h3` element just inside any `section`, the `.after()` method is adding a new `em` element just after the link, and the `.text()` method is replacing the text of any `h1` elements with the text `Hello World`.

```
$('section').prepend('<h3>Featured</h3>');
$('a[target="_blank"]').after('<em>New window.</em>');
$('h1').text('Hello World');
```

DOM Manipulation Methods

- `.after()`
- `.append()`
- `.appendTo()`
- `.before()`
- `.clone()`
- `.detach()`
- `.empty()`
- `.html()`
- `.insertAfter()`
- `.insertBefore()`
- `.prepend()`
- `.prependTo()`
- `.remove()`
- `.replaceAll()`
- `.replaceWith()`
- `.text()`
- `.unwrap()`
- `.wrap()`
- `.wrapAll()`
- `.wrapInner()`

Events

One of the beauties of jQuery is the ability to easily add event handlers, which are methods that are called only upon a specific event or action taking place. For example, the method of adding a class to an element can be set to only occur upon that element being clicked on.

Below is a standard selector, grabbing all of the list items. The `.click()` event method is bound to the list item selector, setting up an action to take place upon clicking any list item. Inside the `.click()` event method is a function, which ensures any actions inside the event method are to be executed. The parentheses directly after the function are available to pass in parameters for the function, in which the `event` object is used in this example.

Inside of the function is another selector with the `.addClass()` method bound to it. Now, when a list item is clicked on that list item, via the `this` keyword, receives the class of `saved-item`.

```
$('li').click(function(event){
  $(this).addClass('saved-item');
});
```

Event Flexibility

The `.click()` event method, along with a handful of other event methods, is actually a shorthand method which uses the `.on()` method introduced in jQuery 1.7. The `.on()` method provides quite a bit of flexibility, using automatic delegation for elements that get added to the page dynamically.

Making use of the `.on()` method the first argument should be the native event name while the second argument should be the event handler function. Looking at the example from before, the `.on()` method is called in place of the `.click()` method. Now the `click` event name is passed in as the first argument inside the `.on()` method with the event handler function staying the same as before.

```
$('li').on('click', function(event){
  $(this).addClass('saved-item');
});
```

Nesting Events

It is possible to have multiple event handlers and triggers, nesting one inside another. As an example, below the `.on()` event method is passed the `hover` argument, thus being called when hovering over any element with the class of `pagination`. Upon calling the `.on()` event the `.click()` event is called on the anchor with the `up` ID.

```
$('.pagination').on('hover', function(event){
  $('a#up').click();
});
```

Event Demo

Using an alert message as a demo, the following code snippets show how to create an alert message and then removing that message based upon clicking the close icon.

HTML

```
<div class="notice-warning">
  <div class="notice-close">&#215;</div>
  <strong>Warning!</strong> I&#8217;m about to lose my cool
</div>
```

JavaScript

```
$('.notice-close').on('click', function(event){
  $('.notice-warning').remove();
});
```

Demo

Event Methods

jQuery provides quite a few methods, all of which are based around registering user behaviors as they interact with the browser. These methods register quite a few events, most popularly, but not limited to, browser, form, keyboard, and mouse events. The most popular of these methods include:

Browser Events

- `.resize()`
- `.scroll()`

Document Loading

- `.ready()`

Event Handler Attachment

- `.off()`
- `.on()`
- `.one()`
- `jQuery.proxy()`
- `.trigger()`
- `.triggerHandler()`
- `.unbind()`
- `.undelegate()`

Event Object

- `event.currentTarget`
- `event.preventDefault()`

- `event.stopPropagation()`
- `event.target`
- `event.type`

Form Events

- `.blur()`
- `.change()`
- `.focus()`
- `.select()`
- `.submit()`

Keyboard Events

- `.focusin()`
- `.focusout()`
- `.keydown()`
- `.keypress()`
- `.keyup()`

Mouse Events

- `.click()`
- `.dblclick()`
- `.focusin()`
- `.focusout()`
- `.hover()`
- `.mousedown()`
- `.mouseenter()`
- `.mouseleave()`
- `.mousemove()`
- `.mouseout()`
- `.mouseover()`
- `.mouseup()`

Effects

Next to events, jQuery also provides a handful of customizable effects. These effects come by the way of different methods, including event methods for showing and hiding content, fading content in and out, or sliding content up and down. All of these are ready to use methods and may be customized as best see fit.

Each effect method has it's own syntax so it is best to reference the jQuery effects documentation for specific syntax around each method. Most commonly though, effects generally accept a duration, easing, and the ability to specify a callback function.

> ### jQuery CSS Animations
>
> Custom animations of different CSS properties can be accomplished in jQuery, although this is a little less relevant as CSS can now handle animations itself. CSS animations offer better performance from a browser processing standpoint and are preferred where possible. jQuery animation effects, with the help of Modernizr, make for a perfect backup solution to any browser not supporting CSS animations.

Effect Duration

Using the `.show()` method as an example, the first parameter available to optionally pass in to the method is the duration, which can be accomplished using a keyword or milliseconds value. The keyword `slow` defaults to `600` milliseconds, while the keyword `fast` defaults to `200` milliseconds. Using a keyword value is fine, but millisecond values may also be passed in directly. Keyword values must be quoted while millisecond values do not.

```
$('.error').show();
$('.error').show('slow');
$('.error').show(500);
```

Effect Easing

In addition to setting the duration in which an effect takes place the easing, or speed at which an animation progresses at during different times within the animation, may also be set. By default jQuery has two keyword values for easing, the default value is `swing` with the additional value being `linear`. The default `swing` value starts the animation at a slow pace, picking up speed during the animation, and then slows down again before completion. The `linear` value runs the animation at one constant pace for the entire duration.

```
$('.error').show('slow', 'linear');
$('.error').show(500, 'linear');
```

jQuery UI

The two easing values that come with jQuery may be extend with the use of different plug-ins, of which may offer additional values. One of the most popular plug-ins is the jQuery UI suite.

On top of new easing values jQuery UI also provides a handful other interactions, effects, widgets, and other helpful resources worth taking a look at.

Effect Callback

When an animation is completed it is possible to run another function, called a callback function. The callback function should be placed after the duration or easing, if either exist. Inside this function new events or effects may be placed, each following their own required syntax.

```
$('.error').show('slow', 'linear', function(event){
  $('.error .status').text('Continue');
});
```

Effect Syntax

As previously mentioned, each effect method has it's own syntax which can be found in the jQuery effects documentation. The duration, easing, and callback parameters outlined here are common, but not available on every method. It is best to review the syntax of a method should you have any questions around it.

Effects Demo

Taking the same events demo from above, the `.remove()` method is now used as part of a callback function on the `.fadeOut()` method. Using the `.fadeOut()` method allows for the alert message to gradually fade out rather than quickly disappearing, then be removed from the DOM after the animation is complete.

HTML

```
<div class="notice-warning">
  <div class="notice-close">&#215;</div>
  <strong>Warning!</strong> I&#8217;m about to lose my cool
</div>
```

JavaScript

```
$('.notice-close').on('click', function(event){
  $('.notice-warning').fadeOut('slow', function(event){
    $(this).remove();
  });
});
```

Demo

Basic Effects

- `.hide()`
- `.show()`
- `.toggle()`

Custom Effects

- `.animate()`
- `.clearQueue()`
- `.delay()`
- `.dequeue()`
- `jQuery.fx.interval`
- `jQuery.fx.off`
- `.queue()`

- .stop()

Fading Effects

- .fadeIn()
- .fadeOut()
- .fadeTo()
- .fadeToggle()

Sliding Effects

- .slideDown()
- .slideToggle()
- .slideUp()

Slide Demo

HTML

```
<div class="panel">
  <div class="panel-stage"></div>
  <a href="#" class="panel-tab">Open <span>&#9660;</span></
</div>
```

JavaScript

```
$('.panel-tab').on('click', function(event){
  event.preventDefault();
  $('.panel-stage').slideToggle('slow', function(event){
    if($(this).is(':visible')){
      $('.panel-tab').html('Close <span>&#9650;</span>');
    } else {
      $('.panel-tab').html('Open <span>&#9660;</span>');
    }
  });
});
```

Demo

Tabs Demo

HTML

```
<ul class="tabs-nav">
  <li><a href="#tab-1">Features</a></li>
  <li><a href="#tab-2">Details</a></li>
</ul>
<div class="tabs-stage">
  <div id="tab-1">...</div>
  <div id="tab-2">...</div>
</div>
```

JavaScript

```
// Show the first tab by default
$('.tabs-stage div').hide();
$('.tabs-stage div:first').show();
$('.tabs-nav li:first').addClass('tab-active');

// Change tab class and display content
$('.tabs-nav a').on('click', function(event){
  event.preventDefault();
  $('.tabs-nav li').removeClass('tab-active');
  $(this).parent().addClass('tab-active');
  $('.tabs-stage div').hide();
  $($(this).attr('href')).show();
});
```

Demo

Resources & Links

- JavaScript For Cats
- A Re-introduction to JavaScript via Mozilla Developer Network
- 30 Days to Learn jQuery via Tuts+ Premium
- Google Hosted Libraries
- jQuery Documentation
- jQuery Fundamentals via Bocoup
- jQuery UI