[Lesson 7.2: JavaScript and JQuery](#h.wt4e21pfcbfl)

[Gem of the Day](#h.cl4cyyskr7a0)

[JavaScript](#h.432w3v5su6vj)

[JavaScript Language Basics](#h.fd18nkmu7gka)

[Variables](#h.8pzbnxvz3z4j)

[Data Types](#h.cxkb8o6zbqq0)

[Null and Undefined](#h.soybf27solvm)

[Scope](#h.hk1ih4hq4yr9)

[Operators](#h.7xol7crzq1qa)

[Conditionals](#h.rj3l0rv2t7q)

[Loops](#h.pol8e62o21l0)

[Functions](#h.p1o5dcf6h697)

[JavaScript in HTML](#h.ow8uuh10mgf)

[JQuery](#h.k452xcryo9x2)

[My First JQuery](#h.xzomxkbntw0c)

[The DOM, or Document Object Model](#h.uv4uhkdm0w56)

[CSS Selectors](#h.u1rudzeo5g58)

[Attributes, CSS, Events, and Animation/Effects](#h.udorodhpi40p)

[Homework](#h.d1o18z98kto)

[Resources](#h.2gz5vyr64vrv)

# Lesson 7.2: JavaScript and JQuery

## Gem of the Day

RuboCop/RefactorCop <http://refactorcop.com/>

## JavaScript

We're going to cover the basics of JavaScript: how to create variables and functions, write loops and conditionals, and including scripts in your HTML page. We'll also cover jQuery, the most popular JavaScript library on the web today.

JavaScript was developed by Netscape in the early 1990s as a way to provide interactivity in web pages. Before JavaScript, web pages were essentially static documents: once you loaded a page, the content didn't change.

Since then, JavaScript has become standardized and evolved tremendously. You can develop entire applications, from the server side to the client side, using only JavaScript.

## JavaScript Language Basics

Double-check that you have node.js installed - run node -v on your command line and confirm that it returns a version (if no version, use Cloud9 or Nitrous as your IDE for now). Then, go to your front-end folder, and create a new file called practice.js. Inside that file, let’s write a very basic program:

var message = "Hello, World!";

console.log(message);

Then, on the command line, type node practice.js. You should see your message print out to the command line like when we used “puts” in Ruby.

### Variables

Creating a variable in JavaScript is called "declaring" a variable. You declare a JavaScript variable with the var keyword (won’t work until we use “var”!). Just like Ruby, JavaScript is an untyped language, so any variable can hold any value at any time.

var x = "Hey";

var y = 10;

console.log(x);

console.log(y);

console.log(x + y);

y = " is for Horses!";

console.log(x + y);

You can declare many variables in one statement:

var x = 1, y = 2, z = "3";

### Data Types

Like Ruby, JavaScript can handle integers, floats, strings, booleans, arrays, and hashes. Actually, because of the nature of JavaScript, every object in JS is in fact a hash. The hash data type just let’s you do the more typical hash methods.

// Numbers

var integer\_num = 1;

var float\_num = 1.23;

console.log(integer\_num + float\_num);

// Strings

var message = "Let's learn JavaScript!";

console.log(message);

// Booleans

var is\_cool = true;

console.log(is\_cool + " dat");

// Arrays

var my\_stuff = [integer\_num, message, is\_cool];

console.log(my\_stuff);

// Hashes

var capitals = {

LA: "Baton Rouge",

TX: "Austin",

GA: "Atlanta"

};

console.log(capitals["LA"]);

### Null and Undefined

JavaScript has two different concepts of “emptiness” - null and undefined:

var x;

// Variables that haven't been initialized are undefined.

console.log(x); // undefined

// But they are not null:

console.log(x === null); // false

// Unless you make them null:

x = null;

console.log(x); // null

console.log(x === null); // true

console.log(x === undefined); // false

// Or use 'type coercion' - 2 equal signs will change the value rather than

// simply compare in Javascript

console.log(x == undefined); // true

### Scope

In JavaScript, objects and functions are also variables. **Scope** is the set of variables, objects, and functions you have access to. JavaScript has **function scope**: The scope changes inside functions.

**Local JavaScript Variables**

Variables declared within a JavaScript function, become LOCAL to the function. Local variables have local scope: They can only be accessed within the function. Since local variables are only recognized inside their functions, variables with the same name can be used in different functions. Local variables are created when a function starts, and deleted when the function is completed.

**Global JavaScript Variables**

A variable declared outside a function, becomes GLOBAL. A global variable has global scope: All scripts and functions on a web page can access it.

**Automatically Global**

If you assign a value to a variable that has not been declared, it will automatically become a GLOBAL variable. This is why you should always declare variables with var to avoid overwriting a global variable with the same name.

// This variable is in the Global scope:

var x = "I'm a global variable called x!";

console.log(x);

// Defining a function called someFunction

function someFunction(){

// This variable only exists inside the function:

var y = "I'm a local variable called y!";

console.log(x);

console.log(y);

// This is bad. Don't do it.

x = "I'm now a BAD local variable called x."

console.log(x);

}

// Calling someFunction

someFunction();

console.log(x); // Returns BAD local x

console.log(y); // Results in an error

### Operators

JavaScript has arithmetic and logical operations, like Ruby:

var x = 10,

y = 5;

console.log(x + y);

console.log(x - y);

console.log(x \* y);

console.log(x / y);

console.log(x % y); // modulus, or remainder

console.log(x > y);

console.log(x < y);

console.log(x === y); // is equal to

console.log(x !== y); // is not equal to

var a = true,

b = false;

console.log(a && b); // and

console.log(a || b); // or

### Conditionals

JavaScript conditional statements are similar to Ruby conditionals, but you need parentheses and curly brackets, and it's else if, not elsif:

var x = 10;

y = 5;

if (x > y){

console.log("x is greater than y");

} else if (x < y){

console.log("x is less than y");

} else {

console.log("x is equal to y");

}

### Loops

JavaScript looks like C when it comes to loops, though modern browsers also support a cleaner forEach loop:

// C-style JavaScript loop

var x = [1, 2, 3, 4, 5];

for (var i = 0; i < x.length; i++){

console.log(x[i]);

}

// For each loop

x.forEach(function (element){

console.log(element);

});

// While loop

var numbers = [1, 2, 3, 4],

i = 0;

while (i < numbers.length){

console.log(numbers[i]);

i++;

}

### Functions

JavaScript functions are like Ruby methods, but they have a few more tricks.

* They create their own local scope.
* They can be stored in variables and passed to other functions, just like any other data type.
* They create closures, which means they keep their surrounding scope with them as they get passed around.
* They can receive an arbitrary number of parameters, regardless of how the function is declared.

Earlier when we talked about scopes, we saw how to define a function and call it. Let’s also learn how to pass some arguments to a function:

// A function with explicit arguments

function sumExplicitly(a, b){

console.log(a + b);

}

sumExplicitly(10, 5); //=> 15

// A function with implicit arguments

function sumImplicitly(){

var total = 0,

i;

for (i = 0; i < arguments.length; i++){

total += arguments[i];

}

console.log(total);

}

sumImplicitly(1, 4, 7, 3, 20); //=> 35

Functions can return values to be used elsewhere. A very important difference to Ruby is that if you don't have a return statement a function, then its return value is undefined.

function sum(a, b){

return a + b;

}

var x = sum(10, 5);

var y = sum(x, sum(20, 30));

console.log(x); //=> 15

console.log(y); //=> 65

As you can see from this last example, you can pass functions as arguments into other functions.

## JavaScript in HTML

When we use JavaScript in HTML, we include it inside <script></script> tags. You can technically place <script> tags anywhere in the<head> or <body> element, but there are a few caveats:

* Scripts block further rendering of the document while they execute, so it's generally a good idea to load them near the end of the <body>.
* Scripts that depend on other scripts need to be loaded in the correct order — e.g., load the jQuery library before your script that uses jQuery.

What’s cool about JavaScript is that console.log is still available to you when we start using the actual browser - find it in DevTools under the Console tab.

<!DOCTYPE html>

<html>

<head>

<title>Practice JavaScript</title>

</head>

<body>

<script type="text/javascript">

alert("It worked!");

console.log("The console still works too!");

</script>

</body>

</html>

## JQuery

jQuery is a fast and concise JavaScript library created by John Resig in 2006 with a nice motto: Write less, do more. jQuery simplifies HTML document traversing, event handling, animating, and Ajax interactions for rapid web development.

Features include:

* DOM manipulation
* Event handling
* Asynchronous requests (Ajax)
* Animations
* Cross-browser compatibility
* Light weight

Just like Bootstrap, you can either download jQuery and include it locally in your project, or you can use the version hosted by their CDN. We'll use the CDN version, so add this to your page just above the existing <script> tag:

<script type="text/javascript" src="<http://code.jquery.com/jquery-1.11.3.min.js>"></script>

### My First JQuery

Let’s write a bit of code first, then talk through it:

<!DOCTYPE html>

<html>

<head>

<title>JavaScript Practice</title>

<script type="text/javascript" src="http://code.jquery.com/jquery-1.11.3.min.js"></script>

</head>

<body>

<div id="my-element">HI</div>

<script type="text/javascript">

$("#my-element")

.show()

.addClass("selectable")

.click(function (e){

alert("You clicked something!");

})

console.log("hi");

</script>

</body>

</html>

Almost everything you'll do in jQuery is implemented through the $() function. You'll also frequently chain function calls together.

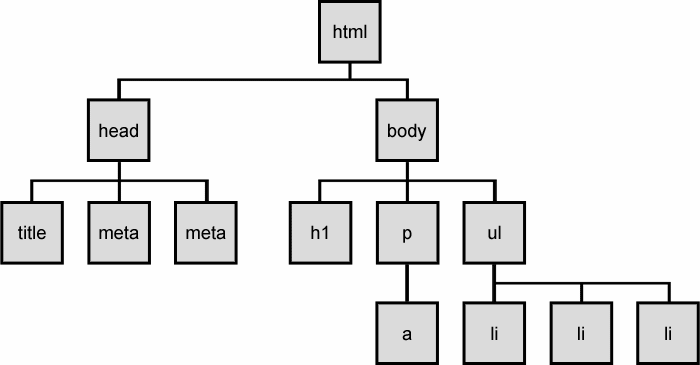
For example, we did these three things to the div element with an ID of “my-element”:

* showed a particular element on the page
* added a CSS class name to it, and
* attached a click event handler to it

### The DOM, or Document Object Model

jQuery's primary purpose is interacting with the HTMLdocument object model (DOM). The DOM is the representation of your HTML document in your computer's memory. You’re already familiar with the DOM though we haven’t really used that terminology yet...

(draw this on board):



The DOM is referred to as a tree since it has many branches. Some terms we need to be familiar with (draw on board with previous image):

* node: a single element in the tree
* root: the top-most node in the tree
* parent: the node exactly one level above me
* child: any node exactly one level below me
* sibling: any node at my level
* ancestor: a node at any level above me
* descendant: a node at any level below me

### CSS Selectors

JQuery uses CSS selectors (who can remember the 3 ways we reference our elements from CSS?)

$("a") //=> all <a> elements  
  
$("li a") //=> only <a> elements that are  
 //=> descendants of <li> elements  
  
$(".special") //=> all elements with a class of "special"  
  
$("#heading") //=> the element with an id of "heading"

### Attributes, CSS, Events, and Animation/Effects

JQuery can manipulate attributes - let’s add some more to our code:

$("#my-element")

.show()

.addClass("selectable")

.click(function (e){

alert("You clicked something!");

})

// Get an attribute's value

var id = $("div").attr("id");

console.log(id); //=> "my-element"

// Set an attribute's value

$("div").attr("id", "your-element");

Now, refresh your page and inspect the div element. What is the id of it? Here are some more [JQuery Attributes](http://api.jquery.com/category/attributes/).

JQuery can also manipulate CSS classes. Let’s play with some [CSS manipulation](http://api.jquery.com/category/css/) (do each one-by-one):

// Set an attribute's value

$("div").attr("id", "your-element");

// Add a class

$("div").addClass("special");

// Remove a class

$("div").removeClass("special");

// Add or remove a class, depending on whether

// it's currently present

$("div").toggleClass("special");

// Check whether a class exists

if ($("div").hasClass("special")){

alert("It's special alright!");

} else {

alert("It's normal.");

}

Now let’s play with [Events](http://api.jquery.com/category/events/) by making our div show the div’s text whenever it is clicked (delete the previous script):

$(function (){

$("#my-element").click(function (e){

var paragraphText = $(e.target).text();

alert(paragraphText);

});

});

Finally, let’s play with Animation, or [Effects](http://api.jquery.com/category/effects/), by making our div slide up and away when we click on it:

$(function (){

$("#my-element").click(function (e){

$(e.target).slideUp();

});

});

Did you notice how in the last 2 examples, we used $(function (){});? Since JavaScript frequently manipulates elements in the DOM, you need to wait until those elements are rendered before you try manipulating them. By including our JQuery code inside that function, it will know to wait until the DOM elements are rendered.

## Homework

1. Create a page that hides all of its <p> tags after 5 seconds. You'll want to use [delay()](http://api.jquery.com/delay/) and [hide()](http://api.jquery.com/hide/).
2. Create a button that shows a [Bootstrap Modal](http://getbootstrap.com/javascript/#modals) when you click it. They list several ways to do it, so try the one under the Via JavaScript heading.

## Resources

JQuery documentation: [JQuery Attributes](http://api.jquery.com/category/attributes/), [CSS manipulation](http://api.jquery.com/category/css/), [Events](http://api.jquery.com/category/events/), [Effects](http://api.jquery.com/category/effects/)

JSFiddle: <http://jsfiddle.net/>