JavaScript & jQuery

Make Your Site Dynamic!

JavaScript Intro

JavaScript is an object oriented programming language that is primarily used to make web pages interactive (though it can be used to run databases, mobile apps, and even robotics). It is also the most popular programming language in the world!

Why is it so popular? Because it runs in all browsers, or client side, so it's functionality works on all devices with a browser. This gets rid of our need for server requests and makes web apps much, much faster.

JavaScript is growing everyday, and is becoming a ubiquitous language across all platforms.

JS is not the most powerful or the most flexible, but very popular and practical. It is intentionally limited.

JavaScript works in **the web browser**

Client vs. Server

The client is your computer, or more specifically, your browser. And remember the browser can only read three languages!

Because of that, any web application that we have written in any other language has to send a request to a remote server. This request takes time. The more requests you make, or the bigger the request is, the slower your application.

JS: A History

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.

Actually, it was created in 10 days. And because it was a rush job, it wasn't fully functional, nor was it standardized.

JS Frameworks

Because JavaScript is a language that has been sort of hacked together over the ages, developers have been trying to make the language more uniform and easier to write.

The common way this is done is through a library (or framework). A JavaScript library is pre-written JS which allows for easier development of common JS practices. Unlike other languages which have just a few frameworks each, JS has TONS of libraries.

The Console

Let's get into some pure JavaScript and run some commands, functions, and programs in the console. Where is your console? In your browser, of course!

Console is another term for terminal, but basically it is an area that reads and executes inputted commands. Fortunately for us, Chrome has a built-in console that reads and executes JavaScript!

Go to any web page, right click, select inspect element, go to the console tab. While here, enter:

```
> console.log("Hello, World!");
```

- We type commands
- Something may be is displayed
- A value is returned

Strings

Instructor Note: If you are running this lesson after going through Ruby, the students should obviously know what a String is.

“Hello World” is what is called a String, a data type frequently used in JavaScript and all programming. A String is a way of representing a sequence of characters. Here are some examples of JavaScript strings:

```
“My name is John”

‘Wow, this is so simple!’

“1234”
```

As you can see above, we surrounded our Strings in single- or double-quotes .

Take special note of the last example, “1234.” Although it has a numeric value in it, you may not be able to perform arithmetic operations on it. We'll touch more on that in a bit.

Numbers

Instructor Note: Once again, if you're teaching in a post-Ruby environment, students know Integers and Floats - however, point out that JavaScript does not descriminate between those two types, and they're all just Numbers.

Within JavaScript, we have numbers as well. These numbers can be with or without decimals. This is the Number data type. Take a look at the examples below:

```
1234

23.4

0.45
```

This is not usually the case in programming languages. Usually whole numbers and decimals are two different data types!! Let's try this out in console:

```
> 1 + 3
<* 4
> 3 + 34.5
<* 37.5
```

Mathmatical Operations

The following are basic arithmetic operators :

- for addition
- for subtraction
- for multiplication / for division

Now let’s use a few of these in console :

```
> 12 * 4
<* 48
> 34 / 3
<* 11.333333333333334
> 8 / 0
<* Infinity
```

From these examples, you can see that one of JavaScript ’s many functions is to perform simple calculations.

Data Type Conversion

In most programming languages, before we combine data types together, we must convert them to match. We can do that in JavaScript.

Converting to String

```
String(456)

String(100.7 + 23)

(7).toString()
```

Converting to Number

```
Number("98")
Number("3.14159")
Number(" ")
Number("54 42") //Returns NaN which stands for Not a Number!
```

Fortunately for us, JavaScript does a pretty good job of Automatic type conversion!!

```
9 + null    // returns 9 - because null is converted to 0
"9" + null  // returns "9null" - because null is converted to "null"
"5" + 2     // returns 52 - because 2 is converted to "2"
"5" - 2     // returns 3 - because "5" is converted to 5
"5" * "2"   // returns 10 - because "5" and "2" are converted to 5 and 2
```

Sometimes it's a little unpredictable, but it usually does what you want it to :)

Writing JavaScript Code

In order for us to run JS programs in Chrome, we have to run it through an HTML file.

```
* Create a new folder called JavaScript.
* In that folder create a new file called conversion.html
* Build out your html structure.
* Open that file in Chrome.
* Open up the console.
```

In your html code, add the script tag to your body:

```
<!-- conversion.html -->
<body>
   <script src="height_to_centimeters.js"></script>
</body>
```

In the same JavaScript folder, create a new file called height_to_centimeters.js

Once this file is opened, enter the following line of code:

```
console.log(72 * 2.54);
```

Once these changes have been made and you've saved your program, refresh your browser to see:

```
182.88        height_to_centimeters.js:1
```

Congrats! You just wrote your first program!

Did you notice the semi-colon?

;

In JavaScript (as in many languages), the semi-colon is very important. It denotes the end of an expression and tells the runtime that particular action is over. We will see these at the end of a lot of lines of code (but not all of them!).

Do not forget your semi-colons!

Variables

In JS, variables are memory locations which hold any data used within a given program. Think of it like a container. Let's try a simple example. Navigate back to your height_to_centimeters.js file to enter the following line of code in place of what we entered before:

```
var a = 72;

console.log(a * 2.54);
```

Refresh the page in Chrome, look at the Console, and you will see that it outputs the same value as before, but returning the value from a different line.

In JS, we declare a variable using the var keyword. This creates a new instance of the variable. We can reassign the variable's value later, but when we do, we do not need to use the var key word again.

```
var x = "Aaron is dangerous";
console.log(x);
x = "Aaron is more dangerous than juggling chainsaws...on fire!";
console.log(x);
```

JS is also Dynamically Typed, which means there is no limitation on what a data type variable can store.

```
var theAnswer = 42;
console.log(theAnswer);
theAnswer = "Abe Lincoln";
console.log(theAnswer);
```

Also, variable names must start with a lowercase letter. It is convention to use camel case if it has more than one word.

Now, let's create a new program called **imperial_to_metric_height.js**. Enter the following code into the program via Sublime:

```
var myName = 'John Smith';

var heightInches = 60;

var weightPounds = 120;

var heightCentimeters = heightInches * 2.54;

var weightKilograms = weightPounds * 0.453592;

console.log(myName + ' is ' + heightCentimeters + ' cm and ' + weightKilograms +
' kg.');
```

As you can see, we've used a bunch of variables to store different information such as name, height, and weight. We then went on to call these variables back in at later points in the program, making use of the automatic data conversion and concatenation.

Link up your new JS file to your conversion.html, and refresh your browser!

```
<!-- conversion.html -->
<script src="imperial_to_metric_height.js"></script>
```

Once the program has been called in the Console, the output should be a sentence containing information about the name, height (centimeters), and weight (kilograms) of John Smith. Bear in mind that **console.log** is responsible for this, as it prints out to the console what was passed to it.

Dynamic Programs

Up until now, we have just been creating static programs. Meaning, if we wanted the outcome of our program to change, we would have to go into our source code and hard code the changes ourselves.

Problem is: that defeats the whole purpose of programming, right? Let's make our programs interactive, by retrieving user responses and using them in our program.

The best way for us to do this in Chrome is to use a function called **prompt()**. Prompt is a built in function that will pop up an alert box, and allow the user to enter data. The data is then stored in memory for our use in the program! With this we must also have a user prompt, to let the user know exactly what we are looking for.

```
var deepThought = prompt("What is the answer to life, the universe, and everything?");
```

Navigating back to **height_to_centimeters.js** in Sublime, let's make use of **prompt()**:

```
var a = prompt("How tall are you in inches?");

console.log(a * 2.54);
```

Refresh your browser, and notice the pop up that now wants your info. Once you enter any value, it saves the value to the variable a and applies it to the rest of the program.

Classroom Challenge: Edit height_to_centimeters.js in Sublime to have it output something like this in Console:

```
> 60 inches = 152.4 centimeters
```

Classroom Challenge:

Instructor Note: Ask students for input

Edit your Imperial_to_metric_height.js to be fully interactive:

```
var myName = prompt("What is your name?");

var heightInches = prompt("How tall are you in inches?");

var weightPounds = prompt("How much do you weigh in pounds?");

var heightCentimeters = heightInches * 2.54;

var weightKilograms = weightPounds * 0.453592;

console.log(myName + ' is ' + heightCentimeters + ' cm and ' + weightKilograms + ' kg.');
```

Functions

We've already used several built in JS functions. It's important to note that the following can all be considered methods: console.log("Hello handsome!"); prompt("Type your name"); String(45); (45).toString(); ("Hello").length ("stop yelling...").toUpperCase();

That's right! All of these are considered functions . This is because they all must be applied to some sort of object (i.e., argument) in order to produce any meaningful output.

A JS function is a block of code designed to perform a particular task. A function is executed when "something" invokes it (calls it).

A function allows us to package many lines of code, and perform those lines in one simple call. We can use this short call to perform the function over and over again, without us writing the code ourselves. This helps keeps things DRY!

*Instructor Note: Make sure the student understand that... javascript:function::ruby:method *

While there are many built in/native functions, we can also build any custom function that we want.

```
function greeting() {
  console.log("Hello Zack!");
}

greeting();
```

We use the function keyword, followed by the functions name and a pair of parenthesis. This is also followed by curly brackets. Whatever is written inside of these brackets is performed when the function is called.

Unlike many other programming languages, we can pass variables into the function.

```
var myName = "Zack";

function greeting() {
  console.log("Hello " + myName);
}

greeting();
```

This is not recommended however. *Instructor Note: just as using a global (@) variable into a Ruby method is possible, but not recommended.*

What is recommended is passing data into the function via parameters/arguments.

```
var myName = "Zack";

function greeting(name) {
  console.log("Hello " + name);
}

greeting(myName);
```

In the function name, we have an argument inside of the parenthesis. This acts as a variable. Then when we call the function, we must pass in data through that argument slot.

We can have as many arguments as you want. Though they must be entered in order.

```
var myName = prompt("What is your name?");
var newLang = prompt("Do you know a language besides English?");
var welcomeSaying = prompt("How do you greet others in that language?")

function greeting(name, language, saying) {
  console.log(saying + " " + name + ", nice to speak with someone who knows " +
language);
}

greeting(myName, newLang, welcomeSaying);
```

A function can also "return" a value.

```
function addItUp(x,y) {
  var z = x + y;
  return z;
  // or simply:
  // return x + y;
}
```

You can then assign the call of that method to a variable:

```
var num = addItUp(40,2);

// you can also still call it
// within console.log() or alert()
```

Change your **imperial_to_metric_height.js** and use a function to help you there!

```
function convertInchesToCentimeters(number) {
    heightCentimeters = number * 2.54
```

```
    return heightCentimeters
}

var myName = prompt("What is your name?");

var heightInches = prompt("How tall are you in inches?");

var weightPounds = prompt("How much do you weigh in pounds?");

var heightCentimeters = heightInches * 2.54;

var weightKilograms = weightPounds * 0.453592;

console.log(myName + ' is ' + convertInchesToCentimeters(heightInches) + ' cm and
' + weightKilograms + ' kg.');
```

Again the curly brackets open and close our function. You can also see that we named the function **convertInchesToCentimeters**, again using camel-casing. We included number as our argument. Finally, we called the function that we created back into the last few lines of the program passing in a variable we create earlier, heightInches. This variable is retrieving user input via the prompt function!

HTML Functions

Let's move out of the console. JavaScript has built-in functions that let us modify HTML & CSS.

First, we need to establish which HTML element we want to affect.

```
document.getElementById()

document.getElementsByClass()

document.getElementsByTagName()
```

Notice how the last two address "Element s "? That's why it's best to just work with IDs!

Once you select something to change, we have to figure out what to do with it.

```
//add or replace HTML text inside an element with a certain ID
document.getElementById("div1").innerHTML = "Something changed here...";

//add a class to an element with a certain ID
document.getElementById("div2").className = "nice-div";

//add an ID to an element with a certain ID
document.getElementById("div3").id = "cool_id";
```

Conditionals

Sometimes, we want JS to perform an action only if a certain condition is met.

```
var num = 45;

if ( num < 50 ) {
    console.log("Less than half.");
}
```

In addition to if , you will also usually tell the program what else to do:

```
var num = 45;

if ( num < 50 ) {
    console.log("Less than half.");
} else {
    console.log("That ain't half bad!");
}
```

JS Activity: MadLibs

Instructor Note: only do if you think there's enough time

Create a Mad Lib Program! * Have at least 10 inputs * Include at least one number

Example Please enter the following. Exclamation: Holy Bananas Name: Zack Verb: jump

Holy Bananas! Is that *Zack*? Wow, they sure do love to *jump*!

```
alert("Please enter the following...");
var excl = prompt("Exclamation");
var name = prompt("Name");
var verb = prompt("Verb");

console.log(exc + "! Is that " + name "? Wow, they sure do love to " + verb +
"!");
```

Create your own story, or steal one from the internetz.

jQuery Intro & Set-Up

jQuery is a JavaScript library... It sits on top of the existing JavaScript language and reduces the amount of code we have to write when we want to use basic functions.

Let's practice using jQuery in a new space...

Step 1: Create a new folder called jQuery. In that folder, let's create an index.html page.

Step 2: Inside index.html, build out the basic structure of a HTML page - include the DOCTYPE, a title, a head and a body.

Step 3: In your , create a link to an external stylesheet called style.css, then create that file in your folder. Next create a JavaScript file called site.js. Link that script file and our HTML file together, and then embed the jQuery library into our file via a CDN.

Note: JavaScript is typically included at the bottom of your HTML page. The reason for this is we want our HTML to load before our JavaScript loads, because again, our JS is manipulating our HTML.

```
<!-- index.html -->
<body>
    <!-- Lots of HTML. Oh yeah! -->
    <script
src="https://ajax.googleapis.com/ajax/libs/jquery/3.1.1/jquery.min.js"></script>
    <script src="site.js"></script>
</body>
```

Step 3 Explained: The first line is pulling in the library of jQuery code from jQuery's website and letting us use it in our site! Hey, thanks!

The second line performs virtually the same function as our element to our stylesheet - after our site pulls in the jQuery library, it will look to our site.js file for any scripts that we write and pull them in so that whatever we write in our script file will function in our webpage, exactly like how when we write CSS in our style.css file.

HTML/CSS Challenge

Instructor Note: Give about 2-5 minutes for students to complete

```
* Create two red squares and one green circle.

* You should have a "square" class and a "circle" id.

* Big Hint: use the border-radius property.
```

Click & Hover Functions

Now that you've got your shapes, go into your script file and type in the following:

```
$('#circle').click(function() {
 alert('Clicked!');
});
```

$ - “string” character. A sign to define/access jQuery

(“#circle”) - our “selector”. This selects what we’ll be applying a function to

.click(function() - tells jQuery when it should apply the function, in this case when the user clicks

{alert(“Click!”);}) - the “function”, or what we actually want to happen - in this case, fire a pop-up alert with the text “Click!”

Now try to select one of your squares instead of your circles, refresh and see if your script still works!

Now, instead of creating an alert when you click on a shape, let’s change some text. In your HTML file, comment out your squares and put in a paragraph element with a short sentence, like...

```
<p>”This is a sentence!”</p>
```

Back in your site.js file, we’re still going to use the top line of our code - selecting our circle and performing a function when the user clicks on it - but we’re going to change what that function is. Let's replace the alert...

```
$('#circle').click(function() {
  $('p').html(“We’ve changed the text!”);
});
```

What if we want the text change when we hover over the circle? We would simply change click to hover on Line 1! It's that easy!

Changing some text is cool, but we can change more stuff too... like a HTML attribute. Insert an image under our

tag, remembering to include the “src=” attribute. Then, in our script file, replace Line 2 with this:

```
$('img').attr('src', '...');
```

Replace that ellipsis with the URL of another image. Now refresh your page and try it out!

Now, we’ll actually work with the styles of our elements. Let’s bring back our squares in the HTML file and get rid of our paragraph and images. Use jQuery to change the squares from red to blue when clicked:

```
$('.square').click(function() {
  $(this).css('background-color', 'blue');
});
```

this? What does that mean? *this *is a JavaScript function that simply means the object that’s been referenced immediately before (so, in this case, 'div').

The CSS function is just like the attr function we used a minute ago, with a CSS attribute/value pair within the parantheses.

jQuery Challenge

Instructor Note: give students 2-5 minutes to complete

Without the instructor showing you what to do, I'd like you to change the width of the squares to 400px after the user hovers over them.

```
$('.square').hover(function() {
  $(this).css('width', '400px');
)}
```

More jQuery Functions

.css() adds the CSS rules into a style attribute for the specified HTML tag, which we learned is not optimal. Why not just add and remove pre-made CSS classes?

.addClass(), **.removeClass()** Take a parameter of a class (no dot/period needed)

.toggleClass() Takes same parameter as above, but will do both on the same clickable ID

.val() Can pull or set the value from a specified input field

.append(), **.prepend()** Adds to text rather than replace (unlike .html() or .text())

.after(), **.before()** Similar to append/prepend, but adds before/after the selector, rather than within

.hide(), **.show()** Essentially adds and takes away CSS rule of display: none;

.fadeIn(), **.fadeOut()** Same as hide/show, but not so immediately (you can set the rate in milliseconds as a parameter)

.fadeToggle() Combines fadeIn/fadeOut into one clickable

.fadeTo() Does fadeOut, but to a certain opacity (given as a second parameter)

.slideUp(), **.slideDown()** Once again, a twist on hide/show

.slideToggle() Combines slideUp/slideDown into one clickable

jQuery UI

jQuery UI is the official home to a lot of great pre-built plugins that save lazy people like you and me some time. There are two files we need to import to be able to use plugins from the UI:

```
<link rel="stylesheet"
href="https://code.jquery.com/ui/1.11.4/themes/smoothness/jquery-ui.css">

<script src="https://code.jquery.com/ui/1.11.4/jquery-ui.min.js"></script>
```

The link tag should go in your underneath your CSS link. Your script should go just before your own script file at the bottom of the .

Now that we've imported these two files, we can use special classes and functions to give us even more possibilities. UI is a lot like Bootstrap, in that it uses pre-built styling and custom classes.

Draggable

To make an element, like a

, draggable, it only takes a few lines of code now that we're using the jQuery UI library. Create a with the ID **"draggable"** and the class **"ui-widget-content"**. Inside that
, create a paragraph with the sentence "Drag me around." In your script file, we're going to select the div by using its ID and call the function **draggable**.

```
$(function() {
  $('#draggable').draggable();
  })
```

Notice that we aren't telling the function when to activate (click, hover, etc.). We skip the whole "selector" process and go straight to the "function". This is because we want the script to be running all the time.

Resizable

We only have to change a few things to turn our

from **draggable** to **resizable**! What do you think we should do?

```
$(function() {
  $('#resizable').resizable();
  })
```

And don't forget to add the "resizable" id to an HTML element on index.html!

Challenge: Sortable

See if you can implement this! Visit https://jqueryui.com/sortable/, and click 'view source' to see how to implement.

```
<html>
  <head>
    <title>Simple Page</title>
  </head>
  <body>
    <p>This is a very basic HTML page</p>
    <script src="script.js"></script>
  </body>
</html>
```

```
alert("Hello smart TTS students");
```

Computers are about input and output!

Javascript is all about the input and outpout on the webpage.

Right now, we have an alert box.

Let's add in some input.

Now change script.js:

```
var name = prompt("What is your name?");
alert("Hello, " + name);
```

We create a variable in JavaScript like:

```
var windSpeed;
var todaysDate;
```

We are carving out memory to hold our variable, but we don't have a value for our variable yet.

```
windSpeed = 20;
// or we could create and initialize our variables
var windSpeed = 20;
```

Strings

In java script, we can use "" or '' to hold strings, but dont mix them

create a string called message

```
var message = "Hello there!"
alert(message);
```

So, we aren't printing out "message", but instead the value that message holds

Basic String Properties

How long is the string?

```
alert(message.length);
```

Condtional Code

The if statement

```
var a = 5;
var b = 10;

if ( a < b ) {
   alert("Yes, a is less than b");
}

if ( a == b ) {
   alert("Yes, a is equal to b");
}
```

switch statement

```
var my_day=new Date();

switch (my_day.getDay())
   {
    case 0:
        console.log("Today is Sunday");
        break;
    case 1:
        console.log("Today is Monday");
        break;

    case 2:
        console.log("Today is Tuesday");
        break;
```

```
    case 3:
        console.log("Today is Wednesday");
        break;

    case 4:
        console.log("Today is Thursday");
        break;

    case 5:
        console.log("Today is Friday");
        break;

    case 6:
        console.log("Today is Saturday");
        break;

    default:
        console.log("value of i is not equal to any given days");
        break;

}
```

Loose typing

variables are declared with var, but have a type that can be changed

```
var z = 1;
z = 'abc'; //would throw an error in strong typed languages
typeof z //string
z = 1
typeof z //number
```

Coersion

variable type is coerced to a type that makes sense when reached

IMPLICIT coercion

```
7 + 7 + 7; // = 21


// First two 7's are calculated... then concatened into a string.
// returning a string value
7 + 7 + "7"; // = "147"

// The entire expression is implicitly converted into a string and concatenated
"7" + 7 + 7; // = "777"
```

Above, when we add Integers (numbers) together, we get a sum of 21. However, in our second example. The first two 7's are calculated and then, converted into a string and concatenated with the string value of 7. In the final example, the entire expression is almost immediately converted into a string.

EXPLICIT coercion:

As you can see, JavaScript is trying to be helpful, and sometimes this is the desired behavior. However, what happens when you retrieve data from a user and it needs to be forcibly coerced? This is where Explicit coercion comes in.

Let's say you are helping a local non-profit raise funds and the pledges are coming in through the website. Obviously, the data is going to come in a string. It then needs to be coerced into a number (so that it can be added).

Here's an example:

```
// perform in browser of JS Bin
var amountRaisedSoFar = 1000;

var newDonation = prompt("How much would you like to donate?");

amountRaisedSoFar = Number(newDonation) + amountRaisedSoFar;

console.log("We have now raised: " + amountRaisedSoFar + "!");
```

Here we are taking the user input in as a string, then converting it into a number when adding it to the amountRaisedSoFar variable.

Type Casting

You can convert from one type to another using these other built-in conversion functions as well.

- `ParseInt`
- `parseFloat()`
- `toString()`

`null` and `undefined`

According to the Mozilla Documentation

> A variable that has not been assigned a value is of type undefined.

```
var something;
typeof something // "undefined"
```

```
**CHALLENGE 1** Do you need more coffee?<br>
(3 minutes)

Write a script that:

- stores the number of cups (that a person has consumed) in a variable
- if the person has had *less than* 3 cups
 - log a message to the console saying: ("Yes I'll take another cup of coffee")
- if not
 - log a message to console saying ("I think I'm okay for now")


**CHALLENGE 1 ANSWER:**

```javascript
var cups = 1

if(cups < 3){
    console.log("Yes. I'll take another cup of coffee!");
} else {
    console.log("I think I'm okay for now.");
}
```

Loose Equality

The double equals `==` tries to ignore the type when comparing. The triple equals `===` takes into account type. These are called loose and strict equality checks. You pretty much always want to use strict.
```

```
var x = 10;
//Type coersion is happening here!
if(x == '10') {
  console.log(true); //true
}

if(x === '10') {
  console.log(true); //false
}
```

Comparison Operators

- `<`
- `>`
- `<=`
- `>=`
- `==`
- `===`
- `!=`
- `!==`
- `!!`

CHALLENGE 2
(10 minutes)

1. Create a variable for the temperature and set it to 80
2. Write a statement that outputs the temperature as "The temperature is 80 degrees";
3. If the temp is greater than 80, output "time to swim" (set temp to 60, 90) and test;
4. Create a precipitation variable and set it to false
5. Only output "time to swim" if temp is greater than 80 and its not raining
6. Set the precipitation variable to 'raining' or 'snowing' and only output 'time to swim' if there is no precipitation
7. Create an 'indoors' variable and set it to true
8. If indoors, then output 'time to swim' regardless of the temp and precip.

CHALLENGE 2 ANSWER:

```
var temp = 85;
var precipitation = false;
var indoors = true;

console.log("The temperature is " + temp + " degrees");

if (temp > 80 && precipitation === false) {
  console.log("time to swim!");
} else if (indoors) {
  console.log("time to swim!");
}
```

Iteration

For loop

```
for (var i = 0; i <= 9; i ++) {
    console.log( i );
}
```

Let us work through each part of the loop.

within the parentheses, there are 3 key things happening:

1. We are **setting** a variables value (in this case, to 0)
2. We are **comparing** the variables value to the desired break-point
3. we are **incrementing** the *value* of the variable on each *iteration*

CHALLENGE 3 - 99 Bottles

(5 minutes)

- Using a for loop.
- Write a simple version of "99 bottles of beer on the wall"
  (note: make sure youre logging the result to the console)
- Once you get the program running, log "Hey! We need more beer!" to the console when your counter hits 0

CHALLENGE 3 ANSWER:

```
var bottle = 99;

for (bottle; bottle >= 0; bottle --){

  if (bottle === 0) {
    console.log("Hey! Go buy more beer!");
  } else {
      console.log(bottle + " bottles of beer on the wall");
  }
}
```

While Statement

```
var x = 0;
while(x < 10) {
  x = x + 1;
}
```

Break vs Continue

A `break` statement jumps *out* of the loop.

```
for(var i = 0; i < 10; i++) {
  console.log(i);
  console.log('before break');
  break; // exits the next loop
  console.log('after break'); //never happens
}
```

A `continue` statement jumps to the next *iteration*

```
for(var i = 0; i < 10; i++) {
  console.log(i);
  console.log('before continue');
  continue; // continues to the next iteration
  console.log('after continue'); //never happens
}
```

Lab Activity

1. rock paper scissors

References:

1. http://javascript.crockford.com/javascript.html
2. http://blog.jeremymartin.name/2008/03/understanding-loose-typing-in.html