# Week 8: Introduction to JavaScript (JS)

**Agenda**

* Introduction to JavaScript
  + What is it?
  + How do we use it in our pages?
  + Basic Concepts
  + Class Example
  + JS Challenges
* Next Week

## Introduction to JavaScript (JS)

**What is it?**

JavaScript is a programming language which can be used to add interactivity to any web page. JS serves as the third main technology to web development on the front-end side of things. It is its own language but there are many APIs built on top of it which enhance its features and make coding easier. We can use it on the client side or server side (backbone.js, node.js, etc.), but for the purposes of this course, we will focus solely on the client-side applications of JS.

**How do we use it in our page?**

Adding JS to a page is simple to do. Like CSS, we can add it inline, internally, or externally. I’ve provided examples of each type in the PowerPoint notes, but I will explain them briefly below.

Inline

When we use inline JS, we place actual JS code inside of an HTML elements attribute. We might use something like the onclick attribute and set it equal to a function we define in our JS code. **DO NOT USE THIS TYPE. IT IS VERY BAD PRACTICE.**

Internally

Internally placed code is located inside of <script> tags which can be placed anywhere in the document. With internally placed code, our onclick function and all related code would be placed inside of the <script> tags. We would use an ID to hook the JS to our HTML.

Externally

Externally placed code would be in an external JS file entirely. You would use a script tag with a source attribute to link your JS/HTML. Like internal JS, the HTML id attributes are used to hook the JS and HTML.

**Best practice** dictates that we place our JS in a **separate file** and place the <script></script> tags directly **before the closing body tag**. The reason we do that is because our HTML needs to load first. If we use a button for a JS function and the JS loads before the button, our JS won’t link to it correctly and users may not be able to use your page as expected. Separating our code from our structure also helps keep our code tidy and easier to maintain in the long run.

**Basic Concepts**

There are many components to JS that I could cover but this week we will start with the basics.

Like HTML and CSS, you can leave comments in your JS code to explain what is happening. To create a single-line comment, you can use two forward slashes (“//”) or an asterisk and forward slash for multi-line comments (“/\* \*/”).

// This is a one-line comment

/\* This comment spans

Multiple lines \*/

Variables are the foundation of JS and without them, we can’t accomplish more complicated things like form validating, modals, image carousels, etc. Variables typically get assigned values and those values are used to add interactivity on our page or reference elements. You can use any name you want for a variable with some exceptions. It can’t begin with a number or be a reserved word in JavaScript (bool, break, etc.)

var x = 5;

The above line is called a JavaScript statement and each one should end in a semi-colon. We start by declaring a variable called x. When we declare a variable, we always use lowercase “var”. The equal sign doesn’t mean that x equals 5. Instead, when you see that symbol, think about it as an assignment. The line above means we are assigning a value of 5 to a variable called x. We could also just say “var x;” which would declare the variable but not give it a value. Please note that variable names are case sensitive so “var x” and “var X” would refer to two different variables.

In that example, the value of 5 has a “type” which all values have. There are six “primitive types” in JS and they are as follows: Boolean, null, undefined, number, string, and symbol. The most common ones are Boolean, number, and string. Everything else besides that is an object and objects have methods to work with them (arrays, functions, etc.).

A **Boolean** value evaluates to either true or false.

var catsAreCool = true;

var mosquitosAreCool = false;

A **Number** type is straight-forward and just reflects an numerical value.

var answerToEverything = 42;

A **String** type is always wrapped in quotation marks.

var myName = “Christopher”;

Unlike some other programming languages, JavaScript doesn’t allow you to declare a variable’s type. Instead, the browser engine determines what it is and goes from there which can lead to weird behaviors. If you were to try and add a string to a number type, it would treat both as a string and combine them. This is called “type-coercion” and you must be careful with your variable types when using them if you don’t want unexpected behaviors to occur.

With these different types, we have operators which we can use to manipulate them. Some of the most common ones are the mathematical operators (\*, /, +, -, %) and the concatenation operator (‘+’ which is used to combine strings). In the example below, I use the concatenation operator to combine strings but I also add a space because our strings don’t include them automatically.

var firstName = “Chris”;

var lastName = “Velez”;

var fullName = firstName + “ ” + lastName; // This would be “Chris Velez”

Having variables is great but sometimes we need to check their values to have certain things occur on our page. Let’s say we have a modal that appears which asks for a user’s age. If they are under 21, they can’t gain access to the website. If they are 21 or older, then they can enter (common for alcohol distribution websites). We can use **if/else** statements to do this.

var myAge = 28;

If (myAge < 21) {

Alert(“Sorry, you are not old enough!”);

} else {

Alert(“You may enter!”);

}

The if() portion checks to see if myAge < 21 is true. If it’s true, then the code inside of the curly brackets is executed. If it is false, it goes to the next else statement (or else if assuming there are multiple options). If the next statement is true, that will be executed. If none of them are true, none of the code gets used.

Another important feature of JavaScript is the ability to perform loops. If we wanted to list the numbers 1-100 in the console, we could say console.log(1); console.log(2); and so on. However, that isn’t very efficient and will result in code bloat. We can accomplish the same result with loops. The two main types of loops are For and While loops. They both contain the same features but are syntactically different.

Each loop requires three parts: Where we are starting, where we are finishing/the condition to check against, and how much we are incrementing/decrementing our number by. In the for loop below, the first part of the loop is where we start (0), the second part is where we go up to (less than or equal to 10), and the third part is what we increment our number by (++ is a shortcut for saying +1).

for(var x = 0; x <=10; x++) {

console.log(x);

}

We start with 0 and check the second part of the loop to see if it’s true. Since 0 is less than or equal to 10, we will console.log it. After it’s logged, we increase x by 1. The loop continues and since 1 is less than or equal to 10, it will run again. This continues until we reach the number 11. Since 11 is neither equal to or less than 10, the loop will exit.

var x = 0;

while(x <= 10) {

console.log(x);

x++;

}

The main difference between this loop (while) and the for loop is that the initial number and incrementing part are in different parts of the loop. You initialize the number before the loop and increment it inside of the loop. While loops are useful when you aren’t sure when your loop needs to end. For loops are more for when you know exactly when the loop should end. Always remember to give your loop a way to close itself. If we removed x++ from any of the loop examples above, x would always be less than or equal to 10. This would cause the loop to run forever and crash the browser.

The last thing I want to talk about is functions. Functions are blocks of code that are wrapped up in a container. The code inside of that container (curly brackets) isn’t executed unless the function is called. In the example below, I’ve created a function called “squareMyNum” which will take in one number as a parameter/argument and then it will return the square of that number (number times itself). After creating the function, I call it by using the name of the function and providing a value for the argument. The result is logged to the console since that was inside of our function. The console.log() method is useful because it allows us to test our values without putting anything on our page.

var squareMyNum = function(num) {

var result = num \* num;

console.log(result);

}

squareMyNum(5); // This would output 25 but only viewable in the console.

Functions don’t need arguments to work but, in some cases, you’ll find that arguments can help you with creating output or manipulating input. The content between the parentheses is where you can place your arguments/parameters. The area between the curly bracket is where you place the code you want to be executed when the function is called. You can pair functions with if/else if/else statements and other JS objects to create robust applications.

I’ve provided coded examples along with explanations in the Class Example section below. Please also read through the Additional Reading articles and the PowerPoint on Blackboard for JavaScript. You don’t need to read the entire page for each link below, there is a lot of content that is covered. Just read the portions that relate to the content above. All those materials should give you an idea of how those things work.

Additional Reading

* <https://htmldog.com/guides/javascript/beginner/makingstuffhappen/> (connecting JS)
* <https://javascript.info/types> (data types)
* <https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Statements/if...else> (if/else statements)
* <https://developer.mozilla.org/en-US/docs/Web/JavaScript/Guide/Loops_and_iteration> (all kinds of loops)
* <https://htmldog.com/guides/javascript/beginner/functions/> (functions)
* <https://eloquentjavascript.net/> (Free eBook on JavaScript - super comprehensive – I would check out parts 1-3 for this week)
* <https://www.w3schools.com/js/default.asp> (Tutorials on everything – use with caution)
* <https://www.javascript.com/> (More tutorials with examples)

**Class Example:**

<https://iinf362.000webhostapp.com/examples/javascript-intro/>

Right-click this page in order to view the page source. I’ve explained things on the page and left comments throughout the JavaScript to help you understand what is going on.

**If you haven’t done so already, please read the “Creating and Viewing our Web Pages.docx” file on Blackboard. You will not be able to submit anything for the assignment without completing that portion first.**

**JS Challenges**

Download the “JS-Challenges.zip” folder from Blackboard under the Lecture Notes for this week or in the Assignments folder. Inside of this zip folder, you’ll find an HTML file, a js folder, and a script.js file. Using these files, your task is to add relevant HTML and JS to complete the challenges listed below. Some of the challenges will display results in the browser and others will produce results in the console. Make sure you check both places to ensure you have completed the assignment. The challenges should be completed exactly as I have them described below.

Challenges

1. Create a form with three input boxes and a button. These input boxes should receive numbers as their values. When the user clicks your button/submit, a message should appear (using alert, console.log, or innerHTML) to give them the following information:
   1. What numbers were entered
   2. The sum of those numbers
   3. The average of those numbers

**Example Input:**

The user types in 3, 5 and 7 and then clicks submit.

**Example Output:**

“You entered the numbers 3, 5, and 7. The total of these numbers is 15 and the average is 5.”

For this challenge, you’ll need to create variables for at least each input and the button. You would then use the mathematical operators and string concatenation to output a final message. You’ll also use a onclick event handler (look at the class example…btn.onclick) to trigger the function. Depending on how you build your inputs, you may also need parseInt…the reason for this is that the type for input boxes is automatically set to strings. If you take the value and parse it as an integer, you’ll be able to use the values as numbers. Make sure to grab the values using document.getElementById() AFTER they click submit (inside of the function). Otherwise, your input values will be empty, and your function won’t work.

1. Create a function called “nameInfo” that takes in a user’s name as a parameter. When that function is called, it should tell the user the following information about their name:
   1. What their name is
   2. How many letters long it is
   3. If it is a short, average, or long name

**Example Function Call/Input:**

nameInfo(“Christopher”);

**Example Output:**

“Your name is Christopher. This name is 11 letters long and considered to be a long name.”

For the purposes of this challenge, names that are less than 5 letters are short, names that are 5-8 letters are average, and names with 9 or more letters are long. You’ll need an if/else if/else statement to handle these possibilities. The “.length” method for strings will help you get the length of the string. When creating your function, make sure it logs the information to the console. You can also have it output to the page if you’d like but the console will be easiest.

1. Create a function called “catdog” that will log the numbers 1-100 to the console. If the number if divisible by 3, you should log “Cat” to the console instead. If the number is divisible by 5, you should log “Dog” to the console. If the number is divisible by 3 and 5, you’ll log “CatDog”. Otherwise, the number should be logged to the console itself. The output should look like the example below but include output for all 100 numbers:

1

2

Cat

4

Dog

Cat

7

8

Cat

Dog

11

Cat

13

14

Catdog

To complete this part, you’ll need to use a loop along with an if/else if/else statement. I want you to do this with a **for loop** in addition to a **while loop**. The output for both loop types will be the same but the syntax will be different as described in the explanations above.

1. Create a function called “numberAdder” that takes in a number as an argument. The function will take the numbers from 1 to the number provided and add them together. After the numbers are added together, it should output the result.

**Example Function Call/Output:**

numberAdder(100); // Output would be 5050 (1 + 2 + 3 + 4 + 5… up to 100)

numberAdder(5); // Output would be 15 (1 + 2 + 3 + 4 + 5)

For this part, you can use a loop and the += operator to continuously add numbers to an original variable you set to 0. However, you can tackle this anyway you see fit. You can use the test numbers provided to ensure that your output is what you expect.

1. Create a paragraph on your page that says “This isn’t even my final form!” (<p>This isn’t even my final form!</p>). Add a JS mouseover event to that paragraph. When the paragraph has a mouse hovering over it, the text inside should change to “This is my **second** form!” (<p>This is my <b>second</b> form!</p>). Add one more event handler so that when that paragraph is double-clicked, the text will say “**THIS** is my final form!” (<p><b>THIS</b> is my final form!</p>).

To do this exercise, you’ll need one paragraph with the text I mentioned above. That paragraph should contain a unique id attribute value, so you can identify it. You’ll then add two separate event handlers (normally not good practice but do it for this assignment)…one will handle onmouseover and the other will handle ondblclick. Look at the onmouseover example provided in the class example for the syntax you’ll need. The “this” keyword will come in handy and you’ll need .innerHTML to change the text. Make sure to bold the words as I have done above. When you use innerHTML, you can place tags inside of the quotation marks and they’ll render as HTML in your page. Since you’re replacing the text of the paragraph, you don’t need to include that tag in your innerHTML declaration.

To receive credit for these challenges, submit a link to your JS Challenges page by Wednesday, March 25th at midnight. The assignment with be called JS Challenges in the assignments folder. It will also be accessible through the Lecture Notes for this week.

## Next Week (March 25th)

Next week, we will dive deeper into JS and some of the more complicated things it can accomplish. Specifically, we’ll cover features like modals, image carousels, form validation, and more. The links below will provide a glimpse into those topics. Make sure to read them to gain a better understanding of how JS can make a page more interactive.

1. <https://www.htmldog.com/guides/javascript/> (Objects, Arrays, DOM)
2. <https://developer.mozilla.org/en-US/docs/Learn/JavaScript/Building_blocks/Events>
3. <https://javascript.info/function-basics>