Week 6: JavaScript (JS) Bootcamp

Agenda

- Overview
  - What's Due?
- JS Bootcamp
  - What is it?
  - How do we use it in our pages?
  - JS Concepts
    - Commenting
    - Variables
    - Data Types
    - Operators
    - Creating Basic Output
    - If/Else Statements
    - Loops
    - Functions
    - Objects
    - Arrays
    - Working with the DOM
    - Events in JS
    - Getting User Input
  - Class Examples
- JS Challenges
- JavaScript Features Evaluation (Two Posts)
- Next Week

Overview

Last week, we finished learning about responsive design with CSS. This week, we will have our third and final bootcamp which will cover JavaScript. Specifically, we'll go over linking JS to our pages, JS syntax, and adding basic interactivity to our pages. By the end of this week, you should be able to add JS to your HTML and generate output to the page or the console.

What's Due

- JS Challenges
- JS Features Evaluation (Two Posts)

JavaScript (JS) Bootcamp

This week will serve as a refresher of some of the fundamental concepts as we have done for HTML and CSS. There will be a lot of content in this week's notes along with code examples and PowerPoint slides. Feel free to browse as much content as necessary. If you have questions, please send an email to me and/or your Peer Educator (Lu) for assistance.

What is JS?

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. It is its own

language but there are many frameworks 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.).

JS works based on the premise of **objects** and **functions**. In the lines below, I demonstrate some pseudo examples of objects along with some functions they could have.

Cat Object (Meow, Lick, Eat, Scratch, Purr)
Person Object (Talk, Run, Read, Write, High-Five)
Car Object (Accelerate, Brake, Turn, Lock, Unlock, Park)

The contents in the parentheses are functions that can be performed by the objects listed on the left. Each of those functions can take what are known as **arguments** or **parameters**. If a cat was to Meow, some "arguments" could be the volume or the frequency of the meow. We could create pseudo code to represent this as show below. The first "argument" is the volume and the second one is the frequency. I've included other examples as well to demonstrate how real-life items can be seen as objects with functions and parameters. I use periods to separate the object from the function and then parentheses are used for the parameters.

Cat.Meow("high", "medium"); // A cat performing the Meow function with the volume argument set to high and the frequency argument set to medium

Person.Talk("low", "spanish"); // A person performing the Talk function with the volume argument set to low and the language argument set to Spanish

Car.Turn("left"); // A car performing the Turn function with the direction argument set to left.

JavaScript has many functions that can be used with the page or objects on the page. These functions sometimes take in parameters and are usually trigged by some sort of **event** on the page. In the examples above, I have shown you objects that perform functions. In the one below, I show what kind of events can be used to trigger functions on a web page.

Objects on a Page (Clicked, Hovered Over, Changed, Loaded, Key Pressed, etc.)

When any of the actions in the parentheses are performed on or with an object on a page, we can trigger a function to create some action. We can provide arguments to the function to change the outcome or result. Before I can provide examples of some JS usage based on events, I will show you how to link it to your HTML pages.

How do we use it on 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 as well below. There's also a link below which demonstrates linking JS using all three methods.

https://www.albany.edu/~cv762525/cinf362/examples/js-linking-example.html

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. **I**

wouldn't recommend this as it isn't good practice. In this example, clicking the button produces a popup message that says "Hello World!"

Remember how I said we can trigger events on objects in our HTML pages? The button is the object and the onclick event is our trigger to the alert function. The alert function accepts what is known as an **argument** or **parameter**. In this case, the parameter is a string with a value of "Hello World!". We will learn about strings later, but just know that without some sort of value, the alert wouldn't output anything.

```
<button onclick="alert('Hello World!');">Click Me!</button>
```

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 can target the button many ways but one of the easiest ways to do so in JS is with the document.getElementById function.

With document.getElementById, we are targeting the "document" object and using its "getElementById" function. getElementById searches the whole document for the provided id value. The id value is known as the parameter or argument that we supply to the function. Our retrieved elements is then linked to a function which contains our alert function.

It does the exact same thing as the button onclick code from above, but it is linked differently. The id attribute value must match the value provided in getElementById or it will not work.

```
<button id="myButton">Click Me!</button>
<script>
      document.getElementById("myButton").onclick = function(){
         alert("Hello World!");
      }
</script>
```

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. Similar to what you do with CSS or any other linked source, you need to make sure the folder and file structure match up in your reference to it. With external JS, the HTML id attributes are used to hook the JS and HTML.

HTML File

```
<button id="myButton">Click Me!</button>
<script src="script.js"></script>
```

Script.js file – Doesn't contain HTML, just JavaScript

```
document.getElementById("myButton").onclick = function(){
   alert("Hello World!");
}
```

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.

JS Concepts
This week, we will cover some basic concepts such as comments, variables, data types, and functions so that you can build some simple interactive components on your web pages. Most of the output from the topics mentioned below will be displayed in the console. If you choose to replicate this code, make sure you check that place out for your output.

Comments
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
Variables are the foundation of JS and without them, we can't accomplish more complicated things like form validation, modals, image carousels, etc. Think of variables as boxes that store data or pieces of information inside of them. Each box holds a value and then we can use the value later for whatever we want.

They 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 statement 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, that is an assignment operator. Operators are used to perform some function or calculation. A single equal sign is used to assign values. That var x = 5; 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.

Data Types
In that example, the value of 5 has a "type" which is true for anything with a value. 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. In this example, we have created two separate variables and assigned a value of true to one of them and a value of false to the other one. You can use **comparison operators** to check if values are true or false.

```
var catsAreCool = true;
var mosquitosAreCool = false;
```

A **Number** type is straight-forward and contains a numerical value. You can use **mathematical operators** with numbers to add, subtract, multiply, or perform other mathematical functions.

```
var answerToEverything = 42;
```

A **String** is a bit of text and is always wrapped inside of quotation marks. String objects have properties or functions such as length, search, split, and indexOf.

In the first line of my example below, I declare a variable called myName and assign it a value of "Christopher." In the next line, I create another variable and assign it the value of myName.length. The length is property that can be used on any string to find its length. We use the period to separate an object and its property or function.

```
var myName = "Christopher";
var myNameLength = myName.length; // This value would be 11 (number type)
```

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.

<u>Operators</u>
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"
```

Mathematical Operators:
(+) This is used for adding numbers
(-) This is used for subtraction
(*) This is used multiplication
(/) This is used for division
(%) This is modulus and is used to calculate the remainder after division is performed

The examples below are based on x and y which have a value of 5 and 4 respectively. The statements aren't actual JS statements but I'm using them to depict the syntax and the output in parenthesis.
var x = 5;
var y = 4;

x + y (9)
x – y (1)
x * y (20)
x / y (1.2)
x % y (1)

For modulus, you perform the division first, and then take the remainder. 5 divided by 4 is 1.2. However, you should think of it like this: 4 goes into 5 one time, and the remainder is 1.

Comparison Operators (in parenthesis):
(>) This is the greater than sign
(<) This is the less than sign
(==) The double equal sign compares values but not data types
(===) The triple equal sign compares vales AND data types…it does a more strict comparison
(!=) This is not equal

Comparison operators are typically used to check values against each other. You can check the values or variables to ensure they match or are different.

Logical Operators:
|| - This represents "or"
&& - This represents "and"
! – This represents "not"

You can use the three logical operators to check conditions. If you need two conditions to be true, you would use &&. If you only needed one of them to be true, you would use the or operator.

Creating Basic Output
You can output information to the page in several ways. Like the document object, the console is another object which is useful for doing testing on an HTML page. Log() is one of the methods of the console object and it can accept many things as a parameter.

For example:
console.log("My name is Chris Velez."); // Outputs whatever is in the string
console.log("My name is " + fullName); // Outputs the same as the line above because we created a variable called fullName earlier.

var myAge = 29;
console.log(myAge); // outputs 29

alert(myAge); // Would create a popup with the value of the myAge variable. Alert is the function and the thing inside of the parenthesis (myAge) is the argument/parameter we provide.

To output something directly on the page (create HTML), you can use "innerHTML." We haven't covered the document object or working with the DOM yet, but the code below is a quick demonstration of how we can change text on a page. Feel free to revisit it once you understand the DOM more.

HTML
This will be changed.

JS
document.getElementById("changeMe").innerHTML = "This is changed.";

In the code above, we are using the document's getElementById method to get an element by its id attribute (changeMe). After we get it, we use the innerHTML property to change the text inside to "This is changed."

If/Else Statements

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 = 29;
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. The first part is in parenthesis and whatever is placed inside of there must evaluate to true or false. Each code block that we want to execute should be separate by curly brackets.

You don't need to use else if or else statements with an if statement. If you only had one condition to check, you could opt to use only an if statement as shown below.

```
var catOwner = false;
if (catOwner = true) {
   alert("You own a cat!");
}
```

Since catOwner is a Boolean variable (true or false), we don't need to check if it's equal to true. The if statement above could also be written like this:

```
if(catOwner) {
   alert("You own a cat!");
}
```

Loops

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 (var x = 0; - we start from 0), the second part is where we go up to (x <= 10; - x must be less than or equal to 10), and the third part is what we increment our number by (x++ - increment our number by 1 each time).

```
for(var x = 0; x <=10; x++) {
        console.log(x);
}
```

Since var x = 0, we start with 0 as our value and check the second part of the loop to see if it's true. Since 0 is less than or equal to 10 (evaluates to true), we will go inside the for loop and execute the code inside of the curly brackets. After we execute this code, we return to the loop and increase x by 1 (x++). 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.

Functions

Normally, code in JS is executed once the browser reaches that line of code. However, functions allow us to run specific code as it is needed. 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.

Another example of a function is shown below. We create a variable called likesCats and assign it a value of true. We then create a function called catLikerChecker which takes in one parameter called "feelings." Inside of this function, we use an if/else statement to use if the parameter provided is true. If it is, we alert the user to say "You like cats!!" and if it's false, we output "You don't like cats!!"

After creating the function, we call/reference the function by its name. We provide the value of "likesCats" as an argument and since it's true, "You like cats!!" would be the output.

```
var likesCats = true;
var catLikerChecker = function(feelings) {
        if (feelings = true) {
                alert("You like cats!!");
        } else {
                alert("You don't like cats!!");
        }
}
catLikerChecker(likesCats);
```

Arrays

Think of these as lists of any kinds of data (which can include other arrays). These data pieces are separated by commas and placed inside of square brackets. Every item inside of an array has an index number which is used to refer to that item. For example, we might have an array of house pets:

```
var housePets = [];
```

The one above is empty and has nothing in it.

var housePets = [‘cat’, ‘dog’, ‘bird’, ‘chinchilla’]

The above array has **four** items, but the index number only goes up to **three.** This is because the first item in any array has an index number of 0. The values are listed below:

housePets[0]; // Refers to cat
housePets[1]; // dog
housePets[2]; // bird
housePets[3]; // chinchilla

You can change values at any point by referring to the item and then assigning it a new value:

housePet[3] = “fish”;

The array would now be housePets[‘cat’, ‘dog’, ‘bird’, ‘fish’]

Like strings, arrays have their own set of methods/functions which can be used on them.

Using housePets.length; would produce 4 for our case.

We can also use things like pop() or push() to remove or add items to our array.

housePets.pop(); // Removes last item in our array – becomes cat, dog, bird
housePets.push(‘cow’); // Adds cow to the end of our array – becomes cat, dog, bird, cow

Overall, you should think of a variable as a box which holds a value. An array is like a box which contains many other boxes inside of it.

Objects
These are things in JS that have their own properties and methods which can be called upon using something called **dot syntax**. Dot syntax is used between objects and functions so that your code knows what to do.

For example, if you have a car, that car has multiple characteristics and things it can perform. We can create an object for a car with the following code:

```
var myCar = {
        name: “Herbie”;
        owner: “none”;
        age: 58;
        honk: function() {alert(“HONK HONK!”);}
};
```

All you need to do is say var variableName = {} and place all methods/properties inside of the curly brackets. Name, owner, and age are all properties while honk is a method. We separate

these with a colon but refer to them as key/value pairs. To get these values and use them, we use **dot syntax** which utilizes a period and the property/method name.

console.log(myCar.name); // Refers to the name key but logs the value "Herbie" to the console
myCar.honk(); // Would cause an alert to appear with "HONK HONK!"
myCar.age; // Refers to the age key and the value of 57

Objects in JS are powerful because almost everything is an object of some sort. Arrays, strings, and other types all have their own properties or methods (length, concat, push, pop, splice, etc.).

We have already worked with objects before. One example we have already seen is the console object. We can use the "log" method/function to create output in the Console.

console.log("Hello World!");

Console is the object we are using and then we place a period after it to signify that we are going to reference some method/function or property. In our example, we are using the log **function** and then providing "Hello World." as the **value** for that function.

Similarly, we can do the same with strings or any other object in JS. In the example below, we create a string with a value of "Chris." After this we create another variable which uses our previously defined string (myName) and the length **property** to get the length of that string. Just like with console or other objects, we use a period to separate the object and the method/function or property.

var myName = "Chris"; // Creates a string with a value of Chris
var myNameLength = myName.length;

Working with the DOM
If you've looked through the JS examples provided in the course, you may have seen that you can use JS to execute functions and create basic output on our page. The main method used in the course examples has been document.getElementById(). In the example below, I have a button with an id of btn. I can trigger some sort of an event (dialog box popup perhaps) when it's clicked by using this method paired with the unique id of the button.

HTML
```
<button id="btn" type="submit">Click Me!</button>
```

JS
```
document.getElementById("btn").onclick = function() {
        // code to execute in here
}
```

Using the document object's "getElementById" method, we can target any **one** HTML element based on its id value. Each id value on our page must be unique and can't be repeated in other elements. This limits us to targeting only one element at a time with this method.

Another useful method is getElementsByClassName. Instead of using the id value of an HTML element, you would provide the class name of an element or multiple elements. In the example below, I select all elements with a class name of "para" and store them in a variable called paras.

```
var paras = document.getElementsByClassName("para");
```

When using this method, we create a collection of all the elements which match the class name provided. A collection is very similar to an array in that it holds multiple values. Let's say we had the following HTML code:

```
<p class="para">First sentence.</p>
<p class="para">Second sentence.</p>
<p class="para">Third sentence.</p>
```

If we use var paras = document.getElementsByClassName("para");, we would have a collection of all three paragraphs above. To refer to the first sentence, we could use paras[0] since the first item in a collection has an index value of 0 (just like arrays). The last paragraph could be referenced with paras[2] since the index values are 0, 1, and 2.

We could loop through each member of the collection with the code below:

```
var paras =  document.getElementsByClassName("para");,
for(var x = 0; x < paras.length; x++) {
        paras[x].style.color = "blue";
}
```

In the second part of the for loop, the condition we check before executing the code inside the loop (x < paras.length), we specify that the loop should only run up to but not including the length of the collection. Since the length of the collection is 3, x < paras.length means x < 2. This makes sense because if we refer to each item in our collection individually, it will look like the code below:

paras[0] – First sentence
paras[1] – Second sentence
paras[2] – Third sentence

During each execution of the loop, we change the elements to have a color of blue by using paras[x]. x just refers to the number which is 0 through 2 in our case.

Another method which can be used to select elements on our page is the querySelector() method. It is like the getElementById method in that it targets only **one** element on the page. However, instead of targeting something by its id value, we provide a CSS selector between the quotation marks. Any valid CSS selectors can be used.

In this example, I target anything with a class of "redPara" and give it a color of red. The querySelector method only targets the very first match and so this red color style wouldn't apply to the second paragraph tag.

```
<p class="redPara">This para will be <span>red<span>.</p>
```

This para will not be red.</p

document.querySelector(".redPara").style.color = "red";

If we only wanted to target the span, we could also do that using this method. We would just change the value provided to ".redPara span" so that it references all span tags inside of anything with a class of "redPara."

The last function we will cover is querySelectorAll(). Just like querySelector, it uses a provided CSS selector to target elements. The main difference is that a list of items is returned just like with getElementsByClassName(). We will reuse the code from the querySelector example as demonstrated below.

This paragraph will be red.
This paragraph will be red.

var redParas = document.querySelectorAll(".redPara span");

Once we create a variable to represent the list of all items which match our CSS selector (span tags inside of elements with the redPara class), we can loop through it to change the style for each individual element.

```
for(var x = 0; x < redParas.length; x++) {
        redParas[x].style.color = "red";
}
```

After executing this code, all span tags within those paragraphs would have that red color applied. We could also refer to just one of those elements inside of the list by referring to the index value (same as arrays or collections).

redParas[0].style.color = "red"; - Only the very first paragraph in our list gets styled with this

Events in JavaScript
Most features built into web pages that use JavaScript rely on some sort of event. For example, if there is a drop-down menu which only appears after some element is clicked, we could say that there is a click event being used.

Another example of an event can be found on Twitter or some other platforms which limit characters with posts. As you type into an input box for a status update, there might be a counter present which tells you how many characters you have typed or have left. In this instance, there is a keypress event being used.

Some of the most popular JavaScript events are listed below:

- Click – A user clicks on an element
- Dblclick – A user double-clicks on an element
- Mouseover – When the pointer is moved onto an element or one of its children
- Keypress – When a user presses a key

    - Keydown – When the user is pressing a key
    - Keyup – When a user releases a key
- Onload – When an HTML object has loaded
- Focus – When an element gets focus

To add an event to an element, you can use the on event syntax as demonstrated below. In this example, we have another button with a class of "btn." We target that button with querySelector and then add a "onclick" event to it.

```
<button class="btn" type="submit">Click me</button>

document.querySelector(".btn").onclick = function () {
        //Some code in here
}
```

We could change "onclick" to "onmouseover", "ondblclick" and other events as we want. After we specify the event to trigger our function, we define the function which is execute once the event is triggered. We would do the same thing for all other events.

Another way to trigger events is by adding an event listener to the element. We could create a variable to store the element and then add an event listener to it to execute some code.

```
<button id="btn" type="submit">Say Hi!</button>
function sayHi() {
        alert("Hi!");
}
var myButton = document.getElementById("btn");
myButton.addEventListener("click", sayHi);
```

The first parameter for the addEventListener() function is the type of event. We use a click event in the example above. We also declare a function called "sayHi" which creates an alert. The second parameter of addEventListener() is the function to execute when the event is triggered. In our case, we use the sayHi() function.

Getting User Input

Most websites have some sort of form which takes in user input. This input may be used to change the view on a website or add data to an existing database. We can use input tags with JavaScript to retrieve a user's input and then manipulate it. In the example below, I have input and button tags with unique ids on them. I use JavaScript to add a click function to the button.

Inside of the function, I create a variable called firstName and assign it the value of the input box's value. To do so, you can use document.getElementById().value. The .value property gets the value of the input box. This would go inside of the function because you want to get the value after the button is clicked. If you get the value before it's clicked (such as when the page loads), the value will be empty.

After I get this value, I can do whatever I want with it. In my example, I simply use an alert to output the name provided. You could also do things like check its length, see if the value was empty, etc. The course examples for JavaScript cover this example in more detail.

```
<input id="firstName" type="text">
<button id="submitName" type="submit">Submit Name</button>

document.getElementById("submitName").onclick = function() {
        var firstName = document.getElementById("firstName").value;
        alert(firstName);
}
```

Class Examples
All the links below point to pages that I created to give you an example of the concepts mentioned above. Feel free to right-click the page and view the source. You can then click the link to the JS file to see my code which has comments placed inside of it. Some code may be inside of the HTML file as well.

All JavaScript examples have been released and are listed below. If you'd like for me to create another example, please request one via email. I can create more examples as needed, but these should cover everything needed for this course.

- https://www.albany.edu/~cv762525/cinf362/examples/js-linking-example.html
- https://www.albany.edu/~cv762525/cinf362/examples/js-concepts-example.html
- https://www.albany.edu/~cv762525/cinf362/examples/js-concepts-example-2.html
- https://www.albany.edu/~cv762525/cinf362/examples/page-output-example.html
- https://www.albany.edu/~cv762525/cinf362/examples/dom-manipulation.html

Here's a link to all course examples:
https://www.albany.edu/~cv762525/cinf362/examples/ (coded examples)
https://www.albany.edu/~cv762525/cinf362/videos/ (videos)

There is also content on Blackboard to help you with understanding/debugging JavaScript code. These items are located on Blackboard → Resources → JS and will help further explain JavaScript and demonstrate some best practices you should adopt moving forward.

Online Resources (If you don't like my content, here are some alternatives)

- https://htmldog.com/guides/javascript/beginner/makingstuffhappen/ (connecting JS)
- 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://www.w3schools.com/js/js_htmldom.asp
- https://developer.mozilla.org/en-US/docs/Learn/JavaScript/Building_blocks/Events
- https://www.tutorialspoint.com/javascript/javascript_events.htm
- https://eloquentjavascript.net/ (Free eBook on JavaScript - super comprehensive)
- https://www.w3schools.com/js/default.asp (Tutorials on everything)

- https://www.javascript.com/ (More tutorials with examples)

Before completing the assignment(s) for this week, please read the "Viewing Your Web Pages.docx" file on Blackboard. You will not be able to submit anything for the assignment without completing that portion first. It is located directly in the Lecture Notes folder.

JS Challenges

Due Sunday, March 6th at midnight

Download the "JS-Challenges.zip" folder from Blackboard under this week's Lecture Notes or the Assignment folder. Inside of the zip folder will be an HTML file to use for this assignment.

Your task is to link to an **external JavaScript file** and then add relevant HTML/JS to complete the challenges listed below. Helpful code can be found in this week's JavaScript example page.

The JS and HTML are not linked for you already. You will need to add a reference to an external script file. To do so, refer to the class notes above about linking to an external JavaScript file (https://www.albany.edu/~cv762525/cinf362/examples/js-linking-example.html). All your JS should go inside of this externally placed file. In assignments after this one, the JS file will be provided/linked for you already.

You will need to add HTML forms to the assignment file. If you are not familiar with them, review the links below for some information:

- https://htmldog.com/guides/html/beginner/forms/
- https://htmldog.com/guides/html/advanced/forms/

Challenges (4 total)

1. Create a form which uses three number inputs and one button. When the button is clicked, the following information should be output to the page.
    a. What numbers were used for the function
    b. The sum of those numbers
    c. The average of those numbers

    Input Example:
    3, 5, and 7
    Example Output:
    "The numbers 3, 5, and 7 were provided. The total of these numbers is 15 and the average is 5."
    For this challenge, you'll need to use the string concatenation operator in addition to the addition and division operators. You'll need to use an on click event for a button and attach a function to it.
2. Create a form which uses one input and one button. When the button for this form is clicked, it should output the following information about their name to the page:
    a. What their name is
    b. How many letters long it is
    c. If it is a short, average, or long name

    Input Example:

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.

3. Create an array called "animals" with 3 animals inside of it. Create a form with one input and one button. This input should take in a new animal name. When the user clicks your button, the animal should get added to your "animals" array, and the array should be output to the page in a list. For this challenge, you'll need to use a click event on the button. After the button is clicked, get the value of the input using .value. From there, you can use the push method to add the retrieved value to the array. After it has been added to the array, you should loop through the list to output each array item as a list item for either an ordered or unordered list.
   I would recommend creating an empty ul tag and giving it a unique id. You can then refer to this ul tag with JS to output your animals. This is demonstrated in the coded examples for the class.
   Example Output (after I type horse in my form – cat, dog, and duck in my array)
   - Cat
   - Dog
   - Duck
   - Horse
4. Create an object called myInfo with the following three properties/methods inside of it: favColor, age, and favQuote. favColor should have a string with your favorite color in it. Age should have a number with your age. favQuote should have a function in it which will create an alert with your favorite quote inside of it.

   After creating this object, use an alert to output your favorite color and tell the user how many years it will be until you are 100.

   Example Output
   "My favorite color is blue, and in 70 years, I will be 100."

   After this alert, refer to your favQuote method for the myInfo object so that another alert is generated with your favorite quote.

   Example Output (after I refer to the favQuote method)
   "Be who you are and say what you feel, because those that mind don't matter, and those that matter don't mind."

If your JavaScript isn't working at all, it's probably because the src value you've provided in the script tag is incorrect. If some JavaScript works, but other parts don't, look at your JavaScript in the console. Inspect the page and then go to the console tab. Any lines of code with an error will appear there. The error is usually a misspelling or incorrect reference to something on the page.

Your webpage is **due on Sunday, March 6th at midnight.** To submit the work for this exercise, visit Blackboard → Course Materials → Lectures Notes for this week's class. You can also go into the "Assignments" folder and the submission area will be titled "JS Challenges." You should be submitting a live link to me; it can be through the UAlbany server or 000webhost. The work submitted will be evaluated based on the rubric explained below.

JS Challenges Rubric – 10pts

- Live link submitted – 1pt
- JS is externally placed and organized/easy to read – 1pt
- HTML is valid – 2pts
- Each challenge is complete – 6pts total or 1.5pts for each challenge

JavaScript Features (Two Posts)

Initial Post due Thursday, March 3rd

Many websites use client-side and server-side scripts in to add features and enhance website usability. Server-side scripts typically interact with databases and make bigger changes to pages which are saved over time. Client-side scripts typically affect things on a less permanent basis and are used to help a person interact with a page (image carousels, drop down menus, resizing images, etc.).

Think about when you sign up for a service. If you click a button which makes another part of a form appear, that is most likely a client-side script. When you finally try to sign up for the service by submitting your info, the page might interact with a database to ensure your chosen username isn't already taken. That could be an example of a server-side script.

Visit a few websites which contain advanced features for users and observe how they respond to user input. In your **initial post,** discuss the following items and mention the name(s) of the website(s) you visited:

If you input content into a form, how does the input affect the content on that page or other pages? Do you think the content or features are made possible with client or server-side scripts? Do you think a database may be involved? Think about the content itself to determine if a database may be involved. What are some of the most popular JS features built into pages based on what you're seeing? What are your favorite features you've seen on websites? What about your least favorite? (autoplay on videos, random popups, etc.) Do those features use client or server-side scripts?

You can see if a page has client-side code by viewing the page source and checking for script tags. Server-side scripts can be detected sometimes by the page's filename extension such as .php, .asp, or .aspx. If the URL contains variables and values separated by "&" and "=", the page is probably using some type of server-side scripting. Check out a few websites to get a good idea of common features or filename extensions.

The initial post is **due Thursday, March 3rd at midnight**. To submit, go to Blackboard → Course Materials → Lecture Notes for this week's class. There will be a discussion area called "JavaScript Features" where you can post your initial post. You can also visit the Discussion Board area directly from the Course Materials folder.

Response Post due Sunday, February 27th
In your response post, visit the websites mentioned by another student and compare them to your own. Were there any similar features being used? Did you notice any client or server-side scripts that they didn't? If there's a database involved, what pieces of information are being captured or used? What were some of the best features you saw? Were there any bad/inconvenient ones?

The response post is **due Sunday, March 6th at midnight**. To submit, go to Blackboard → Course Materials → Lecture Notes for this week's class. There will be a discussion area called "JavaScript Features" where you can post your response post. You can also visit the Discussion Board area directly from the Course Materials folder.

JavaScript Features Rubric – 2pts
The initial post is worth 1.5pts and the response post is worth .5pt for a total of 2 points. I will be evaluating your posts based on the following criteria:

- Did you mention the websites you looked at?
- Were specific features in each the websites mentioned?
- Did your response post contribute to the original post?
    - Avoid summarizing the other person's post or simply saying that it was a good post.
    - Did you mention a specific feature on the other student's website?

Next Week

Next week, we will dive deeper into JS and some of the more complicated things it can accomplish. Specifically, we'll cover topics such as cookies and AJAX. These allow us to work with data in a user's browser or an external location on a server.