14-Intro-To-JavaScript

**JavaScript** is used to handle programming logic and give websites interactivity.

Responsible for its behavior

we start JavaScript (JS).

JS lends itself to a “**functional programming**” paradigm;

a coding paradigm where you can think about functions as first class objects.

This means functions can be passed around as values, functions can pass into other functions as parameters, and one function can return another function.

Deals w immutable data structures

get introduced to these concepts through Python.

There are three functions that facilitate this functional approach to programming:

map, reduce, filter.

There is a short video series that dives into these topics. I’ve organized the videos and the examples worked out into a notebook file posted below.

<https://github.com/the-Coding-Boot-Camp-at-UT/UTAUS201807DATA2/blob/master/class-content/00-ExtraMaterial/FunctionalProgramming/00-FunctionalPrograming-InPython.ipynb>

So why/what functional programming in Python?

The benefits are that we can parallelize data processing in a scalable and readable way.

“Big Data” tools like Spark/Hadoop also have this map/reduce style of processing data.

Checkout out the first video of the Python series on functional programming to learn more

<https://www.youtube.com/watch?v=xJCPpDlk9_w&list=PLP8GkvaIxJP1z5bu4NX_bFrEInBkAgTMr>

10/8/18 day 1 – Manuel

**### Class Objectives**

\* gain familiarity with JavaScript variables, data types, and statements.

\* gain familiarity with basic JavaScript control flow

(functions, loops, if/else statements).

\* gain familiarity with JavaScript arrays.

\* gain familiarity with using and creating functions in JavaScript, including built-in functions.

Todays Objectives

\* Intro to JavaScript

Today's class will introduce students to the basic syntax of JavaScript.

\* Compare JS syntax to Python

\* JavaScript data types:

- Variables

- Arrays

- Functions

- Objects

Resources:

\* Document Object Model (DOM)

- <https://www.tutorialspoint.com/javascript/javascript_w3c_dom.htm>

\* JavaScript tools

- <https://github.com/the-Coding-Boot-Camp-at-UT/UTAUS201807DATA2/blob/master/class-content/00-ExtraMaterial/JavaScript/JavaScript-Tools.ipynb>

\* Functional Programming in Python

- <https://github.com/the-Coding-Boot-Camp-at-UT/UTAUS201807DATA2/blob/master/class-content/00-ExtraMaterial/FunctionalProgramming/00-FunctionalPrograming-InPython.ipynb>

\* Data Visualization Libraries

- D3 based reusable chart library: <https://c3js.org/samples/simple_multiple.html>

- Plotly, built on D3: <https://plot.ly/javascript/>

www.tutorialspoint.com

JavaScript The W3C DOM - Learn Javascript in simple and easy steps. A beginner's tutorial containing complete knowledge of Javascript Syntax Objects Embedding with HTML Validations Cookies Regular Expressions Literals Variables Loops Conditions.

c3js.org

C3.js | D3-based reusable chart library

D3 based reusable chart library

plot.ly

A free open source interactive javascript graphing library. Plotly.js is built on d3.js and webgl and supports over 20 types of interactive charts.

JavaScript is the language of the web. JavaScript…

\* makes it possible to create interactive web pages and visualizations.

\* JavaScript is often used to place API calls to cloud data and services.

\* JavaScript also allows websites to send and receive data from a server, respond to a user's actions on the page, and dynamically modify HTML elements.

\* With JavaScript, it is possible to build interactive sites that do not require the use of the command line interface.

For example, data analytics students from this program have used JavaScript to build internal tools and even client-facing dashboards.

\* JavaScript is everywhere. Even machine learning has been made available for the web browser: [https://js.tensorflow.org/]

\* Bottom line: it is a very marketable and in-demand skill for any data position.

When paired with HTML and CSS,

JavaScript is a programming language that allows developers to add complexity and interactivity (such as animation or dynamic updates) to websites.

**\*\*Note\*\***: JavaScript styling guidelines recommend 4 spaces when indenting code

blocks and CamelCase when naming variables and functions.

<https://www.tutorialspoint.com/javascript/images/html-dom.jpg>

<https://stackoverflow.com/questions/748175/asynchronous-vs-synchronous-execution-what-does-it-really-mean>

The **internet of things**, or **IoT**, is a system of interrelated computing devices, mechanical and digital machines, objects, animals or people that are provided with unique identifiers (UIDs) and the ability to transfer data over a network without requiring human-to-human or human-to-computer interaction.

1.1-Ins\_JavaScript

script-within.html

<!DOCTYPE html>

<html lang="en">

<head>

<title></title>

<meta charset="UTF-8">

<meta name="viewport" content="width=device-width, initial-scale=1">

</head>

<body>

<h1>Open the Chrome Inspector Console!</h1>

<!-- The JavaScript code is placed between a pair of `script` tags inside of the HTML file.

This means the code will run when the webpage has loaded. -->

<script type="text/javascript">

console.log("My script is stored within the HTML!")

</script>

<!-- The `console.log()` function allows the developer to print out a message

to a web browser's built-in console.

This message can be viewed by opening up the inspector & navigating into the console tab. -->

</body>

</html>

script-outside.html

<!DOCTYPE html>

<html lang="en">

<head>

<title></title>

<meta charset="UTF-8">

<meta name="viewport" content="width=device-width, initial-scale=1">

</head>

<body>

<h1>Open the Chrome Inspector Console!</h1>

<!-- src contains the path to the file -->

<script type="text/javascript" src="app.js"></script>

</body>

</html>

<!-- using external JavaScript files is far more common than writing JavaScript

into the HTML itself since this allows developers to easily reuse code. -->

app.js

// This code will run when linked to in HTML

console.log("My script is stored outside of the HTML!");

JavaScript and Python are logically and syntactically similar

python uses snake\_case vs. js using CamelCase ?

Just like in Python,

JavaScript will automatically determine the data type assigned to a variable.

Every line in JavaScript ends with a semicolon;

While this is not technically a requirement for the language, JavaScript programmers conventionally use a semi-colon to conclude a statement.

In Python, Booleans are Capitalized. In JavaScript, they are lowercase.

1.2-Evr\_Python\_to\_JavaScript

.html file

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

<meta name="viewport" content="width=device-width, initial-scale=1.0">

<meta http-equiv="X-UA-Compatible" content="ie=edge">

<title>Python to JavaScript</title>

</head>

<body>

<h1>Open the Chrome Inspector Console!</h1>

<script src="hello-variable-world.js"></script>

</body>

</html>

// All variables in JavaScript must be initialized using the `var <Variable Name> = <Value>`

// syntax. This is in contrast to Python, where variables can be declared without the `var` keyword.

name = "Homer Simpson" vs. below...

// Create a variable called "name" that holds a string

var name = "Homer Simpson";

// Create a variable called "country" that holds a string

var country = "United States";

// Create a variable called "age" that holds an integer

var age = 26;

// Create a variable called "hourlyWage" that holds an integer

var hourlyWage = 15;

// Calculate the "dailyWage" for the user

var dailyWage = hourlyWage \* 8;

// Create a variable that holds a number as a string

var weeklyHours = "40";

// Create a variable called 'weeklyWage' that converts a string into an integer

var weeklyWage = hourlyWage \* parseInt(weeklyHours);

// vs. python

weekly\_wage = hourly\_wage \* int(weekly\_hours)

// A similar operation can be performed with `parseFloat()` for decimals.

// Create a variable called "satisfied" that holds a boolean

var satisfied = true;

// Print out "Hello <name>!" string template literal

console.log(`Hello ${name}!`);

// vs. python f-string

print(f"Hello, {name}!")

// Print out what country the user entered

console.log(`You live in ${country}.`);

// Print out the user's age

console.log(`You are ${age} years old.`);

// Print out the daily wage that was calculated

console.log(`You make ${dailyWage} dollars per day.`);

// Print out the weekly wage that was calculated

console.log(`You make ${weeklyWage} dollars per week.`);

// Using an IF statement to print out whether the users were satisfied

if (satisfied === true) {

console.log("You are satisfied with your pay.");}

else {

console.log("You are not satisfied with your pay.");}

// vs. python

if satisfied:

print("You are satisfied with your pay.")

else:

print("You are not satisfied with your pay.")

conditional checks…

var x = 1;

var y = 10;

// Checks if one value is equal to another... JS uses `===` to denote strict equality.

if (x === 1) {

console.log("x is equal to 1");

}

// vs. python

if x == 1:

print("x is equal to 1")

// JavaScript uses curly braces to define blocks of code.

// This is equivalent to how Python uses whitespace & indentation to define it.

// Checks if one value is NOT equal to another

if (y !== 1) {

console.log("y is not equal to 1");

}

// Checks if one value is less than another

if (x < y) {

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

}

// Checks if one value is greater than another

if (y > x) {

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

}

// Checks if a value is greater than or equal to another

if (x >= 1) {

console.log("x is greater than or equal to 1");

}

// Checks for two conditions to be met using &&

if (x === 1 && y === 10) {

console.log("Both values returned true");

}

// Python combines boolean expressions using logical statements such as `and` and `or`. The JS equivalent of `and` is `&&` while the `or` equivalent is `||`.

// Checks if either of two conditions is met using ||

if (x < 45 || y < 5) {

console.log("One or the other statements were true");

}

// JavaScript uses `if`...`else if`...`else` to chain conditionals.

// This is equivalent to `if`...`elif`...`else` in Python.

if (y < 5) {

console.log("x is less than 10 and y is less than 5");

}

else if (y === 5) {

console.log("x is less than 10 and y is equal to 5");

}

else {

console.log("x is less than 10 and y is greater than 5");

}

// Nested if statements

if (x < 10) {

if (y < 5) {

console.log("x is less than 10 and y is less than 5");

}

else if (y === 5) {

console.log("x is less than 10 and y is equal to 5");

}

else {

console.log("x is less than 10 and y is greater than 5");

}}

1.3-Par\_Loan\_Approver

The solution uses in-document script tags. This is ok for short scripts, but it is

best practice to separate the HTML and JavaScript code into their own files.

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

<meta name="viewport" content="width=device-width, initial-scale=1.0">

<meta http-equiv="X-UA-Compatible" content="ie=edge">

<title>Loan Approver</title>

</head>

<body>

<h1>Open the Chrome Inspector Console!</h1>

<script>

// @NOTE: change these to test the code with different values

var income = 25;

var debt\_income\_ratio = 0.5;

var jobyears = 3;

var criminalrecord = false;

if ((income<30 && debt\_income\_ratio<0.5) || (income>=30 && income<75 && jobyears<1) || (income=>75 && criminalrecord)) console.log("No loan")

else if ((income<30 && debt\_income\_ratio>0.5) || (income>=30 && income<75 && jobyears>1) || (income=>75 && !criminalrecord)) console.log("Loan")

</script>

</body>

</html>

bonus.html

<script>

// @NOTE: change these to test the code with different values

var income = 25000;

var debtIncomeRatio = .5;

var yearsInJob = 3;

var criminalRecord = false;

var goodCredit = true;

if (income < 30000) {

if (debtIncomeRatio < .5) {

console.log("No Loan");}

else {

console.log("Loan");}}

else if(income >= 30000 && income < 75000) {

if (yearsInJob < 1) {

console.log("No Loan");}

else if(yearsInJob >1 && yearsInJob < 5) {

if (goodCredit) {

console.log("Loan");}

else {

console.log("No Loan");

}}

else {

console.log("Loan");}}

else {

if (criminalRecord) {

console.log("No Loan");

}

else {

console.log("Loan");

}}

</script>

1.4-Evr\_JavaScript\_Arrays

arrays in JavaScript, somewhat parallel to Python lists

\* JavaScript arrays, like Python lists, hold items in an ordered fashion.

\* Arrays are mutable: it is possible to add items to an array.

\* Arrays can hold items of different data types, such as integers and strings.

They can even hold other arrays.

\* Like Python, an element in an array can be accessed by its **\*\*index\*\***:

arrays.js called in index.html (not shown here)

// A JavaScript array is much like a Python list

// Here, start with a blank array

var lettersArray = ["a", "b", "c", "d"];

// Display the array in console

console.log("An array of letters:");

console.log(lettersArray);

// Use indexing to access an array item

console.log("Use indexing to access an array item:");

var firstLetter = lettersArray[0];

var secondLetter = lettersArray[1];

console.log(firstLetter);

console.log(secondLetter);

In Python, the `append()` method is used to add an item to a list.

In JavaScript, the **\*\*`push()`\*\*** method is used:

// Use push() to append an item to an array

lettersArray.push("e");

lettersArray.push("f");

console.log("Use push() to append an item to an array:");

console.log(lettersArray);

console.log("==========");

// Use slice() to return selected items of an array…a portion of it

console.log("Use slice() to return selected items of an array");

var slicedArray1 = lettersArray.slice(1);

// Return the first three items of an array

var slicedArray2 = lettersArray.slice(0, 3);

// Return the second and third items of an array

var slicedArray3 = lettersArray.slice(1, 3);

console.log(slicedArray1);

console.log(slicedArray2);

console.log(slicedArray3);

console.log("==========");

// Use join() to return items of an array into a single string

var joinedArray = lettersArray.join(", ");

console.log("Use join() to return items of an array into a single string:");

console.log(joinedArray);

// anotherJoinedArray = lettersArray.join("\*\*\*");

// console.log(anotherJoinedArray);

console.log("==========");

// A JavaScript string, like their Python counterparts, are indexed:

var soundOfMusic = "The hills are alive with the sound of music";

console.log("This is a string:");

console.log(soundOfMusic);

// Use indexing to access a string character

console.log("Use indexing to access a string character:");

console.log(soundOfMusic[0]);

console.log(soundOfMusic[5]);

// Split a string into an array of substrings

// Here, split the string where spaces are found

var soundArray = soundOfMusic.split(" ");

console.log("Use split() to split a string into an array of substrings:");

console.log(soundArray);

An *integrated development environment* (*IDE*) is a software application that provides comprehensive facilities to computer programmers… visual studio code, atom, sublime text

1.5-Ins\_Loops

// Prototypical use case increments loop counter by one on each iteration

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

console.log("Iteration #", i);

}

// First, a variable `i` is used to control the number of loops.

// This typically is initialized to `0`, but it can be set to any starting value.

// Next, a conditional expression is used to determine when to stop the loop iteration.

// In this example, `i` will loop from 0 to 9.

// Finally, an expression is used to either increment

// or decrement the value of `i` at the end of each loop.

// The body of the `for` loop is contained in the curly braces.

// Everything inside of the body will be executed once per loop.

// Looping through an array

var students = ["Johnny", "Tyler", "Bodhi", "Pappas"];

for (var j = 0; j < students.length; j++) {

console.log(students[j]);

}

1.6-Stu\_Movie\_Scores

\* Check the [documentation](https://www.w3schools.com/jsref/jsref\_length\_array.asp) to

find the length of the array.

// Array of movie ratings

var movieScores = [

4.4, 3.3, 5.9, 8.8, 1.2, 5.2, 7.4, 7.5, 7.2, 9.7, 4.2, 6.9];

// Starting a rating count

var sum = 0;

// Arrays to hold movie scores

var goodMovieScores = [];

var okMovieScores = [];

var badMovieScores = [];

// Use a for loop to iterate through the movie scores

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

// Add each score to the ratings count...equivalent to `sum = sum + score;`.

var score = movieScores[i];

sum += score;

// If the movie's rating is greater than 7, add it to the list of good movies

if (score > 7) {

goodMovieScores.push(score);

}

// If the movie's rating is between 5 and 7, add it to the list of "Ok" movies

else if (score <= 7 && score > 5) {

okMovieScores.push(score);

}

// Otherwise, if the movie's rating is less than or equal to 5, add it to the list of bad movies

else {

badMovieScores.push(score);

}

}

// Find the average score

var avg = sum / movieScores.length;

// Store the length of movie ratings

var numGoodMovies = goodMovieScores.length;

var numOkMovies = okMovieScores.length;

var numBadMovies = badMovieScores.length;

// Print results

console.log("---------");

console.log(`There are ${numGoodMovies} good movies.`);

console.log(`There are ${numOkMovies} ok movies.`);

console.log(`There are ${numBadMovies} bad movies.`);

console.log(`The average movie rating is ${avg}.`);

console.log("---------");

2.10-Stu\_Movie\_Score\_Refactored

// Array of movie ratings

var movieScores = [

4.4, 3.3, 5.9, 8.8, 1.2, 5.2, 7.4, 7.5, 7.2, 9.7, 4.2, 6.9];

// Starting a rating count

function addScores(runningSum, currentValue){

return runningSum + currentValue

}

var sum = movieScores.reduce(addScores, 0)

// Arrays to hold movie scores

var goodMovieScores = movieScores.filter(function(score){

return score>7

})

var okMovieScores = movieScores.filter(function(score){

return score >= 5 && score <= 7

})

var badMovieScores = movieScores.filter(function(score){

return score < 5

})

End of 1.6 repeated…

1.7-Ins\_Functions

Functions in JavaScript are designed for certain tasks and are executed when called.

They are defined by the `function` keyword, followed by the function name, and completed with parentheses.

Parameters may be included within the parentheses, separated by commas.

The code executed when the function is called is placed within curly brackets.

// In Python, functions are declared with the `def` keyword, # Simple print statement

def print\_hello():

print("Hello there!")

// then finished by adding a colon with indented code written underneath.

// while JavaScript used `function`. // Simple log statement

function printHello() {

console.log("Hello there!");

}

// Run the code in the `printHello` function

printHello();

// Takes two numbers and adds them

function addition(a, b) {

return a + b;

}

// Log results of addition function

console.log(addition(44, 50));

// Functions can call other functions

function doubleAddition(c, d) {

var total = addition(c, d) \* 2;

return total;

}

// Log results of doubleAddition function

console.log(doubleAddition(3, 4));

// This function accepts a parameter and iterates through an array

function listLoop(userList) {

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

console.log(userList[i]);

}

}

var friends = ["Sarah", "Greg", "Cindy", "Jeff"];

listLoop(friends);

// Javascript built in functions

var longDecimal = 112.34534454;

var roundedDecimal = Math.floor(longDecimal);

console.log(roundedDecimal);

see functions.py for more python comparisons

1.8-Stu\_Stats\_Functions

\* [variance](<https://stats.stackexchange.com/questions/212650/variance->

explanation.html)

\* [standard deviation](<https://www.mathsisfun.com/data/standard-deviation.html>)

// Array of movie ratings

var movieScore = [4.4, 3.3, 5.9, 8.8, 1.2, 5.2, 7.4, 7.5, 7.2, 9.7, 4.2, 6.9];

// a function called `mean` is created that accepts an array as an argument.

// This function iterates over the array, sums the values, then divides by the length of the array.

function mean(arr) {

var total = 0;

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

total += arr[i];

}

var meanValue = total / arr.length;

return meanValue;

}

// Variance can be found by subtracting the mean from each number in the data set,

// squaring the result, and then averaging the square differences.

function variance(arr) {

var meanValue = mean(arr);

var total = 0;

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

total += (arr[i] - meanValue) \*\* 2;

}

var varianceValue = total / arr.length;

return varianceValue;

}

// Standard deviation is the square root of the variance

function standardDeviation(arr) {

var varianceValue = variance(arr);

var standardDeviationValue = Math.sqrt(varianceValue);

return standardDeviationValue;

}

console.log("Movie Statistical Analysis");

console.log("--------------------------");

console.log(`The mean is : ${mean(movieScore)}`);

console.log(`The variance is : ${variance(movieScore)}`);

console.log(`The standard deviation is : ${standardDeviation(movieScore)}`);

console.log("");

Note: for many of these, I have saved just the index.js here,

which was called in the index.html file

**10/10/18 – day 2 Todays Objectives - Manuel**

Today's class will take a deeper look into JS through Objects, ES6, & tables,

performing more sophisticated operations and data manipulation, getting exposure to modern JavaScript programming techniques and best practices.

functional programming, and data munging (modify incrementally) in JavaScript.

\* JavaScript objects

\* JavaScript array functions

- forEach

- map

- reduce

- filter

\* JavaScript Arrow Function

**### Class Objectives**

\* understand and be able to use forEach and callback functions and how to use them.

\* create, update, and iterate JavaScript Objects.

\* understand how to apply map and filter to parse data.

\* be able to create and use arrow functions to simplify code.

\* understand the basic structure of a Bootstrap HTML table.

2.1-Ins\_JavaScript\_Objects… index.js is called index.html (not shown here)

// A JavaScript object is similar to a Python dictionary

// Python dictionaries: \*\*organize\*\* information in `key` and `value` pairings.

// Unlike lists, key-value pairs are \*\*unordered\*\*.

// The `key` is used to \*\*access\*\* the `value`.

var movie = {

name: "Star Wars",

year: 1977,

profitable: true,

sequels: [5, 6, 1, 2, 3, "The Last Jedi"]};

// JavaScript also allows value lookup via dot notation

console.log(movie.name);

console.log(movie.year);

console.log(movie.sequels[0]);

// JS also allows value lookup via bracket notation--note the similarity to Python

console.log(movie["name"]);

// Add a key-value pair to an existing object...much like in Python,

// a property can be added to a JS object simply by specifying a key and assigning a value to it:

movie.rating = 8.5;

console.log(movie);

// Delete a key-value pair

delete movie.sequels;

console.log(movie);

// Check whether a key exists in an object

if ("rating" in movie) {

console.log("This movie has a rating!");}

// Built-in object methods in JavaScript...An array of objects

var people = {

mom: "wilma flintstone",

dad: "fred flintstone",

daughter: "pebbles",

son: "bambam"};

// Display the entire object, both keys and values

console.log(people);

// Display only the keys of the object

console.log(Object.keys(people));

// Display only the values of the object

console.log(Object.values(people));

// Display a key-value pair held in an array

console.log(Object.entries(people));

2.2-Stu\_Word\_Counter

function wordCount(myString) {

// Convert string to an array of words

var stringArray = myString.split(" ");

// An object to hold word frequency

var wordFrequency = {};

// Iterate through the array

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

var currentWord = stringArray[i];

// If the word has been seen before...

if (currentWord in wordFrequency) {

// Add one to the counter

wordFrequency[currentWord] += 1;}

else {

// Set the counter at 1

wordFrequency[currentWord] = 1;} }

console.log(wordFrequency);

return wordFrequency; }

wordCount("I yam what I yam and always will be what I yam");

While both loop over an array, a `for loop` will repeat until a specified condition is

met, while `forEach` will call a function once for each element.

2.3-Ins\_forEach

// Array of student names

var students = ["Johnny", "Tyler", "Bodhi", "Pappas"];

// This function will be called for each element in the array

function printName(name) {

console.log(name);}

// Loop through each student name and call the printName function

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

printName(students[i]);}

// `forEach` automatically iterates (loops) through each item and

// calls the supplied function for that item, and performs an action on each element in the array.

// This is equivalent to the for loop above.

students.forEach(printName);

// You can also define an anonymous function inline

students.forEach(function(name) {

console.log(name);});

// update the value

var list = [{a: "1", b: "10"},

{a: "2", b: "20"},

{a: "3", b:"30"}]

// casts the string values to a number

list.forEach(function(data){

data.a = +data.a

data.b = +data.b})

2.11-Ins\_Map

introduces a powerful tool within the JavaScript arsenal, functional programming.

var theStagesOfJS = ["confidence", "sadness", "confusion", "realization", "debugging", "satisfaction"];

// Using the .map method

var mapSimpleArray = theStagesOfJS.map(function(item) {

return item;});

// the `.map` method created a new array with indices from the existing array.

console.log(mapSimpleArray);

// Map will also provide the index position of the array.

// This is similar to enumerate in Python.

var mapArrayWithIndex = theStagesOfJS.map(function(item, index) {

return `Stage ${index}: ${item}`;});

console.log(mapArrayWithIndex);

// Note: The original array is unchanged

console.log(theStagesOfJS);

// Mapping over an array of objects

var students = [

{ name: "Malcolm", score: 80 },

{ name: "Zoe", score: 85 },

{ name: "Kaylee", score: 99 },

{ name: "Simon", score: 99 },

{ name: "Wash", score: 79 }];

var names = students.map(function(student) {

return student.name;});

var scores = students.map(function(student) {

return student.score;});

// two new arrays were created from the original: one holding student names,

// and the other holding student scores.

// Map vs forEach

// `forEach` executes a function on each element in an array.

// `map` creates a new array with the results of calling a function on each element in the original array.

// Part A

var forEachStages = theStagesOfJS.forEach(function(each, index) {

// the return of forEach is ignored

return `Stage ${index + 1}: ${each}`;});

// undefined, because `forEach` does not automatically create new arrays like `map`.

console.log(forEachStages);

// unaltered

console.log(theStagesOfJS);

// Part B

theStagesOfJS.forEach(function(each, index) {

// The original array is mutated with forEach

theStagesOfJS[index] = `Stage ${index + 1}: ${each}`;

});

// Note that the original array has been altered (mutated)

console.log(theStagesOfJS);

// Challenge Activity!

var princesses = [

{ name: "Rapunzel", age: 18 },

{ name: "Mulan", age: 16 },

{ name: "Anna", age: 18 },

{ name: "Moana", age: 16 }];

// Log the name of each princess, follow by a colon, followed by their age

// forEach: executes a provided function once for each array element

princesses.forEach(function(princess) {

console.log(`${princess.name}: ${princess.age}`);

});

// Create an array of just the names from the princesses array

// map: creates a new array with the results of calling a provided function on every element in the calling array

var names = princesses.map(function(princess) {

return princess.name;

});

console.log("names: ", names);

2.4-Evr\_Arrow\_Functions

**### Lesson - Arrow Functions 2.4**

Refactor JavaScript code using ES6 arrow functions.

[Watch the Video](https://youtu.be/8lZ0EhI3Czo)

var theStagesOfJS = ["confidence", "sadness", "confusion", "realization", "debugging", "satisfaction"];

var students = [

{ name: "Malcolm", score: 80 },

{ name: "Zoe", score: 85 },

{ name: "Kaylee", score: 99 },

{ name: "Simon", score: 99 },

{ name: "Wash", score: 79 } ];

// An Arrow function is a new concise syntax for function

// Arrow functions allow us to drop the `function` keyword and just show the parameters.

// Note: The fat arrow `=>` that was added to indicate an arrow function.

var mapArrow1 = theStagesOfJS.map((item) => {

return item;});

// For functions with a single return line, we can drop the curly braces.

var mapArrow2 = theStagesOfJS.map(item => return item);

// And finally, we can just drop the `return` keyword. The return is implied.

var mapArrow3 = theStagesOfJS.map(item => item);

// Functions with more than one parameter still need the parenthesis

var mapReturn2 = theStagesOfJS.map((item, index) => {

return `Stage ${index}: ${item}`;});

// We can also drop the curly braces here

var mapReturn2 = theStagesOfJS.map((item, index) => `Stage ${index}: ${item}`);

// Map and Arrow makes it really easy to build an array of values for an array of objects

var names = students.map(student => student.name);

var scores = students.map(student => student.score);

// Challenge Activity!

var princesses = [

{ name: "Rapunzel", age: 18 },

{ name: "Mulan", age: 16 },

{ name: "Anna", age: 18 },

{ name: "Moana", age: 16 }];

// log the name of each princess, follow by a colon, followed by their age

// forEach: executes a provided function once for each array element

princesses.forEach(princess => console.log(`${princess.name}: ${princess.age}`));

// create an array of just the names from the princesses array

// map: creates a new array with the results of calling a provided function on every element in the calling array

var names = princesses.map(princess => princess.name);

console.log("names: ", names);

// [When and Why You Should Use ES6 Arrow Functions]

(https://medium.freecodecamp.org/when-and-why-you-should-use-es6-arrow-functions-and-when-you-shouldnt-3d851d7f0b26)

2.5-Ins\_Object\_Iteration

var userInfo = {

name: "Eric",

age: 32,

location: "North America"};

// Use `Object.values` and `forEach` to iterate through keys

Object.keys(userInfo).forEach(key => console.log(key));

// Use `Object.values` and `forEach` to iterate through values

Object.values(userInfo).forEach(value => console.log(value));

// Use `Object.entries` and `forEach` to iterate through keys and values

Object.entries(userInfo).forEach(([key, value]) => console.log(`Key: ${key} and Value ${value}`));

// Array of objects

var users = [

{ name: "Eric", age: 32, location: "North America" },

{ name: "Sally", age: 23, location: "Europe" },

{ name: "Cassandra", age: 27, location: "North America" }];

// Loop through array of objects then each object

users.forEach((user) => {

console.log(user);

// Get the entries for each object in the array

Object.entries(user).forEach(([key, value]) => {

// Log the key and value

console.log(`Key: ${key} and Value ${value}`);

}); });

2.6-Stu\_Object\_Iteration

var recipes = [

{ dish: "Fried fish", spice: "Dorrigo" },

{ dish: "Crab Rangoon", spice: "Akudjura" },

{ dish: "Pickled Okra", spice: "Chili pepper" },

{ dish: "Macaroni salad", spice: "Pepper" },

{ dish: "Apple butter", spice: "Avens" },

{ dish: "Pepperoni Pizza", spice: "Asafoetida" },

{ dish: "Hog fry", spice: "Peppermint" },

{ dish: "Corn chowder", spice: "Akudjura" },

{ dish: "Home fries", spice: "Celery leaf" },

{ dish: "Hot chicken", spice: "Boldo" }];

// Create empty arrays to store the dish and spice values

var dishes = [];

var spices = [];

// Iterate through each recipe object

recipes.forEach(function(recipe) {

// Iterate through each key and value

Object.entries(recipe).forEach(function([key, value]) {

// Use the key to determine which array to push the value to

if (key === "dish") {

dishes.push(value);

}

else {

spices.push(value);

} }); });

// BONUS - Use map to build both arrays of dish and spice values

var dishesMapped = recipes.map(function(recipe){return recipe.dish});

var spicesMapped = recipes.map(function(recipe){return recipe.spice});

console.log(dishesMapped);

console.log(spicesMapped);

2.7-Ins\_Filter

// filter() An array of objects, representing a cartoon family

var simpsons = [{

name: "Homer",

age: 45

}, {

name: "Lisa",

age: 8

}, {

name: "Marge",

age: 43

}, {

name: "Bart",

age: 10

}, {

name: "Maggie",

age: 1 }];

// Create a custom filtering function

function selectYounger(person) {

return person.age < 30; }

// filter() uses the custom function as its argument

var youngSimpsons = simpsons.filter(selectYounger);

// Test

console.log(youngSimpsons);

// filtering is a two-step process: An analogy here might be that

// a detective named `filter` is hired to comb through the data to find people younger than 30.

// The detective, in turn, hires a subcontractor named `selectYounger` to inspect each individual's age.

learning to filter data will be important for completing this week's homework.

2.8-Stu\_Filters

// An array of objects

var roster = [{

name: "Doug",

position: "Quarterback",

madeTeam: true},

{

name: "Antonio",

position: "Tight End",

madeTeam: true},

{

name: "Nick",

position: "Kicker",

madeTeam: false},

{

name: "Ereck",

position: "Offensive Live",

madeTeam: false},

{

name: "AJ",

position: "Line Backer",

madeTeam: true }];

// Create a custom function to return players who made the team

function madeCut(player) {

// return player.madeTeam == true;

// A more concise way to express a boolean conditional

return player.madeTeam; }

// Call the custom function with filter()

var playersOnTeam = roster.filter(madeCut);

// Display the results

console.log(playersOnTeam);

// Determine how many players made the cut, and how many did not

var numberOfPlayers = playersOnTeam.length;

var numberOfCutPlayers = roster.length - numberOfPlayers;

// Display the results

console.log(`${numberOfPlayers} players made the team.`);

console.log(`${numberOfCutPlayers} players were cut.`);

[Bootstrap Tables Documentation] (<https://getbootstrap.com/docs/3.3/css/#tables>)

(<https://www.w3schools.com/html/html_tables.asp>)

2.9-Ins\_Reduce

var prices = [12,18,30,10]

var sum = prices.reduce((sum, price) => sum + price)

console.log(sum)

var names = ['Alice', 'Bob', 'Tiff', 'Bruce', 'Alice']

var countedNames = names.reduce((allNames, name) => {

if (name in allNames) {

allNames[name]++

} else {

allNames[name] = 1

}

return allNames

}, {})

console.log(countedNames)

**10/15/18 – day 3 - Todays Objectives - Manuel**

### Overview

Today's class will introduce students to DOM selection, manipulation, and events using

[D3.js](<https://d3js.org/> ).

\* D3.js is an incredibly powerful visualization library written in JavaScript.

working with data in JavaScript and learning how to manipulate the DOM with D3.

The material covered today is intended to be a gentle introduction to the DOM using helper methods from D3.

\* Today's class uses only a subset of D3.js to perform basic DOM manipulation and event handling…. to select and create HTML elements dynamically.

Data binding is not covered in this lesson as D3 is only being used as an aid to simplify interaction with the DOM. A full week will be dedicated to further exploring D3 in a future unit.

to learning D3.js to build dynamic visualizations in JavaScript.

Today’s Objectives

\* Review 15.1 – Plotly (AD taught this past Sat.)

\* 14.3 - D3.JS Intro

\* Use D3 for basic DOM manipulation.

\* Understand event handling

\* Use `this` to reference elements.

\* Use D3 to attach events to DOM elements.

\* Dynamically manipulate the DOM through events.

\* Dynamically filter tables

### Class Objectives (not repeated in Mauel’s objectives above that were slacked)

\* Students will be able to populate a table using static data structures.

\* Students will understand events.

Resources

\* Document Object Model (DOM)

- <https://www.tutorialspoint.com/javascript/javascript_html_dom.htm>

- <https://www.tutorialspoint.com/javascript/javascript_w3c_dom.htm>

- <https://www.tutorialspoint.com/javascript/images/html-dom.jpg>

\* D3JS

- <https://d3js.org/>

valid events to trigger on

<https://developer.mozilla.org/en-US/docs/Web/Events>

**### Lesson - D3 Table 3.3**

Create a table with D3 and fill it with weather data pulled from an array of objects.

[Watch the Video](https://youtu.be/EfkbksORz-Y)

**### Lesson - Form Filtering 3.9**

Use D3 to filter data based on input from a form field.

[Watch the Video](https://youtu.be/EDk-4-2NtoI)

**# Unit 14 - Intro to Javascript from StudentGuide.md**

**## Objectives**

\* Understand JS fundamentals: arrays, conditionals, loops, functions, objects.

\* Understand functional programming with map, forEach.

\* Work with common data structures.

\* Be introduced to data driven documents (d3.js).

\* Understand how to select elements using d3.select.

\* Use d3 for basic DOM manipulation.

\* Understand how to use callbacks.

\* Understand the structure of html tables.

\* Populate a table using static data structures.

\* Understand events.

\* Use d3 to attach events to DOM elements.

\* Dynamically manipulate the DOM through events.

\* Filter data with JavaScript.

**### Helpful Links**

\* [Interactive JavaScript Sheet](http://htmlcheatsheet.com/js/)

\* [Scrimba Intro to JavaScript](https://scrimba.com/g/gintrotojavascript)

\* [Scrimba ES6+](https://scrimba.com/g/gintrotoes6)

\* [You Don't Know JS (book series)](https://github.com/getify/You-Dont-Know-JS)

\* [JavaScript Tutorial](<https://watchandcode.com/p/practical-javascript>)

**## Advanced Topics**

**### Machine Learning with TensorFlow.js**

Brief Introduction to machine learning in the browser using the TensorFlow.js library.

[tensorflow.js](https://codingbootcamp.hosted.panopto.com/Panopto/Pages/Viewer.aspx?id=96a1d603-321f-40f5-8f29-a8e1000f81f0)

<https://htmlcheatsheet.com/js/>

JavaScript Cheat Sheet contains useful code examples on a single page. Find code for JS loops, variables, objects, data types, strings, events and many other categories.