JAVASCRIPT LANGUAGE CSE 121B

# Week 5: Controlling your Code

## Choice Making and Computational Flow

JavaScript and Python share a set of structures used to control which part of your code is executed, based on specific situations. The idea of situational 'computational flow' control isn't unique to Python or JavaScript. It is found in all languages though the way it is accomplished can be different. Review the boolean operators from Week 02. They are used by the control structures you will see below.

### If

In JavaScript, the if...else if....else statement takes this form,

```
if(boolean expression){
    //code if true
}
else if(boolean expression){
    //code if true
}
else{
    //code if no other match
}
```

Notice the boolean expressions have parenthesis around them, and the JavaScript scope operators are used to indicate what lines of code executes in each condition. Here is a code example that evaluates the number of sales of product a company has for one day. The first condition checks if the number of sales is less than or equal to 50. Since the first check has been done, the second

actually tests if the number of sales is greater than 50 and less than or equal to 1000. The default, 'else', is the number of sales being greater than 1000.

```
if(numSales <= 50){
    console.log("Way to few sales");
}
else if(numSales <= 1000){
    console.log("Average number of sales")
}
else{
    console.log("A good number of sales.")
}
```

There is another way to control your code that behaves similar to an if-else if-else statement. It's called a switch. It can be very helpful if you have large numbers of cases to check. Just be careful using it. It is VERY easy for those not familiar with them to put logic errors in their code using switches.

## Getting Loopy

As mentioned in week 03, JavaScript has built in map, reduce, and filter functions for arrays. It also has a forEach loop that is used with arrays. Go visit this gist on array iteration thoughts for a quick review of some of those array methods we can use to traverse an array.

Maybe, in a different part of the likes application from week 3, you are asked to print each of the like values to the console. You could use the forEach function to do this.

```
likeCounts.forEach(aCount => console.log(aCount))
```

Here is an important piece of advice. NEVER use the forEach method when map, filter, reduce, or some combination of them will accomplish the same task. It should rarely ever be used.

JavaScript has more types of loops that do things similar to the forEach loop. Use of these should be avoided when possible. Instead, use the map, filter, and reduce functions. If you design well by thinking outside the box, you will probably never need to use these or forEach. They are slower anyway.

### The for-in Loop

This is NOT the same as the python for-in loop. Its purpose is to iterate over the properties of an object, allowing you to manipulate or print out the property values associated with each key. We'll use the Lactated Ringers medication object from last week in this example. You could print out each of the values for Lactated Ringers like this.

```
let aMedication = medications['Lactated Ringers']
for(propertyName in aMedication){
    console.log(aMedication[propertyName])
}
```

### The for-of Loop

The for-of loop IS the same as the Python for-in loop. It can, but shouldn't, be used to iterate over the elements of a list, set, map, or other type of data structure. It won't iterate over the properties of a defined object.

```
let ages = [3,5,-1]
for(anAge of ages){
    console.log(anAge)
}
```

### The for Loop

Here is the traditional C-like `for` loop. If you are looping over an array (the most common thing we do with loops) please don't use this! You should really look at `.map`, `.filter`, `.reduce` or `.forEach` as they are MUCH better suited to that. Sometimes you just need to do something (not dealing with arrays) a known number of times. The `for` loop is good for that.

```
for(let i = 1, i <=10, i++){
    console.log(i)
}
```

### The while Loop

Like Python's while loop, JavaScript's while loop works with a single boolean logic check to assess the state of your application by assessing one or more boolean logic statements. As long as the logic statements are true, the loop continues looping. Otherwise it stops. This is useful when you need to loop an unknown number of times, but again, you will rarely need to use this loop if you wisely apply map, filter, and reduce.

```
let total = 0
while(total < 10){
    console.log(total)
    total++
}
```