More JavaScript!

Higher-Order Functions, Callbacks, and Array Methods

Higher-Order Functions

- A Higher-Order Function is any function that operates on any other function, either by taking them in as arguments or returning them.
- In JavaScript, this is facilitated by the fact that functions are First-Class Functions

Higher-Order Functions

```
function outerFunc(cb){
    return cb();
}
```

- In the above example, the Higher-Order function is the `outerFunc` function because it takes in a callback (cb) and returns the invocation of the callback inside of it.

Callback Function

- A callback function is any function that is passed into another function as an argument, which is then invoked inside of the outer function to complete some kind of routine or action.

- Callback functions can be declared functions, function expressions, or even anonymous functions depending on your needs and the context in which you are using them.

Callback Function

```
function outerFunc(cb){
    return cb();
}
```

- Looking at the same example from before, the callback in this case is cb, as it is being passed into the `outerFunc` function and is invoked inside of it.

Array Methods

- To solidify understanding of higher order functions and callbacks, we are going to look at some of the more popular iterative array methods provided for us by JavaScript.

- Some of the most popular are forEach, map, and reduce

array.forEach()

- The forEach method takes a callback and runs it once for each element in the array.
- forEach can also take an optional index argument to keep track of the index that you are currently working with
- forEach function does not mutate the array that it is being operated on, and does not return any value itself.

array.map()

- The map method takes a callback function and creates a new array by performing the callback on each array element.

- map can also take an optional index array.

- map does not mutate the original array. It instead returns a new array of the same length as the original array with the result of operating the callback function.

array.reduce()

- The reduce method takes a callback function and <u>an iterator (which can be any data type)</u> and runs the callback on each array element to reduce it to a single value.

- Your callback should take at least two arguments, which are regularly known as *previous* and *next.* These will be used to reduce each value in the array into the iterator, and as such your callback must return a value to be used on the next iteration.

- Reduce <u>does not</u> mutate the original array, but it does return a new value based on the callback function.