MAKING DECISIONS 2

Intro

Overview

- Arrays
- Combining Arrays & Loops
- Control Flow

Arrays

What Are Arrays?

Arrays can be thought of as lists (or a collection) of data.

They are created using [square brackets], with each element in the array separated by a comma.

These are organized by indexes, which start at 0.

Accessing values in an array

Values inside of arrays are accessed primarily using bracket notation. This allows us to reference a specific index, or location within an array. To do this, we use the following format: array[index]

In most languages, arrays are zero indexed, meaning that the first index, or location, inside of an array is 0. For example:

```
var numArr = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
```

In this example numArr[0] is equal to 1 because 1 is the first item in the array. numArr[5] is equal to 6 because 6 is the 5th index on the array.

This can be a confusing idea at first, but you will have plenty of practice accessing items in arrays. You can also access values on an array by passing in a variable. This can be useful if you don't know exactly which index you will need to access at different times. For example:

```
var namesArr = ['Andrew', 'Jonathan', 'Josh', 'Brandon']

var index = 0

console.log(namesArr[index])
//This will print 'Andrew' to the console.

//However, if we change the value of index later in the file,
//we can access another value.

index = 2
console.log(namesArr[index])
//Because the value of index is now 2, this will print 'Josh' to the console
```

Editing arrays

Arrays have built in functions to easily edit their contents. They will all allow you to edit arrays as the data you are dealing with changes.

All of these functions are invoked by chaining them onto an array variable in the following format: `array.function()`.

Let's dive into them…

- `.push()` allows you to add an item to the end of an array.

```
var namesArr = ['Andrew', 'Jonathan', 'Josh']
namesArr.push('Brandon')

//namesArr now looks like this: ['Andrew', 'Jonathan', 'Josh', 'Brandon']
```

- `.pop()` removes the last item from an array.

```
var namesArr = ['Andrew', 'Jonathan', 'Josh']
var finalName = namesArr.pop()

//namesArr now looks like this: ['Andrew', 'Jonathan']
//.pop can be assigned to a variable which will be the item removed.
//In this example finalName is now equal to 'Josh'
```

- `.shift()` removes the first item from an array.

```
var namesArr = ['Andrew', 'Jonathan', 'Josh']
var firstName = namesArr.shift()

//namesArr now looks like this: ['Jonathan', 'Josh']
//Similar to .pop, .shift can be assigned to a variable.
//firstName will be equal to 'Andrew'
```

- `.unshift()` allows you to add an item to the front of an array.

```
var namesArr = ['Andrew', 'Jonathan', 'Josh']
namesArr.unshift('Brandon')
//namesArr now looks like this: ['Brandon', 'Andrew', 'Jonathan', 'Josh']
```

- `.slice()` makes a shallow copy of the array that it's chained on to. It takes two arguments, the starting and ending indices of the array you want to copy. The ending index is not included in what is copied. This method does not change the initial array.

```
var namesArr = ['Andrew', 'Jonathan', 'Josh']
var someNames = namesArr.slice(0, 2)
//someNames now looks like this: ['Andrew', 'Jonathan']
```

- `.splice()` is the most dynamic of all for editing arrays. It takes 3 arguments:

  1. The index at which we want to begin editing

  2. How many items to remove from the array

  3. Any values to replace at that index

```
var namesArr = ['Andrew', 'Jonathan', 'Josh', 'Brandon', 'Steve']
var removedName = namesArr.splice(1, 1)
//This will remove 1 name from the array at index 1 and assign it to the variable removedName
//namesArr now looks like this: ['Andrew', 'Josh', 'Brandon', 'Steve']
//removedName is equal to 'Jonathan'

//We can continue to edit it, and even put new values in
namesArr.splice(1, 0, 'Charles')
//This will just insert charles at index 1.
//namesArr will now look like this: ['Andrew', 'Charles', 'Josh', 'Brandon', 'Steve']
```

- `.length` All arrays have a length property by default. As the name would suggest this returns the length of the array. Remember that while arrays are zero indexed, length is not. For example:

```
var namesArr = ['Andrew', 'Jonathan', 'Josh']
console.log(namesArr.length)
//Prints 3 to the console
```

- Note: You do not invoke length, as it is not a function but a property.
- Trick: You can use .length to dynamically access the last index in an array. If we wanted the last item in our namesArr array but didn't know its index, we could reference it like this: namesArr[namesArr.length - 1]. This is a good trick to know.

Combining Arrays & Loops

Using a for loop to iterate over an array

We are able to use for loops to dynamically look at each item in an array based on its index. Let's take a look at what this might look like:

```
var numsArr = [1, 2, 3, 4, 5]
for (let i = 0; i < numsArr.length; i++) {
  numsArr[i] += 1
  //The += operator is the same as saying numsArr[i] = numsArr[i] + 1
}

//After the for loop runs, numsArr will look like this: [2,3,4,5,6]
```

- In this way we are able to loop over an array and dynamically access numsArr at index i, executing a block of code on each

Note: For-Loops & Arrays

While for loops are useful for looping over arrays, they are not intrinsically tied together. The important thing to remember, is that a for loop executes a given code block until the provided condition evaluates to false.

Control Flow

What is Control Flow

To put it simply, we can **control** the **flow** of our code by writing logic that directs our code one way or the other, depending on certain logic.

Basic Examples of Control Flow

We have already been using control flow. If-statements and loops are perfect examples of this. For example:

```
if (temperature < 55) {
  console.log("Better get a jacket.")
} else {
  console.log("T-shirt today!")
}
```

In this simple example, we were able to write code in such a way that we can **control** which line will print. In the same way, when we write a loop, we can control how many times our code will run, based on the conditions we provide.

The Importance of Control Flow

At this point, you can probably tell that control flow is important, but what you might not realize yet, is just how much control flow is going to play a part in your day to day coding. It will be used in most of your code going forward. Loops and conditionals are just the start. Functions will become a major part of the process soon. It is important to take the time now to make sure you understand how your code flows, or in other words, in which order your code will run.

Control Flow Example 1

```
// assume there are two variables, player1 and player2. Either of them can have the value "rock", "paper", or "scissors".

if (player1 === player2) {
  console.log("It is a draw")
} else if (player1 === "rock" && player2 === "scissors") {
  console.log("Player 1 wins.")
} else if (player1 === "rock" and player2 === "paper") {
  console.log("Player 2 wins.")
} else if (player1 === "scissors" && player2 === "paper") {
  console.log("Player 1 wins.")
} else if (player1 === "scissors" && player2 === "rock") {
  console.log("Player 2 wins.")
} else if (player1 === "paper" && player2 === "rock") {
  console.log("Player 1 wins.")
} else {
  console.log("Player 2 wins.")
}
```

Control Flow Example 2

```
// What is the final value of guessMe?

let guessMe = 54

while (guessMe < 100) {
    if (guessMe % 4 === 0 || guessMe % 5 === 0) {
        guessMe += 25
    } else if (guessMe % 3 === 0) {
        guessMe -= 27
    } else {
        guessMe += 3
    }
    guessMe += 22
}
```

Control Flow Summary

In that last example we could see how difficult it can be to follow code, even when that code is just 9 lines long. As we introduce functions, switch statements, ternary statements, and more, this will only become more difficult. Take your time to understand how code runs, and how you can control the flow of your code!