JavaScript III

INFO 253A: Front End Web Architecture

Kay Ashaolu

ECMAScript, what is that?

- ECMAScript is technically the JavaScript language standard
- Other languages have adopted some of this standard (e.g. ActionScript)
- Lays out the features of JavaScript to be agreed upon

So many versions...

- Figuring out which browser is running which version of ECMAScript could get daunting
- Browsers also do not simply implement the entire version
- A browser update could add support to single particular feature

What if you actually want to use the newer features

- That person that never updates IE will not be able to execute your JavaScript
- That person that found a way to not automatically update Chrome will not be able to see your site

Solution: Transpiler

- Similar to a compiler, but converts JavaScript to JavaScript
- Converts Javascript code written in a higher version into lower version JavaScript code
- This enables developers to use newer features, and users with older browsers able to execute the code

Example ES6 Code

```
1  class Planet {
2
3    constructor (mass, moons) {
4      this.mass  = mass;
5      this.moons = moons || 0;
6    }
7
8    reportMoons () {
9      console.log(`I have ${this.moons} moons.`)
10   }
11 }
```

Complied ES5 Code

```
1  var _createClass = function () { function defineProperties(target, props) {..
2  defineProperties(Constructor, staticProps); return Constructor; }; }();
3
4  function _classCallCheck(instance, Constructor) {
5          if (!(instance instanceof Constructor)) {
6                  throw new TypeError("Cannot call a class as a function");
7          }
8  }
9
10 var Planet = function () {
11   function Planet(mass, moons) {
12     _classCallCheck(this, Planet);
13
14     this.mass = mass;
15     this.moons = moons || 0;
16   }
17
18   _createClass(Planet, [{
19     key: 'reportMoons',
20     value: function reportMoons() {
21       console.log('I have ' + this.moons + ' moons.');
22     }
23   }]);
24
25   return Planet;
26 }();
```

Using Babel

- Babel is a transpiler that accomplishes conversion
- There is an entire build environment, using webpack 4, babel, and npm to set up
- For this week, please use the latest version of Chrome or Firefox to run your Javascript

Any Questions?

Syntactic Sugar

A lot of improvements to language focuses on changing syntax to make it easier to accomplish a certain goal

Let's talk about some of those features in ES6

Let and Scope

- Let creates a variable with scope
- Scope is a term that defines a boundary where variables live
- Scope is how you can ensure content inside a function is not affected by the outside
- Scope in Javascript is largely defined by curly brackets ('{}')

Let example

```
let a = 50;
let b = 100;
if (true) {
        let a = 60;
        var c = 10;
        console.log(a/c); // 6
        console.log(b/c); // 10
}
console.log(c); // 10
console.log(a); // 50
```

Let example explained

- The variable a is found both in the scope of this script, and in the scope of the if statement block
- The variable a within the block can be considered a different variable than the variable a outside the block

Const

- There a times where you do not want a variable to change after assignment
- For example, if you have a variable that is set to the number PI
- You wouldn't want that variable PI to change during your program

Const Example

```
1 const b = "Constant variable";
2 b = "Assigning new value"; // shows error.
3
4 const LANGUAGES = ['Js', 'Ruby', 'Python', 'Go'];
5 LANGUAGES = "Javascript"; // shows error.
6
7 LANGUAGES.push('Java'); // Works fine.
8 console.log(LANGUAGES); // ['Js', 'Ruby', 'Python', 'Go', 'Java'
```

Const example explained

- The variable LANGUAGES can not be changed
- However, what LANGUAGES points to, if it is mutable can change

Why use let and const?

- Cleaner understanding of the lifespan of a variable
- Reduce coding mistakes by ensuring variables that shouldn't change does not

Arrow Functions

- There is a new way of defining functions
- There are a few reasons for *this* (and that's actually a pun, but you can look that up to figure it out)
- This new way of writing function also helps with clearly defining scope

Arrow Functions Example

```
1  function oldOne(name) {
2          console.log("Hello " + name);
3  }
4
5  oldOne("Kay");
6
7  // New Syntax
8  let newOne = (name) => {
9          console.log("Hello " + name);
10 }
11
12 newOne("Kay");
```

What did that do?

- The parameters are named in the parenthesies outside the name of the function
- Note how you assign a variable to a function (and can use let for scope)

Default Parameters

- Convenient ability to assign parameters to a function a value if not specified by the caller

Default Parameter Example

```
let Func = (a, b = 10) => {
        return a + b;
}
console.log(Func(20)); // 20 + 10 = 30

console.log(Func(20, 50)); // 20 + 50 = 70

let NotWorkingFunction = (a = 10, b) => {
        return a + b;
}
console.log(NotWorkingFunction(20)); // NAN. Not gonna work.
```

What did that do?

- The function Func sets a default value to the second parameter
- You can pass the second parameter or leave it blank
- However order matters. You can't define a default parameter and then the next parameter does not have a default value

For...loop

- Very nice way of looping through a list of elements
- No need to figure out index parameters and value conditions

For...loop

```
let arr = [2,3,4,1];
for (let value of arr) {
        console.log(value);
}
```

For...loop explained

- The variable 'value' is assigned each element of that array once
- Note you do not have access to the index while using this construct

Spread Attributes

- Ability to define a function with a variable number of parameters
- You do not have to pass an array in order to have a variable number of parameters

Spread

```
let SumElements = (...arr) => {
        console.log(arr); // [10, 20, 40, 60, 90]

        let sum = 0;
        for (let element of arr) {
        sum += element;
        }
        console.log(sum);
}

SumElements(10, 20, 40, 60, 90);
SumElements(10, 20, 90);
```

What did that do?

- You can pass a variable number of parameters
- Those parameters are avaiable as an array inside the function

Template Literals

- Template literals makes adding variables to your strings much easier
- Many Languages (like Python and Ruby) has this built into the langauge

Template Literals Example

```
1 let name = "Jon Snow";
2 let msg = `My name is ${name}`;
3 console.log(msg);
```

Destructing Objects and Arrays

- Let’s just get into an example

Destructing Objects Example

```
1 let person = {firstName: "Jon", lastName: "Snow", age: 23}
2 const {firstName, age} = person
3
4 console.log(firstName);
5 console.log(age);
```

Destructing Arrays Example

```
1 let arr = [1,2,3,4,5,6]
2 let [a,b,,d,e] = arr
3
4 console.log(a);
5 console.log(b);
6 console.log(d);
7 console.log(e);
```

What did that do?

- You can do the same thing with arrays
- Order of the array that is the result of destructuring matters
- You can skip what you don't want by leaving that position blank

Questions?