Lecture 1: JavaScript, ES6

Jordan Hayashi

Previous Lecture

- Types
- Coercion
- Objects
- Prototypal Inheritance
- Scope
- JS Execution
- Global Object
- Closures...

ES5, ES6, ES2016, ES2017, ES.Next

- ECMAScript vs JavaScript — Javascript is implementation of ECMAScript
- What do most environments support? — ES5
- Transpilers (Babel, TypeScript, CoffeeScript, etc.) — Takes the newer version code (ES6, ES2016, etc.) and Transpiler converts it to ES5.
- Which syntax should I use?

Closures

- Functions that refer to variables declared by parent function still have access to those variables
- Possible because of JavaScript’s scoping

Immediately Invoked Function Expression

AKA IIF

- A function expression that gets invoked immediately
- Creates closure
- Doesn’t add to or modify global object

First-Class Functions

- Functions are treated the same way as any other value
  - Can be assigned to variables, array values, object values
  - Can be passed as arguments to other functions
  - Can be returned from functions
- Allows for the creation of higher-order functions
  - Either takes one or more functions as arguments or returns a function
  - map(), filter(), reduce()

```
> const x = [0,1,2,3]
> function addOne(number) { return number + 1 }
> addOne(1)
2
> x.map
[Function: map]
> x.map(addOne)
[ 1, 2, 3, 4 ]
```

filter() -->Retains/keeps the values that are true and gets rid of the values that false

```
> const x = [0,1,2,3]
function isGreaterTHanOne(num) { return num >1 }
> isGreaterTHanOne(1)
false
> x.filter(isGreaterTHanOne)
[ 2, 3 ]
```

** Takes array of values and returns one value **

```
> function add(x,y) { return x+y }
> add(1,2)
3
> x
[ 0, 1, 2, 3 ]
> x.reduce(add)
6
```

Synchronous? Async? Single-Threaded?

- JavaScript is a single-threaded, synchronous language
- A function that takes a long time to run will cause a page to become unresponsive
- JavaScript has functions that act asynchronously
- But how can it be both synchronous and asynchronous?

Synchronous --> other features becomes unresponsive since there is something running in the thread already

Asynchronous JavaScript

- Execution stack
- Browser APIs
- Function queue
- Event loop

Execution Stack

- Functions invoked by other functions get added to the call stack
- When functions complete, they are removed from the stack and the frame below continues executing

Asynchronous JavaScript

- Execution stack
- Browser APIs
- Function queue
- Event loop

Asynchronous JavaScript

- Asynchronous functions
  - setTimeout()
  - XMLHttpRequest(), jQuery.ajax(), fetch()
  - Database calls

Callbacks

- Control flow with asynchronous calls
- Execute function once asynchronous call returns value
  - Program doesn’t have to halt and wait for value

Promises

- Alleviate “callback hell”
- Allows you to write code that assumes a value is returned within a success function
- Only needs a single error handler

Async/Await

- Introduced in ES2017
- Allows people to write async code as if it were synchronous

this

- Refers to an object that’s set at the creation of a new execution context (function invocation)
- In the global execution context, refers to global object
- If the function is called as a method of an object, `this` is bound to the object the method is called on

Setting `this` manually

- bind(), call(), apply()
- ES6 arrow notation

Browsers and the DOM

- Browsers render HTML to a webpage
- HTML defines a tree-like structure
- Browsers construct this tree in memory before painting the page
- Tree is called the Document Object Model
- The DOM can be modified using JavaScript

Assignment

- Create a TODO app
- Will use JS DOM manipulation