ES6 JavaScript - What You Need To Know

Destructuring assignment

- `let {a, b} = o` assigns Object `o`'s `a` and `b` properties to variables `a`, `b`
- `let [a, b] = arr` assigns first/second items of Array `arr` to variables `a` and `b`
- Assign defaults with `=`, e.g. `let {max = 5} = options`
- Destructuring can be performed on function arguments.
  `function fn({options = {}, flag = true}) { ... }`

`for .. of` loops

- Works on Iterables, including `Array`, `Map`, `Set` and generators.
- Does not work with objects.
- Use with destructuring assignment and `let`
  `for (let [key, value] of map) { ... }`

`let` / `const`

- Make variables scoped by block, not function
- Use in place of `var`
- `const` prevents re-assignment, but does not make assigned objects immutable

=> arrow functions

- *`argument => returned expression`*
- `this` inside function is equal to `this` where it was defined
  `function() { ... }.bind(this)`
- *returned expression* can be a block
  `x => { console.log('doubling'); return x*2 }`
- Use parentheses for more than one argument
  `(min, x, max) => Math.max(min, Math.min(x, max))`
- Use parentheses when argument is being destructured
  `({x, y}) => Math.sqrt(x*x, y*y)`

Backtick (``) Template Strings

- Interpolate with `${`*`expression`*`}`
  `` `Token token=${identity.get('accessToken')}` ``
- Can be split over multiple lines

... (spread operators / rest parameters)

- In functions parameters, creates an array of remaining arguments
  `function classes(...args) { return args.join(' ') }`
- In function arguments, expands array to actual parameters
  `console.log(...args)`
- Similar to `Function.prototype.apply`, but doesn't modify `this`

New `Array` Methods

- `arr.find(callback[, thisArg])`
  return the first item which when passed to `callback`, produces a truthy value
- `arr.findIndex(callback[, thisArg])`
  return the index of the first item which when passed to `callback` produces a truthy value
- `arr.fill(value[, start = 0[, end = this.length]])`
  fills all the elements of an array from a `start` index to an `end` index
- `arr.copyWithin(target, start[, end = this.length])`
  copies the sequence of items within the array to the position starting with `target`, taken from the position starting with `start`

New Built-in Classes

- `Map` - Map keys to values. Unlike objects, keys don't have to be strings
- `Set` - Store a set, where each stored value is unique
- `Symbol` - Use to make private object/class properties
- `Promise` - Manage callbacks for an event which will occur in the future