ES6 AND BEYOND

Introducing the latest JS goodies and how to use them today

ABOUT ME

- Web developer for over ten years
- Recently became a frontend dev
- First-time speaker

SO WHAT'S ES6

- And ES7 and ES8
- And ES2015, ES2016 and ES2017
- What about ES.Next?

PROPOSAL STAGES

- Stage 4: Finished
- Stage 3: Candidate
- Stage 2: Draft
- Stage 1: Proposal
- Stage 0: Strawman

WHY DO I NEED TO KNOW ALL OF THIS?

Because this is where a lot of the cool features lie

FEATURES

- Shorthands
- New functions, objects, methods
- New concepts

SHORTHAND: `FOR..OF` LOOPS

```
var list = [2, 3, 5, 7, 11]

for (var number of list) {
  console.log(number)
}

var kvStore = new Map({
  key: 'value',
  otherKey: 'other value'
})

for (var item of kvStore) {
  // we'll see how this can be made even nicer later
  console.log(item[0], ':', item[1])
}
```

SHORTHAND: ARROW FUNCTIONS

```
var log = (text) => {
  console.info(text)
}

var addAndDouble = (first, second) => {
  var result = first + second
  return result * 2
}

var add = (first, second) => first + second

log(add(2, 3))

var list = [2, 3, 5, 7, 11]

console.log(
  list.map(item => item ** 2)
)
```

SHORTHAND: FUNCTION BINDING

```
// Syntax thisObject::methodReference(arguments)

// Previously:

$.ajax(url, { success: function (text) { console.log(text) } })

// With new arrow functions

$.ajax(url, { success: (text) => console.log(text) })

// With function bind syntax

$.ajax(url, { success: ::console.log })

// Just syntactic sugar for

$.ajax(url, { success: console.log.bind(console) })
```

SHORTHAND: FUNCTION BINDING

```
// Where find and html are functions that work on "this", similarly to how jQuery works with callbacks

document.querySelectorAll('div.myClass')::find('p')::html('Example')
```

SHORTHAND: DO EXPRESSIONS

```
var a = do {
  if (someValue === null) {
    'null'
  } else if (someValue < 0) {
    'negative'
  } else {
    someValue
  }
}

// REALLY handy in React:

return (
  <div>
    { do {
      if (state.data) {
        <div>
          <table>
            ...
          </table>
        </div>
      } else {
        <div>Loading...</div>
      }
    } }
  </div>
)
```

SHORTHAND: DESTRUCTURING

```
var data = { tag: 'div', properties: { style: 'font-weight: bold' } }

// Pick tag and properties out of data
var { tag, properties } = data

// Rename properties to props
var { tag, properties: props } = data

// Pick tag and properties.style out of data
var { tag, properties: { style } } = data

console.log(tag, ':', style) // => "div: font-weight: bold"

// Remember for..of from earlier?

for (var item of someMap) {
  console.log(item[0], ':', item[1])
}

// Now we can write it like this:

for (var [key, value] of someMap) {
  console.log(key, ':', value)
}
```

SHORTHAND: DEFAULTS

```
function console(text, level = 'info') {
  console[level](text)
}


// options gets an implied `= undefined` here
function ajax(url, method = 'GET', options) {
    // ...
}


var data = { name: 'Jim', age: 28 }

var { name, age = 20, country = 'UK' } = data

console.log(name, age, country) // => 'Jim', 28, 'UK'


var [ first, second = 2 ] = [1]

console.log(second) // => 2
```

SHORTHAND: REST AND SPREAD

```
var data = [2, 3, 5, 7, 11]


// Destructuring - rest
var [ first, second, ...rest ] = data

console.log(first, rest[1]) // => 2, 7


// Constructing - spread
var newData = [ ...rest, second, first ]

console.log(newData) // => 5, 7, 11, 3, 2


// Function arguments - rest
function tail(first, ...rest) {
  return rest
}


// Convert an array to an argument list - spread
console.log(Math.max(...data)) // => 11
```

SHORTHAND: REST AND SPREAD

```
var data = [2, 3, 5, 7, 11]


var data2 = [ ...data ]

data2[1] = 4

console.log(data) // => [2, 3, 5, 7, 11]
console.log(data2) // => [2, 4, 5, 7, 11]


var data = { language: 'javascript', version: 'es5' }

var data2 = { ...data }

data2.version = 'es2015'

console.log(data) // => { language: 'javascript', version: 'es5' }
console.log(data2) // => { language: 'javascript', version: 'es2015' }


// can override object keys on the fly:
var data3 = { ...data, version: 'es2015', addNewKey: true }
```

SHORTHAND: TEMPLATE LITERALS

```
var count = 4, thing = 'apple'

console.log(`I have ${count} ${thing}s`)

console.log(`I have ${count} ${
  do {
    if (count === 1) {
      thing
    } else {
      `${thing}s`
    }
  }
}`)

// Tagged template literal
function upper([ text ]) { // destructuring as an array is sent in
  return text.toUpperCase()
}

console.log(upper`Some string here`)
```

SHORTHAND: OBJECT LITERAL EXTRAS

```
var count = 4, item = 'apple'

// Currently:

var obj = {
  count: count,
  item: item,
}

// If the key and variable have the same name, just use the variable.

var obj = { count, item }

// Defining a function in an object:

var obj = {
  count,
  type: item,

  say() {
    console.log(`I have ${this.count} ${this.type}s`)
  }
}


obj.say() // => "I have 4 apples"
```

SHORTHAND: CLASSES

```
class Quad {
  length = 0
  width = 0
  constructor(length, width) {
    this.length = length
    this.width = width
  }
  area() {
    return this.length * this.width
  }
}

class Quad3D extends Quad {
  depth = 0
  constructor(length, width, depth) {
    super(length, width)
    this.depth = depth
  }
  volume() {
    return this.area() * this.depth
  }
}

var cuboid = new Quad3d(4, 7, 3)
console.log(cuboid.area(), cuboid.volume()) // => 28, 84
```

SHORTHAND: CLASSES

```
class Quad {
  length = 0
  width = 0

  constructor(length, width) {
    this.length = length
    this.width = width
  }

  area = () => this.length * this.width

  getAreaSoon() {
    setTimeout(this.area, 500) // okay this doesn't do anything, but you get the idea
  }
}
```

NEW METHODS: ARRAYS

- Array.prototype.map
- Array.prototype.filter
- Array.prototype.reduce
- Array.prototype.includes
- Array.prototype.find
- Array.prototype.findIndex

NEW FUNCTIONS: ARRAYS

```
var data = [2, 3, 5, 7, 11]

var doubled = data.map(number => number * 2)

var smallish = data.filter(number => number < 7)

var sum = data.reduce((acc, number) => acc + number, 0)

var seven = data.find(number => number > 5) // will return the first it finds or undefined

var idx = data.findIndex(number => number > 5) // ditto

console.log(data.includes(6)) // => false
```

NEW FUNCTIONS: OBJECTS

- Object.keys
- Object.values
- Object.entries

NEW FUNCTIONS: OBJECTS

```
var obj = {
  id: 719,
  name: 'Bob',
  city: 'Birmingham',
}

for (var key of Object.keys(obj)) {
  console.log(key)
}

for (var value of Object.values(obj)) {
  console.log(value)
}

for (var [key, value] of Object.entries(obj)) {
  console.log(key, value)
}
```

NEW OBJECTS: `SET` AND `MAP`

`Set` : like an array but can only contain unique values

`Map` : like an object but can use any object as keys and are inherently iterable

NEW CONCEPTS: MODULES

```
// include a library that has a global effect (e.g. a polyfill)
import 'whatwg-fetch'

// import the default export from a given module
import moment from 'moment'

// import the default and also a named export (Component)
import React, { Component } from 'react'

// import a named export and rename it in this scope
import { PropTypes as P } from 'react'

// export a function as the default export
export default function() {
  console.log(Date.now())
}

// export a constant
export const PI = 3

// export an import!
export moment from 'moment'
```

NEW CONCEPTS: DECORATORS

```
@setDatabaseConnection(db)
class X {
  // ...
}

@logCalls
function y() {
  // ...
}

class Z {
  @writeable(false)
  someProperty = 0
}
```

NEW CONCEPTS: `LET` AND `CONST`

```
let i = 0
i++

console.log(i) // => 1

const j = 0
j++ // errors


const arr = [2, 3, 5, 7, 11]

arr.push(13)
arr.shift()

arr = arr.map(number => number ** 2) // errors


const obj = { id: 9 }

obj.name = 'Sally'
```

NEW CONCEPTS: `LET` AND `CONST`

```
for (const [key, value] of Object.entries(data)) {
  setImmediate(() => console.log(key, value))
}
```

HOW TO USE TODAY

- Code must be transpiled
- Best to use a build process
- Still need polyfills
- Grunt/Gulp + Browserify + Babel
- Webpack + Babel