ES2015 Features

And Your Rails App

Josiah Mory / @kickinbahk

ES2015 - Released June 2016

Added New Syntax For "Complex" Applications

github.com/DrkSephy/es6-cheatsheet/

New Features

Var vs Let/Const

Replacing IIFEs with Blocks

Arrow Functions

Strings

Destructuring

Modules

Parameters

New Features cont.

Classes

Symbols

Maps

WeakMaps

Promises

Generators

New Features cont.

Var vs Let/Const

Replacing IIFEs with Blocks

Arrow Functions

Parameters

Strings

Promises

Generators

Var

vs.

Let & Const

Var is Scoped to the Function

```
function f() {
  for (var i = 2; i < 10; i+=1) {
    console.log("i = " + i);
  }
  console.log(i);
}
f();
```

Let and Const are Scoped to the Block

Const is a Constant Reference to a Value

Immutable when Referencing a Primitive

(String, Num, Bool)

Not Immutable Referencing an Object

(Arrays and Objects)

Less Strict Immutability

Let and Const

```
function myFunc() {
  {
    let x;
    {
      // okay, block scoped name
      const x = "sneaky";
      // error, const
      x = "foo";
    }
    // okay, declared with `let`
    x = "bar";
    // error, already declared in block
    let x = "inner";
  }
}
```

Replacing IIFEs with Blocks

Immediately Invoked Function Expression

Allows for scoping

(To not pollute the global space)

Es5 IIFE:

```
(function () {
    var food = 'Meow Mix';
}());

console.log(food); // Reference Error
```

ES6 Blocks:

```
{
  let food = 'Meow Mix';
}

console.log(food); // Reference Error
```

Arrow Functions

From Coffeescript (Fat Arrow)

Shorter Code...

```
function foo(x,y) {
    return x + y;
}

// versus

var foo = (x,y) => x + y;
```

More importantly is impact of `this`

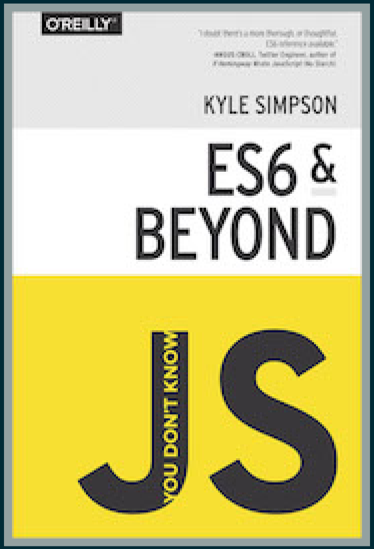
`this` bindings are dynamic so we use the predictability of lexical scope via the self variable.

```
var controller = {
                      makeRequest: function(..){
                        var self = this;
                        btn.addEventListener( "click", function(){
                          // ..
                          self.makeRequest(..);
                        }, false );
                      }
                    };
```

Inside arrow functions, the `this` binding is not dynamic, but is instead lexical

```
var controller = {
                        makeRequest: function(..){
                            btn.addEventListener( "click", () => {
                                // ..
                                this.makeRequest(..);
                            }, false );
                        }
                    };
```

JavaScript.isSexy



O'REILLY

[illegible]

KYLE SIMPSON

ES6 & BEYOND

YOU DON'T KNOW JS

Parameters

Default Parameters

Es5

```
function addTwoNumbers(x, y) {
  x = x || 0;
  y = y || 0;
  return x + y;
}
```

Es6

```
function addTwoNumbers(x=0, y=0) {
  return x + y;
}
```

Rest Parameter

(Indefinite Amount of Args)

Rest Operator (...)

Es5

```
function logArguments() {
  for (var i=0; i < arguments.length; i++) {
    console.log(arguments[i]);
  }
}
```

Es6

```
function logArguments(...args) {
  for (var i=0; i < args.length; i++) {
    console.log(args[i]);
  }
}
```

for loop

```
function logArguments(...args) {
  for (var i=0; i < args.length; i++) {
    console.log(args[i]);
  }
}
```

for...of (Es6)

```
function logArguments(...args) {
  for (let arg of args) {
    console.log(arg);
  }
}
```

Strings

Adds New Methods to the Library

.includes()

Es5

```
var string = 'food';
var substring = 'foo';

console.log(string.indexOf(substring) > -1);
```

Es6

```
const string = 'food';
const substring = 'foo';

console.log(string.includes(substring)); // true
```

.repeat()

Es5

```
function repeat(string, count) {
  var strings = [];
  while(strings.length < count) {
    strings.push(string);
  }
  return strings.join('');
}
```

Es6

```
// String.repeat(numberOfRepetitions)
'meow'.repeat(3); // 'meowmeowmeow'
```

Template Literals

Allows special characters w/o Escaping

Es5

```
var text = "This string contains \"double quotes\" which are escaped.";
```

Es6

```
var text = `This string contains "double quotes" which are escaped.`;
```

String Interpolation

Es5

```
var name = 'Tiger';
var age = 13;

console.log('My cat is named ' + name
          + ' and is ' + age + ' years old.');
```

Es6

```
const name = 'Tiger';
const age = 13;

console.log(`My cat is named ${name} and is ${age} years old.`);
```

New Line Preservation

Es5

```
var text = (
  'cat\n' +
  'dog\n' +
  'nickelodeon'
);
```

Es6

```
let text = ( `cat
dog
nickelodeon`
);
```

Promises

A Promise object represents a value that may not be available yet.

Allow replacing Callbacks with Promises

Makes for more readable Code

Callbacks

```
func1(function (value1) {
  func2(value1, function (value2) {
    func3(value2, function (value3) {
      func4(value3, function (value4) {
        func5(value4, function (value5) {
          // Do something with value 5
        });
      });
    });
  });
});
```

Promises

```
func1(value1)
  .then(func2)
  .then(func3)
  .then(func4)
  .then(func5, value5 => {
    resolve(5+1);// Do something with value 5
    reject();
  })
  .catch(error);
```

Generators

New Type of Function

Standard Function is "Run to Completion"

With ES6 generators, we have a different kind of function

These new functions may be paused, and resumed later

This allows other code to run during these paused periods

It can be paused by using the `yield` keyword inside the Generator

Nothing from the outside of a Generator can stop it

Once paused, only something outside can restart it

You would do this using the `return` statement.

This enables 2-way message passing, to and from the Generator

(Two different naming conventions)

```
function *foo() {
  // ..
}
```

or

```
function* foo() {
  // ..
}
```

It is just a normal function, with different keywords `yield` is referred to as a:

`yield expression`

What we send back in is the result of the `yield expression`

```
function *foo() {
  yield 1;
  yield 2;
  yield 3;
  yield 4;
  yield 5;
}
```

To step through values, we need an iterator

```
var it = foo();
```

We have the .next()

```
var it = foo();

console.log(it.next()); // { value:1, done:false }
```

If we keep interating

```
console.log(it.next()); // { value:2, done:false }
console.log(it.next()); // { value:3, done:false }
console.log(it.next()); // { value:4, done:false }
console.log(it.next()); // { value:5, done:false }
console.log(it.next()); // { value:undefined, done:true }
```

Let's look at a slightly more complex example

```
function *foo(x) {
  var y = 2 * (yield (x + 1));
  var z = yield (y / 3);
return (x + y + z);
}

var it = foo( 5 );

// note: not sending anything into next() here
console.log( it.next() );        // { value:6, done:false }
console.log( it.next( 12 ) );   // { value:8, done:false }
console.log( it.next( 13 ) );   // { value:42, done:true }
```

The Basics Of ES6 Generators

https://davidwalsh.name/es6-generators

Adding ES2015 to Your Rails App

2 gems are Necessary

```
#Gemfile
gem "sprockets"
gem "sprockets-es6"
```

Add to top of `application.js`

```
require 'sprockets/es6'
```

Install Presets

```
npm install babel-preset-es2015 --save-dev
```

Create `.babelrc` config and enable Plugin(s)

```
echo '{ "presets": ["es2015"] }' > .babelrc
```

Es6 functionality is added to any `.es6` file

Instructions From Babel Website

(https://babeljs.io/docs/setup/#rails)

ES2015 Spec