ES5 -> ES2017

THE EVOLUTION OF JAVASCRIPT

Created by Chris Rink

WHY "JAVA"-SCRIPT

THE YEAR - 1995

- **May -** Mocha is invented in Netscape by Brendan Eich
- **September -** Renamed to Livescript
- **December -** Renamed to Javascript (Because Java was popular and the cool kid at that time)

- **1996 -** Javascript is taken to a standardization in ECMA From Now on ECMA is the Spec. Javascript is an implementation.
- **1997 -** ECMA-262 (ECMAScript 1)
- **1998 -** ECMAScript 2 -
- **1999 -** ECMAScript 3 - Now with Regex
- **Early 2000s -** ECMAScript 4 - Abandoned. Because ya know, Politics

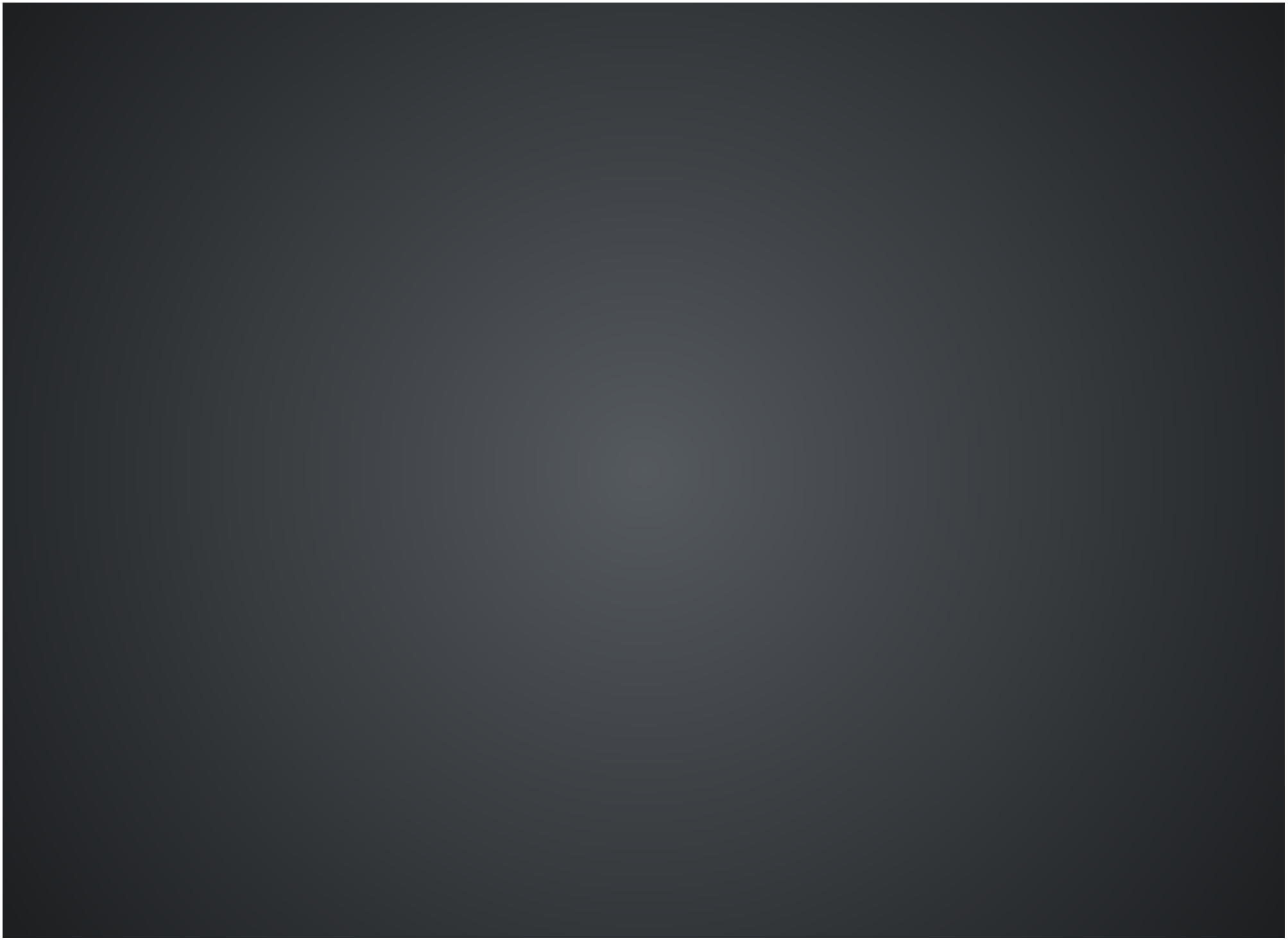
ES5

- **2008** ECMAScript 3.1 Proposed
- **2009** ECMAScript 3.1 proposal renamed to ECMAScript 5

We got

1. Strict Mode
2. Object getter/setter
3. Object Statics - freeze, create, defineProperties, getPrototypeOf
4. Array Statics - IndexOf- Woot!, Map, filter, reduce, forEach
5. Function - bind!

[https://kangax.github.io/compat-table/es5/]



LET THERE BE ES6

Many of the features in ES4 got moved into Harmony(named because people finally got along)

1. Scope
2. Classes
3. Modules
4. Iterators
5. Syntactic Suger

...and so much more

SCOPE ISSUES

```
if(YesNo){ //Function Scope
    var x = 42;
}

console.log(x); // 42 - because variable hoisting

if(YesNo){ //Block Scope
    let x = 42;
}

console.log(x); // Exception: Meaning of Life not found
```

IMMUTABILITY

No guarantee of immutability in ES5.

```
var x = 5;
x= 10;
```

But now we can do

```
const jedi = { isSith: false };
jedi = true; //TypeError

jedi.isSith=true ; //Shouldn't work. Darkside leads to hate
Object.freeze(jedi);
jedi.isSith=true; //Fails, Powerful you have become
```

STRING

```
"Hello".startsWith("Hell"); //TRUE
"Goodbye".endsWith("bye"; //TRUE
"Jar".repeat(2); //JarJar
"abcdef".includes("bcd");
```

NUMBER

```
Number.MAX_SAFE_INTEGER
Number.MIN_SAFE_INTEGER

Number.isNaN();
Number.isFinite();
Number.isInteger();
Number.isSafeInteger();
Number.parseFloat();
Number.parseInt();
```

ARRAY

```
Array.from(gen(6));
// let aRay = Array [0, 1, 2, 3, 4, 5, 6]

let aRay = Array.from(gen(6), function(x){ return x*x});
//[0 ,1, 4, 9, 16, 25, 36]

aRay.keys();
// [0,1,2,3,4,5,6]

aRay.entries();
//[[0,0],[1,1],[2,4], [3,9] ....]
```

THE FOR ...IN PROBLEM

```
//ES5
Array.prototype.foo = 1;
var  aRay = [1, 2, 3]
aRay[5] = 'bar'
for(var i in aRay){
    if(aRay.hasOwnProperty(i)){
        console.log(i);
    }
}

//  '0'
//  '1'
//  '2'
//  '5'
```

- Need to check property is not inherited
- We mix empty indexes. (3,4)?

FOR ...OF

```
//ES6
Array.prototype.foo = 1;
let aRay = [1, 2, 3]
aRay[5] = 'bar'
for(let i of aRay){
   console.log(i);
}

//  '0'
//  '1'
//  '2'
// 'undefined';
// 'undefined';
//  '5'
```

-Better

ITERATOR

```
let it = [1, 2, 3][Symbol.interator]();
it.next(); // {value 1, done: false}
it.next(); // {value 2, done: false}
it.next(); // {value 3, done: false}
it.next(); // {value undefined, done: true}
```

GENERATORS

```
let factory = function*(){
    yield 1;
    yield 2;
    yield 3;
    yield 4;
}

let it = factory();
it.next(); // {value 1, done: false}
it.next(); // {value 2, done: false}
it.next(); // {value 3, done: false}
it.next(); // {value undefined, done: true}
```

Generators are factories for Iterators

GENERATORS ARE ITERATOR AND ITERABLE

```
function zeroOneTwo(){
   return [0,1,2]
}
//or
function zeroOneTwo(){
    yield 1;
    yield 2;
    yield 3
}
```

Both output the same

```
var aRay = zeroOneTwo();
for(var i of aRay){
    console.log(i);
}
// '1', '2', '3'
```

NEW FUNCTION SYNTAX

```
//ES5
function estimateStory(factor){
    return function(factor, estimate = 1){
        return factor * estimate;
    }
}

let chrisEstimate = estimateStory(2*2);
const story = chrisEstimate(3);
// Outputs 12
```

```
//ES6
chrisEstimate = (factor) => (factor,estimate = 1) => {
    return factor * estimate;
};
```

FUNCTION DECLARATIONS

```
let log = statement => console.log(statement);

let namedLog = (stmt,name) => console.log(`$(name) said: $(stmt)`

let complexLog = (statement) => {
    if(statement.includes("error")){
        console.error(statement);
    }else{
        console.log(statement);
    }

}
```

PHAT ARROW

```
doThis(callback() => {
    return this.doThat();
});

//is eqivilent to:
callback.bind(this);

//or
$.proxy(callback,function() {
        return this.doThat();
    },this);
```

FUNCTION DEFAULTS

```
//ES5
return function(factor, estimate){
    if(!estimate){
        estimate = 1;
    }
    return factor * estimate;
}
//ES6
return function(factor, estimate = 1){
    return factor * estimate;
}
return function(factor, estimate = Math.rand()){
    return factor * estimate;
}
```

We can use functions, functions are lazy

THE ...REST OPERATOR

```
let [ , , ...y] = [1, 2, 3, 4, 5];
// y = [3, 4, 5];

function sum(...nums){
    console.log(nums);
    return nums.reduce((total,num) => total + num),0)
}

const tots = sum(1,2,3,4,5);
//  LOG: [1,2,3,4,5]
//  tots = 15;
```

THE ...SPREAD OPERATOR

```
//These are the same
Object.assign === $.extend === _.extend

They can be used to clone and shallow merge
let merged = Object.assign({},{a:1},{a:2, b:3});
//merged === { a:2, b:3 }

But now we can do:

merged = {
    ...a,
    ...b
}

This also works on functions
function f(x, y, z) {
```

OBJECT UPGRADES

Now you can do:

```
const hasLightSaber = true;
const property = 'hasBeard';
const master = {
 __proto__: jedi,  //Sets the prototype that you extend from.
 [property]: true,  //computed keys
 hasLightSaber, // shorthand for adding key using function key.
 forcePush(){}  // functions too!
}

master.isSith; //false
master.hasBeard; //true
master.hasLightsaber; //true
master.forcePush() //KaPOWWWW!
```

GET CLASSY

We finally have OO

```
class Jedi{
    constructor(){
        this.isSith= false; //this gets called when I call "new"
    }

    toString(){
        return `Anger is the path to ${(this.isSith) ? "being Awe
    }
}

class Sith extends Jedi{
    constructor(){
        super();
       this.isSith = true;
    }
}
```

GET CLASSIER

```
let yoda = new Jedi();
let vader = new Sith();

console.log(yoda.toString());
//Anger is the path the the darkside.

console.log(vader.toString());
//Anger is the path to being Awesome!

console.log(vader instanceOf Sith); // TRUE
console.log(vader instanceOf Jedi); // Also TRUE
```

MODULES EXPORTING

AMD to ES6 Equivalent

```
//sets primary export. similar
define(['jquery'],function($){
    return myFunction
}
export default myFunction
```

But now we can do:

```
export default myFunction;
export otherFunction;
export andAnother;
```

MODULES IMPORTING

AMD to ES6 Equivalent

```
//sets primary export. similar
import name from path
```

But now we can do:

```
import {f1, f2 } from path

import * from path
require([path],function(es6){
   const main = es6.default
}
```

THE ASYNC PROBLEM

What we want to do.

```
function getData(cb){
    let results = requestFromServer();
    let values = formatResults(results);
    return calulcatedValues(values);
}
```

But these are async calls, so how do we work around it?

CALLBACK HELL

Chaining Async functions sucks

```
function getData(cb){
    requestFromServer(
        formatResults(
            calculateValues(cb)
        )
    )
}
```

We also need to handle errors at each level.

THE PROMISED LAND

```
function getData(cb){
    return requestFromServer
        .then(function(res){
            return formatResults(res);
        })
        .then(function(values){
            return calulateValues(values)
        })
    )
}
```

Better - Promise changes allow use to bubble errors up.

PROMISES IN PARALLEL

We can also do parallel operations.

```
function getDinner(){
return Promise.all(
        callJoe(),
        callAngie(),
        callMom(),
        callDad(),
        "Spicy Bowl"
    )
```

Returns a promise!

GENERATORS*

```
function* getData(cb){
     let results = yield requestFromServer();
     let values = yield formatResults(results);
     yield calulcatedValues(values);
}
```

Better - Getting closer but I need some middleware to run this.

ASYNC-AWAIT

```
async function getData(cb){
     let results = await requestFromServer();
     let values = await formatResults(results);
     return calulcatedValues(values);
}
```

WHAT!!!! This is what we wanted at the beginning. backed by promises.

NOT COVERED

- Symbols
- Map/WeakMap
- Set/WeakSet
- Proxy
- Reflection

REFERENCES

[https://babeljs.io/learn-es2015/]
[https://www.slideshare.net/hesher/es2015-es6-overview]
[https://www.slideshare.net/domenicdenicola/es6-the-awesome-parts] [https://github.com/joshburgess/not-awesome-es6-classes]