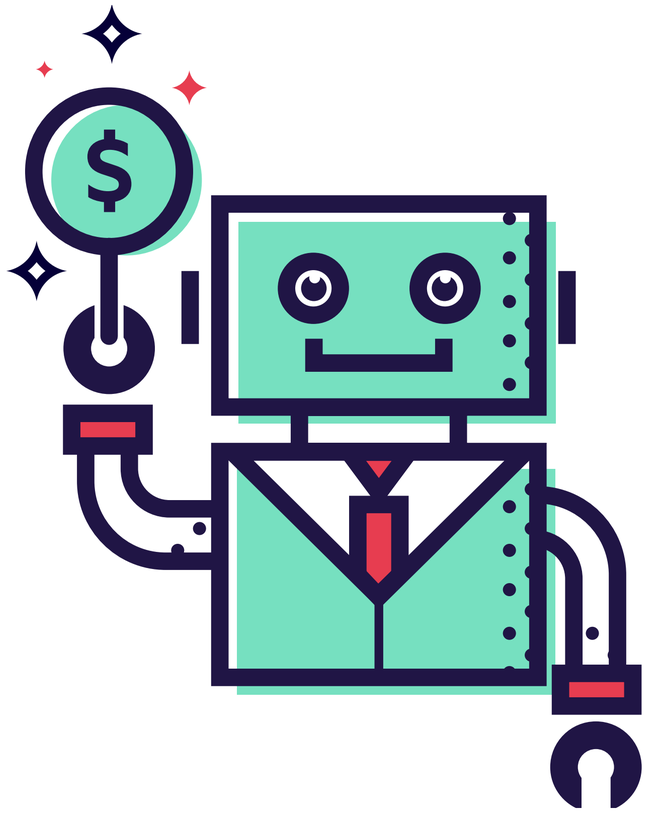
SHARETHROUGH

ES6 For Fun & Profit

Mark Meyer

Software Engineer at Sharethrough

mmeyer@sharethrough.com



What is ECMAScript 6?

Also known as ECMAScript2015

Latest standard of ECMAScript, ratified in June 2015. It's the first update since ECMAScript 5 in 2009. The fifth ratified version of the spec. (ECMAScript 4 was abandoned due to being too ambitious)

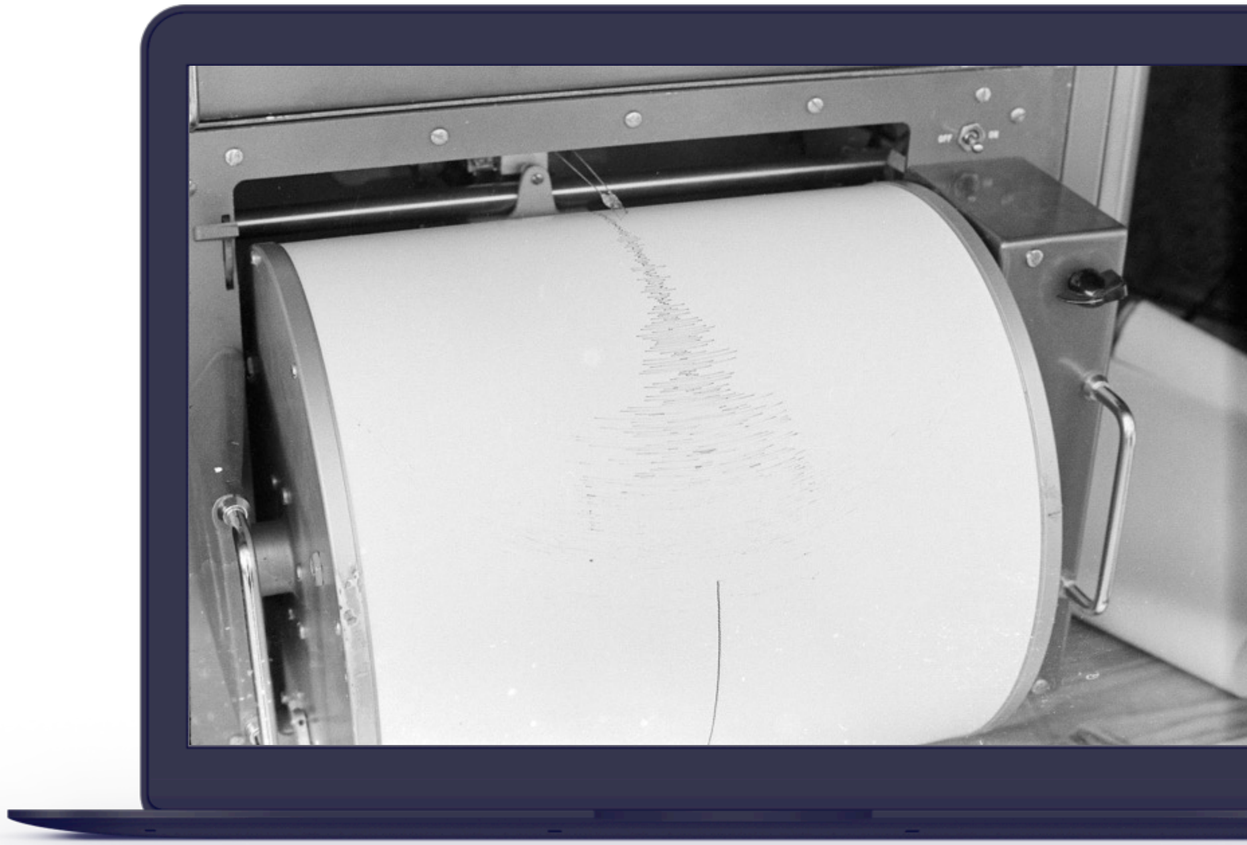
What does the spec cover?

- Syntax - parsing rules, keywords, operators, etc.
- Types - boolean, string, number, etc.
- Prototypes and Inheritance
- The Standard Library - JSON, Math, Array methods

SHARETHROUGH

ES6 Will Change the Way You Write JS Code

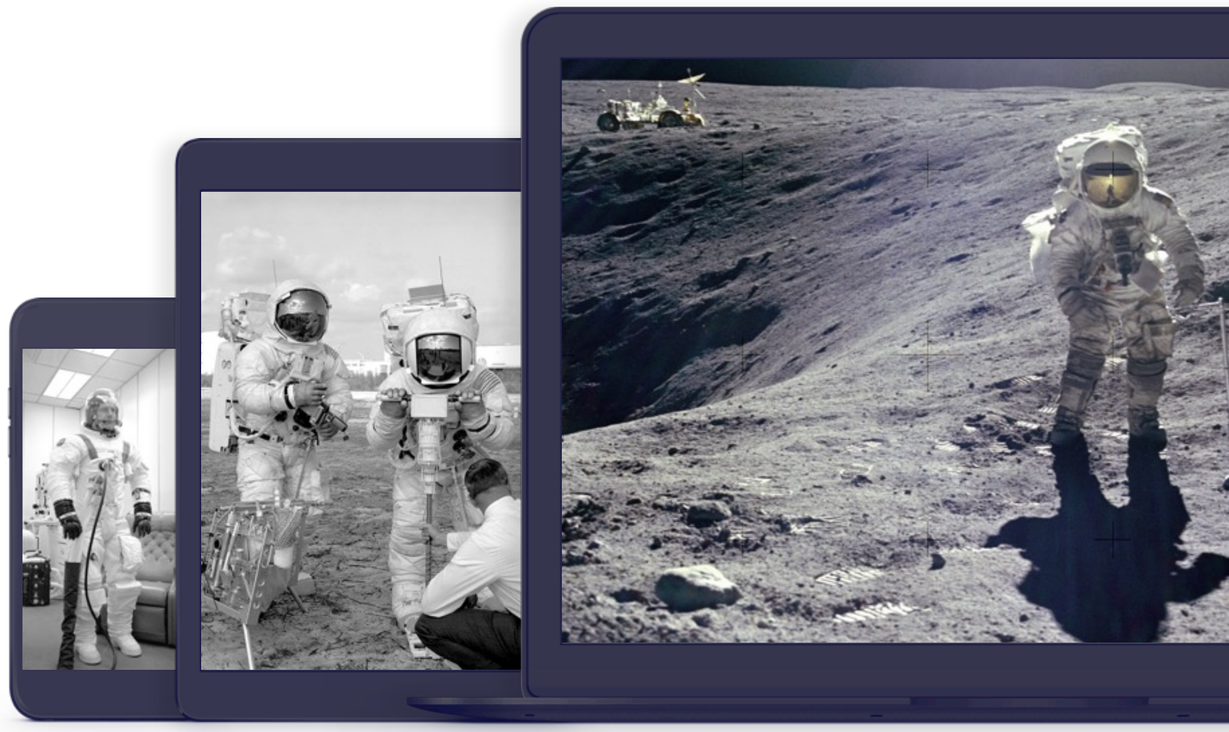
Convenience Changes to Brain Melting Concepts



Quoted From Jason Orendorff

Influences From Many Languages

- CoffeeScript
- Python
- C++
- Java
- BASIC
- E
- Lisp
- Smalltalk
- CommonJS, AMD

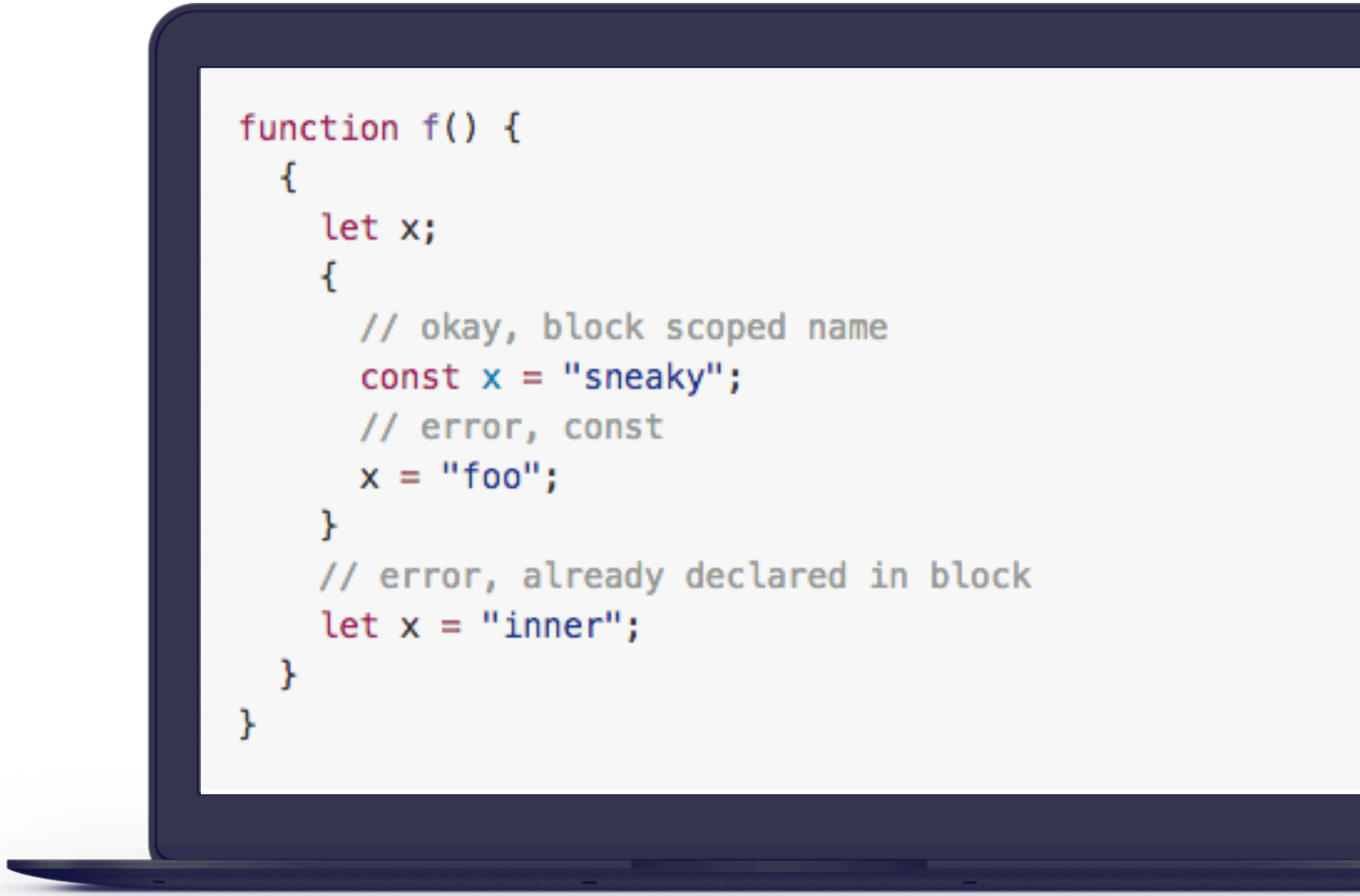


Let’s get to the features!

Let & Const

Let is the new *var*

- Block Scoped
- Not hoisted
- Not added to global object
- Redeclaration is a syntax error
- Loops created a fresh binding of the variable
- Const variables can only be assigned once at declaration



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

```
var MyClass = (function() {

  // module scoped symbol
  var key = Symbol("key");

  function MyClass(privateData) {
    this[key] = privateData;
  }

  MyClass.prototype = {
    doStuff: function() {
      ... this[key] ...
    }
  };

  return MyClass;
})();

var c = new MyClass("hello")
c["key"] === undefined
```

Symbols

The First Primitive Type since ES1

- Unique, Immutable values
  - Symbol('foo') !== Symbol('foo')
- Can't assign properties
- Used like Strings as names for object properties
- Exposed via reflection *Object.getOwnPropertySymbols*

Arrows

CoffeeScript's best feature goes standard

- Share lexical *this* with surrounding code.

```
// Expression bodies
var odds = evens.map(v => v + 1);
var nums = evens.map((v, i) => v + i);
var pairs = evens.map(v => ({even: v, odd: v + 1}));

// Statement bodies
nums.forEach(v => {
  if (v % 5 === 0)
    fives.push(v);
});

// Lexical this
var bob = {
  _name: "Bob",
  _friends: [],
  printFriends() {
    this._friends.forEach(f =>
      console.log(this._name + " knows " + f));
  }
}
```

Template Strings

String Interpolation Arrives

Adds support for Multiline Strings, Interpolation, and Tags to prevent injection

Classes

Syntactic Sugar on Prototypes

- Inheritance
- Constructors
- Super Calls
- Instance Methods
- Class (Static) Methods
- Getters & Setters

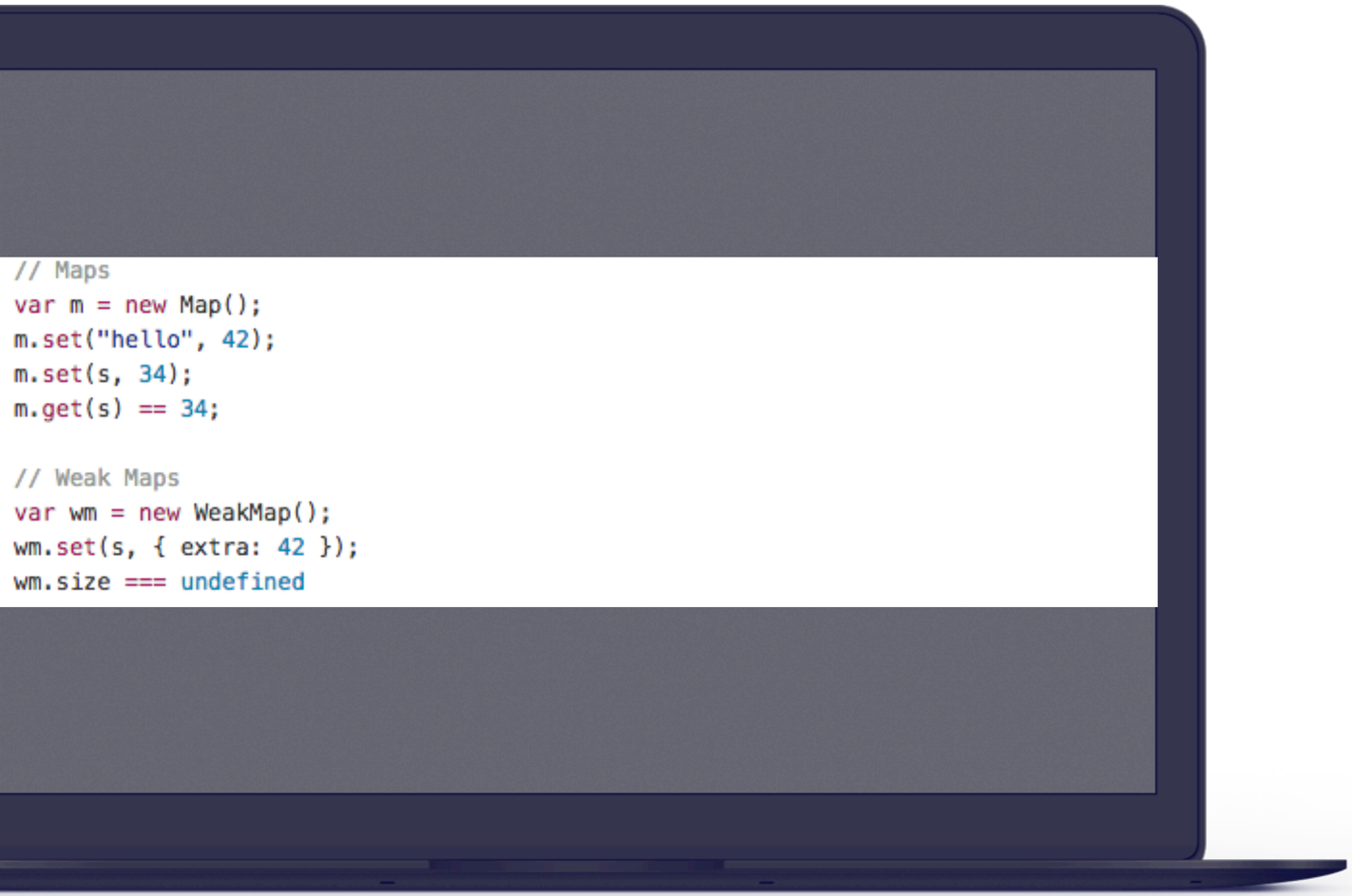
```
class SkinnedMesh extends THREE.Mesh {
  constructor(geometry, materials) {
    super(geometry, materials);

    this.idMatrix = SkinnedMesh.defaultMatrix();
    this.bones = [];
    this.boneMatrices = [];
    //...
  }
  update(camera) {
    //...
    super.update();
  }
  get boneCount() {
    return this.bones.length;
  }
  set matrixType(matrixType) {
    this.idMatrix = SkinnedMesh[matrixType]();
  }
  static defaultMatrix() {
    return new THREE.Matrix4();
  }
}
```

Subclassing Built-In Classes

Array, Date, and DOM Elements can be subclassed

At vero eos et accusamus et iusto odio dignissimos ducimus qui blanditiis praesentium voluptatum deleniti atque corrupti quos dolores et quas molestias excepturi sint occaecati cupiditate.

```
// Pseudo-code of Array
class Array {
    constructor(...args) { /* ... */ }
    static [Symbol.create]() {
        // Install special [[DefineOwnProperty]]
        // to magically update 'length'
    }
}

// User code of Array subclass
class MyArray extends Array {
    constructor(...args) { super(...args); }
}

// Two-phase 'new':
// 1) Call @@create to allocate object
// 2) Invoke constructor on new instance
var arr = new MyArray();
arr[1] = 12;
arr.length == 2
```



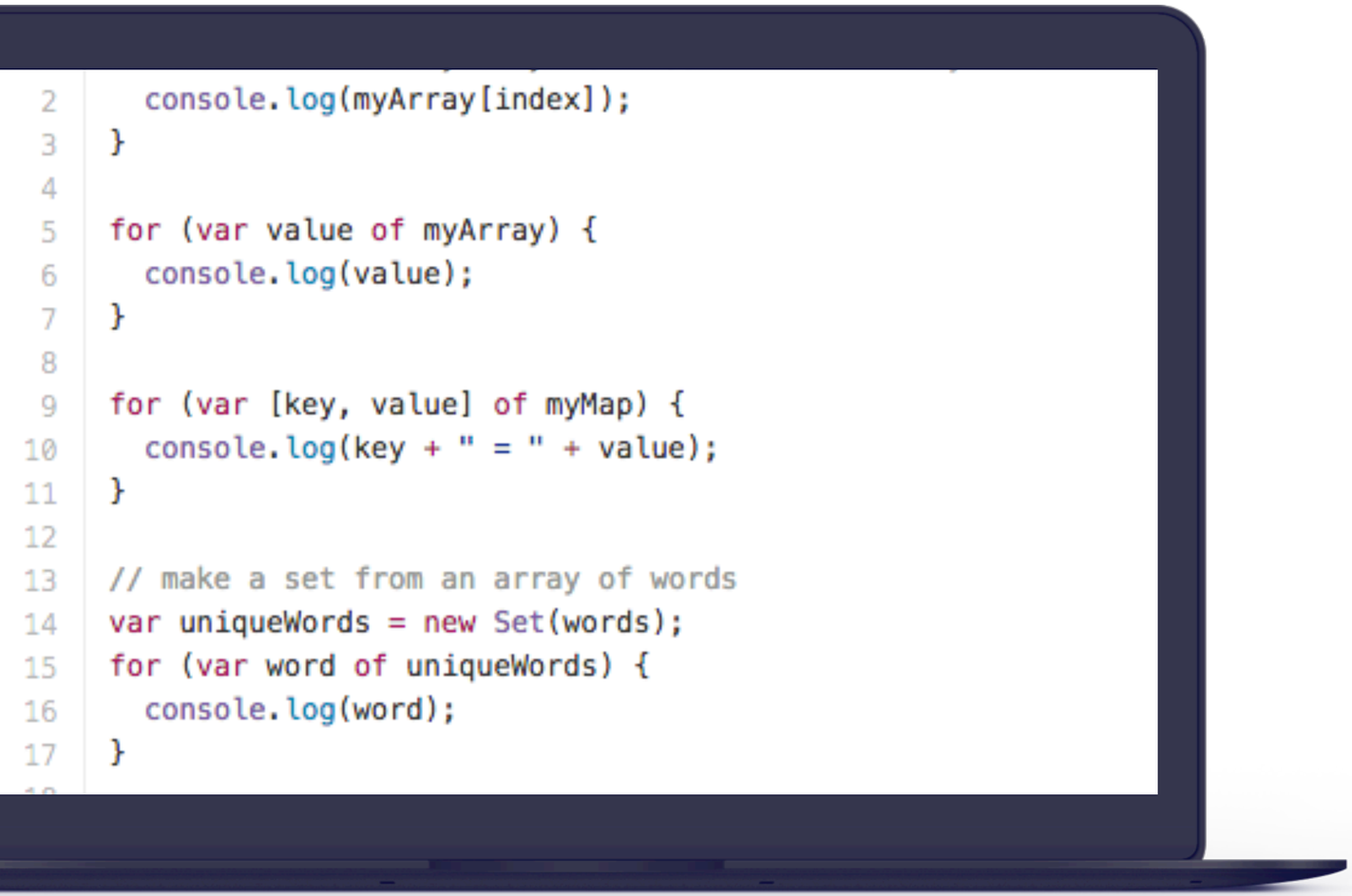
Maps + WeakMaps

- Key, Value Stores
- Maps have no prototype, whereas objects have a prototype, thus have default keys
- Keys of an object can only be Strings or Symbols
- Keys of a map can be any type
- Maps have *size* method to get number of key value pairs
- WeakMaps can only have object keys
- WeakMap keys are weakly held, so they will be GC’d if no other reference exists

Sets + WeakSets

Unique, Iterable Collection

- Collection of unique values
- Can be iterated in insertion order
- Sets have *size* method to get number of key value pairs
- WeakSets can contain objects only
- WeakSets create no strong references to the objects



```
  console.log(myArray[index]);
}

for (var value of myArray) {
  console.log(value);
}

for (var [key, value] of myMap) {
  console.log(key + " = " + value);
}

// make a set from an array of words
var uniqueWords = new Set(words);
for (var word of uniqueWords) {
  console.log(word);
}
```

For … Of

For … In mistakes no more!

Iterate over objects, sets, and maps retrieving what you probably meant all along

Custom Iteration

Customize For…Of Iteration with custom iterator method

```
let fibonacci = {
  [Symbol.iterator]() {
    let pre = 0, cur = 1;
    return {
      next() {
        [pre, cur] = [cur, pre + cur];
        return { done: false, value: cur }
      }
    }
  }
}

for (var n of fibonacci) {
  // truncate the sequence at 1000
  if (n > 1000)
    break;
  console.log(n);
}
```

```
function timeout(duration = 0) {
    return new Promise((resolve, reject) => {
        setTimeout(resolve, duration);
    })
}

var p = timeout(1000).then(() => {
    return timeout(2000);
}).then(() => {
    throw new Error("hmm");
}).catch(err => {
    return Promise.all([timeout(100), timeout(200)]);
})
```

Promises

No more libraries required

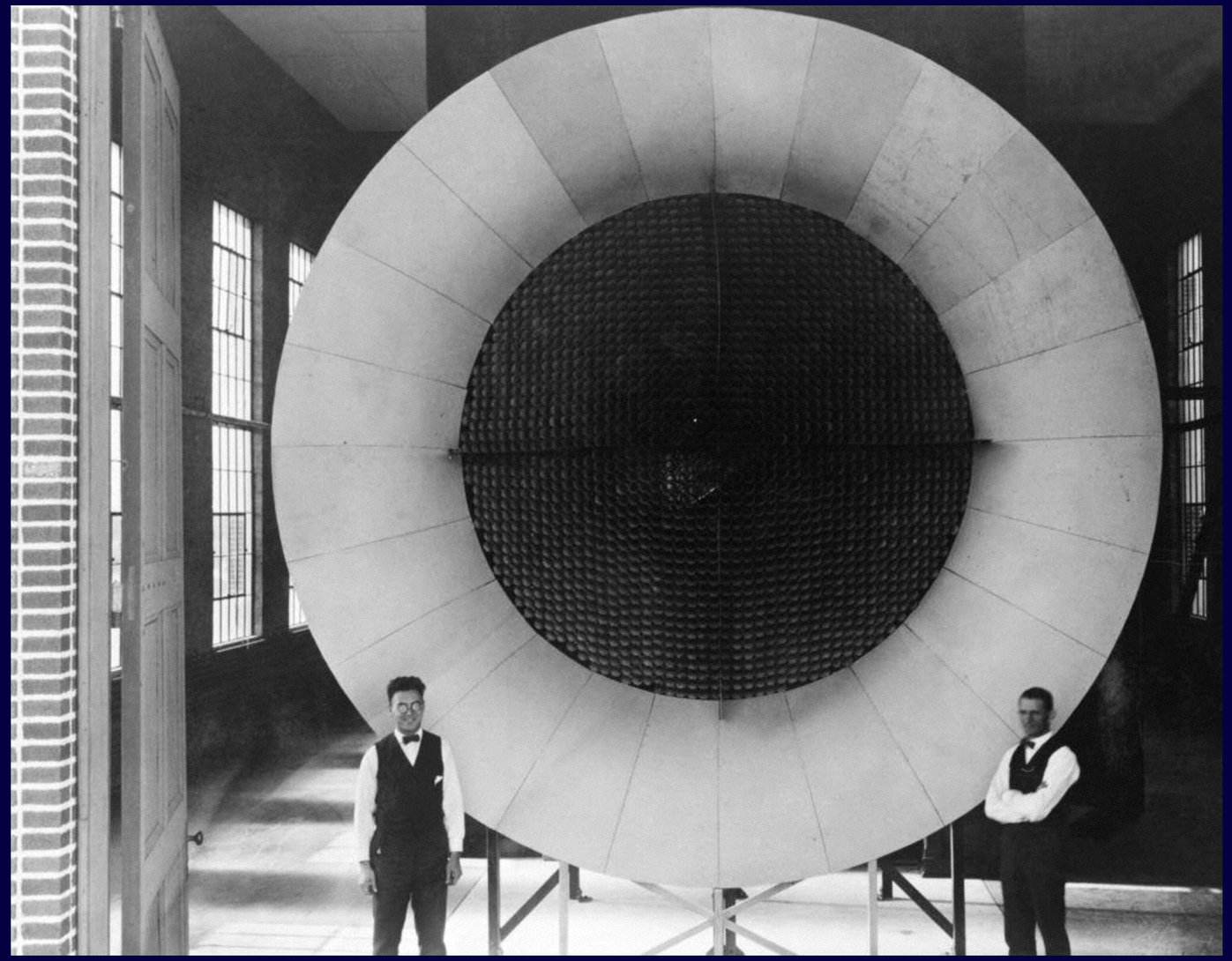
Modules

- Export any top-level function, class, var, let, or const.
- Implicitly async model – no code executes until requested modules are available and processed.

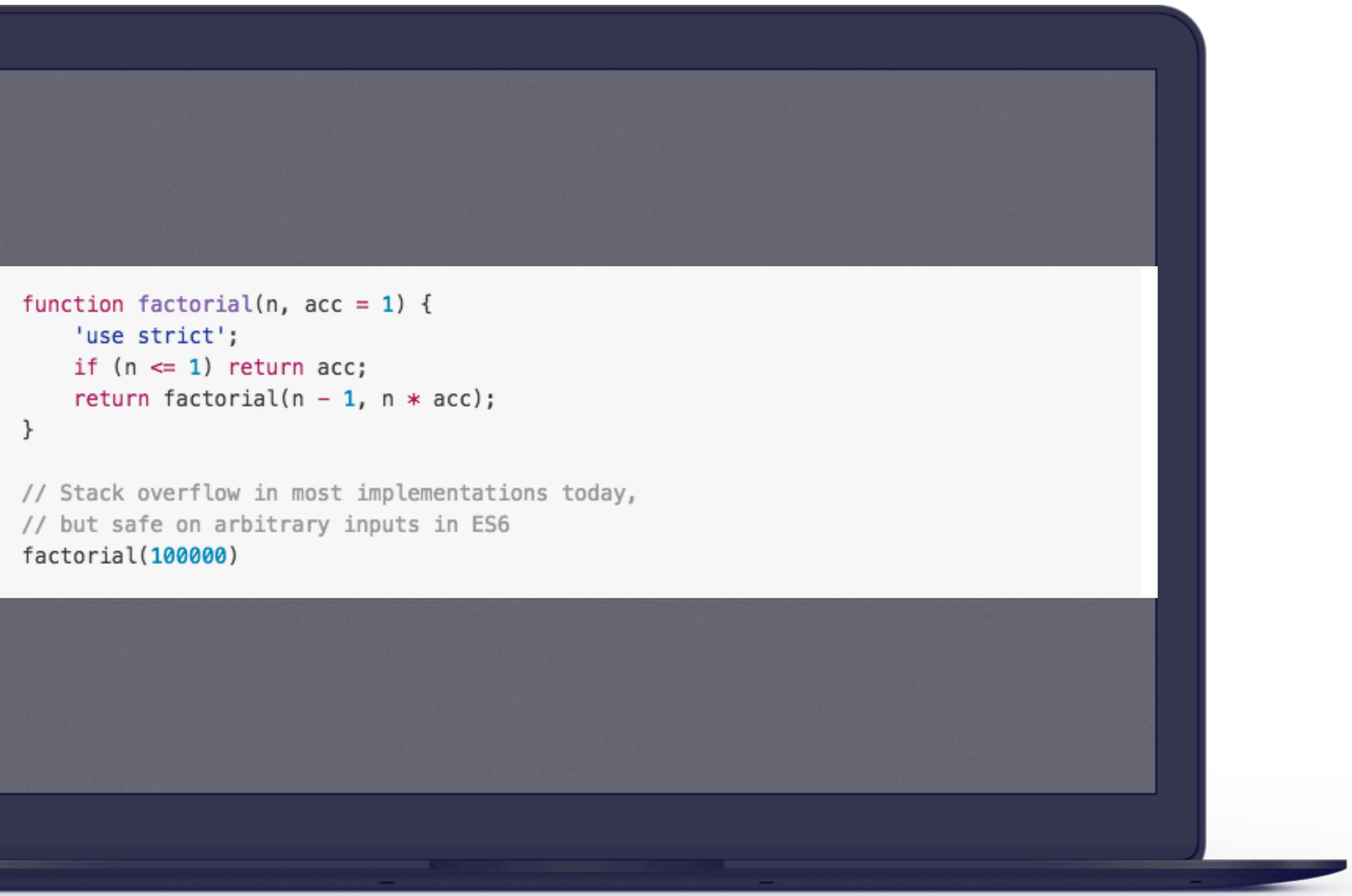
```
// lib/math.js
export function sum(x, y) {
  return x + y;
}
export var pi = 3.141593;
```

```
// app.js
import * as math from "lib/math";
alert("2π = " + math.sum(math.pi, math.pi));
```

```
// otherApp.js
import {sum, pi} from "lib/math";
alert("2π = " + sum(pi, pi));
```



And Now For Some Brain Melting



Tails Calls

Stack Overflow No Mo'!

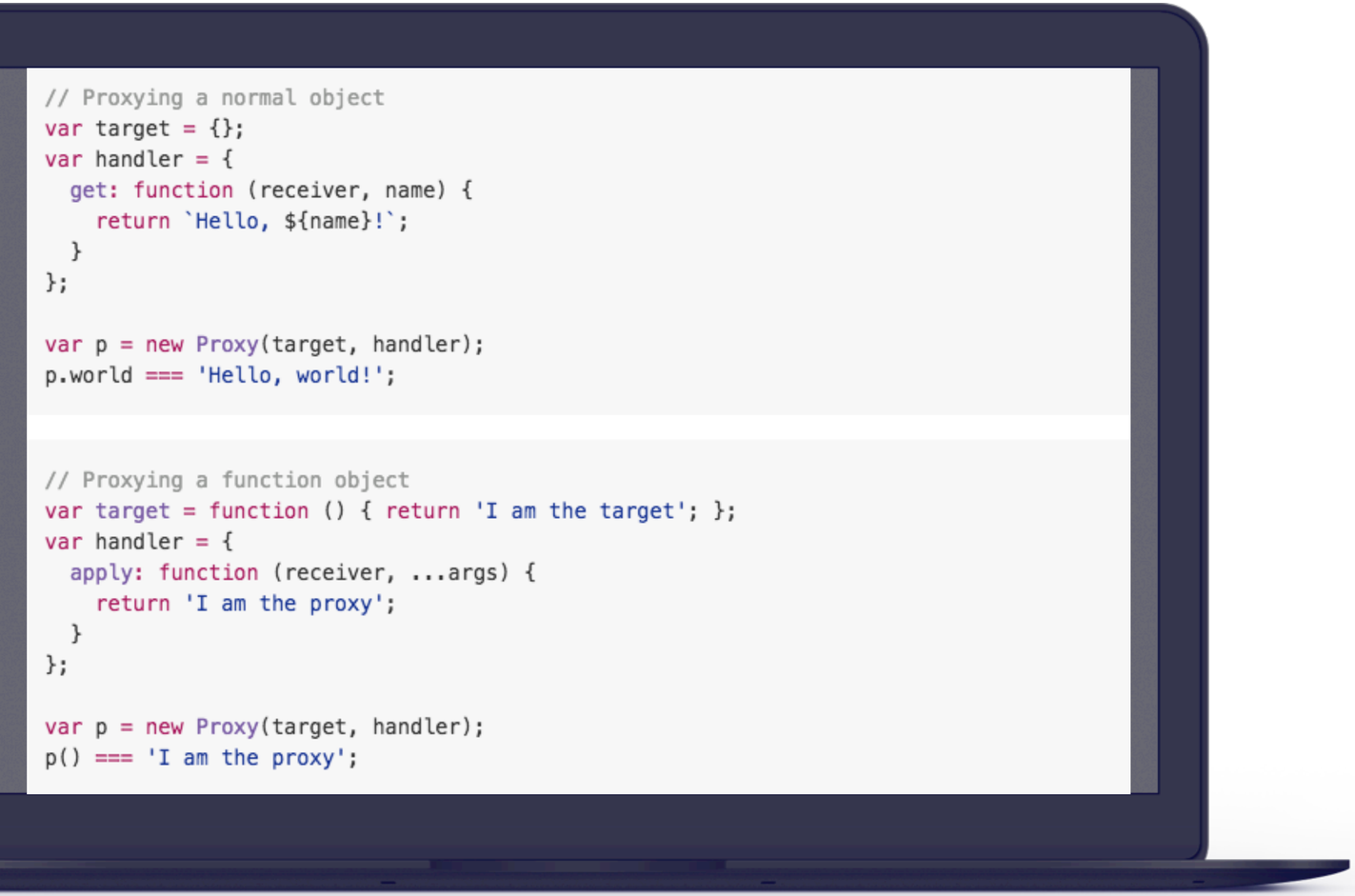
Recursive algorithms guaranteed to not grow the stack unboundedly

Destructuring

Binding via Pattern Matching

- Useful on arrays or objects
- Soft fail returning *undefined* like a normal property lookup

```
// list matching
var [a, , b] = [1,2,3];

// object matching
var { op: a, lhs: { op: b }, rhs: c }
       = getASTNode()

// object matching shorthand
// binds `op`, `lhs` and `rhs` in scope
var {op, lhs, rhs} = getASTNode()

// Can be used in parameter position
function g({name: x}) {
  console.log(x);
}
g({name: 5})

// Fail-soft destructuring
var [a] = [];
a === undefined;

// Fail-soft destructuring with defaults
var [a = 1] = [];
a === 1;
```



```
// Proxying a normal object
var target = {};
var handler = {
  get: function (receiver, name) {
    return `Hello, ${name}!`;
  }
};

var p = new Proxy(target, handler);
p.world === 'Hello, world!';


// Proxying a function object
var target = function () { return 'I am the target'; };
var handler = {
  apply: function (receiver, ...args) {
    return 'I am the proxy';
  }
};

var p = new Proxy(target, handler);
p() === 'I am the proxy';
```

Proxies

Wrap your objects for more power

- Intercept calls and redirect
- Log when accessing methods
- Profile how long calls take
- Very useful for mocks and stubs like Jasmine implements

Generators

Yield to the power

- A subclass of iterators with a *next* and *throw* interface
- *Yield* returns a value while the generator object maintains the current stack frame so that it can be called back into
- Not multithreaded

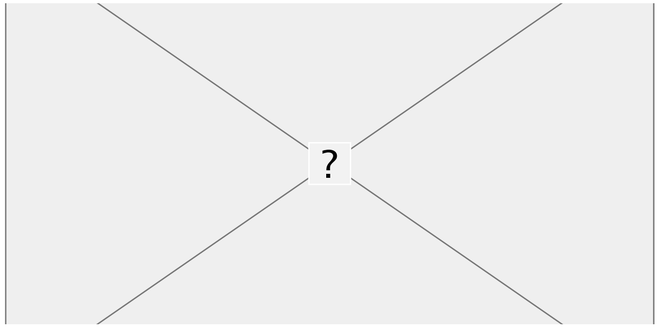
```
var fibonacci = {
  [Symbol.iterator]: function*() {
    var pre = 0, cur = 1;
    for (;;) {
      var temp = pre;
      pre = cur;
      cur += temp;
      yield cur;
    }
  }
}

for (var n of fibonacci) {
  // truncate the sequence at 1000
  if (n > 1000)
    break;
  console.log(n);
}
```

ES6 FTW!

A major upgrade that will change the way you write javascript

ES6 In Depth

https://hacks.mozilla.org/category/es6-in-depth/



ES6 Features

https://github.com/lukehoban/es6features#unicode

New Old Stock

http://nos.twnsnd.co/