
ES6: THE FUTURE OF JAVASCRIPT
Created by Adam Epling

AGENDA

- What is it?
- ES6 Additions
  - Sugar
  - Built-in Lib Improvements
  - Promises
  - Generators
  - Classes
  - Modules
- How Can I Use This Right Now?

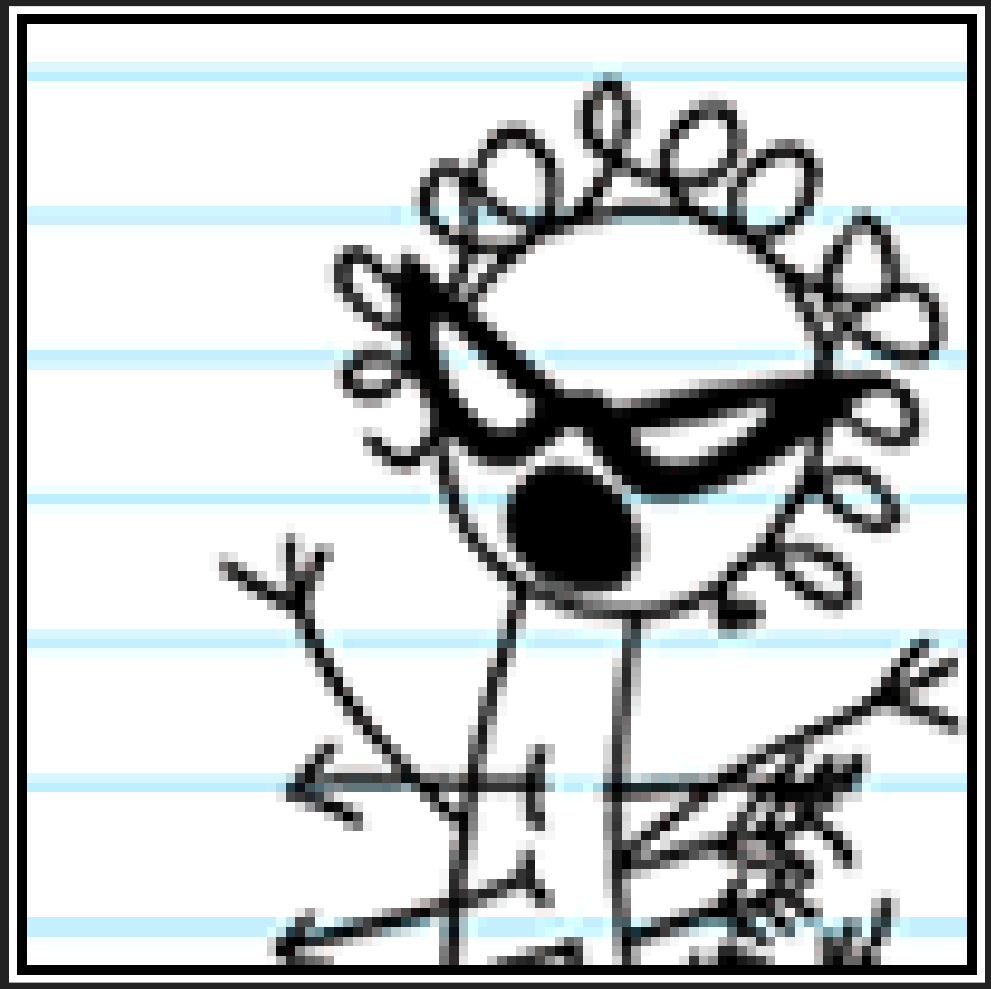
WHAT IS ECMASCRIPT?

- ECMAScript is the international standard that defines JavaScript (and languages like ActionScript have implemented it as well).
- It's up to each browser vendor to implement the standard
- ES6 has been in the works in some form or another since 2008. It was "feature frozen" in Aug. 2014; the spec is in the process of being finalized, with a target publish date in June 2015.
- Major browsers have already rolled out many of the proposed features in preparation.

SO LET'S LOOK AT SOME OF WHAT WE'LL BE GETTING WITH ES6...


FYDISNEYMISFITS | TK-THE-
SUGAR

ARROW FUNCTIONS



ARROW FUNCTIONS

TAKE THIS:

```
var add = function (a, b) {
  return a + b;
}
```

AND TURN IT INTO THIS:

```
var add = (a, b) => a + b;
```


LEXICAL "THIS"
This guy!

THE PROBLEM

```
function Person() {
  // The Person() constructor defines `this` as itself.
  this.age = 0;

  setInterval(function growUp() {
    // In nonstrict mode, the growUp() function defines `this`
    // as the global object, which is different from the `this`
    // defined by the Person() constructor.
    this.age++;
  }, 1000);
}

var p = new Person();
```

UNTIL NOW, WE GOT AROUND IT BY DOING THIS...

```
function Person() {
  var self = this; // Some choose `that` instead of `self`.
                   // Choose one and be consistent.
  self.age = 0;

  setInterval(function growUp() {
    // The callback refers to the `self` variable of which
    // the value is the expected object.
    self.age++;
  }, 1000);
}
```

BUT IN ES6, IT'S ALL GOOD...

```
function Person() {
  this.age = 0;

  setInterval(() => {
    this.age++; // |this| properly refers to the person object
  }, 1000);
}

var p = new Person();
```


LET, CONST, AND
BLOCK SCOPE

FUNCTIONAL SCOPE

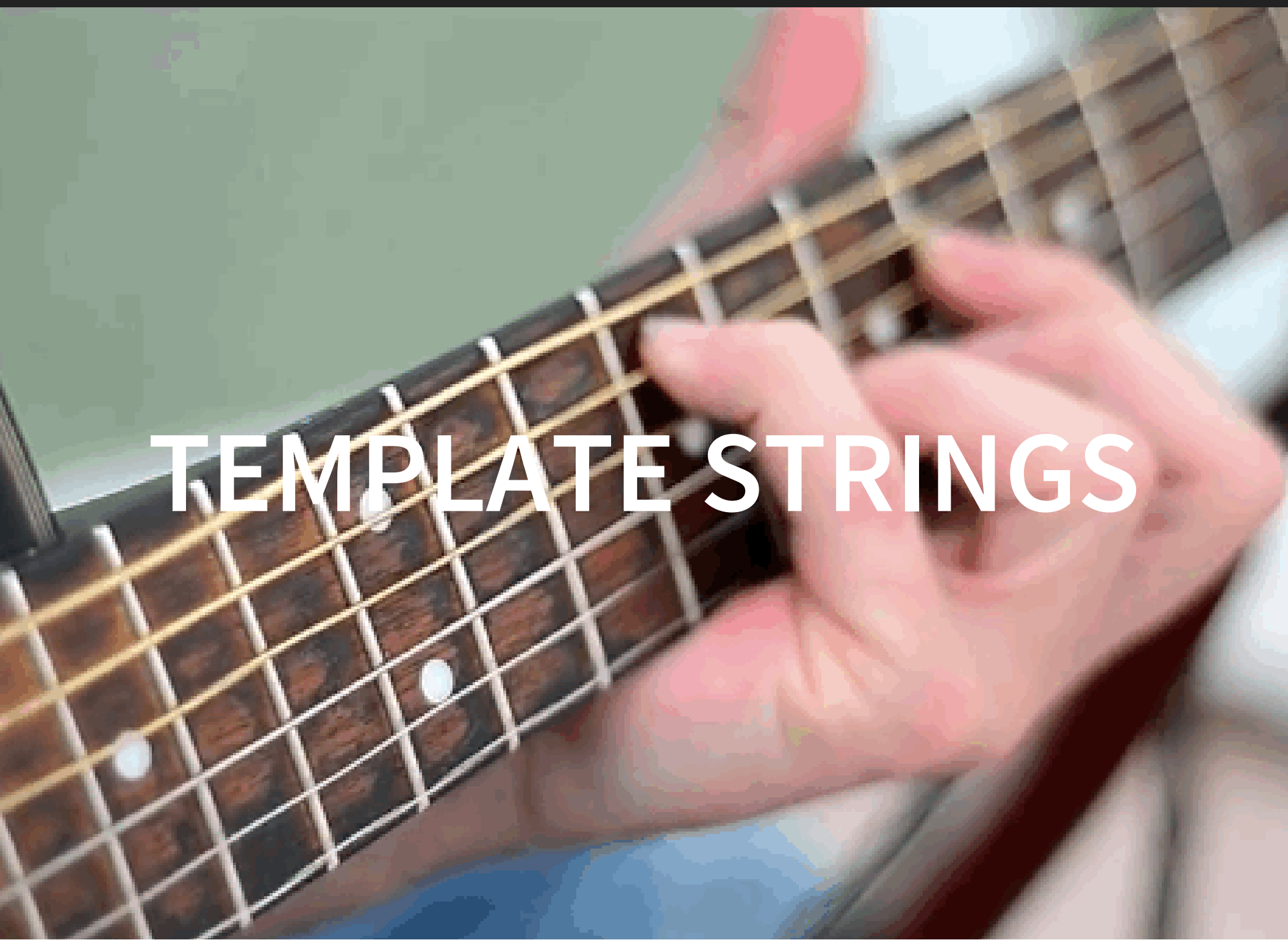
```
function () {
  for (var i = 0; i < 5; i++) {
    //...
  }
  // Will print 5, even though
  // we're outside of the block
  // in which i was declared.
  console.log(i);
}
```

BLOCK SCOPING WITH "LET"

```
function () {
  for (let i = 0; i < 5; i++) {
    //...
  }
  // ReferenceError: i is not defined
  console.log(i);
}
```

CONSTANTS

```
function () {
  const hooray = 'Hooray!';
  hooray = 'Boo!';

  // In strict mode, you'll get
  // TypeError: Assignment to constant variable.
  // In non-strict mode, prints 'Hooray!'
  console.log(hooray);
}
```


TEMPLATE STRINGS

WHAT WE DO NOW...

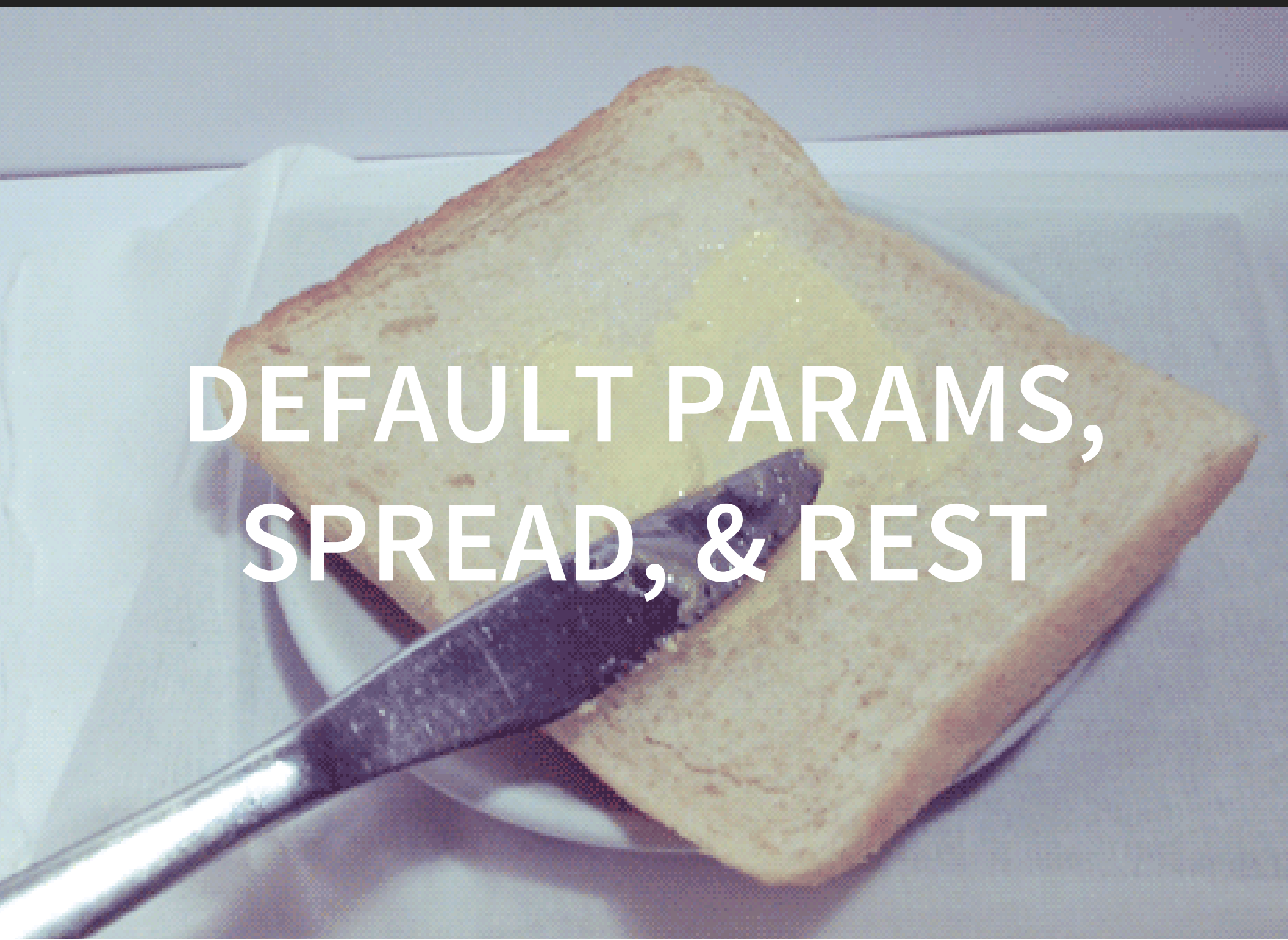
```
var name = 'Roger';
var job = 'Shrubber';
console.log('My name is ' + name + '. I am a ' + job +
      '.\n They call me "' + name + ' the ' + job + '".');
// My name is Roger. I am a Shrubber.
// They call me "Roger the Shrubber".
```

IN ES6...

```
var name = 'Roger';
var job = 'Shrubber';
console.log(`My name is ${name}. I am a ${job}.
             They call me "${name} the ${job}".`);
// My name is Roger. I am a Shrubber.
// They call me "Roger the Shrubber".
```

FEATURES

- Interpolation with ${}
- Multiline
- No need to escape special characters


DEFAULT PARAMS,
SPREAD, & REST

DEFAULT PARAMS

```
var f = function (x, y=12) {
  return x + 7;
}

// Or, using an arrow function...
var f = (x, y=12) => x + y;

console.log(f(3)); // 15
```

SPREAD OPERATOR

```
function addThree(x, y, z) {
  return x + y + z;
}
var nums = [1, 2, 3];
addThree(...nums); // returns 6

var parts = ['shoulder', 'knees'];
var lyrics = ['head', ...parts, 'and', 'toes'];
```

SPREAD OPERATOR

```
// A better "push"
// ES5 way to add to existing array
var arr1 = [0, 1, 2];
var arr2 = [3, 4, 5];
// Append all items from arr2 onto arr1
Array.prototype.push.apply(arr1, arr2);

// ES6 way
var arr1 = [0, 1, 2];
var arr2 = [3, 4, 5];
arr1.push(...arr2);
```

REST PARAMETERS

```
function sortRestArgs(...theArgs) {
  return theArgs.sort();
}

console.log(sortRestArgs(5,3,7,1)); // 1,3,5,7

function add(...nums) {
  return nums.reduce(function (a, b) {
    return a + b;
  });
}

console.log(add(1,2,3,4)); // 10
```


BUILT-IN LIBRARY
IMPROVEMENTS

ARRAYS

```
var nums = [1, 2, 3, 4, 5];
var sum = 0;
for (var i of nums) {
  sum += i;
}
console.log(sum) // 15

Array.from(document.querySelectorAll('*')) // Returns a real Array
[0, 0, 0].fill(7, 1) // [0,7,7]
[1, 2, 3].find(x => x == 3) // 3
[1, 2, 3].findIndex(x => x == 2) // 1
["a", "b", "c"].entries() // iterator [0, "a"], [1,"b"], [2,"c"]
["a", "b", "c"].keys() // iterator 0, 1, 2
["a", "b", "c"].values() // iterator "a", "b", "c"
```

MAPS & SETS

```
// Sets
var s = new Set();
s.add("hello").add("goodbye").add("hello");
s.size === 2;
s.has("hello") === true;

// Maps
var m = new Map();
m.set("hello", 42);
m.set(s, 34);
m.get(s) == 34;
```

STRINGS

```
var getty = 'Four score and seven years ago';

console.log(getty.includes('ore')); // true
console.log(getty.startsWith('Five')); // false
console.log(getty.endsWith('go')); // true

var ha = "HA";
console.log(ha.repeat(4)); // HAHAHAHA
```


PROMISES
ReactionGIFS

WHERE WE ARE NOW:

CALLBACKS

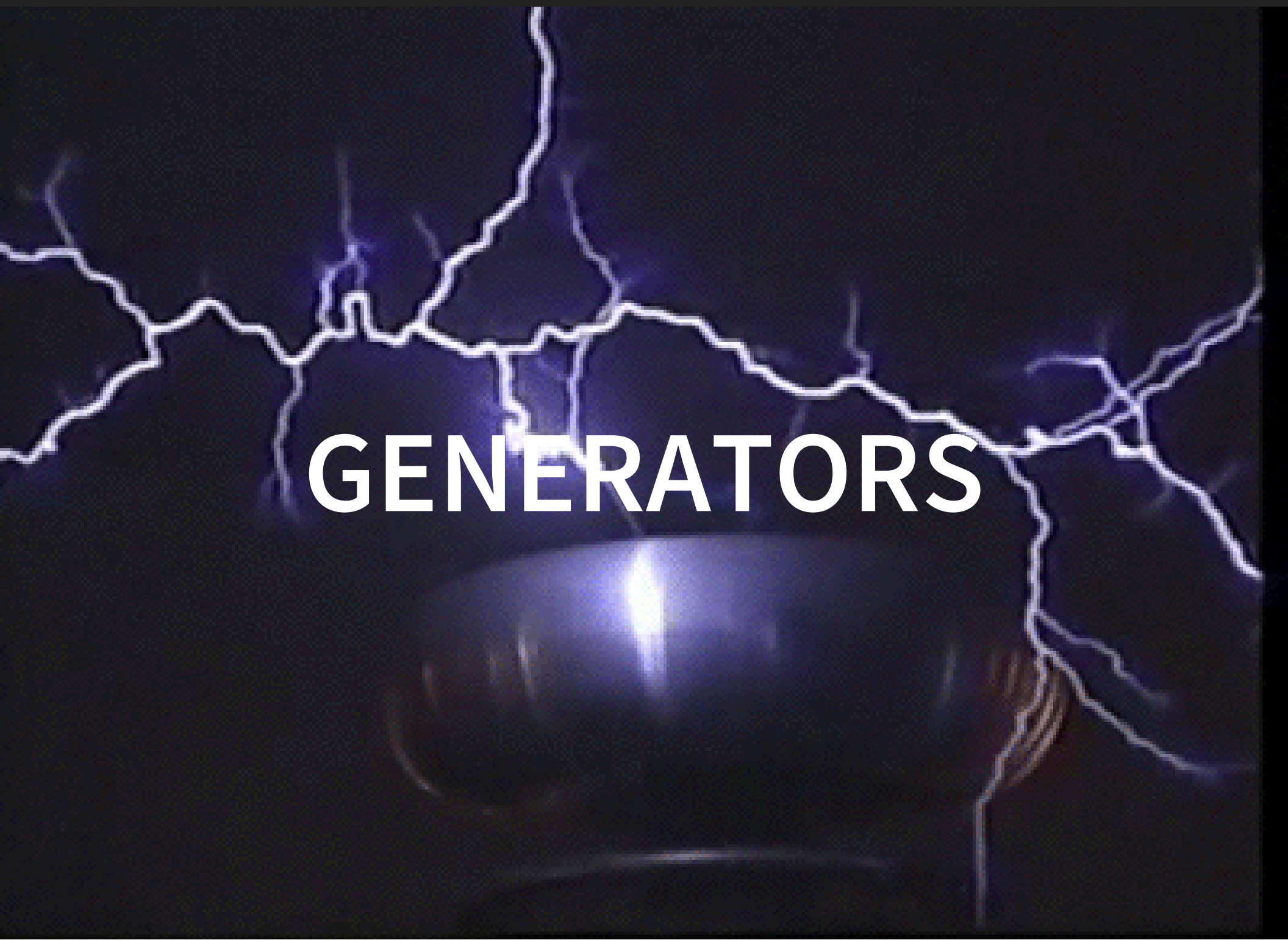
```
doThings(function (result) {
  doMoreThings(result, function (more) {
    pyramidOfDoom(more, function () {
      alsoKnownAs(function () {
        callbackHell('aaaaaaaaah!');
      });
    });
  });
});
```

PROMISES!

```
doThings().then( result => {
  return doMoreThings(result);
}).then(() => {
  return doEvenMoreThings();
}).then(() => {
  return rockOn();
}).catch( err => {
  console.error(err); // If anything went wrong with any of
                      // the calls, the whole chain fails.
});;
```

Sometimes we may want to make a bunch of async requests simultaneously, then once all of them return something, proceed.

```
Promise.all([doStuff(), doMoreStuff(), doEvenMoreStuff()])
  .then( results => {
    console.log(results[0]); // Result of doStuff()
    console.log(results[1]); // Result of doMoreStuff()
    console.log(results[2]); // Result of doEvenMoreStuff()
  })
  .catch( err => {
    console.error(err);
  });
```


GENERATORS

THEY'RE A LITTLE TOUGH TO EXPLAIN, SO....

https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Statements/function*

Paladin

CLASSES

Enter the new member's name.

Character Name AC Hit Cnd SpPt C

THE CURRENT WAY(S)

```
var Person = function (firstName) {
  this.firstName = firstName;
};

Person.prototype.sayHello = function() {
  console.log("Hello, I'm " + this.firstName);
};

var person1 = new Person("Alice");
var person2 = new Person("Bob");

// call the Person sayHello method.
person1.sayHello(); // Hello, I'm Alice
person2.sayHello(); // Hello, I'm Bob
```

THE CURRENT WAY(S)

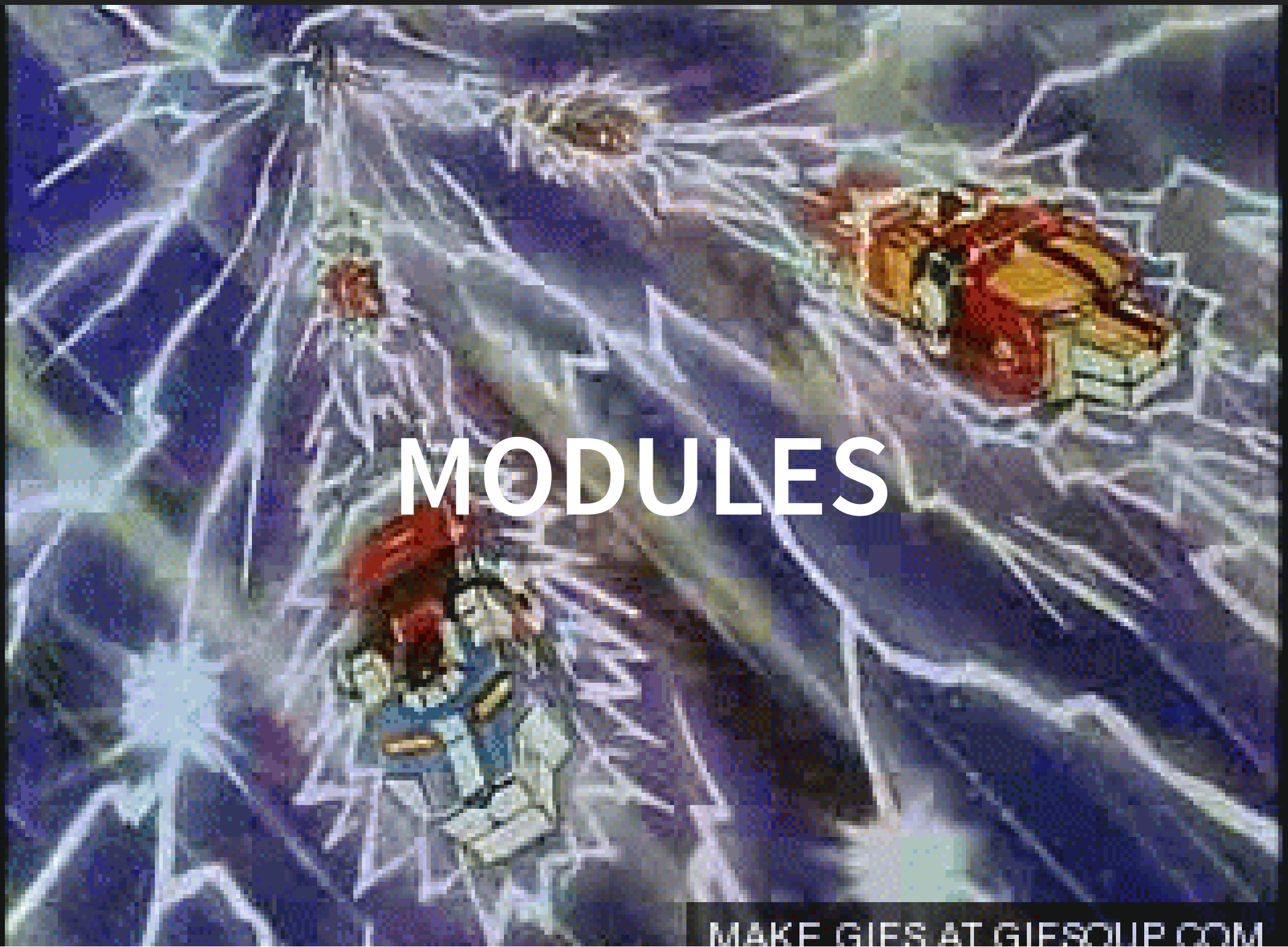
```
var person = {
  name: 'Bob',
  sayHello: function () {
    return "Hello, I'm " + this.name;
  }
};

console.log(person.sayHello()); // Hello, I'm Bob
```

THE ES6 WAY

```
class Person {
  constructor (name) {
    this.name = name;
  }

  sayHello() {
    return `Hello, I'm ${this.name}`;
  }
}

var adam = new Person('Adam');
console.log(adam.sayHello()); // Hello, I'm Adam
```

THE ES6 WAY (CONT.)

```
class Vehicle {
  constructor(name) {
    this.kind = 'Vehicle';
    this.name = name;
  }
  printName() {
    console.log(this.name);
  }
}

class Car extends Vehicle {
  constructor(name) {
    super(name); //call the parent method with super
    this.kind = 'Car';
  }
```


MODULES
MAKE GIFS AT GIFSOUP.COM

CURRENT METHODS

We can do this in Node...

```
// Inside file printer.js
module.exports = {
  print: function (thingToPrint) {
    console.log(thingToPrint);
  }
}

// Inside some other file...
var printer = require('printer');
printer.print('hooray for modularity!'); // hoooray for modularity!
```

CURRENT METHODS

We can do the same in the browser using one of these:

RequireJS: http://requirejs.org/

Browserify: http://browserify.org/

Webpack: http://webpack.github.io/

BUT ES6 HAS MODULES BUILT-IN!

```
// Inside file printer.js
export class Printer {
  print (thingToPrint) {
    console.log(thingToPrint);
  }
}

// Inside some other file...
import * as printer from 'printer';
printer.print('hooray for modulartity!'); // hoooray for modularity!
```

PARTIAL IMPORTS

```
// Inside file triathlon.js
export function run () {
  console.log('Running!');
}
export function bike () {
  console.log('Biking!');
}
export function swim () {
  console.log('Swimming!');
}

// Inside some other file...
import {swim} from 'triathlete';
swim(); // Swimming!
```


HOW CAN I GET ALL
THIS RIGHT NOW?!

SUPPORT

Some features already have great browser support.
("for...of" array iteration, template strings, promises)

https://kangax.github.io/compat-table/es6/

SUPPORT

Who's already using it?

- IO.js (fork of NodeJS)
- Aurelia (created by some members of the Angular 2.0 team)
- Angular 2.0
- ReactJS v0.13 beta Supports ES6 classes. Planning for full ES6 support soon.

ES6 final spec is expected to be approved in June. After that, it's up to the browsers to implement.

BUT IF YOU DON'T WANT TO WAIT, YOU CAN USE...

"TRANSPILERS"

BABEL (FORMERLY KNOWN AS 6-TO-5)

- Runs as one of your gulp/grunt/etc. build tasks

https://babeljs.io

TRACEUR

- Embedded as a runtime script
- Compiles to ES5 when page loads

https://github.com/google/traceur-compiler

MORE INFO

https://github.com/lukehoban/es6features
https://developer.mozilla.org/en-US/docs/Web/JavaScript/New_in_JavaScript

Some of the code samples in this presentation may or may not have been lifted directly from these sites, because when it comes to contriving examples...


Well, I guess that's everything.