

JAVASCRIPT

2099

The Future and Present of JS

I'm talking to you today about the present and future of javascript

CHRISTOPHER PLUMMER



Developer at Upstatement

introduce self, blah blah blah

JAVASCRIPT

ECMAScript

Any conversation about the JavaScript language begins with ECMAScript. ECMAScript is the standard specification for JavaScript. It dictates how the language should function.

ECMAScript == JavaScript

JavaScript is ecmascript.

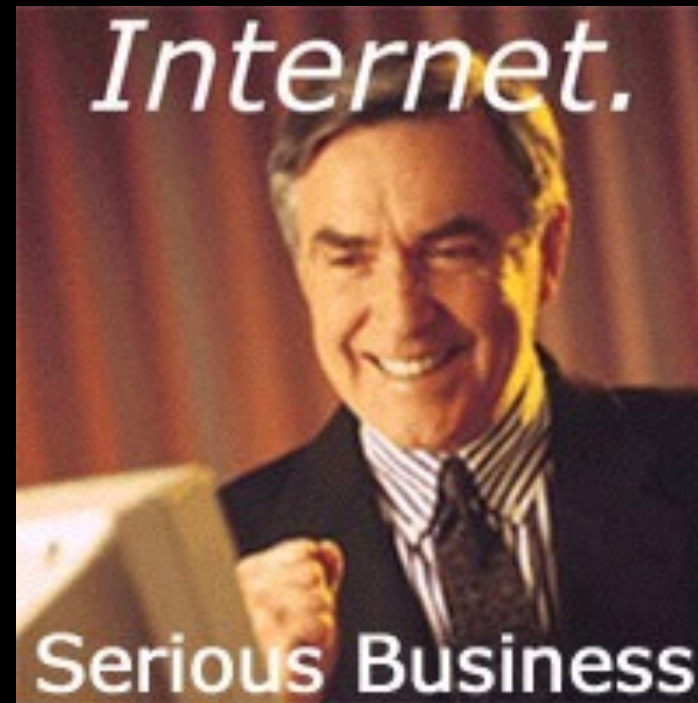
ECMAScript === JavaScript

Sorry. JavaScript triple equals ECMAScript.

***“ECMAScript WAS ALWAYS
AN UNWANTED TRADE
NAME THAT SOUNDS LIKE
A SKIN DISEASE.”***

Brendan Eich

(read quote). ECMAScript is serious business.



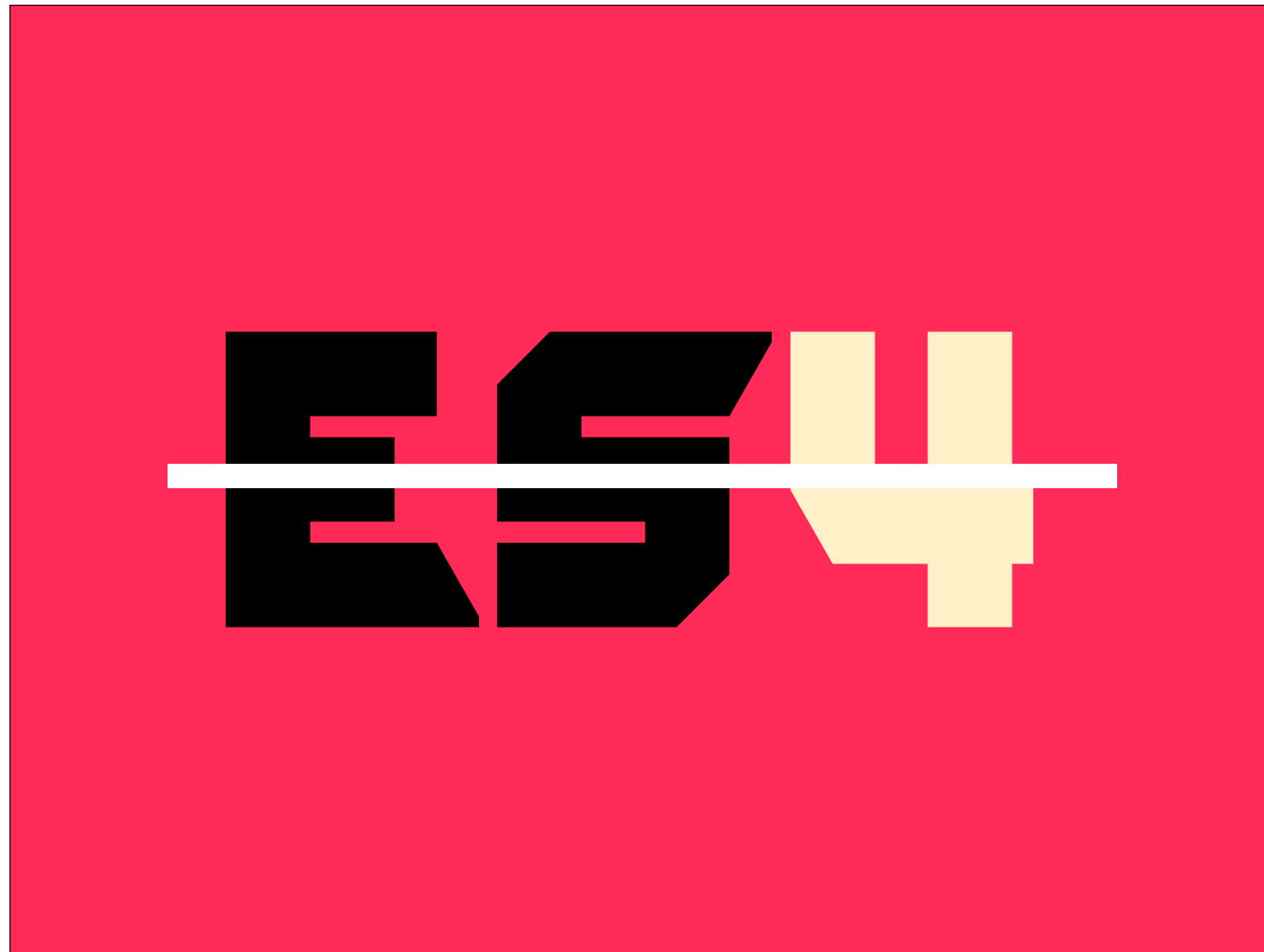
There is a standards body made up of engineers from companies such as Google, Apple, Microsoft, media companies like Netflix, non-profits like mozilla, open source projects like jQuery and smaller consultancies.

The logo consists of the letters 'E' and 'S' in a bold, black, sans-serif font. The 'E' and 'S' are connected at the top and bottom, with a small gap in the middle. The 'E' has a horizontal bar that extends to the right, and the 'S' has a horizontal bar that extends to the left, meeting the 'E' bar. The logo is centered on a solid red background.

This working group drafts the spec for the language. The first standard was drafted in 1997. Change has been slow!

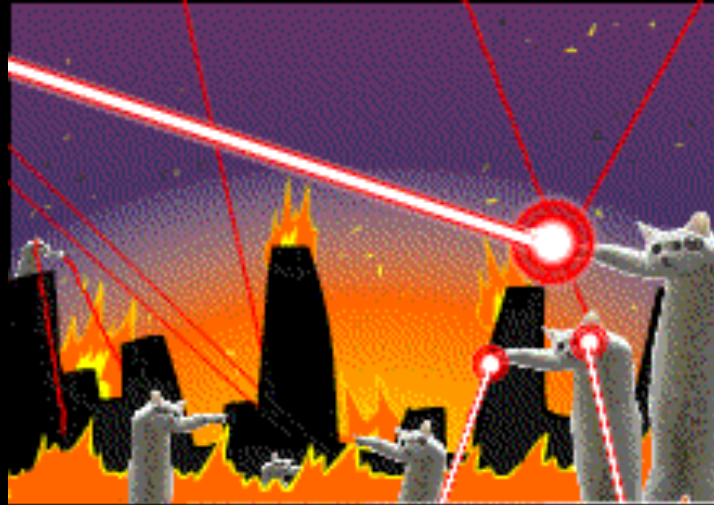
The logo for ECMAScript 3 (ES3) is displayed on a solid red rectangular background. The text "ES3" is rendered in a bold, blocky, sans-serif typeface. The letters "E" and "S" are black, while the number "3" is a bright yellow. The characters are closely spaced and centered within the red field.

A update, ECMAScript 3, was published in 1999. If you are supporting IE 8, you're JS is ES3. You're using a spec from 17 years ago.



ES4 was supposed to be the next major update to the language, but conflicts within the ECMAScript Working Group assigned to write the spec caused the specification to be abandoned in 2008 after 10 years of work.


There was a lack of process to achieve consensus and feature creep.



Remember that this was the era of the Second Browser War, when browser manufacturers had proprietary features and implementations. Anyone who writes **CSS** even today knows how hard it is for browsers to reach consensus on how a feature should be implemented.

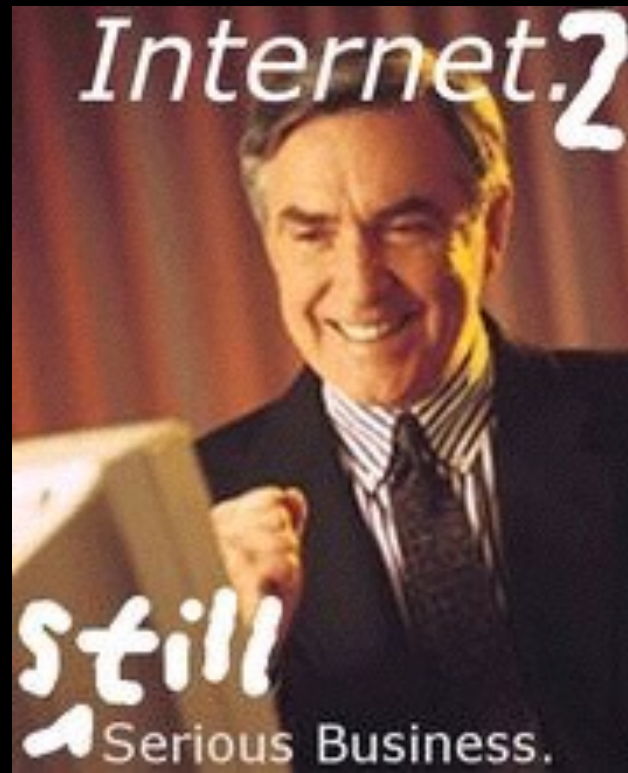
HARMONY

The factions eventually settled their differences and worked out a process called Harmony for changing the language. I'll tell you about that process later.



TC-39

They formed the ECMA Technical Committee 39: TC 39...



...and TC-39 is serious business.

The logo for ECMAScript 5 (ES5) is displayed on a solid red rectangular background. The text "ES5" is rendered in a bold, blocky, sans-serif typeface. The "E" and the first "S" are black, while the second "S" is a bright yellow. The letters are closely spaced and have a slightly irregular, hand-drawn appearance.

In 2009 TC-39 published a draft for ES5. The first update in since 1999. If you're supporting IE8 you may have missed some of features in ES5.

HIGHER ORDER ARRAY FUNCTIONS

```
['flux', 'flux', 'flux', 'capacitor'].filter( function(item) {  
  return item === 'capacitor';  
}); // ['capacitor']
```

```
['1.0', '1.21'].forEach( function(item) {  
  console.log(item + ' gigawatts!!!1 ');  
});  
// 1.0 gigawatts!!!1 1.21 gigawatts!!!1
```

such as higher order Array functions

OBJECT.CREATE()

```
var alien = {  
  name: 'Bob',  
  announce: function() {  
    return 'I am ' + this.name;  
  }  
}  
  
var marklar = Object.create(alien, {  
  name: {  
    value: 'marklar',  
    configurable: false  
  }  
});  
  
marklar.name = 'bob';  
marklar.announce(); // 'marklar'
```

OTHER ITEMS

- ★ `getters & setters`
- ★ `Object.keys()`
- ★ `'use strict';`
- ★ `JSON.parse,`
`JSON.stringify`
- ★ `Date.now()`
- ★ `legal trailing commas`

There were a number of other small changes, but the big change happened in the next release ES6.



If you practice resume driven development, this part of the talk will be especially important for you.

The logo for ECMAScript 6 (ES6) is displayed. It features the letters 'ES' in a bold, black, sans-serif font, followed by the number '6' in a bold, yellow, sans-serif font. The entire logo is centered on a solid red rectangular background.

Last year, TC-39 published a new draft, ECMAScript 6.



Or is it EcmaScript 2015? I'll explain the name change.

ES6 took so long to complete, it had a lot features...

- ★ arrows
- ★ classes
- ★ enhanced object literals
- ★ template strings
- ★ destructuring
- ★ default + rest + spread
- ★ let + const
- ★ iterators + for..of
- ★ generators
- ★ unicode
- ★ modules
- ★ module loaders
- ★ map + set + weakmap + weakset
- ★ proxies
- ★ symbols
- ★ subclassable built-ins
- ★ promises
- ★ math + number + string + array + object APIs
- ★ binary and octal literals
- ★ reflect api
- ★ tail calls

...Maybe too many. And it took too long to draft: 6 years. So TC-39 decided to make smaller releases more often, every year.

The logo for ES2015, featuring the text "ES2015" in a bold, sans-serif font. The "ES" is black, and "2015" is yellow. The logo is centered within a red rectangular background.

ES2015

Now it's called ES2015 to reflect the pace and future releases will follow this naming convention.

The logo for ES2016, featuring the text "ES2016" in a bold, sans-serif font. The "ES" is black, and the "2016" is yellow. The logo is centered within a red rectangular background.

ES2016

This landed already. Did you miss it?

The logo for ES2017, featuring the letters 'ES' in black and the year '2017' in yellow, all in a bold, sans-serif font, centered on a red rectangular background.

ES2017

And this is just a year away.

The logo features the text "ES 2099" in a bold, stylized, blocky font. The "ES" is black, and "2099" is bright pink. The entire logo is centered within a light yellow rectangular box with a thin black border.

ES 2099

And it's not stopping any time soon.



At this rate of change, you're probably worried about maintaining your skills as well as your applications.

WHY BOTHER LEARNING THIS STUFF?

You can get plenty done now with old-school JavaScript and jQuery, right?



Let the kids have their toys! Why bother learning it?



***BECAUSE I
SAID SO,
THAT'S WHY.***

This isn't some new javascript framework. This isn't a trend or a fad. This is the language. Unless you plan on leaving web development, you're going to run into this sooner or later. Core libraries, frameworks and tools are all moving in this direction.

These new features also make the language so much easier to work with. They address some of the problems born of javascript's rapid creation. These features borrow some of the best ideas from other languages and libraries.

WHAT'S IN **ES2015**

So what features am I talking about? There's not enough time to cover everything, so I'm just going to talk about some of the things I use everyday.

*THINGS THAT
CHANGE HOW YOU*

WRITE JS

First, the small things that change the way you write JS. Syntax things.

TEMPLATE STRINGS

```
var captain = {  
  name: 'Kirk'  
};  
var location = 'mountain';  
  
'Captain ' + captain.name + ' is climbing the ' + location + '.';  
  
// "Captain Kirk is climbing the mountain."
```

```
`Captain ${captain.name} is climbing the ${location}.`;`  
  
// "Captain Kirk is climbing the mountain."
```

ARROW SYNTAX

```
[1, 2, 3, 4, 5].filter(function(num) {  
  return num > 2;  
}); // [3, 4, 5]
```

```
[1, 2, 3, 4, 5].map(function(num, index) {  
  return num + index;  
}); // [1, 3, 5, 7, 9]
```

```
[1, 2, 3, 4, 5].filter(num => num > 2);  
// [3, 4, 5]
```

```
[1, 2, 3, 4, 5].map((num, index) => num + index);  
// [1, 3, 5, 7, 9]
```

ARROW SYNTAX

```
var Runner = {  
  name: 'Logan',  
  run: function() {  
    var self = this;  
  
    [1, 2, 3].forEach(function(i) {  
      console.log(`${self.name} is running ${i}! `);  
    });  
  }  
};  
  
Runner.run();
```

```
var Runner = {  
  name: 'Logan',  
  run: function() {  
    [1, 2, 3].forEach( i => {  
      console.log(`${this.name} is running ${i}! `);  
    });  
  }  
};
```

DEFAULT ARGUMENTS

```
function farnsworth(audience) {  
  audience = audience || 'everyone';  
  console.log(`Good news ${audience}!`);  
};  
  
// Good news everyone!
```

```
function farnsworth(audience = 'everyone') {  
  console.log(`Good news ${audience}!`);  
};  
  
// Good news everyone!
```

SPREAD OPERATORS

```
function getFirstStarship() {  
  var starships = arguments;  
  return starships[0];  
};  
  
getFirstStarship('galactica', 'enterprise', 'serenity');  
// 'galactica'
```

```
function getFirstStarship(...starships) {  
  return starships[0];  
};  
  
getFirstStarship('galactica', 'enterprise', 'serenity');  
// 'galactica'
```

SPREAD OPERATORS II

```
function exampleFunction(arg1, arg2) {  
  return console.log(arg1, arg2);  
};  
  
function callExample() {  
  return exampleFunction.apply(null, arguments);  
};
```

```
function callExample() {  
  return exampleFunction(...arguments);  
};  
  
callExample('benjamin', 'sisko');  
// 'benjamin sisko'
```

DESTRUCTURING

```
var Runner = {  
  name: 'logan',  
  job: 'run'  
};  
  
var name = Runner.name;  
var job = Runner.job;  
// 'logan'
```

```
var { name, job } = Runner;
```


OBJECT LITERAL SYNTAX

```
var makeRobot = function(name, service) {  
  return {  
    name: name,  
    service: service  
  };  
}
```

```
var makeRobot = function(name, service) {  
  return { name, service };  
}
```

OBJECT LITERAL SYNTAX II

```
var deathStar = {  
  location: 'alderaan',  
  demonstrateFirepower: function(planet) {  
    this.destroy(planet);  
  }  
}
```

```
var deathStar = {  
  location: 'alderan',  
  demonstrateFirepower(planet) {  
    this.destroy(planet);  
  }  
}
```

CLASS SYNTAX

```
function Jedi(name) {  
  this.name = name;  
};  
  
Jedi.prototype.attack = function(target) {  
  console.log(`${this.name} punches ${target.name}`)  
}  
  
var obiWan = new Jedi('obi-wan');  
var sith = new Jedi('vader');  
  
sith.attack = function(target) {  
  console.log(`${this.name} shoots lightning at ${target.name}`)  
};  
  
sith.attack(obiWan); // vader shoots lightning at obi-wan  
obiWan.attack(sith); // obi-wan punches vader
```

CLASS SYNTAX

```
class Jedi {  
  constructor(name) {  
    this.name = name;  
  }  
  
  attack(target) {  
    console.log(`${this.name} punches ${target.name}`);  
  }  
};  
  
class Sith extends Jedi {  
  attack(target) {  
    console.log(`${this.name} shoots lightning ${target.name}`);  
  }  
}  
  
var obiWan = new Jedi('obi-wan');  
var sith = new Sith('vader');
```

***THINGS THAT
CHANGE HOW YOU***

USE JS

So what functional changes have been made?

PROMISES

```
getAJAX('spacejam.com', function(data) {  
  if (data.err) {  
    console.log(err);  
  } else {  
    return res;  
  }  
});
```

```
getAJAX('spacejam.com')  
  .then( data => data )  
  .catch( err => {  
    console.log(err);  
  });
```

OBJECT.ASSIGN()

```
var brundle = {  
  species: 'human',  
  name: 'brundle'  
};  
  
var fly = {  
  species: 'fly'  
};  
  
var brundleFly = Object.assign(brundle, fly);  
  
// { species: 'fly', 'name': 'brundle' }
```

MODULES

```
var Enterprise = (function(WarpCore, $, undefined) {  
  var engineering = WarpCore.init();  
  return {  
    engineering: engineering  
  };  
  
})(window.WarpCore, window.jQuery)
```

```
import WarpCore from 'lib/warp-core';  
import * as $ from 'node_modules/jquery';  
  
let Enterprise = {  
  engineering: WarpCore.init()  
}  
  
export default Enterprise;
```


LET, CONST

```
var PLANET = { type: 'apes' };

if (PLANET.type !== '') {
  var DrZaius = new Ape();
  DrZaius.judge();
}

DrZaius.judge();
```

```
const PLANET = { type: 'apes' };

if (PLANET.type !== '') {
  let DrZaius = new Ape();
  DrZaius.judge();
}

PLANET = null;
// Uncaught TypeError: Assignment to constant variable.

DrZaius.judge();
// Uncaught ReferenceError: DrZaius is not defined
```

SET

```
var replicants = ['roy','pris','leon','roy'];  
replicants.push('pris');  
  
console.log(replicants);  
// ['roy','pris','leon','roy','pris']
```

```
var replicants = new Set(['roy','pris','leon','roy']);  
replicants.add('pris');  
  
console.log(replicants); // ['roy','pris','leon']  
replicants.toString()   // object Set
```

ARRAY.FROM()

```
var arr = [];  
var len = 3;  
  
for (var i = 0; i < len; i++) {  
  arr[i] = i;  
}  
  
console.log(arr) // [0,1,2]
```

```
Array.from({length: 3}, (value, key) => key); // [0,1,2]
```

OTHER ITEMS

- ★ Symbols
- ★ `for..of`
- ★ Unicode
- ★ subclassable built-ins
- ★ hyperbolic geometry
- ★ binary, octal literals
- ★ proxies
- ★ tail calls
- ★ Reflect API
- ★ Maps, WeakMaps, WeakSets

The logo for ES2016, featuring the text "ES2016" in a bold, stylized font. The "ES" is black and the "2016" is yellow, all set against a red background.

ES2016

ES2016 is here already! It only adds two features, which is good. Smaller releases, remember?



One is an exponentiation operator. the next is...

ARRAY.INCLUDES()

```
var replicants = ['roy', 'pris', 'leon'];  
  
var isLeonReplicant = replicants.indexOf('leon') > -1;  
  
console.log(isLeonReplicant); // true
```

```
var replicants = ['roy', 'pris', 'leon'];  
  
var isDeckerReplicant = replicants.includes('decker');  
  
console.log(isDeckerReplicant); // false
```

...array.includes.

WHEN CAN I USE THIS MAGIC SAUCE?

When can I use this magic sauce? The good news is you can use most of these features today, provided you don't have to support Internet Explorer. Seriously. You'll have to wait until IE 11 dies to use these.

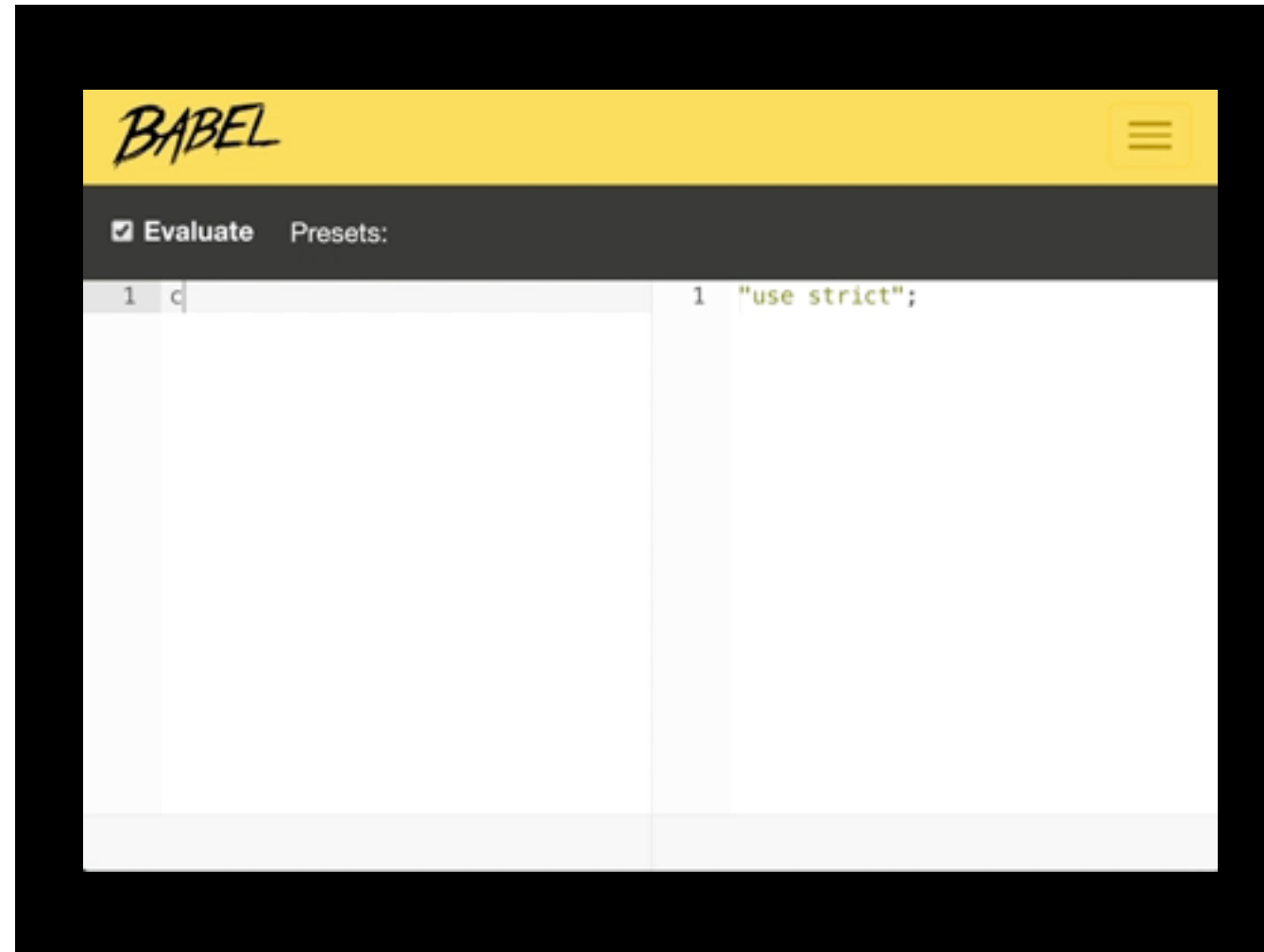
***BUT I WANT
IT NOW!***

...

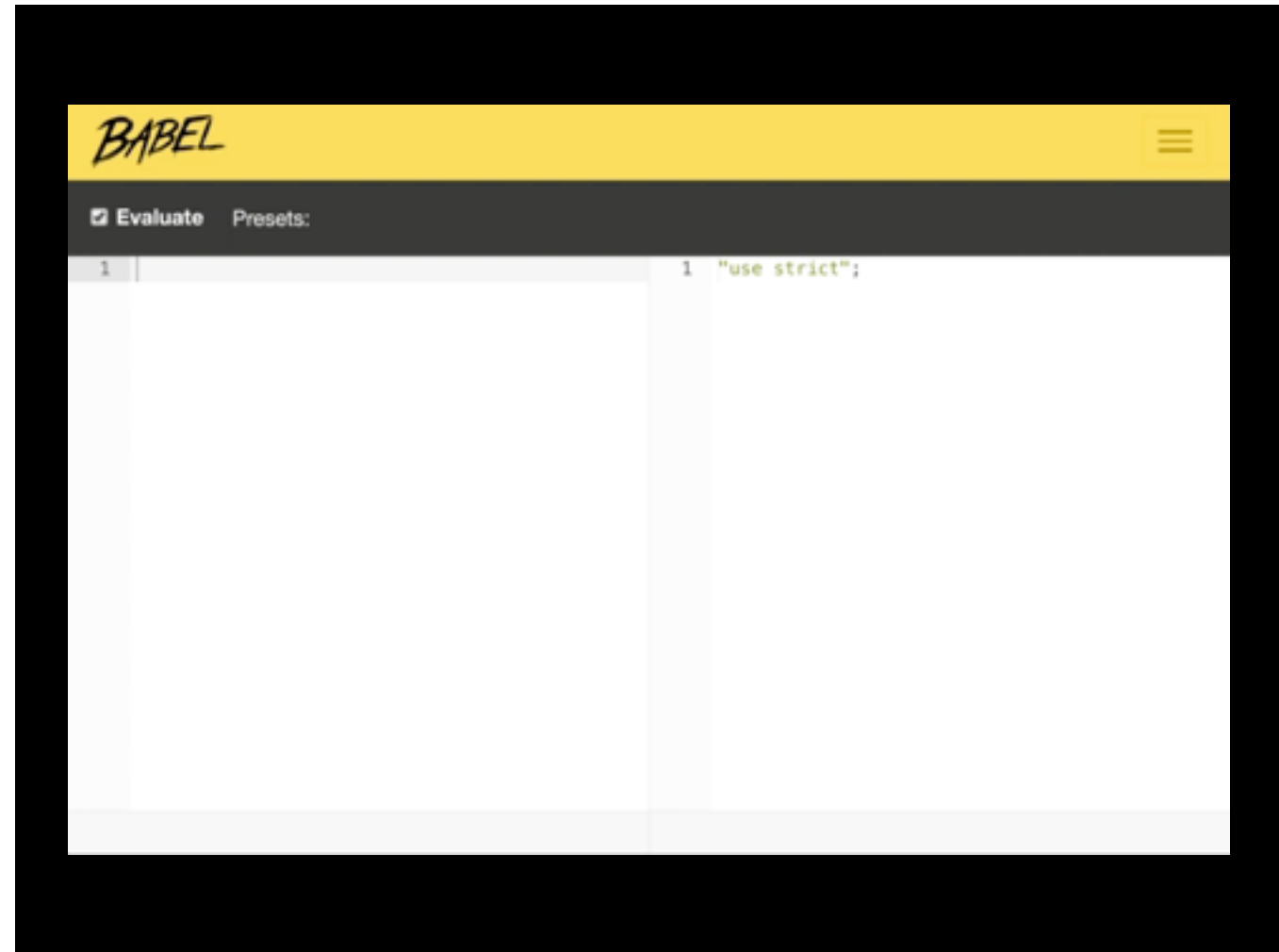
Good news is that there is an option for you.

The logo features the word "BABEL" in a large, yellow, stylized font with a hand-drawn, sketchy appearance. Below it, the text "6TO5" is written in a similar yellow, stylized font. The entire logo is set against a solid red rectangular background.

It's Babel. Babel is going to transform ES2015 into ES5 to provide backwards compatibility. In a sense it works a lot like SASS, transpiling one language into another.



const let



destructuring arrays

TOMORROW'S JAVASCRIPT PROBLEMS, TODAY!

Babel delivers tomorrow's javascript problems, today!

So what does any of this have to do with wordpress?

WP + BABEL = 

You can use babel with wordpress and I'll show you how.

```
NERD_tree_1 s/j/app.js s/j/module.js X
" Press ? for help

.. (up a dir)
/Users/plumcakes/Sites/wordcamp/javascript-2099-demo/
└ dist/
└ src/
    docker-compose.yml
    functions.php
    package.json
    style.css
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
Session: camp 2 1 1:..ipt-2099-demo# 2:vim* 19 Jul 17:22
```

Here is a bare-bones wordpress theme. I have made two js modules using the module syntax and some other new features. App.js imports a module named HelloWorld.

The Hello World module exports a function which returns an object. On document.onload we'll call the announce method on that object to replace some text on the wp homepage.



TOOLS

If you're in the anti-tools crowd get a coffee and then wait a couple years until all these ES2015 features land in your browser matrix. The rest of us will push on.



First we need a module loader. We'll use browserify for this. ``npm install -g browserify``.

Browserify is going to bundle our js and dump the results in a directory called dist. It's like concatenating javascript. Then we can simply `enqueue_script` our bundled javascript. If we try to run browserify we get an error. We need to transpile ES2015's syntax using babel.



There's a Babel plugin for Browserify called Babelify. This plugin recognizes the new module syntax and bundles our dependencies for us. We install it locally with `npm install babelify`.

We tell browserify to use babelify by setting a transform option on the terminal command. Let's bundle again. Here's I'm setting the transform option to use babelify. Oops! It didn't work

The problem is that we need to tell Babel what flavor of JS we're using: ES 2015, ES 2016?

```
→ javascript-2099-demo git:(start-over) X browserify \
src/js/app.js -o dist/js/app.js \
--transform [ babelify ]

/Users/plumcakes/Sites/wordcamp/javascript-2099-demo/src/js/app.js:1
import HelloWorld from './module';
^
ParseError: 'import' and 'export' may appear only with 'sourceType: module'
→ javascript-2099-demo git:(start-over) X
```

Session: camp 1 1

1:..ipt-2099-demo* 2:vim-

19 Jul 18:01

So we have to install a package of babel transforms. In this case we need both ES2015 and ES2016. We npm install those and pass in as arguments to our call to the babel transform.

Here I'm setting the preset option for babel. Press enter and let's see what happened, by looking at the output. Seems legit. We can see that it's bundled and transpiled our code. It looks a lot like that demo I showed earlier.

```
→ javascript-2099-demo git:(start-over) X  
  
Session: camp 1 1 1:...ipt-2099-demo* 2:vim- 19 Jul 18:06
```

We just have to change the script that we queue. If we open up the browser, we get the expected result.

That's tedious running that command every time you make a change. Let's automate it.



You could use Grunt or Gulp or Broccoli or whatever.



You can also just use plain old NPM Scripts. Let's do that and keep it simple.

A terminal window with a dark background. The prompt is `→ javascript-2099-demo git:(start-over) X`. The terminal is mostly empty. At the bottom, there is a status bar with the text `Session: camp 1 1`, `1:..ipt-2099-demo* 2:..ipt-2099-demo-`, and `19 Jul 20:13`.

First, install browserify as a local dependency. This let's us run it with npm scripts. `npm install browserify`. you can uninstall the global browserify if you want to.

In `package.json` specify your babel config. You can set `source maps` to `true` for if you want a `sourcemap`. and then I'm specifying the presets we want for babel `es2015` and `es2016` just like we did in the command line.

Then just specify the terminal commands we want to run to build our js in the `build` property. In this case, run the same browserify task we ran from the command line.

To build, we just need to `npm run build` in the terminal. And it works!



```
→ javascript-2099-demo git:(start-over) ✕
```

Session: camp 1 1 1:..ipt-2099-demo* 2:..ipt-2099-demo- 19 Jul 20:33

So our terminal command is shorter, but we really want it to run automatically every time a source javascript file is changed.

[press play]

You'll just need to install one final npm package.

npm install onchange. onchange is a library that allows node to watch for changes. We can specify a watch task in our script file and we start it up in the terminal with npm run watch.

Now when we save a file, we build new javascript assets!

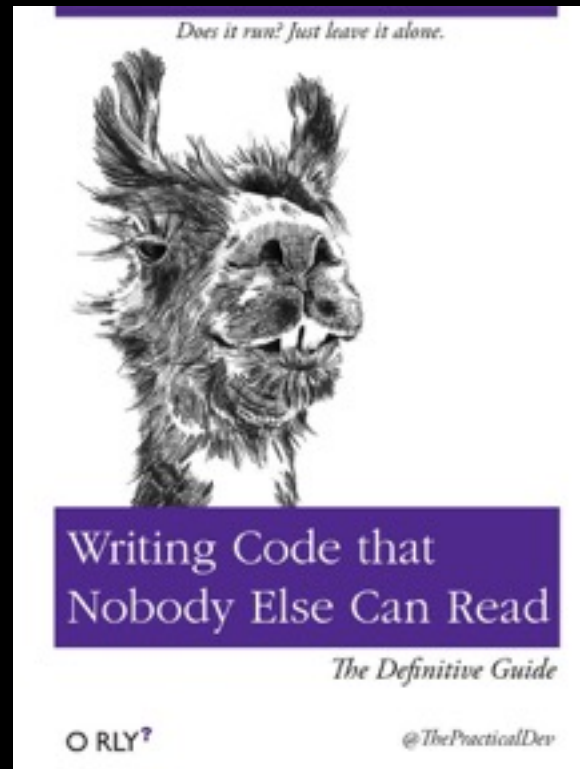


We did it without the need for a separate build dependency like gulp or grunt. It's possible to cache the builds and run other parallel tasks with npm scripts...

EXAMPLE REPO

[https://github.com/signal-intrusion/
javascript-2099-demo](https://github.com/signal-intrusion/javascript-2099-demo)

...I won't go into that here, but I have a demo repo you can fork.



Then you can confuse your co-workers and clients with new JS features and syntax!


THE FUTURE

What does the future hold? How does TC-39 decide what features to add to the language?

PROPOSAL PROCESS

`https://github.com/tc39/proposals`

Members make proposals and they go through a series of reviews, definition, and testing before they are accepted.

|  | Proposal | Champion | Stage |
|---|---|--|-------|
| | SIMD.JS - SIMD APIs + polyfill | John McCutchan, Peter Jensen, Dan Gohman, Daniel Ehrenberg | 3 |
| | Async Functions | Brian Terlson | 3 |
| | Trailing commas in function parameter lists and calls | Jeff Morrison | 3 |
| | Function.prototype.toString revision | Michael Ficarra | 3 |
| | Asynchronous Iterators | Kevin Smith | 2 |
| | function.sent metaproperty | Allen Wirfs-Brock | 2 |
| | Rest/Spread Properties | Sebastian Markbage | 2 |
| | Shared memory and atomics | Lars T Hansen | 2 |
| | System.global | Jordan Harband | 2 |
| | Lifting Template Literal Restriction | Tim Disney | 2 |
| | ArrayBuffer.transfer | Luke Wagneer & Allen Wirfs-Brock | 1 |
|  | export * as ns from "mod"; statements | Lee Byron | 1 |
|  | export v from "mod"; statements | Lee Byron | 1 |
|  | Class and Property Decorators | Yehuda Katz and Jonathan Turner | 1 |

Proposals move through stages starting at 0, called “straw-man”. Stage 4 represents a complete feature that is shipping.

IN order for a proposal to advance they must meet certain criteria. For example, to advance to Stage 1 a proposal needs a sponsor and examples of usage. To advance from stage 3 to 4 proposals need acceptance tests and significant usage in real world applications. At any point the proposal may be withdrawn, and I show you why that’s important in a bit.

DECORATORS

`https://github.com/wycats/
javascript-decorators`

Decorators is one proposal in Stage 1. It has it's own github repo you can comment, file issues, submit your own PR's for the proposal.

DECORATORS

```
function readonly(target, key, descriptor) {  
  descriptor.writable = false;  
  return descriptor;  
}  
  
class Robot {  
  @readonly  
  name() { return `${this.first} ${this.last}`; }  
}  
  
let klatu = new Robot();  
klatu.name = 'Barada Nicto';  
  
// Cannot assign read only property 'name'
```

Decorators are expressions that return a function and allow us to dynamically add or modify the function that follows to object methods and classes.

SYSTEM.GLOBAL

`https://github.com/tc39/
proposal-global`

System.global is a stage 2 proposal. While so much of js these days is meant to remove global state, system global wants to create a namespace for it.

SYSTEM.GLOBAL

```
window.$.toString();

// 'function (selector,context){return new
jQuery.fn.init(selector,context)}'
```

```
System.global.$.toString();

// 'function (selector,context){return new
jQuery.fn.init(selector,context)}'
```

Currently one of the global scopes in the browser is window, and you may have done a window.query request once in your career.

But window doesn't exist in node.js. node uses global. Shell's like d8 or jsc have neither window nor global.

— — — —

This proposal would create a new global namespace, system.global that could be used in node, the browser, or elsewhere.

SIMD.JS

`https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/SIMD`

Proposals which are further along have their specs hosted at MDN. The Single Instruction Multiple Data proposal, or SIMD, is much more low level language feature that will improve performance in 3D graphics, video, and cryptography and other domains by providing parallelized computation and new primitives. SIMD is in stage 3 and has a polyfill as all proposals above stage-1 must, so you could start using it today!

ASYNC FUNCTIONS

`https://github.com/tc39/
ecmascript-asyncawait`

Async functions are major Stage 3 proposal which will have a big impact on how you write async code.

ASYNC FUNCTIONS

```
getAJAX('spacejam.com')
  .then( data => {
    return sendAJAX('derpy-api.biz', data);
  }).then( res => {
    console.log(res);
  });
```

```
async function() {
  let data = await getAjax('spacejam.com');
  let res = await sendAjax('derpy-api.biz');
  console.log(res)
}
```

You may be using promises which were made available in es2015. In this example we're requesting some data from spacejam, then sending that data to our derby api then logging the response. If you start chaining promises together beyond this, you may be annoyed at the verbosity and the ambiguity, especially when you have to catch errors.

— — — —

This proposal simplifies async programming. The `async` keyword wraps the return value of a function in a promise. The `await` keyword takes a promise and waits for it's value to be returned.



Many proposals from stage 0 through 3 are available to to use as transforms in babel. This is one of the primary means of testing them in real-world applications, people like you and me using them with babel. This testing is exactly what is required to move to stage 4.



~~Object.observe~~

Anyone ever want to watch an object for changes and then trigger an action elsewhere in the application? There was a proposal to do just this. It looked real promising in the early days of two-way data binding. Seems like a good idea right?

But `Object.observe` came at the cost of performance and increased application complexity. The immutable JS crowd hated this thing. So TC-39 withdrew the proposal before it ended up in the language. They prevented a lengthy and expensive deprecation process later on.

But, `Object.observe` reached stage-2 before it was withdrawn and had even found its way into node.js.

So it's important to use any proposal below stage-3 carefully. Keep in mind that what you're doing is potentially app breaking in the future.

THE FUTURE *AGAIN*

JavaScript will continue to evolve even as Web Assembly and asm open up the browser to other languages. It's not going away, but it is getting better.

***THE HUMAN
ADVENTURE IS
JUST BEGINNING...***

CHRISTOPHER PLUMMER



`chris.plummer@upstatement.com`

`@IntrusionSignal`

`github.com/signal-intrusion`

`@signalintrusion` [Ember Community Slack]

demo: `https://github.com/signal-intrusion/
javascript-2099-demo`