ES6 is so 2015

meet ES2016!

ES6 is so 2015

meet ES2016!

Paul Verbeek


Booking.com
workingatbooking.com

nlhtml5.org

ECMAScript

ECMAScript?

ECMA-262

(a timeline)

December 1995 Announcement of JavaScript by Sun and Netscape

ECMA-262

(a timeline)

December 1995 Announcement of JavaScript by Sun and Netscape

"JavaScript will be the most effective method to connect HTML-based content to Java applets.

- Bill Joy, co-founder Sun

https://web.archive.org/web/20070916144913/http://wp.netscape.com/newsref/pr/newsrelease67.html

ECMA-262

(a timeline)

December 1995	Announcement of JavaScript by Sun and Netscape
March 1996	Netscape Navigator 2.0, with JavaScript
August 1996	Internet Explorer 3.0, with JScript
June 1997	First edition of ECMAScript

ECMA-262

(a timeline)

December 1995 Announcement of JavaScript by Sun and Netscape

March 1996 Netscape Navigator 2.0, with JavaScript

August 1996 Internet Explorer 3.0, with JScript

June 1997 First edition of ECMAScript

"ECMAScript was always an unwanted trade name that sounds like a skin disease.

- Brendan Eich, creator of JavaScript

https://mail.mozilla.org/pipermail/es-discuss/2006-October/000133.html

ECMA-262

(a timeline)

December 1995	Announcement of JavaScript by Sun and Netscape
March 1996	Netscape Navigator 2.0, with JavaScript
August 1996	Internet Explorer 3.0, with JScript
June 1997	First edition of ECMAScript
August 1998	ES2, to align with ISO/IEC 16262
December 1999	ES3, added regex, try/catch and more
~~**July 2008**~~	~~ES4, Abandoned due to complexity~~
December 2009	ES5, strict mode, JSON, get/set, and more
June 2011	ES5.1, to align with ISO/IEC 16262 3rd edition
June 2015	ES6/ES2015, a lot of stuff!
2016	ES2016, not lot of stuff!

The TC39 process

AKA. bunch of white guys

The TC39 process

Stage 0: Strawman
Free-form ideas, reviewed in TC39 meetings

Stage 1: Proposal
Formally accepted proposal

Stage 2: Draft
Has description of syntax and semantics

Stage 3: Candidate
Spec text complete, has at least 2 implementations

Stage 4: Finished
Ready for standard, passes unit tests

ECMAScript 2016

ECMA-262 7th edition

The ECMAScript® 2016 Language Specification

Following slides will mostly contain code

Exponentiation Operator

```
x ** y

Math.pow(x, y);


const squared = 2 ** 2;
// same as  2 * 2


let cubed = 3;
cubed **= 3;
// same as  cubed = 3 * 3 * 3
```

Exponentiation Operator

Array.prototype.includes

```
['a', 'b', 'c'].indexOf('a') >= 0; // true
['a', 'b', 'c'].indexOf('d') >= 0; // false


['a', 'b', 'c'].includes('a'); // true
['a', 'b', 'c'].includes('d'); // false
```

```
['a', 'b', NaN].indexOf(NaN) >= 0; // false
['a', 'b', NaN].includes(NaN); // true

[ , , ].indexOf(undefined) >= 0; // false
[ , , ].includes(undefined); // true
```

Wait? `includ` **not** `contains?`

Array.prototype.includes





ECMA-262 8th edition

The ECMAScript® 2017 Language Specification

Not really…

Stage 4: Finished

Ready for standard, passes unit tests

- Object.values/Object.entries
- String padding
- Object.getOwnPropertyDescriptors()
- Async functions
- Trailing commas in function parameter lists and calls

https://github.com/tc39/proposals/blob/master/finished-proposals.md

Object.values / Object.entries

```
const obj = { name: 'Paul', age: 30 };

const keys = Object.keys(obj);
// ['name', 'age']


const values = Object.values(obj);
// ['Paul', 30]

const entries = Object.entries(obj);
// [['name', 'Paul'], ['age', 30]]



for (let [key,value] of Object.entries(obj)) {
    console.log(`${key}: ${value}`);
}

// 'name': 'Paul'
// 'age': 30
```

Object.values / Object.entries

String padding

insert left-pad joke

```
'1'.padStart(3, '0');
// '001'

'1'.padEnd(3, '0');
// '100'


‘TakeOff'.padEnd(13, 'Conf');
// ‘TakeOffConfCo'



'foo'.padStart(8);
// '     foo'
```

String padding

Object.getOwnPropertyDescriptors()

```
const obj = {
    [Symbol('foo')]: 123,
    get bar() { return 'abc' },
};
console.log(Object.getOwnPropertyDescriptors(obj));


/*
 {
   [Symbol('foo')]: {
     value:        123,
     writable:     true,
     enumerable:   true,
     configurable: true
   },
   bar: {
     get:          [Function: bar],
     set:          undefined,
     enumerable:   true,
     configurable: true
   }
 }
*/
```

1. Copying properties into an object

```
const o1 = { a: 1 };
const o2 = { b: 2 };

Object.assign(o1, o2);

console.log(o1);
// { a: 1, b: 2 }
```

1. Copying properties into an object

```
const source = {
    set foo(value) { console.log(value); }
};

console.log(Object.getOwnPropertyDescriptor(source, 'foo'));
// { get:          undefined,
//   set:          [Function: foo],
//   enumerable:   true,
//   configurable: true }


const target = {};
Object.assign(target, source);

console.log(Object.getOwnPropertyDescriptor(target, 'foo'));
// { value:        undefined,
//   writable:     true,
//   enumerable:   true,
//   configurable: true }
```

1. Copying properties into an object

```
const source = {
    set foo(value) {
        console.log(value);
    }
};


const target = {};
Object.defineProperties(
    target,
    Object.getOwnPropertyDescriptors(source)
);

console.log(Object.getOwnPropertyDescriptor(target, 'foo'));
// { get:          undefined,
//   set:          [Function: foo],
//   enumerable:   true,
//   configurable: true }
```

2. Cloning objects

```
const obj = {
    [Symbol('foo')]: 123,
    get bar() { return 'abc' },
};

const clone = Object.create(
    Object.getPrototypeOf(obj),
    Object.getOwnPropertyDescriptors(obj)
);
```

Object.getOwnPropertyDescriptors()

Async functions

```
doWork();
console.log('working');

// 'working'
```

```
function doWork() {
    console.log('start');



}

doWork();
console.log('working');
// 'start'
// 'working'
```

```
function doSomethingAsync(callback) {




}

function doWork() {
    console.log('start');
    doSomethingAsync();


}

doWork();
console.log('working');
// 'start'
// 'working'
```

```
function doSomethingAsync(callback) {
    if (typeof callback === 'function') {
        setTimeout(function () {
            callback('something');
        }, 1000);
    }
}

function doWork() {
    console.log('start');
    doSomethingAsync(function (value) {
        console.log(value);
    });
}

doWork();
console.log('working');
// 'start'
// 'working'
// 'something'
```

```
function doSomethingAsync() {
    return new Promise(function (resolve, reject) {
        setTimeout(function () {
            resolve('something');
        }, 1000);
    });
}

function doWork() {
    console.log('start');
    doSomethingAsync().then(function (value) {
        console.log(value);
    });
}

doWork();
console.log('working');
// 'start'
// 'working'
// 'something'
```

```
function doSomethingAsync() {
    return new Promise(function (resolve, reject) {
        setTimeout(function () {
            resolve('something');
        }, 1000);
    });
}

async function doWork() {
    console.log('start');
    var value = await doSomethingAsync();
    console.log(value);

}

doWork();
console.log('working');
// 'start'
// 'working'
// 'something'
```

```
function doSomethingAsync() {
    return new Promise(function (resolve, reject) {
        setTimeout(function () {
            resolve('something');
        }, 1000);
    });
}

async function doWork() {
    console.log('start');
    var value = await doSomethingAsync();
    console.log(value);
    value += await doSomethingAsync();
    console.log(value);
}

doWork();
console.log('working');
// 'start'
// 'working'
// 'something'
// 'somethingsomething'
```

```
function doSomethingAsync() {
    return new Promise(function (resolve, reject) {
        setTimeout(function () {
            reject('you failed! hah!');
        }, 1000);
    });
}

async function doWork() {
    console.log('start');
    var value;
    try {
        value = await doSomethingAsync();
    } catch (e) {
        value = 'error: ' + e.message;
    }
    console.log(value);
}

doWork();
console.log('working');
// 'start'
// 'working'
// 'you failed! hah!'
```

https://ponyfoo.com/articles/understanding-javascript-async-await

Async functions

Trailing commas in function parameter lists and calls

Trailing commas in objects and arrays

```
let obj = {
    first: 'Jane',
    last: 'Doe',
};

let arr = [
    'red',
    'green',
    'blue',
];
console.log(arr.length); // 3
```

In function parameter lists and calls

```
function foo(
    param1,
    param2,
) {}


foo(
    'abc',
    'def',
);
```

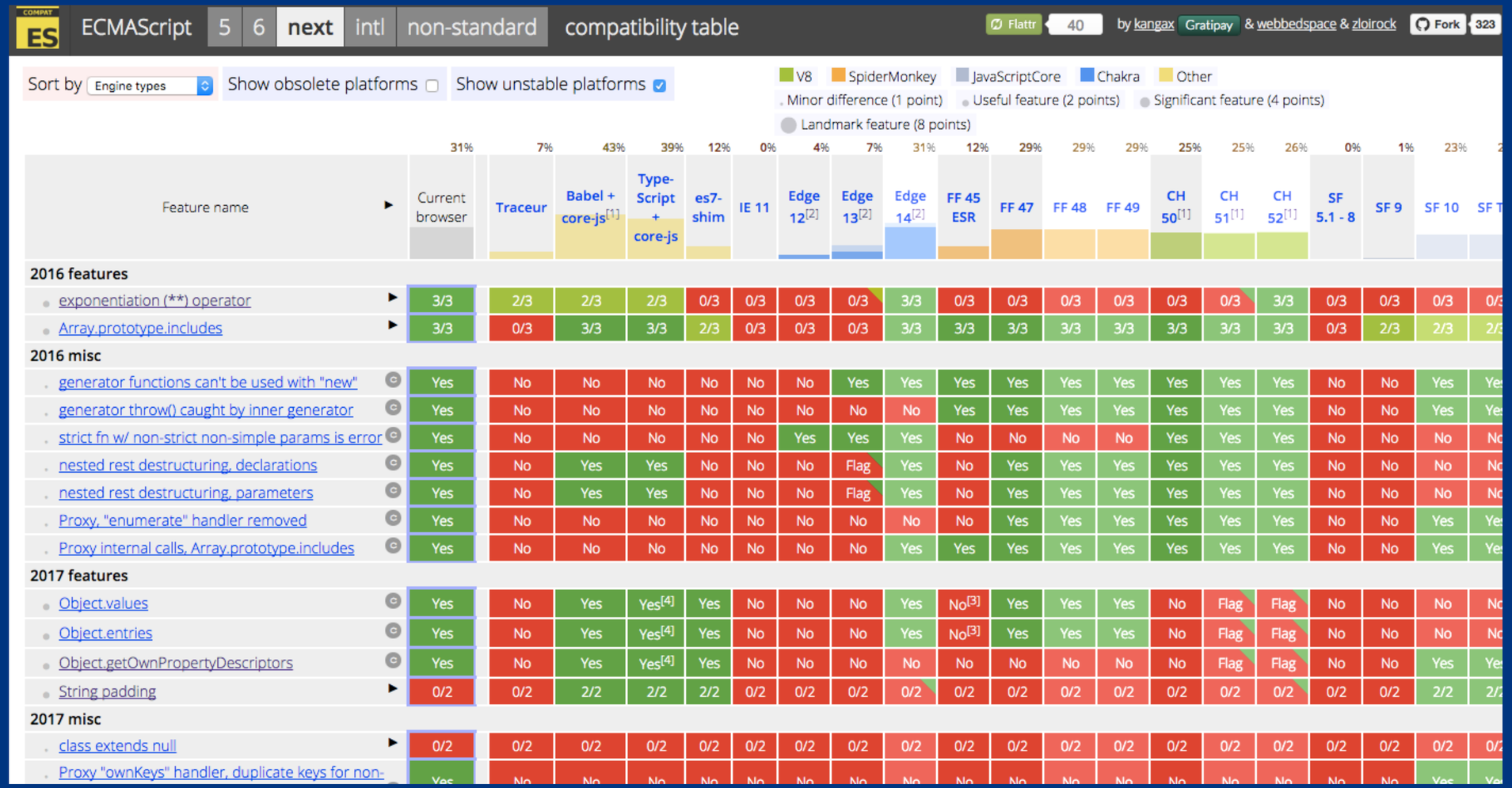
Trailing commas in function parameter lists and calls

Stage 3: Candidate

Spec text complete, has at least 2 implementations

- SIMD.JS - SIMD APIs + polyfill
- `Function.prototype.toString` revision
- Lifting Template Literal Restriction
- global
- Rest/Spread Properties
- Asynchronous Iteration

Edge is pretty awesome 😃



http://kangax.github.io/compat-table/esnext/

Status: https://github.com/tc39/ecma262

Specs: https://tc39.github.io/ecma262/

https://ponyfoo.com/

http://www.2ality.com/

Let’s stop calling it ‘ECMAScript x’

and start call it JavaScript again!


Booking.com
workingatbooking.com


Expedia®

 @_paulverbeek

 @nlhtml5

 verbeek.p@gmail.com

 705649555

Questions?