

ECMASCRIPT 2018 Y MÁS ALLÁ

 Pablo Magaz
<https://pablomagaz.com>

 github.com/pmagaz

 twitter.com/pablo_magaz



The logo for "Back to the Future" features the words "BACK TO THE FUTURE" in a bold, stylized font. The letters are primarily yellow with blue outlines, set against a white background. The word "TO" is smaller and positioned between "BACK" and "THE". "THE" is written in a cursive script, while "FUTURE" is in a bold, sans-serif font. A series of diagonal, wavy lines extend from the right side of the "K" in "BACK" towards the right edge of the "U" in "FUTURE", creating a sense of motion or travel. The entire logo is enclosed in a thin black border.

BACK
TO THE FUTURE™

傳 Array flatten...

```
const nestedArray = [1, 2, [3, 4]];
nestedArray.flat();
// [1, 2, 3, 4]
```

Dynamic imports...

```
import(`./my-modules/${ dynamicModuleName }.js`);
```

管道 Pipes...

```
const functionOne = x => x;
```

```
const functionTwo = y => y;
```

```
let result = 'Hello Codemotion'
```

```
|> functionOne
```

```
|> functionTwo
```

“El TC39 es el comité encargado de seleccionar qué especificaciones se incorporan al standard”

STAGE 0:
Presentación

STAGE 1:
Propuesta formal

STAGE 2:
Borrador

STAGE 3:
Candidato

STAGE 4:
Seleccionado
para el standard

BABEL

Plugins / Presets



Stage-0



Stage-1



Stage-2



Stage-3



ECMAScript 2018

Async Iterators

倦 Iterable...

```
const iterable = 'Hello Codemotion';
const iterable2 = ['Hello', 'Codemotion'];
```

```
for (let x of iterable) {
    console.log(x);
}
// H
// e
// l...
```

Iterator pattern...

```
const simpleIterator = data => {
  let cursor = 0;
  return {
    next: () => ( cursor < data.length ? data[cursor++] : false )
  }
}
```

```
const consumer = simpleIterator(['Hello', 'Codemotion']);
console.log(consumer.next()); // 'Hello'
console.log(consumer.next()); // 'Codemotion'
console.log(consumer.next()); // false
```

傳 Iterator...

```
const iterable = ['Hello', 'Codemotion'];
const iterator = iterable[Symbol.iterator]();

console.log(iterator.next()); // { value: 'Hello', done: false }
console.log(iterator.next()); // { value: Codemotion', done: false }
console.log(iterator.next()); // { value: undefined, done: true }
```

傳 Generator...

```
function* generator() {  
    yield 'Hello';  
    yield 'Codemotion';  
}  
  
const iterator = generator();  
console.log(iterator.next()) // { value: 'Hello', done: false }  
console.log(iterator.next()) // { value: 'Codemotion', done: false }  
console.log(iterator.next()) // { value: undefined, done: true }
```

傳 Async / Await...

```
async function myAsyncFunction() {  
    const req = await fetch(`myUrl`);  
    const data = await req.text();  
    return JSON.parse(data);  
}
```

Iterables + **Iterators** + **Generators** + **Async/await**

“Los iteradores asíncronos nos permiten iterar sobre estructuras de datos asíncronos”

```
const promise1 = new Promise(resolve => resolve('Hello'));

const promise2 = new Promise(resolve => resolve("Codemotion"));

async function* asynclerator () {
    yield await promise1
    yield await promise2
}

async function getAsyncData () {
    for await (let x of asynclerator()) {
        console.log(x); // Hello, Codemotion
    }
}
```

for await of

```
const promise1 = new Promise(resolve => resolve('Hello'));

const promise2 = new Promise(resolve => resolve("Codemotion"));

async function* asynclterator () {
    yield await promise1
    yield await promise2
}

async function getAsyncData () {
    for await (const x of asynclterator()) {
        console.log(x); // Hello, Codemotion
    }
}
```

```
const promise1 = new Promise(resolve => resolve('Hello'));

const promise2 = new Promise(resolve => resolve("Codemotion"));

async function* asynclerator () {
    yield await promise1
    yield await promise2
}

const asynclerator = asynclerator()

asynclerator.next()
.then(value => asynclerator.next()) // { value: 'Hello', done: false }
.then(value => asynclerator.next()) // { value: 'Codemotion', done: false }
```

傳 Object Spread & Rest properties

```
let { a, b, ...c } = { a: 'Hello', b: 'Codemotion', x: true, y: false };
console.log(a); // Hello
console.log(b); // Codemotion
console.log(c); // { x: true, y: false }
```

Promise Finally

```
fetch('https://pablomagaz.com/api/posts')  
  .then(res => {  
    processResponse(res);  
  })  
  .catch(err => {  
    handleErrors(err);  
  })  
  .finally(() => {  
    hideLoading();  
});
```



💡 Grupos de captura...

```
const myDate = /([0-9]{2})-([0-9]{2})-([0-9]{4})/;  
const match = myDate.exec('23-11-2018');  
const day = match[0]; // 01  
const month = match[1]; // 03  
const year = match[2]; // 2018
```

RegExp Named Capture Groups

```
const myDate = /(?<day>[0-9]{2})-(?<month>[0-9]{2})-(?<year>[0-9]{4})/;  
const match = myDate.exec('23-11-2018');  
const day = match.groups.day; // 01  
const month = match.groups.month; // 03  
const year = match.groups.year; // 2018
```

傍 Lookaround Assertions...

(?=€) // Lookahead assertion 100€

(?<=€) // Lookbehind assertion €100

傍 Lookbehind Assertions

```
const euroPrice = /(?<=\u20AC)(?<price>\d+(?:\.\d+)?)$/u // \u20AC € Unicode
console.log(euroPrice.exec('€100').groups.price); // 100
console.log(euroPrice.exec('100€ ')); // null
```



BACK
TO THE FUTURE™

The logo for "Back to the Future" features the words "BACK", "TO THE", and "FUTURE" stacked vertically. "BACK" is at the top in a large, bold, yellow font with a red outline. "TO THE" is in the middle in a smaller, blue font with a yellow outline. "FUTURE" is at the bottom in a large, bold, red font with a yellow outline. A blue lightning bolt graphic is positioned behind the letters "C" and "K" in "BACK". A small "TM" symbol is located at the end of "FUTURE".

Stage 3

(más allá)

提案: Dynamic Imports

```
import(`./my-modules/${ dynamicModuleName }.js`)
.then(module => {
  module.doWickedStuff();
})
.catch(err => console.log(err));
```

Fake private properties & methods...

```
class MyClass {  
    _myFakePrivateProp = 1;  
  
    this._myFakePrivateMethod(){  
        return false;  
    }  
}
```

傍 Proposal: Private fields & methods

```
class MyClass {  
    a = 1; // Public property  
    #b = 2; // Private property  
    #increment(){ // Private method  
        this.#b++;  
    }  
}  
  
const myObject = new MyClass();  
myObject.#b; // ERROR!  
myObject.#increment(); // ERROR!
```

待 Proposal: Array Flat & FlatMap

```
const nestedArray = [1, 2, [3, 4]];
nestedArray.flat();
// [1, 2, 3, 4]
```

```
const myArr =['Back to',",",'the future'];
myArr.flatMap(x => x.split(' '));
// ["Back","to","","the","future"];
```

Stage 2

(mucho más allá)

修士 Proposal: Top level Await

```
const myHandler = await asyncHandler();
```

...

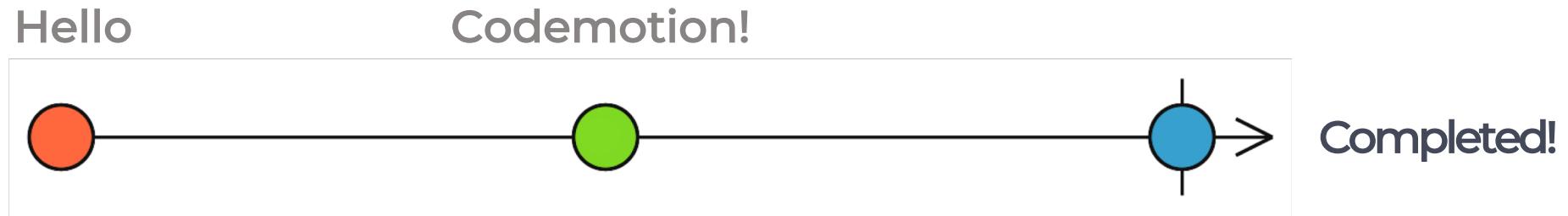
```
let myModule;  
try {  
    myModule = await import(`./modules/${ dynamicModuleName }.js`);  
} catch {  
    foo();  
}
```

Stage 1

(muchísimo más allá)

💡 Proposal: Observable

```
const myObservable = new Observable(observer => {  
  observer.next('Hello');  
  observer.next('Codemotion!');  
  observer.complete();  
});
```



提案 Proposal: Observable

```
myObservable.subscribe({  
    next(value) { console.log(value) }, // Hello, Codemotion!  
    error(err) { console.log(err) }, //  
    complete() { console.log(`Done!`) } // Done!  
});
```

↑ Proposal: Pipeline Operator

|>

Proposal: Pipeline Operator

```
const double = n => n * 2;  
const increment = n => n + 1;
```

```
let result = 5  
    |> double  
    |> increment  
    |> double
```

Gracias.
¿Preguntas?



Slides: pablomagaz.com/#talks



github.com/pmagaz



twitter.com/pablo_magaz