What's new in ES2019?

Let's look at what new features will be coming to JavaScript

WE'LL COVER THE FOLLOWING

We'll look at what's included in the latest version of `ECMAScript`: ES2019 in this chapter.

`Array.prototype.flat()` / `Array.prototype.flatMap()`

`Array.prototype.flat()` will flatten the array recursively up to the depth that we specify. If no depth argument is specified, 1 is the default value. We can use `Infinity` to flatten all nested arrays.

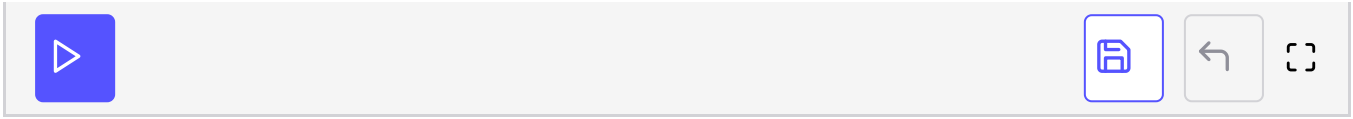
```
const letters = ['a', 'b', ['c', 'd', ['e', 'f']]];
// default depth of 1
console.log(letters.flat());
// ['a', 'b', 'c', 'd', ['e', 'f']]

// depth of 2
console.log(letters.flat(2));
// ['a', 'b', 'c', 'd', 'e', 'f']

// which is the same as executing flat with depth of 1 twice
console.log(letters.flat().flat());
// ['a', 'b', 'c', 'd', 'e', 'f']
```

```
// Flattens recursively until the array contains no nested arrays


console.log(letters.flat(Infinity));
// ['a', 'b', 'c', 'd', 'e', 'f']
```

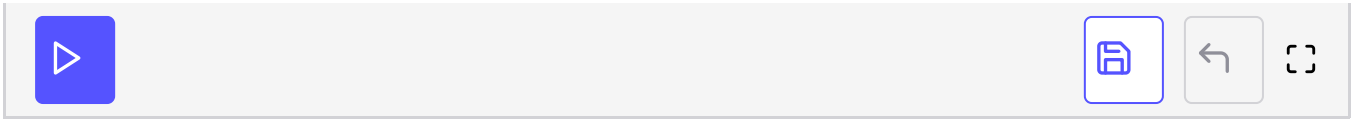


`Array.prototype.flatMap()` is identical to the previous one with regards to the way it handles the depth argument, but instead of simply flattening an array, with `flatMap()` we can also map over it and return the result in the new array.

```
let greeting = ["Greetings from", " ", "Vietnam"];

// let's first try using a normal `map()` function
greeting.map(x => console.log(x.split(" ")));
// ["Greetings", "from"]
// ["", ""]
// ["Vietnam"]


greeting.flatMap(x => console.log(x.split(" ")))
// ["Greetings", "from", "", "", "Vietnam"]
```



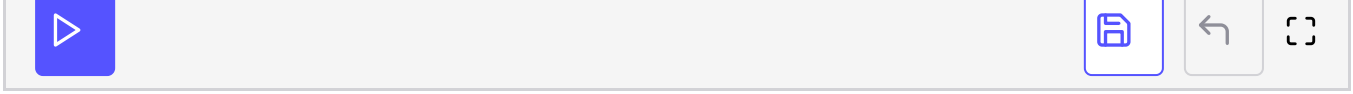
As you can see, if we use `.map()` we will get a multi level array, which is a problem that we can solve by using `.flatMap()`. This will also flatten our array.

`Object.fromEntries()`

`Object.fromEntries()` transforms a list of key-value pairs into an object.

```
const keyValueArray = [
  ['key1', 'value1'],
  ['key2', 'value2']
]

const obj = Object.fromEntries(keyValueArray);
console.log(obj);
// {key1: "value1", key2: "value2"}
```



We can pass any iterable as argument of `Object.fromEntries()`, whether it's an `Array`, a `Map` or other objects implementing the iterable protocol.

You can read more about the iterable protocol here.

`String.prototype.trimStart()` / `.trimEnd()`

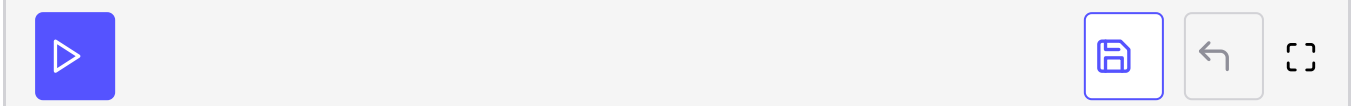
`String.prototype.trimStart()` removes white space from the beginning of a string while `String.prototype.trimEnd()` removes them from the end.

```
let str = "    this string has a lot of whitespace   ";

str.length;
// 42

str = str.trimStart();
console.log(str);
// "this string has a lot of whitespace   "
console.log(str.length);
// 38

str = str.trimEnd();
console.log(str);
// "this string has a lot of whitespace"
console.log(str.length);
// 35
```



We can also use `.trimLeft()` as an alias of `.trimStart()` and `.trimRight()` as an alias of `.trimEnd()`.

Optional Catch Binding

Prior to ES2019, including an exception variable in your `catch` clause. E2019 allows you to omit it.

```
// Before
try {
    ...
```

```
} catch(error) {
   ...
}


// ES2019
try {
   ...
} catch {
   ...
}
```

This is useful when you want to ignore the error. For a more detailed list of use cases for this I highly recommend this article.

`Function.prototype.toString()`

The `.toString()` method returns a string representing the source code of the function.

```
function sum(a, b) {
  return a + b;
}

console.log(sum.toString());
// function sum(a, b) {
//    return a + b;
//  }
```




It also includes comments.

```
function sum(a, b) {
  // perform a sum
  return a + b;
}

console.log(sum.toString());
// function sum(a, b) {
//   // perform a sum
//   return a + b;
// }
```

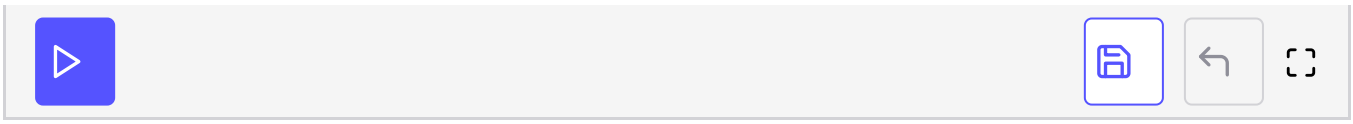



`Symbol.prototype.description`

`.description` returns the optional description of a `Symbol` Object.

```
const me = Symbol("Alberto");
console.log(me.description);
// "Alberto"

console.log(me.toString());
//  "Symbol(Alberto)"
```



Get ready for a coding challenge to test these concepts.