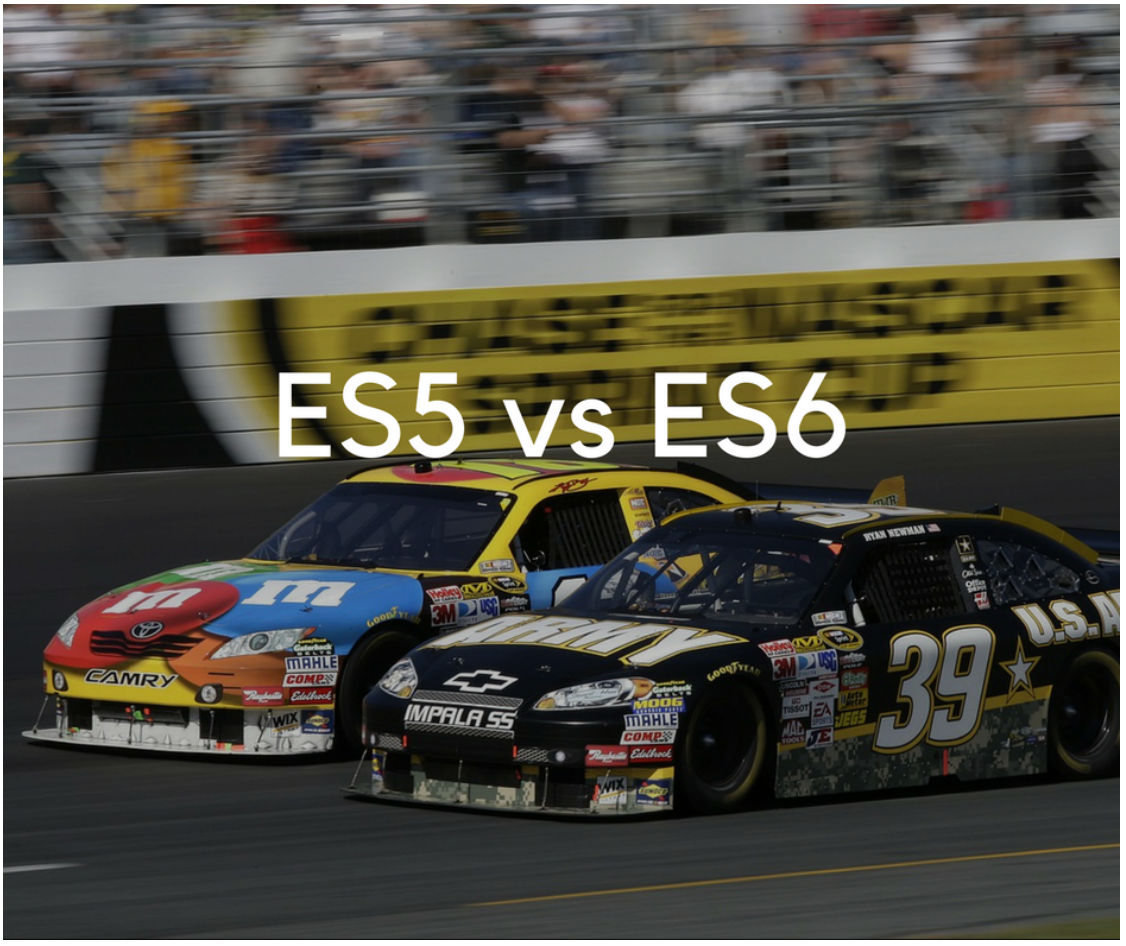
ES5 vs ES6 (With example code)

We have came so far

Manoj Singh Negi Follow
Sep 17, 2017 · 6 min read



Pixabay & canva

I have just launched **ProgrammerJournal.com**. **A standalone blog where I write about Javascript, Web development and software development.**

Reducing code with Arrow Function

Arrow functions brought a lot of clarity & code reduction to Javascript. Let's take a look at different ways we can define function now.

Here is the ES5 version

```
function greetings (name) {
 return 'hello ' + name
}
```

Now take a look at different ways we can define function in ES6

```
const greetings = (name) => {
 return `hello ${name}`;
}
```

You can see the difference we don't have to use the `function` keyword to define the function now but that's now a big improvement right ?

Have a look at another way we can define a function in ES6

```
const greetings = name => `hello ${name}`;
```

This is some code reduction we are talking about earlier. In ES6 if your function has a single parameter then you can altogether ditch the parenthesis around the parameter.

Another thing to notice that we don't have to write the `return` keyword to return the computed value, in ES6 if you don't write function body inside braces the computed expression automatically get returned when the function will be executed.

Manipulating objects in ES6 vs ES5

Objects get a major overhaul in ES6. Things like object destructuring and rest/spread operators made it working with objects very easy now. Let's jump to the code and try to merge two objects in ES5.

```
var obj1 = { a: 1, b: 2 }
var obj2 = { a: 2, c: 3, d: 4}
var obj3 = Object.assign(obj1, obj2)
```

We have to merge the object using `Object.assign()` which takes both objects as input and outputs the merged object. Let's take a look how we can tackle this problem in ES6.

```
const obj1 = { a: 1, b: 2 }
const obj2 = { a: 2, c: 3, d: 4}
```

```
const obj3 = {...obj1, ...obj2}
```

Simple isn't it ? The spread operator makes merging objects a breeze for the developer.

Let's take a look at object destructuring now. If you have to extract multiple values from ES5 you have to write 3–4 lines of code like this:

```
var obj1 = { a: 1, b: 2, c: 3, d: 4 }
var a = obj1.a
var b = obj1.b
var c = obj1.c
var d = obj1.d
```

Time consuming 😕. Oh wait! we have ES6 here to rescue us.

```
const obj1 = { a: 1, b: 2, c: 3, d: 4 }
const {
  a,
  b,
  c,
  d
} = obj1
```

Cool! Last but not least look at another new feature for introduced for objects.

We define a object like this in ES5

```
var a = 1
var b = 2
var c = 3
var d = 4

var obj1 = { a: a, b: b, c: c, d: d }
```

In ES6 you will do something like this:

```
var a = 1
var b = 2
var c = 3
var d = 4

var obj1 = { a, b, c, d }
```

Yeah, if the name of the key and the variable we are going to assign to that key are same you can use this shorthand.

Promises vs Callbacks

Javascript is an Async language we all know that. This feature gives us a lot of freedom when we write code. We have a non-blocking architecture in our hand because of which we can write non-dependent code easily.

Here is an example how we write an Async function in ES5

```
function isGreater (a, b, cb) {

  var greater = false
  if(a > b) {
    greater = true
  }
  cb(greater)

}

isGreater(1, 2, function (result) {
  if(result) {
    console.log('greater');
  } else {
    console.log('smaller')
  }
})
```

Above we defined a function named `isGreater` , which takes three arguments `a` , `b` and `cb` . When executed the function check if `a` greater than `b` and make the `greater` variable `true` , if not `greater` stays `false` . After that `isGreater` calls the callback function `cb` and pass the `greater` variable as the argument to the function.

In the next piece of code we call the `isGreater` function pass it `a` and `b` alongside our callback function. Inside callback function we check if the result is `true` or `false` and shows message according to it. Now let's see how ES6 have handled this.

```
const isGreater = (a, b) => {
 return new Promise ((resolve, reject) => {
  if(a > b) {
   resolve(true)
  } else {
   reject(false)
  }
 })
}

isGreater(1, 2)
 .then(result => {
    console.log('greater')
```

```
})
.catch(result => {
   console.log('smaller')
})
```

The ES6 Promises allows us to `resolve` and `reject` a request. Whenever we resolve a request it will call the callback provided in `then` and whenever we reject a request it will call the `catch` callback.

The ES6 promises are better then callback because it allows us to distinguish easily between a `success` and `error` so we don't have to check again for things in our callback function.

Exporting & Importing Modules

exporting and importing module syntax changed completely with the introduction of ES6 specification. Let's take a look how we export a module in ES5

```
var myModule = { x: 1, y: function(){ console.log('This is ES5')
}}

module.exports = myModule;
```

Here is the ES6 implementation

```
const myModule = { x: 1, y: () => { console.log('This is ES5') }}

export default myModule;
```

The ES6 syntax is more readable here. Alongside the `export` keyword ES6 also comes up with `export default` later on this firstly let's take a look how importing a module changed in ES6.

Here is the ES5 version

```
var myModule = require('./myModule');
```

Here is the ES6 version

```
import myModule from './myModule';
```

Cool ha !!

Okay so as promised now let's talk about the `export default` . When you export something using default we will import a module like this.

```
import myModule from './myModule';
```

The above line means something like this, we exported a module by default and we have to import that whole module in your source file.

But ES6 also provides us with an ability to `export` and `import` multiple child modules or variables from a single module.

So in your module file you will `export` your module something like this

```
export const x = 1;
export const y = 2;
export const z = 'String';
```

And `import` them something like this

```
import {x, y, z} from './myModule';
```

Pretty neat ha ? Here we used ES6 object destruction to import multiple child modules from one single module.

Hi, My name is Manoj Singh Negi. I am a Javascript Developer and writer follow me at Twitter or Medium.

I am available to give a public talk or for a meetup hit me up at justanothermanoj@gmail.com if you want to meet me.

Really loved this article ?

Please **subscribe to my blog**. You will receive articles like this one directly in your Inbox frequently.

Here are more articles for you.

1. What are HTML Custom Elements
2. Trying out Shadow Dom
3. Building a React component as an NPM module
4. Throttle function in javascript with the example
5. React Context API Explained
6. Introduction to Haskell Ranges
7. HOC (Higher-order components) in ReactJS explained

Peace.

codeburst.io

JavaScript ES6 React Promises Web Development