ES6 for beginners

Srebalaji Thirumalai Follow
Sep 29, 2017 · 7 min read

ES6 for beginners

In this post, I will cover some new features in ES6. It will be helpful if you are new to ES6 or learning front-end frameworks.

Topics I'm gonna cover in this post

1. Let and Const
2. Arrow functions
3. Default parameters
4. for of loop
5. Spread attributes
6. Maps
7. Sets

8. Static methods
9. Getters and Setters

Let

let is similar to var but let has scope. let is only accessible in the block level it is defined.

```
if (true) {
 let a = 40;
 console.log(a); //40
}
console.log(a); // undefined
```

In the above example variable ‘a’ is defined inside If statement and so it’s not accessible outside the function.

Another example:

```
let a = 50;
let b = 100;
if (true) {
 let a = 60;
 var c = 10;
 console.log(a/c); // 6
 console.log(b/c); // 10
}
console.log(c); // 10
console.log(a); // 50
```

Const

Const is used to assign a constant value to the variable. And the value cannot be changed. Its fixed.

```
const a = 50;
a = 60; // shows error. You cannot change the value of
const.
```

```
const b = "Constant variable";
b = "Assigning new value"; // shows error.
```

Consider another example.

```
const LANGUAGES = ['Js', 'Ruby', 'Python', 'Go'];

LANGUAGES = "Javascript"; // shows error.

LANGUAGES.push('Java'); // Works fine.
console.log(LANGUAGES); // ['Js', 'Ruby', 'Python', 'Go',
'Java']
```

This may be little confusing.

Consider in this way. Whenever you define a const variable, Javascript references the address of the value to the variable. In our example the variable 'LANGUAGES' actually references to the memory allocated to the array. So you cannot change the variable to reference some other memory location later. Throughout the program it only references to the array.

Arrow Function

Functions in ES6 have changed a bit. I mean the syntax.

```
// Old Syntax
function oldOne() {
 console.log("Hello World..!");
}

// New Syntax

var newOne = () => {
 console.log("Hello World..!");
}
```

The new syntax may be confusing a little bit. But I will try to explain the syntax.

There are two parts of the syntax.

The first part is just declaring a variable and assigning the function (i.e) () to it. It just says the variable is actually a function.

Then the second part is declaring the body part of the function. The arrow part with the curly braces defines the body part.

Another example with parameters.

```
let NewOneWithParameters = (a, b) => {
 console.log(a+b); // 30
}
NewOneWithParameters(10, 20);
```

I don't think I need to give explanation for this. Its straightforward.

Default Parameters:

If you are familiar with other programming languages like Ruby, Python then default parameters isn't new to you.

Default parameters are parameters which are given by default while declaring a function. But it's value can be changed when calling the function.

Example

```
let Func = (a, b = 10) => {
 return a + b;
}
Func(20); // 20 + 10 = 30
```

In the above example, we are passing only one parameter. The function makes use of the default parameter and executes the function.

Consider another example:

```
Func(20, 50); // 20 + 50 = 70
```

In the above example, the function takes two parameters and the second parameter replaces the default parameter.

Consider another example:

```
let NotWorkingFunction = (a = 10, b) => {
 return a + b;
}
NotWorkingFunction(20); // NAN. Not gonna work.
```

When you are calling the function with parameters they get assigned in the order. (i.e) the first value gets assigned to the first parameter and the second value gets assign to the second parameter and so on..

In the above example, the value 20 gets assigned to parameter ‘a’ and ‘b’ is not having any value. So we are not getting any output.

But,

```
NotWorkingFunction(20, 30); // 50;
```

Works fine.

For of loop

for..of is very similar to for..in with slight modification.

for..of iterates through list of elements (i.e) like Array and returns the elements (not their index) one by one.

```
let arr = [2,3,4,1];
for (let value of arr) {
 console.log(value);
}
```

```
Output:
2
3
4
1
```

Note that the variable ‘value’ outputs each element in the array not the index.

Another Example

```
let string = "Javascript";
for (let char of string) {
 console.log(char);
}
```

```
Output:
J
a
v
a
s
c
r
i
p
t
```

Yes. It works for string too.

Spread attributes

Spread attributes help to spread the expression as the name suggests. In simple words, it converts a list of elements to an array and vice versa.

Example without spread attributes:

```
let SumElements = (arr) => {
 console.log(arr); // [10, 20, 40, 60, 90]
```

```
let sum = 0;
 for (let element of arr) {
 sum += element;
 }
 console.log(sum); // 220.
}
```

```
SumElements([10, 20, 40, 60, 90]);
```

Above example is straightforward. We are declaring a function to accept array as parameter and returning its sum. Its simple.

Now consider the same example with spread attributes

```
let SumElements = (...arr) => {
 console.log(arr); // [10, 20, 40, 60, 90]
```

```
let sum = 0;
 for (let element of arr) {
 sum += element;
 }
 console.log(sum); // 220.
}
```

```
SumElements(10, 20, 40, 60, 90); // Note we are not passing
array here. Instead we are passing the elements as
arguments.
```

In the above example, the spread attribute converts the list of elements (i.e) the parameters to an array.

Another Example:

```
Math.max(10, 20, 60, 100, 50, 200); // returns 200.
```

Math.max is a simple method that returns the maximum element from given list. It doesn't accept an array.

```
let arr = [10, 20, 60];
Math.max(arr); // Shows error. Doesn't accept an array.
```

So lets use our savior.

```
let arr = [10, 20, 60];
Math.max(...arr); // 60
```

In the above example, the spread attribute converts the array to list of elements.

Maps

Map holds key-value pairs. It's similar to an array but we can define our own index. And indexes are unique in maps.

Example:

```
var NewMap = new Map();
NewMap.set('name', 'John');
NewMap.set('id', 2345796);
NewMap.set('interest', ['js', 'ruby', 'python']);
```

```
NewMap.get('name'); // John
NewMap.get('id'); // 2345796
NewMap.get('interest'); // ['js', 'ruby', 'python']
```

I think the above example is self explanatory.

Other interesting features of Maps are all indexes are unique. And we can use any value as key or value.

Example:

```
var map = new Map();
map.set('name', 'John');
map.set('name', 'Andy');
map.set(1, 'number one');
map.set(NaN, 'No value');
```

```
map.get('name'); // Andy. Note John is replaced by Andy.
map.get(1); // number one
map.get(NaN); // No value
```

Other useful methods used in Map:

```
var map = new Map();
map.set('name', 'John');
map.set('id', 10);
```

```
map.size; // 2. Returns the size of the map.
```

```
map.keys(); // outputs only the keys.
map.values(); // outputs only the values.
```

```
for (let key of map.keys()) {
 console.log(key);
}
```

```
Output:
name
id
```

In the above example, map.keys() returns the keys of the map but it returns it in Iterator object. It means that it can't be displayed as it is. It should be displayed only by iterating.

Another example:

```
var map = new Map();

for (let element of map) {
 console.log(element);
}

Output:
['name', 'John']
['id', 10]
```

The above example is self explanatory. The for..of loop outputs the key-value pair in array.

We can optimise it a little bit.

```
var map = new Map();

for (let [key, value] of map) {
 console.log(key+" - "+value);
}

Output:
name - John
id - 10
```

Sets

Sets are used to store the unique values of any type. Simple..!

Example

```
var sets = new Set();
sets.add('a');
sets.add('b');
sets.add('a'); // We are adding duplicate value.
```

```
for (let element of sets) {
 console.log(element);
}
```

```
Output:
a
b
```

Note that no duplicate values are displayed. Unique values are displayed.

And also note that sets are iterable objects. We have to iterate through the elements to display it.

Other useful methods:

```
var sets = New Set([1,5,6,8,9]);
```

```
sets.size; // returns 5. Size of the set.
sets.has(1); // returns true.
sets.has(10); // returns false.
```

In the above example, size is self-explanatory. There is another method ‘has’ which return a boolean value based on whether the given element is present in the set or not.

Static methods

Most of you have already heard about static methods. Static methods are introduced in ES6. And it is pretty much easy to define it and use it.

Example:

```
class Example {
 static Callme() {
 console.log("Static method");
 }
}
Example.Callme();
```

```
Output:
Static method
```

Note that I didn't use the keyword 'function' inside Class.

And I can call the function without creating any instance for the class.

Getters and Setters

Getters and setters and one of the useful feature introduced in ES6. It will come in handy if you are using classes in JS.

Example without getters and setters:

```
class People {

constructor(name) {
 this.name = name;
 }
 getName() {
 return this.name;
 }
 setName(name) {
 this.name = name;
 }
}

let person = new People("Jon Snow");
console.log(person.getName());
person.setName("Dany");
console.log(person.getName());

Output:
Jon Snow
Dany
```

I think the above example is self-explanatory. We have two functions in class People that helps to set and get the name of the person.

Example with getters and setters

```
class People {

constructor(name) {
 this.name = name;
 }
 get Name() {
 return this.name;
 }
 set Name(name) {
 this.name = name;
```

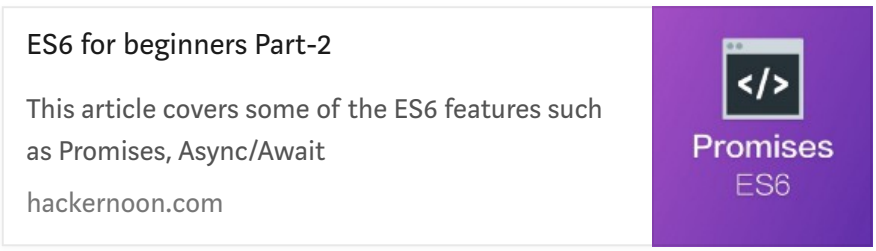
```
 }
}

let person = new People("Jon Snow");
console.log(person.Name);
person.Name = "Dany";
console.log(person.Name);
```

In the above example, you can see there are two functions inside class People with ‘get’ and ‘set’ properties. The ‘get’ property is used to get the value of the variable and ‘set’ property is used to set the value to the variable.

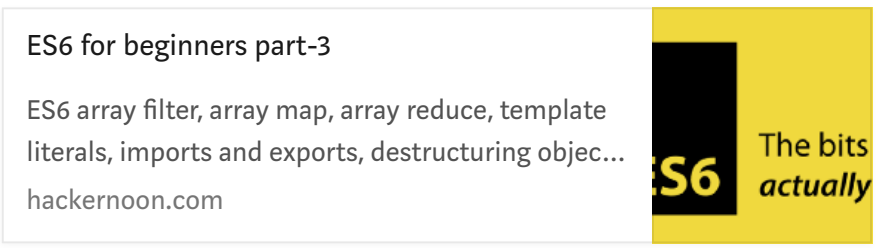
And you can see that getName function is called without parenthesis. And setName function is called without parenthesis and it’s just like assigning a value to the variable.

Thank you for your time. Hope you enjoyed the article. :) :)

I have covered about **promises** and **async/await** in the second part. Check it out



I have covered much more interesting topics such as array map, array filter, reduce, imports/exports, destructuring, etc in the next part. Check it out



. . .

Originally published at medium.com on September 29, 2017.