ES6 for beginners part-3

Srebalaji Thirumalai Follow
Dec 10, 2017 · 8 min read

ES6 for beginners

Part 1 of the article appeared here

Part 2 of the article appeared here

Topics I'm gonna cover in this article

1. Array map
2. Array filter
3. Array reduce
4. Template literals
5. Imports and exports
6. Destructuring objects and arrays
7. Extend and super

Array Map

Map operator is used to do a specific operation on all elements of an array and it returns an array with modified elements.

It's very easy to implement. Let's see an example.

```
let arr = [1,2,3,4,5];

let modifiedArr = arr.map(function(element, index, arr) {
  return element * 10;
});
console.log(modifiedArr);

Output:
[10, 20, 30, 40, 50]
```

As you can see **map** takes a function with three parameters.

1. First parameter is the element itself.
2. Second parameter is the index of the element.
3. Third parameter is the whole array.

And also note we have to return some value in the end. Which will be the modified value of that element. If you didn't return anything then the particular element will be undefined.

One more thing I like to add is the second and third parameter is only optional. Only the first parameter is mandatory.

```
let modifiedArr = arr.map(function(element) {
  return element * 10;
});
```

As you can see map takes a function with a single parameter.

okay.. Let's try to write the map operator with arrow functions.

```
let modifiedArr = arr.map((element, index) => {
  console.log("index "+index);
```

```
    return element * 10;
  });
  console.log(modifiedArr);

  Output:
  index 0
  index 1
  index 2
  index 3
  index 4
  [10, 20, 30, 40, 50]
```

I think I don't have to explain much. It's simple.

One last example.

```
let modifiedArr = arr.map(element => element * 10);
console.log(modifiedArr);
```

Cool right :)

If you know arrow functions well, it should be easy to understand. But if you don't get it I will try to explain.

There are two things to explain here.

1. If you have a function with single parameter, you don't need (). In our case **element** is the parameter.
2. And If you have single line as the body of the function you don't need {} and also JS will implicitly returns the value after executing the function. You don't have to use **return** keyword.

Array Filter

Array filter is used to filter the whole array based on some condition. Array filter takes each element of an array and checks with the given condition. If the element passes the condition it keeps the element in the array else it removes the element.

Let's see some example.

```
let arr = [1, 2, 3, 4, 5, 6]
let modifiedArr = arr.filter(function(element, index, array)
{
  return element % 2 == 0
});
console.log(modifiedArr);

Output:
[2, 4, 6]
```

As you can see in the above example, **filter** like **map** takes three parameter.

And we have to return a boolean value for each element of the array. If you won't return any boolean value at the end then the filter takes it as **false** and deletes the element.

okay.. Let's try it in arrow function.

```
let modifiedAarr = arr.filter((element, index) => element%2
== 0)
```

Hope you got it. And don't forgot to notice, only the first parameter is mandatory. Other two parameters are optional.

Array Reduce

Array reduce is used to aggregate all the elements of an array and return a single value.

Let's see some example

```
let arr = [1,2,3,4,5,6]
let total= arr.reduce(function(sum, element, index, array) {
  return sum + element;
},0);
console.log("total is "+total);

Output:
total is 21
```

Unlike filter and map, reduce takes a function with four parameters and also a additional element. In our case it's 0.

Let’s see how it works.

The first parameter is the aggregator element. It has the sum of elements at any given point. And it’s initial value is defined as the additional element. In our case it’s 0.

Then the second, third and fourth parameter are same as filter and map.

Like filter and map you have to return the end result.

Let’s see another example.

```
let arr = [1,2,3,4,5,6];
let totalSum = arr.reduce(function(sum, element, index,
array) {
  console.log(sum+"+"+element+"="+sum+element);
  return sum + element;
}, 10);
console.log("Total sum is "+ totalSum);
```

```
Output:
10 + 1 = 11
11 + 2 = 13
13 + 3 = 16
16 + 4 = 20
20 + 5 = 25
25 + 6 = 31
Total sum is 31
```

Hope you got the example. Note that I have set the initial value of the aggregator element **sum** to 10 initially.

Let’s write the same code with arrow functions.

```
let totalSum = arr.reduce((sum, element) => element+sum, 0)
```

Hope you got it. It’s straight forward.

Unlike filter and map, the first two parameters are mandatory. Other two are optional.

Here’s the jsfiddle to play around. :)

Template Literals

If you know other scripting languages like ruby, python then template literals aren't new for you.

They are newer syntax to make it easy for adding expressions.

Template literals are used to execute any JS expressions.

Let's see some example

```
let name = "Jon Snow";
let msg = `My name is ${name}`;
console.log(msg);

Output:
My name is Jon Snow
```

Simple right.

You can also have multi-line strings.

```
let msg = `My name
is ${name}`;

console.log(msg);

Output:
My name
is Jon Snow
```

Let's see other example

```
let name = "Srebalaji";
let languages = () => {return "Ruby, Js, Java, Python"}

let msg = `My name is ${name}
My age is ${20+3}
And I code in ${languages()}`

Output:
My name is Srebalaji
My age is 23
And I code in Ruby, Js, Java, Python
```

Hope you got it. Its straight forward.

You can add any expression easily and also multi-line strings.

Imports and Exports

Importing and exporting modules in ES6 is one of the useful features you will see in modern front-end libraries.

I highly recommend to play around this feature in this Plunk. The environment is already set-up in that plunk.

okay.. So how imports and exports work in ES6.

Exports are used in modules to explicitly exporting some variables or functions or classes. (i.e) if you export a variable it can be used in other modules.

Imports are used to import variable, functions, classes from other modules.

If you don't get stay with me. Let's see some example.

```
app.js
export let name = "Jon"
export let age = 23

index.js
import {name, age} from './app'
console.log(name);
console.log(age);

index.html
<script src="./index.js"></script>

Output:
Jon
23
```

In the above example, we have defined two variables **name** and **age** and have exported it.

In another file we have imported the variables and accessed their values.

Simple right.

Let’s go deeper

```
app.js
export default const name = "Jon"

index.js
import name from './app.js'
console.log(name);

Output:
Jon
```

In the above code, you can see we have used a new keyword **default**. Default values are mostly used if you need to export a single value or function or object from a module. And there can be only one default value in a module.

One more thing about default values. Since there will be only one default value in a module you can use any name to reference it during import.

Example

```
import n from './app.js'
console.log(n);

Output:
Jon
```

As you can see we have referenced the default value as **n** here.

Let’s go deeper.

```
app.js
let a = 10;
let b = 2;
let sum = () => a+b;
```

```
export {a,b}
export default sum
```

```
index.js
import * as variables from './app'
import addition from './app' // default value
console.log(variables.a);
console.log(variables.b);
console.log(addition());
```

```
Output:
10
2
12
```

In the above example you can see we have exported two variables and a function. And we have imported all the variables using *.

There are two things to remember while importing.

1. If you are using * to import values then you have to use **alias** (i.e) names that will refer to imported values. In our example we have used **variables** as alias.
2. Using * to import values doesn't import default value. You have to import it separately.

```
import addition, * as variables from './app'
```

If you need to import default value and other values in a single line, you can use the above syntax.

Hope you got it. :)

Destructuring objects and arrays

Destructuring is one of the useful feature in ES6. And it's very simple to use.

Let's some example.

```
let person = {firstName: "Jon", lastName: "Snow", age: 23}
```

```
const {firstName, lastName, age} = person
```

```
console.log(firstName);
console.log(lastName);
console.log(age);

Output:
Jon
Snow
23
```

In the above code, you can see we have an object **person** with multiple keys.

And we have created three variables firstName, lastName, age (which is same as object keys.) from the object itself.

To put it in simple words, we have created three variables from extracting keys from the object.

Let’s see some other example

```
let person = {firstName: "Jon", lastName: "Snow", age: 23}

const {firstName} = person
console.log(firstName);

Output:
Jon
```

In the above example, you can see we have extracted only the required values from the object.

```
let person = {firstName: "Jon", lastName: "Snow", age: 23}

const {firstName: name, age} = person
console.log(name);
console.log(age);

Output:
Jon
23
```

In the above example, you can see we have defined a new variable **name** and it is assigned with **firstName**.

Hope you got it. It’s simple.

Let’s see how to destructure array.

```
let arr [1, 2, 3, 4]

const [a, b, c, d] = arr;

console.log(a);
console.log(b);
console.log(c);
console.log(d);

Output:
1
2
3
4
```

Hope you got the above code. It’s simple.

We are assigning each element of an array to a variable.

Let’s see another example.

```
let arr = [1,2,3,4,5,6]

let [a,b,,d,e] = arr

console.log(a);
console.log(b);
console.log(d);
console.log(e);

Output:
1
2
4
5
```

In the above code, you can see we have skipped the third element of the array. Other than that everything is same as the previous example.

Let’s see another example.

```
let person = {firstName: "Jon", lastName: "Snow", age: 23}

let displayName = ({firstName, lastName:last}) => {
  console.log(`${firstName} - ${last}`);
}

displayName(person);

Output:
Jon - Snow
```

Hope you got it. Its straight forward.

Extend and Super

If you have experience coding in OOPS, then extend and super aren’t new for you.

Extend is used to create sub-class from the main class. The sub-class inherits all the properties of the main class and can also modify the properties of the main class.

```
class Person{
 constructor(firstName, lastName, age) {
   this.firstName = firstName;
   this.lastName = lastName;
   this.age = age;
 }
 displayName() {
  return `${this.firstName} - ${this.lastName}`;
 }
}

class Employee extends Person {
 constructor(firstName, lastName, age, salary) {
  super(firstName, lastName, age);
  this.salary = salary;
 }
 displaySalary() {
  return `${this.salary}`;
 }
 displayName() {
  return super.displayName();
 }
 displayAge() {
  return this.age;
 }
}

let manager = new Employee("Jon", "Snow", 23, 100);
console.log(manager.displaySalary());
console.log(manager.displayName());
console.log(manager.displayAge());

Output:
100
Jon Snow
23
```

In the above code, you can see we have defined a class **Person** with a constructor and a simple method.

And then we have defined another class **Employee** which is a sub-class inherited from **Person**. We have used **extends** to achieve this. Hope you are clear till this.

And then we have used **super** keyword to call the constructor of the parent class. And we also have called the method declared in the parent class using **super**.

Note: you can use **this** in the sub-class only after calling **super**. If you use **this** before calling **super** in the sub-class you will get **RefrenceError**.

So, we have achieved three things in the above code

1. We used **extends** to create a sub-class from the parent class.
2. We used **super** to call the constructor of the parent class.
3. We used **super** to call the method defined in the parent class.

Hope you got it :)

Here's the jsfiddle to play around.

If you enjoyed this article try to give some claps and share it :) :)

. . .

Here's the related articles I wrote about ES6

https://hackernoon.com/es6-for-beginners-f98120b57414