ECMAScript 6
(ES6 or ES 2015)

Tính năng mới của ES6

1. Let & Const
2. Arrow Function
3. Rest Params & Spread Operator
4. Default Params
5. Destructuring
6. Template Strings
7. Object literals
8. For of
9. OOP
10. Import & Export

1. Let

- Dùng để khai báo biến (thay thế và khắc phục 1 số nhược điểm của var)
- Có thể gán giá trị nhiều lần (re-assign)

1. Const

- Dùng để khai báo 1 hằng
- Chỉ có thể gán giá trị 1 lần (non re-assign)

1. Phân biệt: var, let, const

- **Declare**

```
// 1. Declare
var a = 1;
let b = 2;
const c = 3;
const e;
console.log(a,b,c);
// Kết quả:
// a = 1
// b = 2
// c = 3
// e = Uncaught SyntaxError: Missing initializer in const declaration
```

- Khi khai báo const ko gán giá trị thì sẽ báo lỗi

1. Phân biệt: var, let, const

- **Re-declare**

```
// 2. Redeclare
var a = 1;
var a = 3;

let b = 2;
let b = 3;

const c = 3;
const c = 4;

console.log(a,b,c)
// Kết quả
// a = 3
// b = Uncaught SyntaxError: Identifier 'b' has already been declared
// c = Uncaught SyntaxError: Identifier 'c' has already been declared
```

- Let không được khai báo lại, nhưng vẫn có thể thay đổi giá trị
- Const không được khai báo lại or thay đổi giá trị

1. Phân biệt: var, let, const

- **Hoisting**: khi thực hiện những khai báo biến là var thì sẽ được đưa lên trên cùng

```
// 3.Hoisting
console.log(a);
var a = 1

console.log(b);
let b  = 1

// Kết quả
// a = undefined
// b = Uncaught ReferenceError: b is not defined
```

- Let và const khắc phục hoisting của var

1. Phân biệt: var, let, const

Function scope:

- phạm vi khai biến bên trong một hàm. Biến bên trong scope sẽ không lấy được từ bên ngoài

```
// 4. Function scope
(function(){
    var a = 0;
})
console.log(a)
// Kết quả
// a = Uncaught ReferenceError: a is not defined
```

1. Phân biệt: var, let, const

- **Block scope**: phạm vi khai báo biến bên trong một { … }. Biến bên trong scope sẽ không lấy được từ bên ngoài.

```
// Block scope
if(true){
    var x = 10;
}
console.log(x)
// Kết quả
// x = 10
```

```
if(true){
    let y = 10;
}
console.log(y)
// Kết quả
// y = Uncaught ReferenceError: y is not defined
```

- Như chúng ta thấy, var không tuân thủ theo block scope, nên let được thay thế để khắc phục.

1. Phân biệt: var, let, const

- **Block scope**: VD cụ thể thực tế hơn muốn kiểm tra nút nào được click

```
<button>Nút A</button>
<button>Nút B</button>
<button>Nút C</button>
```

```
var myBtn = document.getElementsByTagName("button");
for(var i = 0; i< myBtn.length; i++){
    myBtn[i].onclick = function() {
        alert("Nút số " + (i+1))
    }
}
```

```
var myBtn = document.getElementsByTagName("button");
for(let i = 0; i< myBtn.length; i++){
    myBtn[i].onclick = function() {
        alert("Nút số " + (i+1))
    }
}
```

- K

Khi khai báo var trong vòng lặp for, chúng ta click 3 nút đều trả về kết quả = 3

Khi khai báo let trong vòng lặp for, chúng ta được kết quả như ý muốn

Không được khai báo const sẽ báo lỗi vì khi hết 1 vòng lập giá trị sẽ bị thay đổi

2. Arrow Function

Arrow function là function được viết rút gọn từ khóa function thay bằng dấu mũi tên. Trong trường hợp hàm chỉ có 1 lệnh return thì ta có thể lượt bỏ chữ return và {}

- **Non-parameter**

```
// es5
function test(){
    return "ok"
}
// es6
let test = () => {
    return "ok"
}
// or
let test = () => "ok"
```

- **Parameters**

```
// es5
function test(a){
    return a
}
// es6
let test = (a) => {
    return a
}
// or
let test = a => a
```

2. Arrow Function

Trước ES6, mỗi khai báo hàm đều có một giá trị **this** tách biệt. Với hàm mũi tên trong ES6, giá trị của this chính là this trong tầm vực gần nhất với nó (lexical this). Do đó chúng ta không cần phải khai báo biến tạm hay dùng .bind nữa.
Hàm mũi tên cũng rất hữu ích khi thao tác trên mảng và tiến hành chuyển đổi dữ liệu, giúp mã nguồn dễ đọc và rõ ràng hơn.

```
1   //Xây dựng đối tượng dựa trên function es5
2   let hocVien = {
3       hoTen:'Nguyen Văn A',
4       lop:'FrontEnd XXX',
5       layThongTinHocVien: function(){
6           function hienThiThongTin(){
7               console.log(`Họ tên: ${this.hoTen} - Lớp: ${this.lop}`);
8           }
9           hienThiThongTin();
10      }
11  }
12  //dùng đối tượng hocVien gọi phương thức layThongTinHocVien()
13  hocVien.layThongTinHocVien();
14  //=> this.hoTen = '' this.Lop = ''
```

```
19  //Xây dựng đối tượng dựa trên function es5
20  let hocVien1 = {
21      hoTen:'Nguyen Văn A',
22      lop:'FrontEnd XXX',
23      layThongTinHocVien: function () {
24          hienThiThongTin = () => { //Sử dụng function ES6
25              console.log(`Họ tên: ${this.hoTen} - Lớp: ${this.lop}`);
26          }
27          hienThiThongTin();
28      }
29  }
30  //dùng đối tượng hocVien gọi phương thức layThongTinHocVien()
31  hocVien1.layThongTinHocVien();
32  //=> this.hoTen = 'Nguyen Văn A' this.Lop = 'FrontEnd XXX'
```

- hocVien.layThongTinHocVien(): là global context vì nó không dc gọi bởi 1 object. Cho nên biến this sẽ là của object window
- hocvien1.layThongTinHocVien(): context của layThongTinHocVien() là hocVien
- **Khi sử dụng hàm es6, layThongTinHocVien() ở hocVien1 sẽ ko có context, và biến this sẽ là biến this của object hocVien**

Nên sử dụng arrow function khi nào?

- Sử dụng function trong global scope trong Object.prototype properties
- Sử dụng class cho object constructors.
- Sử dụng => ở những chố còn lại

3. Rest Params

- **Rest:** Các tham số truyền vào sẽ hợp thành 1 mảng, dùng khi không biết có bao nhiêu tham số đầu vào một hàm

```
//Tính tổng các tham số dc truyền vào
function sum(...numbers){
    return numbers.reduce((a,b) => {
        return a+b
    },0)
}

let kq = sum(1,2,3,4,5,6,7)
console.log(kq);
```

4. Spread Operator

- **Spread:** toán tử 3 chấm, dùng để thêm phần tử vào mảng hoặc thêm thuộc tính vào object, ngược với rest nó nhận vào mảng và trả ra từng phần tử

```
// Spread
const a = [1,2,3]
const b = [0, ...a,4]
console.log(b);
// kết quả
// b = [1,2,3,4]
```

```
let a = {
    name:"truc",
    age: 20
}

let b = {...a,gender:'Male'}

console.log(b);
// kết quả
// b = {name:"truc",age:20,gender:"Male"}
```

5. Default Params

- Cho phép set giá trị mặc định tham số **(parameters)** của hàm nếu như không có đối số **(arguments)** truyền vào

```
let getInfo = (ten = "Truc", tuoi = 18) => {
    console.log(ten,tuoi)
}
getInfo();
// Kết quả
// ten = "Truc"
// tuoi = 18
```

6. Destructuring

- **Array Destructuring**

```
// Array destructuring
let [a,b] = [1,2]
console.log(a,b);
// Kết quả
// a = 1
// b = 2
```

- **Object Destructuring**

```
// Object destructuring
let people = {
    name: "Truc",
    age: "18"
}
let {name:name,age:age} = people
// or
let {name,age} = people
console.log(name,age)
// Kết quả
// name = "Truc"
// age = 18
```

7. Template Strings

- Tạo một string vừa tĩnh vừa động
- String nằm trong dấu **`…`**
- Truyền giá trị động trong **${ … }**

```
let getName = (name) => {
    console.log(`Hello ${name}`)
}
getName("Truc")
// Kết quả
// Hello Truc
```

8. Object literals (object chân phương)

- ES6 nâng cấp object chân phương, cho phép bạn khai báo tắt thuộc tính của object với biến cùng tên, và khai báo phương thức cho object.

index.jsx ▸ ...

```
1   //Khai báo theo ES5
2   let hoTen = 'Nguyễn Văn A';
3   let lop = 'FrontEndXX';
4   let hocVien = {
5       hoTen:hoTen,
6       lop:lop,
7       layThongTinHocVien: () => {
8           return 'Họ tên: ' + this.hoTen + " Lớp : "+ this.lop;
9       }
10  }
11  //Khai báo theo ES6
12  let hocVien_es6 = {
13      hoTen,  //Tên biến trùng tên với tên thuộc tính ta có thể khai báo tắt
14      lop,
15      layThongTinHocVien: () => {
16          return 'Họ tên: ' + this.hoTen + " Lớp : "+ this.lop;
17      }
18  }
19
```

8. Object literals (object chân phương)

Ngoài ra từ ES6 bạn cũng có thể khai báo thuộc tính cho object một cách linh động bằng cách sử dụng cú pháp [].

index.jsx ▸ ...

```jsx
//Khai báo thuộc tính động cho object
let propHoTen = 'hoTen';
let propLop = 'lop';
let hocVien = {
    [propHoTen]:'Nguyễn Văn A',
    [propLop]:'FrontEnd XXX',
    layThongTinHocVien: () => {
        return 'Họ tên: ' + this.hoTen + " Lớp : "+ this.lop;
    }
}
```

9. For … in

- **For … in** … duyệt mảng theo index

```
// For in
let listFriend = ["Truc","Tuan","Khai","Hieu","Van"]
for(let friend in listFriend){
    console.log(friend);
}
// Kết quả
// 0 1 2 3 4
```

9. For … of

- **For … on** … duyệt mảng theo từng phần tử

```
// For in
let listFriend = ["Truc","Tuan","Khai","Hieu","Van"]
for(let friend of listFriend){
    console.log(friend);
}
// Kết quả
// Truc
// Tuan
// Khai
// Hieu
// Van
```

10. OOP

- **Class**

```
// es5
function Student(name){
    this.name = name
}
let student = new Student('truc');
console.log(student.name);

// es6
class Student{
    constructor(name){
        this.name = name
    }
}
let student = new Student('Truc')
console.log(student.name);
```

10. OOP

- **Class Inheritance**

```
// es5
function Mother(name){
    this.name = name
}
Mother.prototype.colorEyes = function(){
    console.log('red')
}

function Me(name){
    // Mượn hàm để trả về 1 hàm (kế thừa các thuộc tính của Mother)
    Mother.apply(this,arguments);
}

// Kế thừa các methods của Mother
Me.prototype = new Mother;
Me.prototype.colorSkin = function(){
    console.log('brown')
}
let me = new Me('truc')

console.log(me.name);
me.colorEyes();
me.colorSkin();
// Kết quả
// Truc
// red
// brow
```

```
// es6 - inherit
class Mother{
    constructor(name){
        this.name = name
    }
    colorEyes(){
        console.log('red');
    }
}

class Me extends Mother{
    colorSkin(){
        console.log('brown')
    }
}

const me = new Me('Truc')
console.log(me.name);
me.colorEyes();
me.colorSkin();
// Kết quả
// red
// brow
```

- Ta thấy ES6 giúp code chúng ta dễ dọc, dễ hiểu và tường minh hơn, như những ngôn ngữ thuần OOP

10. OOP

- **Method Overriding**

```
// ES5 - method overriding
function Person(name) {
    this.name = name;
}
Person.prototype.getName = function() {
    return this.name;
}

var reader = new Person('Truc');
reader.getName = function() {
    let baseName = Person.prototype.getName.call(this);
    return "Hello " + baseName
}
console.log(reader.getName());
// Kết quả
// "Hello Truc"
```

```
// ES6 - method Overriding
class Person{
    constructor(name){
        this.name = name
    }
}

Person.prototype.getName = function(){
    return this.name;
}

class Monkey extends Person{
}
Monkey.prototype.getName = function(){
    return "Hello " + this.name
}
let monkey = new Monkey('Khai');
console.log(monkey.getName())
```

10. OOP

- **Super():** đại diện cho Lớp cha để gọi lại constructor hoặc phương thức

```
// ES5 - Super()
function People(name,age){
    this.name = name;
    this.age = age
}
People.prototype.getInfo = function(){
    console.log(`${this.name} ${this.age} tuổi`);
}

function Monkey(name,age,eat){
    // Super thuộc tính
    People.apply(this,arguments);
    this.eat = eat
}

Monkey.prototype = new People;
let getInfo = People.prototype.getInfo;
Monkey.prototype.getInfo = function(){
    // Super phương thức
    getInfo.apply(this,arguments)
    console.log(this.eat);
}
let monkey = new Monkey('Truc',18,'Chuoi');
monkey.getInfo();
```

```
// ES6 - Super()
class People{
    constructor(name,age){
        this.name = name;
        this.age = age;
    }
    getInfo(){
        console.log(`${this.name} ${this.age} tuổi!`)
    }
}

class Monkey extends People{
    constructor(name,age,eat){
        super(name,age)
        this.eat = eat;
    }
    getInfo(){
        super.getInfo();
        console.log(this.eat)
    }
}

let monkey = new Monkey('KhecKhec',18,'chuoi')
monkey.getInfo();
```

11. Import & Export

```
// page: index.js
// export es5
var a = "1"
module.exports = a
// export es6
var b = "2"
export default b;
// or
export var c;

// page: test.js
// import es5
require('./index');
// import es6
import b from './index'
// or
export {c} from './index'
```

- Nếu export default, khi ta import có thể đặt tên biến tùy ý (không có dấu {})
- Nếu export không có từ khóa default, khi import ta phải đặt tên biến giống với tên đã export và có {}
- Có thể export nhiều biến bằng cách: export {...}

***Một số hàm xử lý mảng [array] trong ES6 (Đọc thêm)

- ## filter()

Phương thức filter () trả về **kết quả là một MẢNG** với tất cả các phần tử vượt qua kiểm tra được thực hiện bởi hàm được cung cấp.

```
let mangSanPham = [
    {MaSP:1,TenSP:'Sony Xperia XZ2',Gia: 17500000,HangSX:'SONY'},
    {MaSP:2,TenSP:'Sony Xperia XZ1',Gia: 15500000,HangSX:'SONY'},
    {MaSP:3,TenSP:'Sony Xperia XZPremium',Gia: 18500000,HangSX:'SONY'},
    {MaSP:4,TenSP:'Google Pixel XL',Gia: 27500000,HangSX:'GOOGLE'},
    {MaSP:5,TenSP:'Google Pixel 2',Gia: 17500000,HangSX:'GOOGLE'},
    {MaSP:6,TenSP:'Samsung Galaxy Note 9',Gia: 17500000,HangSX:'SAMSUNG'},
    {MaSP:7,TenSP:'Samsung Galaxy S10 Plus',Gia: 27500000,HangSX:'SAMSUNG'},
    {MaSP:8,TenSP:'Samsung Galaxy S10 5G',Gia: 37500000,HangSX:'SAMSUNG'},
]

//Lấy danh sách các điện thoại thuộc hãng sản xuất là SONY
//Dùng hàm filter với callback truyền vào trả về sanPham nào thỏa điều kiện HangSX === 'SONY'
let mangDienThoaiSony = mangSanPham.filter(sanPham => sanPham.HangSX === 'SONY');

console.log(mangDienThoaiSony)
```

Kết quả

```
▼ (3) [{…}, {…}, {…}]
  ▶ 0: {MaSP: 1, TenSP: "Sony Xperia XZ2", Gia: 17500000, HangSX: "SONY"}
  ▶ 1: {MaSP: 2, TenSP: "Sony Xperia XZ1", Gia: 15500000, HangSX: "SONY"}
  ▶ 2: {MaSP: 3, TenSP: "Sony Xperia XZPremium", Gia: 18500000, HangSX: "SONY"}
    length: 3
  ▶ __proto__: Array(0)
```

***Một số hàm xử lý mảng [array] trong ES6 (Đọc thêm)

- ## find()

Phương thức find () trả về **kết quả là một đối tượng** với phần tử vượt qua kiểm tra được thực hiện bởi hàm được cung cấp.

```
let mangSanPham = [
    {MaSP:1,TenSP:'Sony Xperia XZ2',Gia: 17500000,HangSX:'SONY'},
    {MaSP:2,TenSP:'Sony Xperia XZ1',Gia: 15500000,HangSX:'SONY'},
    {MaSP:3,TenSP:'Sony Xperia XZPremium',Gia: 18500000,HangSX:'SONY'},
    {MaSP:4,TenSP:'Google Pixel XL',Gia: 27500000,HangSX:'GOOGLE'},
    {MaSP:5,TenSP:'Google Pixel 2',Gia: 17500000,HangSX:'GOOGLE'},
    {MaSP:6,TenSP:'Samsung Galaxy Note 9',Gia: 17500000,HangSX:'SAMSUNG'},
    {MaSP:7,TenSP:'Samsung Galaxy S10 Plus',Gia: 27500000,HangSX:'SAMSUNG'},
    {MaSP:8,TenSP:'Samsung Galaxy S10 5G',Gia: 37500000,HangSX:'SAMSUNG'},
]

//Lấy ra đối tượng có MaSP == 2
//Dùng hàm find với callback truyền vào trả về sanPham nào thỏa điều kiện MaSP === 2
let dienThoaiXZ1 = mangSanPham.find(sanPham => sanPham.MaSP === 2);

console.log(dienThoaiXZ1)
```

Kết quả

```
▶ {MaSP: 2, TenSP: "Sony Xperia XZ1", Gia: 15500000, HangSX: "SONY"}
```

Lưu ý: Nếu có hơn 2 object thỏa điều kiện trên nó sẽ trả về object đầu tiên

***Một số hàm xử lý mảng [array] trong ES6 (Đọc thêm)

- ## findIndex()

Phương thức findIndex () trả về **kết quả là chỉ số phần tử** ứng với vị trí phần tử trong mảng. Nếu không có phần tử nào thỏa điều kiện.

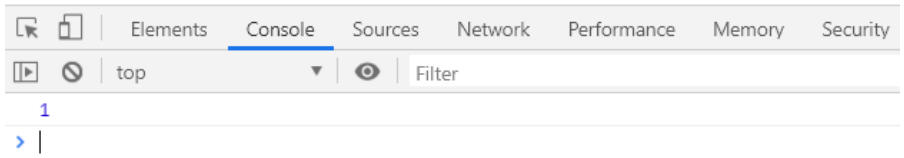
```
let mangSanPham = [
    {MaSP:1,TenSP:'Sony Xperia XZ2',Gia: 17500000,HangSX:'SONY'},
    {MaSP:2,TenSP:'Sony Xperia XZ1',Gia: 15500000,HangSX:'SONY'},
    {MaSP:3,TenSP:'Sony Xperia XZPremium',Gia: 18500000,HangSX:'SONY'},
    {MaSP:4,TenSP:'Google Pixel XL',Gia: 27500000,HangSX:'GOOGLE'},
    {MaSP:5,TenSP:'Google Pixel 2',Gia: 17500000,HangSX:'GOOGLE'},
    {MaSP:6,TenSP:'Samsung Galaxy Note 9',Gia: 17500000,HangSX:'SAMSUNG'},
    {MaSP:7,TenSP:'Samsung Galaxy S10 Plus',Gia: 27500000,HangSX:'SAMSUNG'},
    {MaSP:8,TenSP:'Samsung Galaxy S10 5G',Gia: 37500000,HangSX:'SAMSUNG'},
]

//Lấy ra đối tượng có MaSP == 2
//Dùng hàm findIndex với callback truyền vào trả về index của sanPham nào thỏa điều kiện MaSP === 2
let indexDienThoaiXZ1 = mangSanPham.findIndex(sanPham => sanPham.MaSP === 2);

console.log(indexDienThoaiXZ1); //Nếu không có đối tượng nào thỏa điều kiện sẽ trả về indexDienThoaiXZ1 = -1

//Có thể xóa
mangSanPham.splice(indexDienThoaiXZ1,1);
//Hoặc thao tác với đối tượng đó
mangSanPham[indexDienThoaiXZ1].TenSP = 'Tên sản phẩm cập nhật';
```

Kết quả



***Một số hàm xử lý mảng [array] trong ES6 (Đọc thêm)

- ## foreach()

Phương thức thực thi một hàm 1 lần cho mỗi phần tử. Nôm na mảng có 8 phần thử sẽ thực thi hàm đó 8 lần, hàm nhận tham số đầu vào là từng phần tử của mảng và vị trí foreach((item,index)=>{})

```
let mangSanPham = [
    {MaSP:1,TenSP:'Sony Xperia XZ2',Gia: 17500000,HangSX:'SONY'},
    {MaSP:2,TenSP:'Sony Xperia XZ1',Gia: 15500000,HangSX:'SONY'},
    {MaSP:3,TenSP:'Sony Xperia XZPremium',Gia: 18500000,HangSX:'SONY'},
    {MaSP:4,TenSP:'Google Pixel XL',Gia: 27500000,HangSX:'GOOGLE'},
    {MaSP:5,TenSP:'Google Pixel 2',Gia: 17500000,HangSX:'GOOGLE'},
    {MaSP:6,TenSP:'Samsung Galaxy Note 9',Gia: 17500000,HangSX:'SAMSUNG'},
    {MaSP:7,TenSP:'Samsung Galaxy S10 Plus',Gia: 27500000,HangSX:'SAMSUNG'},
    {MaSP:8,TenSP:'Samsung Galaxy S10 5G',Gia: 37500000,HangSX:'SAMSUNG'},
]

//Dùng hàm foreach để làm công việc console.log(item)

mangSanPham.forEach((sanPham,index) => {
    console.log(index , sanPham);
    //console.log( Object.getOwnPropertyNames(sanPham)); //Lấy ra danh sách thuộc tính của object
    //['MaSP','TenSP','Gia','HangSX']
})
```

Kết quả

```
0 ▶ {MaSP: 1, TenSP: "Sony Xperia XZ2", Gia: 17500000, HangSX: "SONY"}
1 ▶ {MaSP: 2, TenSP: "Sony Xperia XZ1", Gia: 15500000, HangSX: "SONY"}
2 ▶ {MaSP: 3, TenSP: "Sony Xperia XZPremium", Gia: 18500000, HangSX: "SONY"}
3 ▶ {MaSP: 4, TenSP: "Google Pixel XL", Gia: 27500000, HangSX: "GOOGLE"}
4 ▶ {MaSP: 5, TenSP: "Google Pixel 2", Gia: 17500000, HangSX: "GOOGLE"}
5 ▶ {MaSP: 6, TenSP: "Samsung Galaxy Note 9", Gia: 17500000, HangSX: "SAMSUNG"}
6 ▶ {MaSP: 7, TenSP: "Samsung Galaxy S10 Plus", Gia: 27500000, HangSX: "SAMSUNG"}
7 ▶ {MaSP: 8, TenSP: "Samsung Galaxy S10 5G", Gia: 37500000, HangSX: "SAMSUNG"}
```

***Một số hàm xử lý mảng [array] trong ES6 (Đọc thêm)

- ## map():

  Hàm map tương tự hàm foreach() nhưng khác ở chỗ hàm map có giá trị trả về là 1 mảng mới được tạo ra từ các đối tượng return trong callback function.

```
let mangSanPham = [
    {MaSP:1,TenSP:'Sony Xperia XZ2',Gia: 17500000,HangSX:'SONY'},
    {MaSP:2,TenSP:'Sony Xperia XZ1',Gia: 15500000,HangSX:'SONY'},
    {MaSP:3,TenSP:'Sony Xperia XZPremium',Gia: 18500000,HangSX:'SONY'},
    {MaSP:4,TenSP:'Google Pixel XL',Gia: 27500000,HangSX:'GOOGLE'},
    {MaSP:5,TenSP:'Google Pixel 2',Gia: 17500000,HangSX:'GOOGLE'},
    {MaSP:6,TenSP:'Samsung Galaxy Note 9',Gia: 17500000,HangSX:'SAMSUNG'},
    {MaSP:7,TenSP:'Samsung Galaxy S10 Plus',Gia: 27500000,HangSX:'SAMSUNG'},
    {MaSP:8,TenSP:'Samsung Galaxy S10 5G',Gia: 37500000,HangSX:'SAMSUNG'},
]

//Dùng hàm map() tương tự foreach nhưng giá trị hàm map() là trả về 1 mảng mới các đối tượng được return trong callback

let mangCopy = mangSanPham.map((sanPham,index) => {
    console.log(index,sanPham);
    return sanPham;
});

console.log(mangCopy);
```

Kết quả

```
0 ▶ {MaSP: 1, TenSP: "Sony Xperia XZ2", Gia: 17500000, HangSX: "SONY"}
1 ▶ {MaSP: 2, TenSP: "Sony Xperia XZ1", Gia: 15500000, HangSX: "SONY"}
2 ▶ {MaSP: 3, TenSP: "Sony Xperia XZPremium", Gia: 18500000, HangSX: "SONY"}
3 ▶ {MaSP: 4, TenSP: "Google Pixel XL", Gia: 27500000, HangSX: "GOOGLE"}
4 ▶ {MaSP: 5, TenSP: "Google Pixel 2", Gia: 17500000, HangSX: "GOOGLE"}
5 ▶ {MaSP: 6, TenSP: "Samsung Galaxy Note 9", Gia: 17500000, HangSX: "SAMSUNG"}
6 ▶ {MaSP: 7, TenSP: "Samsung Galaxy S10 Plus", Gia: 27500000, HangSX: "SAMSUNG"}
7 ▶ {MaSP: 8, TenSP: "Samsung Galaxy S10 5G", Gia: 37500000, HangSX: "SAMSUNG"}
```

***Một số hàm xử lý mảng [array] trong ES6 (Đọc thêm)

- ## reduce():

Hàm reduce thực thi n lần so với n phần tử của mảng nhằm tạo ra 1 giá trị mới (có thể là 1 biến, 1 mảng, một object … tùy theo xử lý return trong hàm).

```
let mangSanPham = [
    {MaSP:1,TenSP:'Sony Xperia XZ2',Gia: 17500000,HangSX:'SONY'},
    {MaSP:2,TenSP:'Sony Xperia XZ1',Gia: 15500000,HangSX:'SONY'},
    {MaSP:3,TenSP:'Sony Xperia XZPremium',Gia: 18500000,HangSX:'SONY'},
    {MaSP:4,TenSP:'Google Pixel XL',Gia: 27500000,HangSX:'GOOGLE'},
    {MaSP:5,TenSP:'Google Pixel 2',Gia: 17500000,HangSX:'GOOGLE'},
    {MaSP:6,TenSP:'Samsung Galaxy Note 9',Gia: 17500000,HangSX:'SAMSUNG'},
    {MaSP:7,TenSP:'Samsung Galaxy S10 Plus',Gia: 27500000,HangSX:'SAMSUNG'},
    {MaSP:8,TenSP:'Samsung Galaxy S10 5G',Gia: 37500000,HangSX:'SAMSUNG'},
]
```

Kết quả trả về là 1 con số tổng tiền

```
//reduce((thamso1, thamso2, thamso3, thamso4) => {},x)
//reduce có 2 tham số 1 là callback
//tham số 1: Giá trị khởi tạo ban đầu của kết quả (có thể là mảng, là object, là 1 con số, là 1 chuỗi)
//tham số 2: phần tử của mảng (ở đây là đối tượng sanPham)
//tham số 3: index (vị trí phần tử của mảng)
//tham số 4 nếu có: là chính mảng đó
//Ví dụ 1: Dùng hàm reduce để tính tổng tiền tất cả sản phẩm
//tham số 2 của reduce (x) là giá trị khởi tạo ban đầu => ở đây là x = 0 vì tổng tiền bắt đầu từ 0
let resultTongTien = mangSanPham.reduce((tongTien,sanPham,index) => {
    tongTien = tongTien + sanPham.Gia;
    return tongTien; //Cứ mỗi lần chạy callback sẽ trả về tổng tiền lần kế tiếp tổng tiền sẽ trở thành tham số đầu tiên (cộng dồn lên
},0);

console.log(resultTongTien); // 179000000
```

***Một số hàm xử lý mảng [array] trong ES6 (Đọc thêm)

- ## reduce():

Hàm reduce thực thi n lần so với n phần tử của mảng nhằm tạo ra 1 giá trị mới (có thể là 1 biến, 1 mảng, một object … tùy theo xử lý return trong hàm).

```
let mangSanPham = [
    {MaSP:1,TenSP:'Sony Xperia XZ2',Gia: 17500000,HangSX:'SONY'},
    {MaSP:2,TenSP:'Sony Xperia XZ1',Gia: 15500000,HangSX:'SONY'},
    {MaSP:3,TenSP:'Sony Xperia XZPremium',Gia: 18500000,HangSX:'SONY'},
    {MaSP:4,TenSP:'Google Pixel XL',Gia: 27500000,HangSX:'GOOGLE'},
    {MaSP:5,TenSP:'Google Pixel 2',Gia: 17500000,HangSX:'GOOGLE'},
    {MaSP:6,TenSP:'Samsung Galaxy Note 9',Gia: 17500000,HangSX:'SAMSUNG'},
    {MaSP:7,TenSP:'Samsung Galaxy S10 Plus',Gia: 27500000,HangSX:'SAMSUNG'},
    {MaSP:8,TenSP:'Samsung Galaxy S10 5G',Gia: 37500000,HangSX:'SAMSUNG'},
]
```

Kết quả trả về là 1 mảng mới

```
//Ví dụ 2: kết quả không phải là 1 con số mà là 1 mảng điện thoại sony
let resultDTSony = mangSanPham.reduce((mangSony,sanPham,index) => {
    if(sanPham.HangSX === 'SONY')
    {
        mangSony.push(sanPham);
    }
    return mangSony;
},[]); //x đóng vai trò là giá trị ban đầu mảng rỗng
console.log(resultDTSony);
/* resultDTSony mang giá trị
[
{MaSP:1,TenSP:'Sony Xperia XZ2',Gia: 17500000,HangSX:'SONY'},
{MaSP:2,TenSP:'Sony Xperia XZ1',Gia: 15500000,HangSX:'SONY'},
{MaSP:3,TenSP:'Sony Xperia XZPremium',Gia: 18500000,HangSX:'SONY'}
] */
```

- ## reduceRight():

Giống hệt reduce duyệt mảng từ phải qua trái

***Một số hàm xử lý mảng [array] trong ES6 (Đọc thêm)

- reverse():

Hàm reverse là hàm trả về 1 mảng đảo ngược mảng ban đầu

```
let mangSoNguyen = [1,2,3,5];

let reverseMang = mangSoNguyen.reverse();
console.log(reverseMang);
//[5,3,2,1]
```

***Một số hàm xử lý mảng [array] trong ES6 (Đọc thêm)

- sort():

Hàm sort dùng để sắp xếp mảng theo thứ tự tăng dần hoặc giảm dần. Có thể ứng dụng để sắp xếp các mảng đối tượng dựa vào giá trị các thuộc tính.

```
let mangSanPham = [
    { MaSP: 1, TenSP: 'Sony Xperia XZ2', Gia: 17500000, HangSX: 'SONY' },
    { MaSP: 2, TenSP: 'Sony Xperia XZ1', Gia: 15500000, HangSX: 'SONY' },
    { MaSP: 3, TenSP: 'Sony Xperia XZPremium', Gia: 18500000, HangSX: 'SONY' },
    { MaSP: 4, TenSP: 'Google Pixel XL', Gia: 27500000, HangSX: 'GOOGLE' },
    { MaSP: 5, TenSP: 'Google Pixel 2', Gia: 17500000, HangSX: 'GOOGLE' },
    { MaSP: 6, TenSP: 'Samsung Galaxy Note 9', Gia: 17500000, HangSX: 'SAMSUNG' },
    { MaSP: 7, TenSP: 'Samsung Galaxy S10 Plus', Gia: 27500000, HangSX: 'SAMSUNG' },
    { MaSP: 8, TenSP: 'Samsung Galaxy S10 5G', Gia: 37500000, HangSX: 'SAMSUNG' },
]

//Sắp xếp theo thuộc tính tên là chuỗi a -> z
let mangSPSortTheoTen = mangSanPham.sort((sp, spTiepTheo) => {
    let tenSP = sp.TenSP.toLowerCase();
    let tenSPTiepTheo = spTiepTheo.TenSP.toLowerCase();
    if (tenSP < tenSPTiepTheo) {
        return -1;
    }
    if (tenSPTiepTheo > tenSP) {
        return 1;
    }
    return 0;
})
console.log(mangSPSortTheoTen);
```

Kết quả

```
(8) [{…}, {…}, {…}, {…}, {…}, {…}, {…}, {…}]
  0: {MaSP: 2, TenSP: "Sony Xperia XZ1", Gia: 15500000, HangSX: "SONY"}
  1: {MaSP: 5, TenSP: "Google Pixel 2", Gia: 17500000, HangSX: "GOOGLE"}
  2: {MaSP: 6, TenSP: "Samsung Galaxy Note 9", Gia: 17500000, HangSX: "SAMSUNG"}
  3: {MaSP: 1, TenSP: "Sony Xperia XZ2", Gia: 17500000, HangSX: "SONY"}
  4: {MaSP: 3, TenSP: "Sony Xperia XZPremium", Gia: 18500000, HangSX: "SONY"}
  5: {MaSP: 4, TenSP: "Google Pixel XL", Gia: 27500000, HangSX: "GOOGLE"}
  6: {MaSP: 7, TenSP: "Samsung Galaxy S10 Plus", Gia: 27500000, HangSX: "SAMSUNG"}
  7: {MaSP: 8, TenSP: "Samsung Galaxy S10 5G", Gia: 37500000, HangSX: "SAMSUNG"}
  length: 8
  __proto__: Array(0)
```

***Một số hàm xử lý mảng [array] trong ES6 (Đọc thêm)

- sort():

Sort theo giá

```
let mangSanPham = [
    { MaSP: 1, TenSP: 'Sony Xperia XZ2', Gia: 17500000, HangSX: 'SONY' },
    { MaSP: 2, TenSP: 'Sony Xperia XZ1', Gia: 15500000, HangSX: 'SONY' },
    { MaSP: 3, TenSP: 'Sony Xperia XZPremium', Gia: 18500000, HangSX: 'SONY' },
    { MaSP: 4, TenSP: 'Google Pixel XL', Gia: 27500000, HangSX: 'GOOGLE' },
    { MaSP: 5, TenSP: 'Google Pixel 2', Gia: 17500000, HangSX: 'GOOGLE' },
    { MaSP: 6, TenSP: 'Samsung Galaxy Note 9', Gia: 17500000, HangSX: 'SAMSUNG' },
    { MaSP: 7, TenSP: 'Samsung Galaxy S10 Plus', Gia: 27500000, HangSX: 'SAMSUNG' },
    { MaSP: 8, TenSP: 'Samsung Galaxy S10 5G', Gia: 37500000, HangSX: 'SAMSUNG' },
]


//Sắp xếp theo giá trị là Giá tăng dần
let mangSPSortTheoGia = mangSanPham.sort((sp, spTiepTheo) => {
    return sp.Gia - spTiepTheo.Gia;
})
console.log(mangSPSortTheoGia);
```

Kết quả