REACT NATIVE

CYBERSOFT.EDU.VN

Hướng dẫn thực hiện: Trương Tấn Khải - www.cybersoft.edu.vn

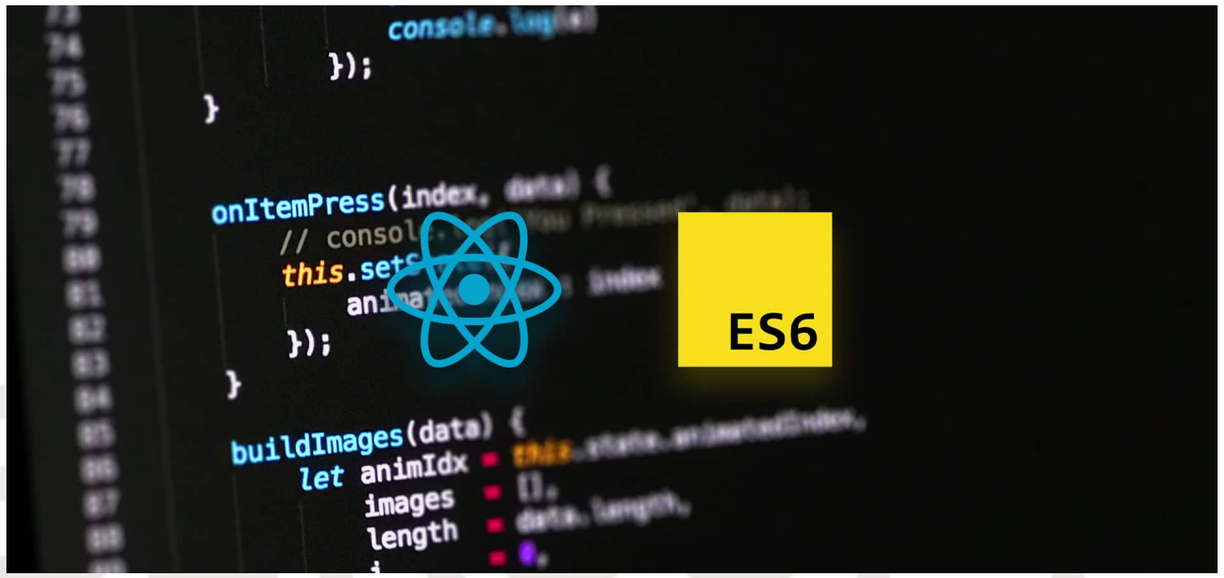
ES 6,7 - ECMASCRIPT (JAVASCRIPT nâng cao)

Hướng dẫn thực hiện: Trương Tấn Khải - www.cybersoft.edu.vn

ES6, 7 ? TẠI SAO PHẢI HỌC ES6, 7?

✦ES 6, 7 là gì ?

ES6, ES7 là chữ viết tắt của ECMAScript 6, đây được coi là một tập hợp các kỹ thuật nâng cao của **Javascript** và là phiên bản mới nhất của chuẩn ECMAScript.



✦Tại sao phải học es6 ?

React Native sử dụng ES6/7 để các developers viết ứng dụng trên nó. Tuy nhiên Facebook chỉ lấy 1 phần (subset) của ES6/7 và sử dụng Babel để làm transpiler (chuyển code và biên dịch sang theo từng hệ điều hành). Xem thêm: https://facebook.github.io/react-native/docs/javascript-environment.html

Tổng quan về es6

1. Cách khai báo biến trong ES6
2. Arrow Function
3. Default Parameter Values
4. Rest Parameter
5. Spread Operator
6. Destructuring
7. Template String
9. String template
10. For in for of
12. OOP
13. Tìm hiểu Proxies
14. ES6 import và export
15. Một số hàm xử lý mảng (Array) trong ES6
16. *Một số kiến thức nâng cao tìm hiểu thêm trong quá trình làm dự án. (Proxies, Promise, Sets, WeakSets, Map, Weak Map, Async, Await, Iterators,generators …)*

1. Cách khai báo biến trong ES6

1. Phân biệt var, let và const

Var

- Dùng để khai báo biến
- Có thể khai báo lại
- Có thể gán giá trị nhiều lần (re-assign)

```
var firstName = 'Cybersoft’;
firstName = 'Cybersoft1'; //ok
var firstName = 'Cybersoft2'; //ok
```

1. Cách khai báo biến trong ES6

Let

- Dùng để khai báo biến giống var
- Có thể gán giá trị nhiều lần (re-assign)
- Let không được khai báo lại

```
let name = 'Cybersoft';
name = 'Cybersoft1'; //ok
let name = 'Cybersoft1'; //
Identifier 'name' has already been decla
red
```

1. Cách khai báo biến trong ES6

Const

- Dùng để khai báo 1 hằng
- Chỉ có thể gán giá trị 1 lần (non re-assign)
- Const không được khai báo lại
- Khi khai báo const không gán giá trị sẽ báo lỗi

```
const PI = 3.1414;
PI = 3.14; // Assignment to consta
nt variable.
const PI = 3.14;// Identifier 'PI'

has already been declared
const numPI; // Missing initialize
r
in const declaration
```

1. Cách khai báo biến trong ES6

Hoisting

Hoisting là hành động mặc định của Javascript, nó sẽ chuyển phần khai báo **var** lên phía trên cùng.

Ví dụ 1:

```
console.log(x);
x = 5; // gán 5 cho
x
```

Kết quả: x is not defined

Kết quả không có gì lạ vì x chưa được khai báo

Ví dụ 2:

```
var x; //khai báo x
console.log(x);
x = 5; //gán 5 cho x
```

Kết quả: Undefined

x chưa được gán giá trị khi chạy console

1. Cách khai báo biến trong ES6

Ví dụ 3:

```
console.log(x);
var x = 5; // khai báo
x và gán 5 cho x
```

Lỗi này là x is not defined hay là Undefined ?

Thật bất ngờ, kết quả là Undefined

1. Cách khai báo biến trong ES6

Giải thích:

- Javascript sẽ tự tách `var x = 5` thành 2 phần
  - Khai báo: **var x**
  - Gán giá trị **x = 5**
- Phần khai báo sẽ được di chuyển lên trên cùng theo Hoisting.

```
// Tách làm 2 phần
console.log(x);
var x;
x = 5;
```

```
// Di chuyển khai báo l
ên cùng
var x;
console.log(x);
x = 5;
```

1. Cách khai báo biến trong ES6

➢Let và const khắc phục hoisting của var

```
console.log(a);
let a = 5;
```

Kết quả:

Cannot access 'a' before initialization

Vì a chưa được khai báo

```
let a;
console.log(a)
;
a = 5;
```

Kết quả: Undefined

a chưa được gán giá trị khi chạy

console

1. Cách khai báo biến trong ES6

1. Function scope

- Một biến được khai báo trong một function sẽ được dùng ở bất cứ đâu trong function đó.
- Bên ngoài sẽ không gọi được biến đó.

```
function weight() {
    var w = 100;
    console.log(w + 'kg');
    if (w > 60) {
        console.log('Bạn nặng: '
+w+ 'kg. Người bạn hơi gầy!');
    }
}
weight();

console.log(w); //
w is not defined
```

1. Cách khai báo biến trong ES6

1. Block scope

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

```
function blockScope(){
    let b = true;
    //Block Scope
    if(b){
        let x =10;
    }
    console.log(x);//x is not defined
}
blockScope();
```

1. Cách khai báo biến trong ES6

➢ var là function scope, còn const và let là block scope

- Từ ví dụ của function scope hãy thử đảo thứ tự của var

```
function weight() {
    if (100 > 60) {
        //Block Scope
        var w = 100;
        console.log(w + 'kg');
    }
    //Out of Block Scope
    console.log('Bạn nặng: '+w+ '
kg. Người bạn hơi gầy!');
}
weight();
```

```
100kg
Bạn nặng: 100kg. Người bạn hơi gầy!
```

weight() vẫn hoạt động, tại sao vậy?

1. Cách khai báo biến trong ES6

➢ Giải thích:

- var là function scope nên mặc dù biến được khai báo trong mệnh đề if thì chỉ cần biến vẫn nằm trong function thì vẫn được sử dụng bình thường
- var không tuân thủ theo block scope, nên let được thay thế để khắc phục.

```
function weight() {
    if (100 > 60) {
        //Block Scope
        let w = 100;
        console.log(w + 'kg');
    }
    //Out of Block Scope
    console.log('Bạn nặng: '+w+ 'kg
. Người bạn hơi gầy!');//
w is not defined
}
weight();
```

2. Arrow Function

- Trong ES6, bên cạnh cách khai báo hàm bằng từ khóa "function" thì chúng ta có thể lượt bỏ từ khóa "function" và thay bằng dấu mũi tên " =>"
- Trong trường hợp hàm chỉ có 1 lệnh return thì ta có thể lượt bỏ chữ return và "{} ".
- Nếu chỉ có 1 biến thì có thể lượt bỏ "() "
- Tương tự như ES5, function của ES6 cũng có 2 loại: Non-parameter và Parameter

1. Non-parameter

```
//ES5
function hello() {
    return "Hello A Sử";
}
console.log(hello());

//ES6
let hello = () =>{
    return "Hello A Sử";
}
console.log(hello());

// Lượt bỏ return and {}
let hello = () =>"Hello A Sử";
console.log(hello());
```

2. Parameter

```
//ES5
function hello(name) {
    return "Hello "+name;
}
console.log(hello('A Sử'));

//ES6
let hello = (name) =>{
    return "Hello " +name;
}
console.log(hello('A Sử'));

//or
let hello = name =>"Hello "+name;
console.log(hello('A Sử'));
```

2. Arrow Function

Lỗi cú pháp

➢ Arrow phải nằm cùng hàng với tên hàm

```
let hello = (name) // Sai
=> {
    return "Hello " +name;
}

let hello = (name) => // Đúng
{
    return "Hello " +name;
}
```

```
let hello = (name) => { // Đúng
    return "Hello " +name;
}

let hello = (name) // Sai
=> "Hello " +name;

let hello = (name) => // Đúng
"Hello " +name;
```

➢ Nếu muốn xuống hàng mà không gây ra lỗi thì dùng cách bên dưới

```
let hello = ( // Đúng
    name
) => {
    return "Hello " +name;
}
```

2. Arrow Function

➢ Khi arrow function bên trong một hàm hoặc sử dụng dạng một biến thì phải dùng cặp đóng để bao quanh lại hoặc gán function cho một biến

```
console.log(typeof () => "Hello A Sử");// Cú pháp sai
console.log(typeof (() =>"Hello A Sử"));// Cú pháp đúng

let hello = () => "Hello A Sử";
console.log(typeof hello); // Cú pháp đúng
```

Giải pháp cho ví dụ 2 bằng ES5:

chúng ta khai báo một biến tạm _bind

để chứa giá trị của this

```
//Xây dựng đối tượng dựa trên function ES5
let hocVien = {
    hoTen: "Hoàng Thùy Mị",
    lop: "Ngu Van 12",
    layThongTinHocVien: function() {
      console.log(this);
      let _bind = this;
      function hienThiThongTin() {
        console.log(`Ho ten: ${_bind.hoTen} - lop: ${_bind.lop}`);
      }
      hienThiThongTin();
    }
  };
  hocVien.layThongTinHocVien();
```

2. Arrow Function

Giải pháp cho ví dụ 2 bằng ES6:

- Chúng ta không cần phải khai báo biến tạm hay .bind nữa
- Arrow function không tạo ra this context, và biến this sẽ là biến this của object hocVien1

```
    //
Xây dựng đối tượng dựa trên function ES6
    let hocVien1 = {
     hoTen: "Hoàng Thùy Mị",
     lop: "Ngu Van 12",
     layThongTinHocVien: function() {
       let hienThiThongTin = () => {
         console.log(`Ho ten: ${this.hoTen} -
 lop: ${this.lop}`);
       };
       hienThiThongTin();
     }
};
hocVien1.layThongTinHocVien();
```

3. Default Parameter Values

Các phiên bản trước của JavaScript, Default Value không được hỗ trợ nên chúng ta phải kiểm tra parameter có giá trị hay không.

```
function getUserInfo (name , age ){
  name = name || 'default text';
  age = age || '0';
  if(age > 0 && age < 30){
    console.log(name + " còn trẻ.");
  }else{
    console.log(name,age);
  }
}
getUserInfo();
```

ES6 Cho phép set giá trị mặc định tham số **(parameters)** của hàm nếu như không có đối số **(arguments)** truyền vào

```
let getUserInfo = (name ="Mị", age = 18 )=>{
  if(age < 30){
    console.log(name + " còn trẻ.");
  }
}
getUserInfo();
```

4. Rest Parameter

- Dùng để khai báo một hàm với số lượng tham số không xác định.
- Các tham số truyền vào sẽ hợp thành 1 mảng
- Để khai báo chúng ta đặt 3 dấu chấm trước biến đại diện.

```
function sum(...numbers) {
  let kq = 0;
  for (let i = 0; i < numbers.length; i++) {
    kq += numbers[i];
  }
  console.log(kq);
}
sum(1, 2, 3, 4, 5, 6, 7, 8, 9, 10);
```

- Trong javascript không có khái niệm overload function (hàm chồng) Vì vậy để sử được khái niệm này, javascript sinh ra khái niệm restParameter để thay thế. Xem ví dụ bên dưới.

```
var hienThiThongTin = (...thongTinHocVien)=>{
    switch(thongTinHocVien.length){
      case 2:{
          console.log('Tên HV: ',thongTinHocVien[0])
          console.log('Lớp:',thongTinHocVien[0])
        };break;
        case 3:{
            console.log('Tên HV',thongTinHocVien[0],'Lớp',thongTinHocVien[1],'Trung tâm',thongTinHocVien[2]);
        };break;
        default : {
            console.log('Tên HV', thongTinHocVien[0]);
        }
    }
}
hienThiThongTin('Nguyễn văn a');
hienThiThongTin('Nguyễn văn a','FE31');
hienThiThongTin('Nguyễn văn a','FE31','Cybersoft');
```

5. Spread Operator

- Spread operator có cú pháp giống Rest Parameter
- Khác với rest nó nhận vào mảng và trả ra từng phần tử
- Dùng để thêm phần tử vào mảng hoặc thêm thuộc tính vào object

```
// Mảng
let mangC = [1, 2, 3, 4];
let mangD = [...mangC];

mangD.push(5);
mangD.push(6);
mangD.push(7);

console.log(mangD);
//kết quả:(7) [1, 2, 3, 4, 5, 6, 7]
console.log(mangC);
//kết quả:(4) [1, 2, 3, 4]
```

```
//Object
let obj = {
  name: "Nguyen",
  age: "18"
};

let objNew = { ...obj, gender: "Female"
};
console.log(objNew);
//kết quả:
{name: "Nguyen", age: "18", gender: "Fem
ale"}
```

5. Spread Operator

- Thay thế function split chuyển 1 chuỗi string thành 1 Array .

```
let str = "foo"
let chars = [ ...str ]
console.log(chars);
//kết quả: (3) ["f", "o", "o"]
```

6. Destructuring

- Là một kỹ thuật dùng để lấy một hay nhiều giá trị của phần tử của mảng hoặc thuộc tính của object, sau đó gán các giá trị này cho các biến khai báo trước.
- Cú pháp ngắn gọn hơn, tránh lặp lại code.

1. Object Destructuring

- Dùng để lấy ra các thuộc tính của object và gán giá trị của từng thuộc tính cho các biến với tên cho trước.
- Các biến được khai báo tương ứng với property của object
- Ta dùng dấu : để gán tên khác cho biến khi không muốn dùng tên trùng với property của object

```
let pet = {
  name: 'Gâu Đần',
  age: 3,
  breed:'Golden Retriever',
  size:{
    weight:'30kg',
    height:'56cm'
  }
}
// ES5
// var name = pet.name;
// var age = pet.age;
// var size = pet.size;

//ES6 - destructuring
let {name,age} = pet;
let {weight, height} = pet.size;
console.log(name, age); //Gâu Đần, 3
console.log(weight, height); //30kg, 56cm

let {weight: w, height: h} = pet.size;
console.log(w, h); //30kg, 56cm
```

6. Destructuring

1. Array Destructuring

- Cách sử dụng với array tương tự như với object tuy nhiên thay vì dùng {} thì ta dùng []
- Các biến được khai báo tương ứng với vị trí các phần tử của mảng

```
let topProgrammingLang = ['JavaScript', 'Python',
'Java'];
let [first, second, third] = topProgrammingLang;

console.log(first); //JavaScript
console.log(second); //Python
console.log(third); //Java
```

7. Template String

- Template String hay còn gọi Template Literals là cú pháp khai báo String trong ES6
- Cho phép chúng ta sử dụng multiline String , biến, biểu thức, hàm bên trong string mà không phải thông qua phép cộng string.
- String sẽ nằm trong dấu `…`
- Để truyền biến trong string dùng ${...}

```
let pet = "Cá";
let action = "bơi";
// Cách cũ
console.log("Mình là " + pet + ", việc của mình là " + action + " .");
// Dùng template string
console.log(`Mình là ${pet}, việc của mình là ${action}.`);

let a =2;
let b = 5;
console.log(`Sum ${a + b}`);
```

7. Template String

Multiline String in ES6

- Ở ES5, cú pháp để xuống dòng một đoạn text sẽ dài và phức tạp.
- ES6 có cú pháp ngắn gọn và không cần nhiều dấu cộng và dấu nháy đơn

```
// Cách cũ
let content = 'Thanh xuân như một tách trà,
\n'
+ 'Biết làm React mới là thanh xuân\n'

console.log(content);

// Dùng template string
let content = `Thanh xuân như một tách trà,
Biết làm React mới là thanh xuân.`
console.log(content);
```

8. Object Literal

Object thường được tạo bằng cú pháp {}, bên trong là các property và method.

```
var name = "Mị";
var myObj = {
    name: name,
    sayHi: function () {
        console.log("Hi, my name is " + this.name
)
    }
};
```

- Trong ES6, cú pháp tạo object được đơn giản hóa hơn.
- Cho phép khai báo tắt thuộc tính của object với biến cùng tên,

```
let name = "Mị";
let myObj = {
    name,
    sayHi() {
        console.log("Hi, my name is " + this.name)
    }
};

myObj.sayHi()
//Hi, my name is Mị
```

- Khai báo tắt phương thức bằng cách lượt bỏ function

8. Object Literal

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

```
let name = 'ten';
let myObj = {
    [name]:"A Phủ",
    sayHi() {
        console.log("Hi, my name is " + this.ten
    }
};

myObj.sayHi()
//Hi, my name is A Phủ
```

9. For in for of

- **For ... in**... là cách duyệt mảng đã được sử dụng trong ES5
- Cơ chế duyệt mảng theo index

```
let currencies = ['VND', 'USD', 'SGN', 'AUD
'];

for (let index in currencies) {
    console.log(index, currencies[index]);
}
// 0 VND
// 1 USD
// 2 SGN
// 3 AUD
```

- **For ... in**... duyệt object trên các thuộc tính (dùng để duyệt các mảng thể hiện ở dạng object

```
//json Object thể hiện dạng Array
var menuDrinks = {
    "327R9YAkI46nWx64eH4J":{
        "id":1,
        "name":"orange juice",
        "price":30
    },
    "CafQ90Rb8l618Bt3VoKW": {
        "id":1,
        "name":"lemon juice",
        "price":40
    },
    "FJg9fXIPt1vMtEVkSBHL":{
        "id":1,
        "name":"watermelon juice",
        "price":50
    }
}
```

```
//Duyệt object
for(var id in  menuDrinks){
    console.log("id",id);
    console.log("item", menuDrinks[id]);
}
```

```
id 327R9YAkI46nWx64eH4J
item ▶ {id: 1, name: "orange juice", price: 30}
id CafQ90Rb8l618Bt3VoKW
item ▶ {id: 1, name: "lemon juice", price: 40}
id FJg9fXIPt1vMtEVkSBHL
item ▶ {id: 1, name: "watermelon juice", price: 50}
```

9. For in for of

- **For … of** … là cách duyệt mảng mới của ES6
- Cơ chế duyệt mảng theo từng phần tử (thường dùng để duyệt mảng dạng đối tượng)

```
var menuDrinks = [
    [
        {
            "id": 1,
            "name": "orange juice",
            "price": 30
        },
        {
            "id": 1,
            "name": "lemon juice",
            "price": 40
        },
        {
            "id": 1,
            "name": "watermelon juice",
            "price": 50
        }
    ]
]

for(var item of menuDrinks){
    console.log('item',item)
}
```

```
let currencies = ['VND', 'USD', 'SGN', 'AUD'
];

for (let value of currencies) {
  console.log(value);
}
// VND
// USD
// SGN
// AUD
```

```
item ▼(3) [{…}, {…}, {…}] i
       ▶0: {id: 1, name: "orange juice", price: 30}
       ▶1: {id: 1, name: "lemon juice", price: 40}
       ▶2: {id: 1, name: "watermelon juice", price: 50}
         length: 3
       ▶__proto__: Array(0)
```

10. OOP

- Javascript ES5 không hỗ trợ class như các ngôn ngữ lập trình hướng đối tượng khác. Thay vào đó, Javascript mô phỏng các class thông qua các function và prototype.

```
function Student(name){
  this.name =name;
}

let student =new Student("Mị");
console.log(student.name);
```

- Ở ES6 bạn đã có thể lập trình hướng đối tượng với class trên Javascript.

```
class Student{
  constructor(name){
    this.name = name;
  }
}

let student = new Student('Mị');
console.log(student.name);
```

Class Inheritance

Ở ES5 cú pháp kế thừa sẽ phức tạp và gây khó hiểu.

```
//ES5
function Mother(name){
  this.name = name;
}
Mother.prototype.colorEyes = function(){
  console.log('Black');
}

function Me(name){
  //Kế thừa thuộc tính của Mother
  Mother.apply(this,arguments);
}
//Kế thừa các phương thức của Mother
Me.prototype = new Mother;
Me.prototype.colorSkin = function(){
  console.log('White');
}
let me = new Me('Mị');

console.log(me.name); //Mị
me.colorEyes(); //Black
me.colorSkin(); //White
```

Class Inheritance

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

```
//ES6
class Mother{
  constructor(name){
    this.name = name;
  }
  colorEyes(){
    console.log('Black');
  }
}
class Me extends Mother{
  colorSkin(){
    console.log('White');
  }
}

let me = new Me('Mị');
console.log(me.name);//Mị
me.colorEyes();//Black
me.colorSkin();//White
```

10. OOP

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

var reader = new Person('Khai');
reader.getName = function(){
  let baseName = Person.prototype.g
etName.call(this);
  return "Hello" + baseName;
}
console.log(reader.getName());
```

```
// ES6 - method overriding
class Person{
  constructor(name){
    this.name=name;
  }
}
Person.prototype.getName = function(){
  return this.name;
}

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

Super()

Super đại diện cho class cha giúp gọi lại constructor hoặc phương phức của class cha, mà phương thức đó đã bị overriding trong lớp con.

```
// ES5 - Gọi lại thuộc tính và phương thức
function People(name,email) {
    this.name = name;
    this.email = email;
}
People.prototype.getInfo = function () {
    console.log(`${this.name} ${this.email}
`);
}

function Reader(name,email,book){
    //Gọi lại thuộc tính của class cha
    People.apply(this,arguments);
    this.book = book;
}
Reader.prototype = new People;
var getInfo = People.prototype.getInfo;
Reader.prototype.getInfo = function(){
    // Gọi lại phương thức của class cha
    getInfo.apply(this,arguments);
    console.log(this.book);
}

var reader = new Reader('Mị','mi@gmail.com','N
gữ Văn 12');
reader.getInfo();
```

```
// ES6 - Gọi lại thuộc tính và phương thức
class People {
    constructor(name, email) {
        this.name = name;
        this.email = email;
    }
    getInfo(){
        console.log(`${this.name} $
{this.email}`)
    }
}

class Reader extends People {
    constructor(name,email,book){
        //Gọi lại thuộc tính của class cha
        super(name,email);
        this.book = book;
    }
    getInfo(){
        // Gọi lại phương thức của class cha
        super.getInfo();
        console.log(this.book);
    }
}

let reader = new Reader('Mị','mi@gmail.com','N
gữ Văn 12');
reader.getInfo();
```

10. ES6 import và export

- Trong JavaScript, câu lệnh import và export hỗ trợ lập trình viên có thể quản lý code theo từng mô-đun.
- Mô-đun là một phương pháp lập trình mà trong đó các đoạn code liên quan được tách ra các phần khác nhau
- Trước ES6, các thư viện JavaScript dùng require() và module.exports() thay cho import và export (ví du: Node.js)

Import và Export bằng ES6

- 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 {…}

```
//page: sum.js
//exports ES6
function sum(a,b) {
    return a + b;
  }

export default sum;
 // export không default
 export {sum};


//page: index.js
// import ES6
import callSum from "./sum.js";
console.log("Cong 2 so:" + callSum(5,3));

// import không default
import {sum} from "./sum.js";
console.log("Cong 2 so:" + sum(5,3));
```

11. Một số hàm xử lý mảng thông dụng

filter()

- Phương thức filter () trả về kết quả là một MẢNG với tất cả các phần tử thỏa mãn điều kiện được khai báo trong hàm.

```
let phoneArray = [
  {id: "1",tenSP: "Note 8",hang: "SamSung",gia:
"20000"},
  {id: "2",tenSP: "Vivo",hang: "Oppo",gia: "1000
0"},
  {id: "3",tenSP: "Black Berry",hang: "Black Ber
ry"},
  {id: "4",tenSP: "Iphone X",hang: "Apple",gia:
"20000"},
  {id: "5",tenSP: "Iphone 11",hang: "Apple",gia:
 "32000"}
];
let iphone = phoneArray.filter(phone => phone.ha
ng === "Apple");
console.log(iphone);
```

```
▼(2) [{…}, {…}] i
  ▶0: {id: "4", tenSP: "Iphone X", hang: "Apple", gia: "20000"}
  ▶1: {id: "5", tenSP: "Iphone 11", hang: "Apple", gia: "32000"}
    length: 2
  ▶__proto__: Array(0)
```

11. Một số hàm xử lý mảng thông dụng

find()

- Phương thức find () trả về kết quả là một đối tượng thỏa mãn điều kiện được khai báo trong hàm.
- Lưu ý: Nếu có hơn 2 object thỏa điều kiện nó sẽ trả về object đầu tiên. Nếu không có phần tử nào thoả điều kiện thì trả về undefine

```
let phoneArray = [
  {id: "1",tenSP: "Note 8",hang: "SamSung",gia: "20000"},
  {id: "2",tenSP: "Vivo",hang: "Oppo",gia: "10000"},
  {id: "3",tenSP: "Black Berry",hang: "Black Berry"},
  {id: "4",tenSP: "Iphone X",hang: "Apple",gia: "20000"},
  {id: "5",tenSP: "Iphone 11",hang: "Apple",gia: "32000"}
];

let iphone = phoneArray.find(phone => phone.hang === "Apple");
console.log(iphone);
```

```
▶ {id: "4", tenSP: "Iphone X", hang: "Apple", gia: "20000"}
```

11. Một số hàm xử lý mảng thông dụng

findIndex()

- Phương thức findIndex () trả về kết quả là chỉ số (index) ứng với vị trí phần tử trong mảng. Nếu không có phần tử nào thỏa điều kiện thì kết quả trả về là -1

```
let phoneArray = [
  {id: "1",tenSP: "Note 8",hang: "SamSung",gia: "20000"},
  {id: "2",tenSP: "Vivo",hang: "Oppo",gia: "10000"},
  {id: "3",tenSP: "Black Berry",hang: "Black Berry"},
  {id: "4",tenSP: "Iphone X",hang: "Apple",gia: "20000"},
  {id: "5",tenSP: "Iphone 11",hang: "Apple",gia: "32000"}
];

let iphone = phoneArray.findIndex(phone => phone.hang ===
" Black Berry ");
console.log(iphone);//2
```

11. Một số hàm xử lý mảng thông dụng

foreach()

- Với mỗi phần tử hàm sẽ thực thi 1 lần. Ví dụ mảng có 8 phần thử thì sẽ thực thi hàm đó 8 lần, hàm nhận tham số đầu vào là từng phần tử của mảng và vị trí của phần tử đó trong mảng.

```
let phoneArray = [
  {id: "1",tenSP: "Note 8",hang: "SamSung",gia: "20
000"},
  {id: "2",tenSP: "Vivo",hang: "Oppo",gia: "10000"}
,
  {id: "3",tenSP: "Black Berry",hang: "Black Berry"
},
  {id: "4",tenSP: "Iphone X",hang: "Apple",gia: "20
000"},
  {id: "5",tenSP: "Iphone 11",hang: "Apple",gia: "3
2000"}
];

phoneArray.forEach((sp,index)=>{
  console.log(index,sp);
});
```

```
0 ▶ {id: "1", tenSP: "Note 8", hang: "SamSung", gia: "20000"}
1 ▶ {id: "2", tenSP: "Vivo", hang: "Oppo", gia: "10000"}
2 ▶ {id: "3", tenSP: "Black Berry", hang: "Black Berry"}
3 ▶ {id: "4", tenSP: "Iphone X", hang: "Apple", gia: "20000"}
4 ▶ {id: "5", tenSP: "Iphone 11", hang: "Apple", gia: "32000"}
```

11. Một số hàm xử lý mảng thông dụng

map()

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

```
let phoneArray = [
  {id: "1",tenSP: "Note 8",hang: "SamSung",gia: "20000"},
  {id: "2",tenSP: "Vivo",hang: "Oppo",gia: "10000"},
  {id: "3",tenSP: "Black Berry",hang: "Black Berry"},
  {id: "4",tenSP: "Iphone X",hang: "Apple",gia: "20000"},
  {id: "5",tenSP: "Iphone 11",hang: "Apple",gia: "32000"}
];

let copyArray = phoneArray.map((sp,index)=>{
  return sp;
});

console.log(copyArray);
```

```
0 ▶ {id: "1", tenSP: "Note 8", hang: "SamSung", gia: "20000"}
1 ▶ {id: "2", tenSP: "Vivo", hang: "Oppo", gia: "10000"}
2 ▶ {id: "3", tenSP: "Black Berry", hang: "Black Berry"}
3 ▶ {id: "4", tenSP: "Iphone X", hang: "Apple", gia: "20000"}
4 ▶ {id: "5", tenSP: "Iphone 11", hang: "Apple", gia: "32000"}
```

11. Một số hàm xử lý mảng thông dụng

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).
- Reduce() có 2 tham số là hàm callback và giá trị khỏi tạo của tham số 1 của hàm callback. Hàm callback có 4 tham số:
  - Tham số 1: Giá trị return của hàm
  - Tham số 2: Phần tử của mảng
  - Tham số 3: vị trí (index) của phần tử trong mảng
  - Tham số 4: chính là hàm đang duyệt

```
let phoneArray = [
  {id: "1",tenSP: "Note 8",hang: "SamSung",gia: 20000},
  {id: "2",tenSP: "Vivo",hang: "Oppo",gia: 10000},
  {id: "3",tenSP: "Black Berry",hang: "Black Berry", gia:30000},
  {id: "4",tenSP: "Iphone X",hang: "Apple",gia: 20000},
  {id: "5",tenSP: "Iphone 11",hang: "Apple",gia: 32000}
];

let total = phoneArray.reduce((sum,sp,index,array)=>{
    sum = sum + sp.gia;
    return sum;
},0);
console.log(total);//112000
```

reduceRight()

reduceRight() giống như reduce() nhưng hàm này sẽ duyệt mảng từ phải qua trái.

```
let phoneArray = [
  {id: "1",tenSP: "Note 8",hang: "SamSung",gia: 20000},
  {id: "2",tenSP: "Vivo",hang: "Oppo",gia: 10000},
  {id: "3",tenSP: "Black Berry",hang: "Black Berry", gia:30000},
  {id: "4",tenSP: "Iphone X",hang: "Apple",gia: 20000},
  {id: "5",tenSP: "Iphone 11",hang: "Apple",gia: 32000}
];
```

```
let resultArray = phoneArray.reduceRight((newArray,sp,index)=>{
     if(sp.hang === "Apple"){
      newArray.push(sp);
     }
     return newArray;
  },[]);
  console.log(resultArray);
```

```
(2) [{…}, {…}]
▶ 0: {id: "5", tenSP: "Iphone 11", hang: "Apple", gia: 32000}
▶ 1: {id: "4", tenSP: "Iphone X", hang: "Apple", gia: 20000}
```

11. Một số hàm xử lý mảng thông dụng

reverse()

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

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

let newArray = intArray.reverse();
console.log(newArray);//
[5, 4, 3, 2, 1]
```

11. Một số hàm xử lý mảng thông dụng

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 phoneArray = [
  {id: "1",tenSP: "Note 8",hang: "SamSung",gia: 20000},
  {id: "2",tenSP: "Vivo",hang: "Oppo",gia: 10000},
  {id: "3",tenSP: "Black Berry",hang: "Black Berry", gia:300
00},
  {id: "4",tenSP: "Iphone X",hang: "Apple",gia: 20000},
  {id: "5",tenSP: "Iphone 11",hang: "Apple",gia: 32000}
];

let newArray = phoneArray.sort((sp, next)=>{
  return sp.gia - next.gia;
});
console.log(newArray);
```

```
(5) [{…}, {…}, {…}, {…}, {…}]
▶ 0: {id: "2", tenSP: "Vivo", hang: "Oppo", gia: 10000}
▶ 1: {id: "1", tenSP: "Note 8", hang: "SamSung", gia: 20000}
▶ 2: {id: "4", tenSP: "Iphone X", hang: "Apple", gia: 20000}
▶ 3: {id: "3", tenSP: "Black Berry", hang: "Black Berry", gia: 30000}
▶ 4: {id: "5", tenSP: "Iphone 11", hang: "Apple", gia: 32000}
```