



ES6

CYBERLEARN

(ECMAScript 2015)

CYBERLEARN.VN

Nội Dung

01 ECMAScript ES6 là gì?

02 Cách khai báo biến trong ES6

03 Arrow Function

04 Default Parameter Values

05 Rest Parameter

06 Spread Operator

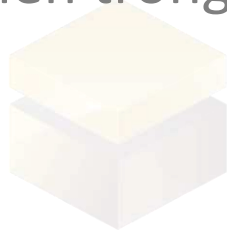
07 Destructuring

08 Template String

09 Object Literal

10 for of vs for in

11 ES6 import và export



CYBERLEARN
ĐÀO TẠO CHUYÊN GIA LẬP TRÌNH



01

ES6 là gì?



CYBERLEARN
ĐÀO TẠO CHUYÊN GIA LẬP TRÌNH

ES6 là gì?

➤ Gia phả nhà ECMAScript

- ES6 là chữ viết tắt của ECMAScript 6
- Ra đời năm 2015 nên có tên chính thức là ECMAScript 2015
- Là tập hợp nhiều kỹ thuật nâng cao của Javascript.
- ES6 hỗ trợ cho việc code trở nên dễ dàng và tường minh hơn giúp giải quyết các hạn chế của phiên bản trước đó (ES5)



ES6 là gì?

➤ Browser Support for ES6

- Internet Explorer không hỗ trợ ES6.



				
Chrome 58	Edge 14	Firefox 54	Safari 10	Opera 55
Jan 2017	Aug 2016	Mar 2017	Jul 2016	Aug 2018



02



CYBERLEAS.N
ĐÀO TẠO CHUYÊN GIA LẬP TRÌNH

Cách khai báo biến trong ES6

Cách khai báo biến trong ES6

- Trong ES6 chúng ta sẽ không sử dụng **var** mà thay vào đó sẽ sử dụng **const** và **let**.

Tại sao ES6 tạo thêm cách khai báo biến mới?



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
```

var thật dễ dãi !



Phân biệt var, let và const

➤ 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  
//Identifier 'name' has already been declared  
let name = 'Cybersoft1';
```

Let không dễ dự
như var đâu nha!



Phân biệt var, let và const

➤ Const

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

Const khó tính
lắm!



```
const PI = 3.1414;  
PI = 3.14; // Assignment to constant variable.  
const PI = 3.14; // Identifier 'PI' has already been declared  
const numPI; // Missing initializer in const declaration
```

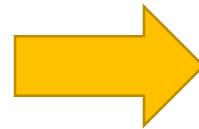
Phân biệt var, let và const

➤ Hoisting với var, let và const

Hoisting là hành động mặc định của Javascript, nó sẽ di chuyển các khai báo biến và functions lên đầu. Cụ thể, khi một file javascript compiled bởi browser, nhưng trước khi nó thực sự executed thì các khai báo sẽ được stored in memory và được đẩy lên đầu file

Code chúng ta thấy:

```
console.log(sum(2,3));  
console.log(city);  
  
function sum(x,y){  
    ...return x + y;  
}  
var city = "Ho Chi Minh";
```



Code khi browser chạy file javascript:

```
function sum(x,y);  
var city;  
  
console.log(sum(2,3));  
console.log(city);  
  
function sum(x,y){  
    ...return x + y;  
}  
var city = "Ho Chi Minh";
```

Phân biệt var, let và const

➤ Hoisting với var, let và const

- Khi gọi các biến trước khi khai báo:
 - **var** có giá trị ban đầu là undefined. Đó là bởi vì hoisting đã đưa các khai báo biến vào bộ nhớ và khởi tạo chúng với giá trị **undefined*** (chưa xác định)

Memory

Name	Value
name	uninitialized
address	uninitialized
city	undefined

Output

> undefined

```
console.log(name);  
console.log(address);  
console.log(city);  
  
const NAME = "Cybersoft";  
let address = "456 Su Van Hanh";  
var city = "Ho Chi Minh";
```

*Undefined là giá trị mặc định nếu ta khai báo biến nhưng không gán giá trị.

Vd: var x; => x = undefined

Phân biệt var, let và const

➤ Hoisting với var, let và const

- Khi gọi các biến trước khi khai báo:
 - **let** hoặc **const** khi được hoist không khởi tạo với giá trị undefined mà đưa vào Temporal Dead Zone và không được khởi tạo tới khi câu lệnh định nghĩa chúng được chạy tới
 - **let** hoặc **const** sẽ thông báo là **Uncaught ReferenceError: Cannot access '...' before initialization**

Memory

Name	Value
name	uninitialized
address	uninitialized
city	undefined

Output

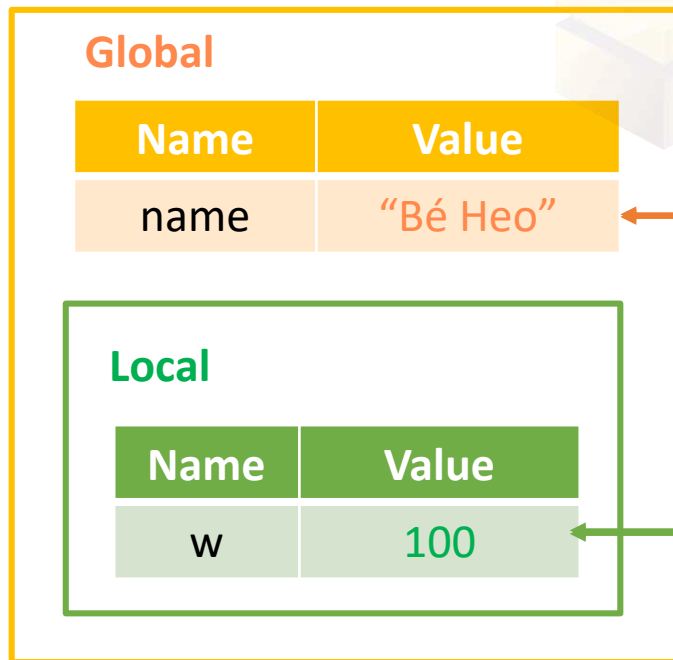
> uninitialized

```
console.log(name);  
console.log(address);  
console.log(city);  
  
const NAME = "Cybersoft";  
let address = "456 Su Van Hanh";  
var city = "Ho Chi Minh";
```

Phân biệt var, let và const

➤ Scope

- **Global Scope:** biến khai báo ở scope này sẽ có phạm vi sử dụng toàn file (biến toàn cục)
- **Local Scope:** biến khai báo ở scope này sẽ có phạm vi sử dụng bên trong function mà nó được khai báo (biến cục bộ).



```
//Khai báo biến
var name = "Bé Heo";
//Khai báo hàm
function weight(){
    var w = 100;
    console.log(name+": " + w + 'kg');
}
//Gọi hàm
weight(); Bé Heo:100kg
console.log(name); Bé Heo
console.log(w); w is not defined
```

Phân biệt var, let và const

➤ Var trong local scope

- Khi 1 biến var được khai báo trong function thì nó sẽ có phạm vi function scope
- Function scope: biến đó sẽ được dùng ở bất cứ đâu trong function đó.
- Bên ngoài sẽ không gọi được biến đó.

Global	
Local	
Name	Value
w	100
nickname	"Bé Heo"

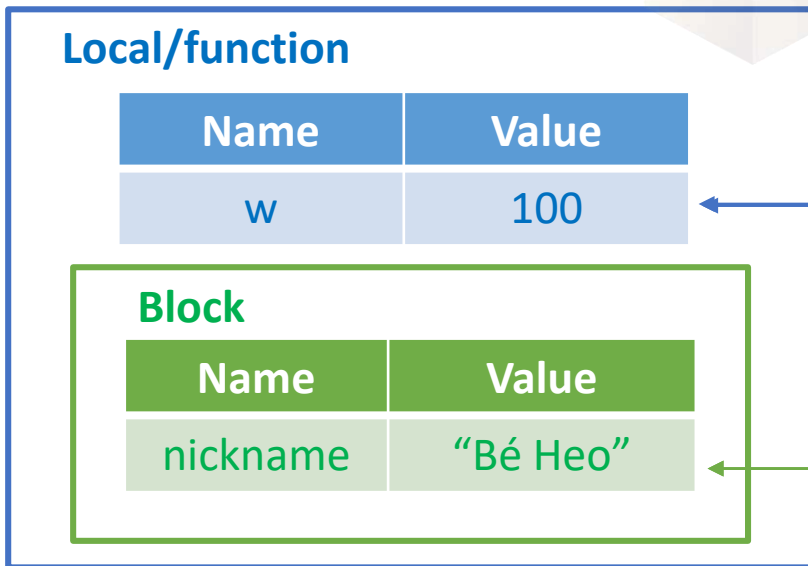
```
//Khai báo hàm
function weight() {
  var w = 100;
  if (w > 60) {
    var nickname = "Bé Heo";
    console.log(nickname + " nặng: "
      + w + "kg.");
  }
  console.log(nickname +
    " hơi gầy! Ráng ăn uống thêm.");
}

//Gọi hàm Bé Heo nặng: 100 kg
weight(); Bé Heo hơi gầy! Ráng ăn uống thêm.
console.log(w); .. is not defined
console.log(nickname); .. is not defined
```

Phân biệt var, let và const

➤ Let và const trong local scope

- Ngoài khái niệm global và local scope thì let và const sẽ có thêm block scope
- Khi 1 biến let hoặc const được khai báo trong mệnh đề if hoặc vòng lặp thì nó sẽ có phạm vi block scope
- Block scope: biến đó chỉ được sử dụng trong mệnh đề if hoặc vòng lặp. Bên ngoài sẽ không gọi được biến đó.



```
//Khai báo hàm
function weight() {
  let w = 100;
  if (w > 60) {
    let nickname = "Bé Heo";
    console.log(nickname + " nặng: "
      + w + "kg.");
  }
  console.log(nickname +
    " hơi gầy! Ráng ăn uống thêm.");
}

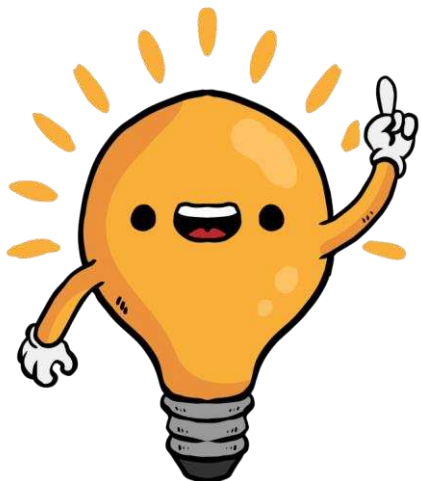
//Gọi hàm
weight();
console.log(w);
console.log(nickname);
```

Bé Heo nặng: 100 kg
nickname is not defined

Tại sao ES6 tạo thêm cách khai báo biến mới?

	var	let	const
Khai báo	Cho khai báo lại	Không khai báo lại	Không khai báo lại
Gán giá trị	Cho gán lại giá trị	Cho gán lại giá trị	Không gán lại giá trị
Hoisting	Khai báo di chuyển lên đầu	Khai báo vẫn ở vị trí như trong code	Khai báo vẫn ở vị trí như trong code
Scope	Function Scope	Block Scope	Block Scope

- Dùng var khi cần chạy trên hệ thống không hỗ trợ ES6.
- Let tốt hơn var vì nó giới hạn scope và không cho khai báo trùng. Có thể biến tất cả khai báo var thành let mà không có bất kỳ thay đổi ngữ nghĩa nào.
- Const được đánh giá là tốt hơn so với let. Mỗi const có thể được thay thế bằng một biến let, sử dụng const có thể tránh bị khai báo và gán lại giá trị.





03



CYBERLEARN
ĐÀO TẠO CHUYÊN GIA LẬP TRÌNH

Arrow Function

Khai báo function

➤ Function trong Javascript có 2 cách khai báo:

- Function declarations khai báo bắt đầu là function operator, sau đó gán cho nó một cái tên:

```
function sum(x,y){  
    console.log(x + y);  
}
```

- Function expressions khai báo bằng cách tạo một variable sau đó gán cho nó với một anonymous function:

```
var sum = function(x,y){  
    console.log(x + y);  
}
```

() => { }

Arrow Function

- Trong ES6, bên cạnh cách khai báo hàm bằng từ khóa "function" thì chúng ta có thể lược bỏ từ khóa "function" và thay bằng dấu mũi tên "=>"



```
function hello(name){  
  ... return "Hello" + name;  
}
```



```
let hello = (name) => {  
  ... return "Hello" + name  
}
```

const =

ES6

(lang=
'es',
version
=6)



Arrow Function

- Nếu chỉ có 1 biến thì có thể lược bỏ "()" "

```
let hello = (name) => {  
  ... return "Hello" + name  
}
```



```
let hello = name => {  
  ... return "Hello" + name;  
}
```

Arrow Function

- Trong trường hợp hàm chỉ có 1 lệnh return thì ta có thể lược bỏ chữ return và "{}".

```
let hello = name => {  
    ...return "Hello " + name;  
}
```



```
let hello = name => "Hello " + name;
```

Arrow Function - Lỗi cú pháp hay gặp

➤ Lỗi cú pháp hay gặp:

- Arrow phải nằm cùng hàng với tên hàm:

Wrong

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

Right

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

Arrow Function - Lỗi cú pháp hay gặp

➤ Lỗi cú pháp hay gặp:

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

Right

CYBERLEARN
ĐÀO TẠO CHUYÊN GIA LẬP TRÌNH

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


Arrow Function - Lỗi cú pháp hay gặp

➤ Lỗi cú pháp hay gặp:

- Khi arrow function bên trong một hàm hoặc sử dụng dạng một biến thì phải dùng cặp đóng mở để bao quanh lại hoặc khai báo function expressions :



Wrong

```
console.log(typeof () => "Hello Mị");
```

Right

```
console.log(typeof (() => "Hello Mị"));  
  
let hello = () => "Hello Mị";  
console.log(typeof hello);
```

JavaScript “this”

Con trỏ This

- **this** trong javascript đôi khi rất phức tạp và rối rắm. Trong trường hợp các function lồng nhau hoặc callback function, chúng ta cần xác định context (ngữ cảnh) của **this**
- Mỗi function trong javascript định nghĩa 1 ngữ cảnh của **this**

Con trỏ This

- Context (ngữ cảnh) của **this** trong đoạn code dưới đây là gì?

```
//Xây dựng đối tượng dựa trên function ES5
let hocVien = {
  ... hoTen: "Hoàng Thùy Mị",
    lop: "Ngu Van 12",
  ... layThongTinHocVien:function(){
    ... console.log("Họ tên: "+this.hoTen+" - Lớp: "+this.lop);
    ... }
  ... }

hocVien.layThongTinHocVien();
```



Con trỏ This

Đáp án: con trỏ **this** được sử dụng trong method **layThongTinHocVien**, và method này được định nghĩa trong đối tượng **hocVien**, vậy nên **this** sẽ tham chiếu đến đối tượng **hocVien** và ta có thể truy xuất các thuộc tính của đối tượng thông qua **this**.



```
//Xây dựng đối tượng dựa trên function ES5
let hocVien = {
  ...hoTen: "Hoàng Thùy Mị",
  lop: "Ngu Van 12",
  ...layThongTinHocVien:function(){
  ...  console.log("Họ tên: "+this.hoTen+"-Lớp: "+this.lop);
  ...  }
  ...}

hocVien.layThongTinHocVien();
```



Nhược điểm của this trong closure function

- Kết quả của hàm **layThongTinHocVien** và context (ngữ cảnh) của **this** trong đoạn code dưới đây là gì?

CYBERLEARN

```
//Xây dựng đối tượng dựa trên function ES5
var hocVien = {
  ...hoTen: "Hoàng Thùy Mị",
  ...lop: "Ngu Van 12",
  ...diemThi:[10,9,9,6],
  ...layThongTinHocVien:function(){
    ...this.diemThi.map(function(diem,index){
    ...  console.log("Họ tên: "+this.hoTen+" - Lớp: "+this.lop);
    ...  console.log("Điểm thi "+index+": "+diem);
    ...  });
    ...}
  ...}

hocVien.layThongTinHocVien();
```



Nhược điểm của this trong closure function

- Kết quả: **Họ tên:** undefined – **Lớp:** undefined
- Ngữ cảnh của **layThongTinHocVien** là **hocVien** nên **this** ở đây tham chiếu đến đối tượng **hocVien**
- **this** bên trong anonymous function của hàm **map** không thể truy cập **this** bên ngoài nên **this** ở đây không thể xác định (undefined)



```
//Xây dựng đối tượng dựa trên function ES5
var hocVien = {
  ...hoTen: "Hoàng Thùy Mị",
  ...lop: "Ngu Van 12",
  ...diemThi:[10,9,9,6],
  ...layThongTinHocVien:function(){
    ...this.diemThi.map(function(diem,index){
      ...console.log("Họ tên: "+this.hoTen+"-Lớp: "+this.lop);
      ...console.log("Điểm thi "+index+": "+diem);
    });
  }
}

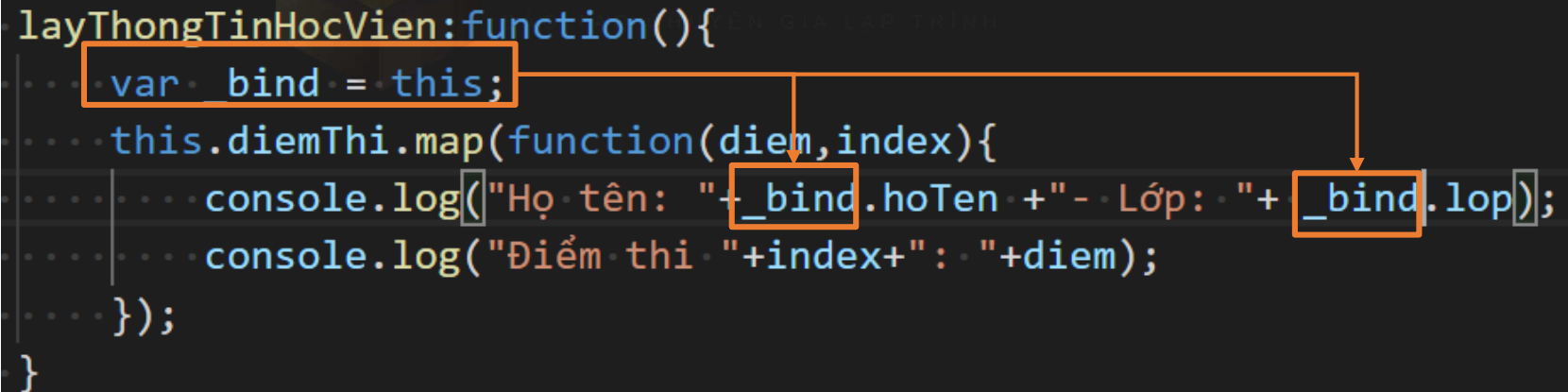
hocVien.layThongTinHocVien();
```

Nhược điểm của this trong closure function

➤ Giải pháp bằng ES5:

1. Chúng ta khai báo một biến tạm **_bind** để chứa giá trị của **this**, dùng biến tạm này để truy xuất các thuộc tính của đối tượng **hocVien**


```
layThongTinHocVien:function(){  
    var bind = this;  
    this.diemThi.map(function(diem,index){  
        console.log("Họ tên: "+_bind.hoTen+"- Lớp: "+_bind.lop);  
        console.log("Điểm thi "+index+": "+diem);  
    });  
}
```



Nhược điểm của this trong closure function

➤ Giải pháp bằng ES5:

2. Chúng ta dùng hàm **bind()** thì đối tượng **this** mới chính là **hocVien**



```
layThongTinHocVien:function(){ .....  
  this.diemThi.map(function(diem,index){  
    console.log("Họ tên: "+this.hoTen+"-Lớp: "+this.lop);  
    console.log("Điểm thi "+index+": "+diem);  
  }.bind(this));  
}
```


Nhược điểm của this trong closure function

➤ Giải pháp bằng ES6:

- Chúng ta không cần phải khai báo biến tạm `_bind` hay dùng hàm `bind()` nữa
- Arrow function không tạo ra this context, và biến this sẽ là biến this của đối tượng **hocVien**



```
layThongTinHocVien:function(){.....  
  this.diemThi.map((diem,index) => {  
    console.log("Họ tên: "+this.hoTen+"- Lớp: "+this.lop);  
    console.log("Điểm thi "+index+": "+diem);  
  });  
}
```



Arrow function làm method cho object

- Nếu chuyển method (phương thức) **layThongTinHocVien** sang arrow function thì kết quả còn đúng ko?



CYBERLEARN
ĐÀO TẠO CHUYÊN GIA LẬP TRÌNH

```
let hocVien = {  
  hoTen: "Hoàng Thùy Mị",  
  lop: "Ngu Van 12",  
  diemThi: [10, 9, 9, 6],  
  layThongTinHocVien: () => {  
    this.diemThi.map((diem, index) => {  
      console.log("Họ tên: " + this.hoTen + " - Lớp: " + this.lop);  
      console.log("Điểm thi " + index + ": " + diem);  
    });  
  }  
}  
  
hocVien.layThongTinHocVien();
```



Arrow function làm method cho object

- Chúng ta sẽ nhận được lỗi:

Uncaught TypeError: Cannot read property 'map' of undefined

Arrow function không định nghĩa giá trị `this` của riêng nó, nên `this` bên trong hàm **layThongTinHocVien** sẽ không được định nghĩa, như vậy không nên sử dụng arrow

function cho method

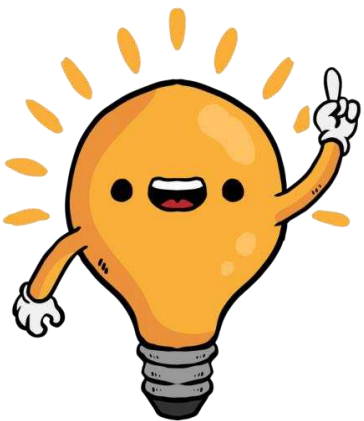


```
let hocVien = {
  hoTen: "Hoàng Thùy Mị",
  lop: "Ngu Van 12",
  diemThi: [10, 9, 9, 6],
  layThongTinHocVien: () => {
    this.diemThi.map((diem, index) => {
      console.log("Họ tên: "+this.hoTen+" - Lớp: "+this.lop);
      console.log("Điểm thi "+index+" : "+diem);
    });
  }
}

hocVien.layThongTinHocVien();
```

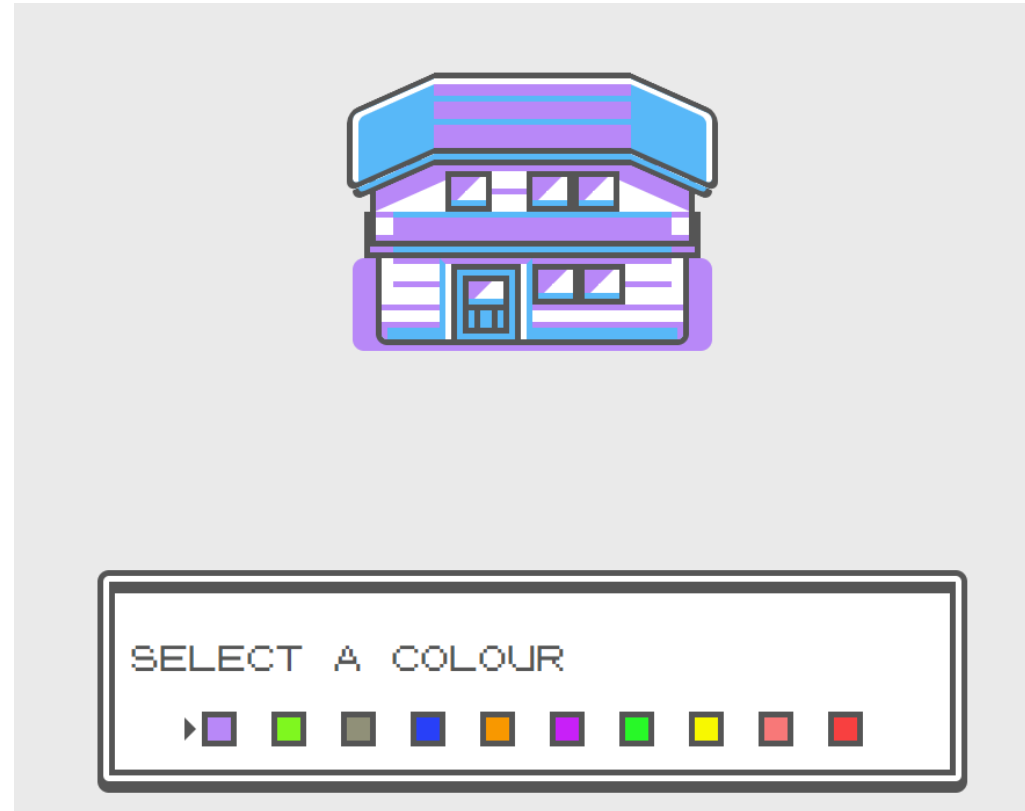
So sánh Function declaration và arrow function

	Function declaration	Arrow function
Cú pháp	Sử dụng từ khóa function	Ngắn gọn hơn, sử dụng ký tự =>
this	Định nghĩa giá trị this dựa vào nơi function chứa this được gọi	Không định nghĩa giá trị this
Hoisting	function declaration được hoisted nên bạn có thể gọi hàm trước khi định nghĩa nó.	Arrow function không được hoisted: nghĩa là bạn phải định nghĩa arrow function trước khi sử dụng nó.
Method Object	Phù hợp	Không phù hợp



Bài tập Arrow Function

- Xây dựng ứng dụng đổi màu
 - Load danh sách ô màu
 - Khi click vào ô màu, ngôi nhà sẽ đổi màu theo



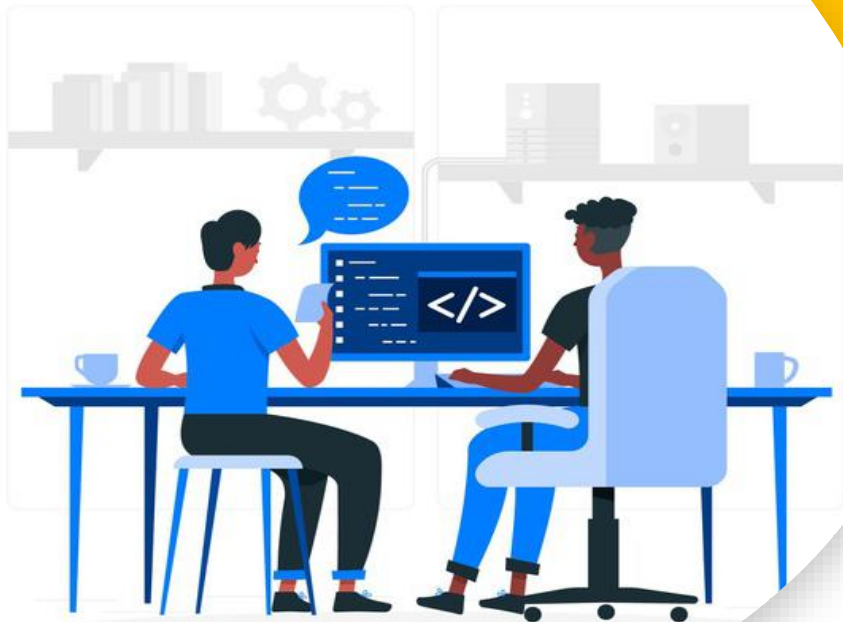


04



CYBERLEARN
ĐÀO TẠO CHUYÊN GIA LẬP TRÌNH

Default Parameter Values



Default Parameter Values

- Các phiên bản trước của JavaScript, Default Value (giá trị mặc định) không được hỗ trợ nên chúng ta phải kiểm tra parameter có giá trị hay không.
- Đối với ES6, chúng ta có thể gán giá trị mặc định cho tham số

Default Parameter Values

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



```
let getUserInfo = (name = "Mì", age = 18 )=>{  
  ...  if(age > 0 && age < 30){  
  ...|    ...  console.log(name + " đang còn trẻ."  
  ...|    ...|    ...|    ...|    ...|    + name + "muốn đi chơi");  
  ...  }  
  ...}  
  ...getUserInfo();
```




05

Rest Parameter



CYBERLEARN
ĐÀO TẠO CHUYÊN GIA LẬP TRÌNH

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 listPet(...pets) {  
  pets.forEach(function(pet) {  
    console.log(pet);  
  });  
}
```

```
listPet('🐱', '🐰', '🐶', '🐠', '🐤');
```

Bài tập Rest Parameter

- Viết chương trình tính điểm trung bình cho cả 2 khối lớp 1 và 2. Cả 2 khối lớp dùng chung 1 hàm tính điểm trung bình

Khối lớp 1

Toán	8
Lý	8
Hóa	8

Tính

8.00

Khối lớp 2

Văn	5
Sử	5
Địa	5
English	5

Tính

5.00



06

Spread Operator



CYBERLEARN
ĐÀO TẠO CHUYÊN GIA LẬP TRÌNH

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: "Female"}
```

Spread Operator

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



CYBERLEARN

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

Bài tập Spread Operator

- Xây dựng hiệu ứng jump hover. Khi hover vào từng chữ, chữ sẽ được phóng to.

Hover Me!



***H*overMe!**

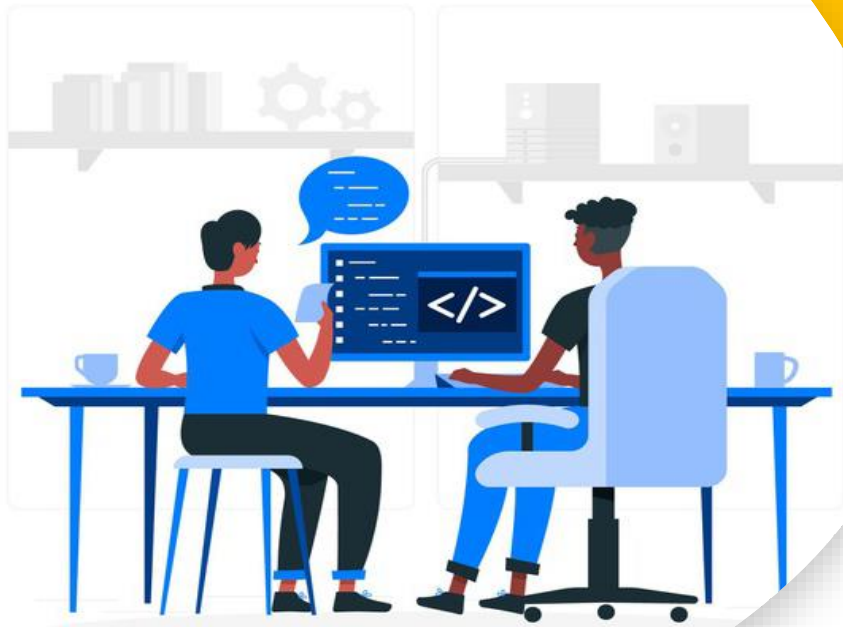


07



CYBERLEARN
ĐÀO TẠO CHUYÊN GIA LẬP TRÌNH

Destructuring



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.

Destructuring

➤ 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

Destructuring

➤ Object Destructuring

```
let pet = {  
  .... name: 'Gâu Đần',  
  .... age: 3,  
  .... breed: 'Golden Retriever',  
  .... size: {  
    .... weight: '30kg',  
    .... height: '56cm'  
  }  
}  
.... // Lấy thuộc tính bằng ES5  
.... var name = pet.name;  
.... var age = pet.age;  
.... var size = pet.size;  
.... console.log(name, age, size); //Gâu Đần, 3, object
```

```
let pet = {  
  .... name: 'Gâu Đần',  
  .... age: 3,  
  .... breed: 'Golden Retriever',  
  .... size: {  
    .... weight: '30kg',  
    .... height: '56cm'  
  }  
}  
.... //ES6 - destructuring  
.... let {name, age} = pet;  
.... let {weight, height} = pet.size;  
.... console.log(name, age); //Gâu Đần, 3  
.... console.log(weight, height); //30kg, 56cm  
.... //Đặt tên mới cho biến  
.... let {weight: w, height: h} = pet.size;  
.... console.log(w, h); //30kg, 56cm
```

Destructuring

➤ 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

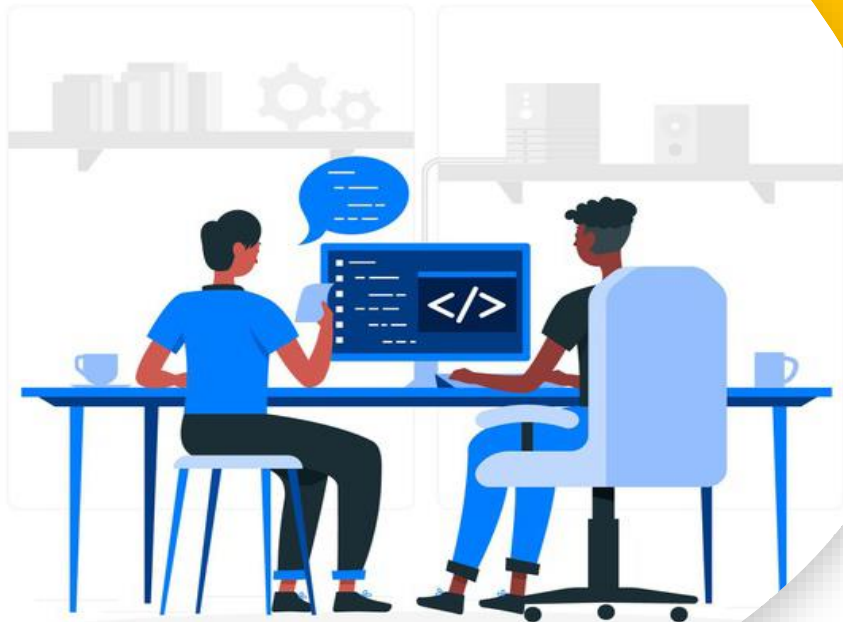
```
let topProgrammingLang = ['JavaScript', 'Python', 'Java'];  
let [first, second, third] = topProgrammingLang;  
console.log(first); //JavaScript  
console.log(second); //Python  
console.log(third); //Java
```

08



CYBERLEARN
ĐÀO TẠO CHUYÊN GIA LẬP TRÌNH

Template String



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.

Template String

- String sẽ nằm trong dấu ``...``
- Để truyền biến trong string dùng `${...}`
- Ở ES5, cú pháp để xuống dòng một đoạn text sẽ dài và phức tạp.
- ES6 có cú pháp ngắn gọn, không cần nhiều dấu cộng và dấu nháy đơn

```
let pet = "Cá";
let action = "bơi";
// Cách ES5
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}`);
```

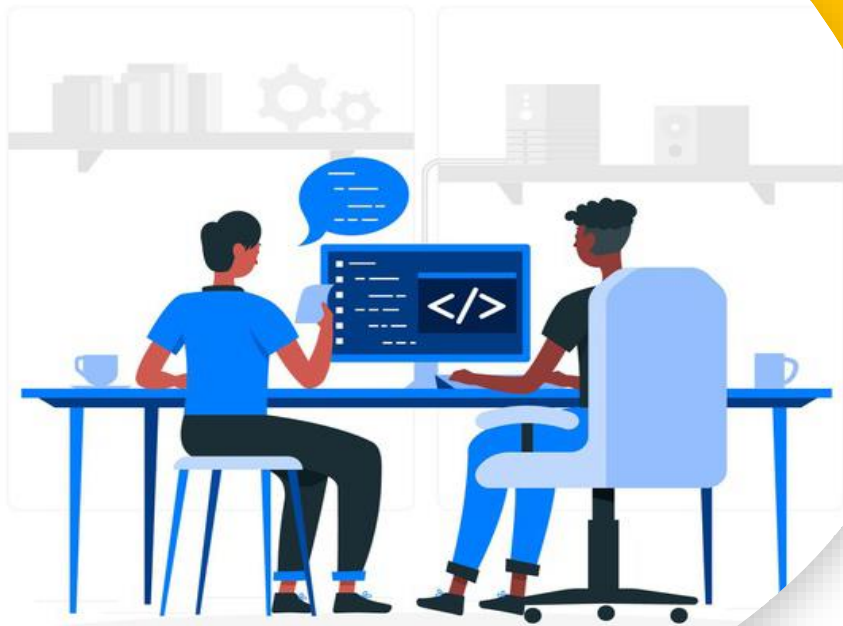


09



CYBERLEARN
ĐÀO TẠO CHUYÊN GIA LẬP TRÌNH

Object Literal



Object Literal

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

CYBERLEARN
ĐÀO TẠO CHUYÊN GIA LẬP TRÌNH

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

Object Literal

- Trong ES6, cú pháp tạo object được đơn giản hóa hơn.
- Cho phép khai báo tắt thuộc tính của object với biến cùng tên,
- Khai báo tắt phương thức bằng cách lược bỏ function



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

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

Object Literal

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



CYBERLEARN
ĐÀO TẠO CHUYÊN GIA LẬP TRÌNH

```
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
```

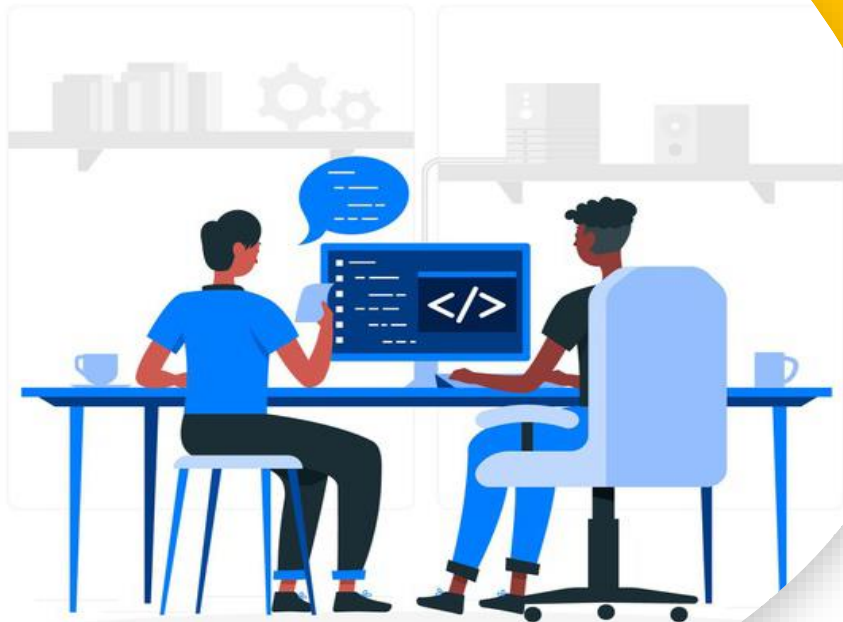


10

for of vs for in



CYBERLEARN
ĐÀO TẠO CHUYÊN GIA LẬP TRÌNH



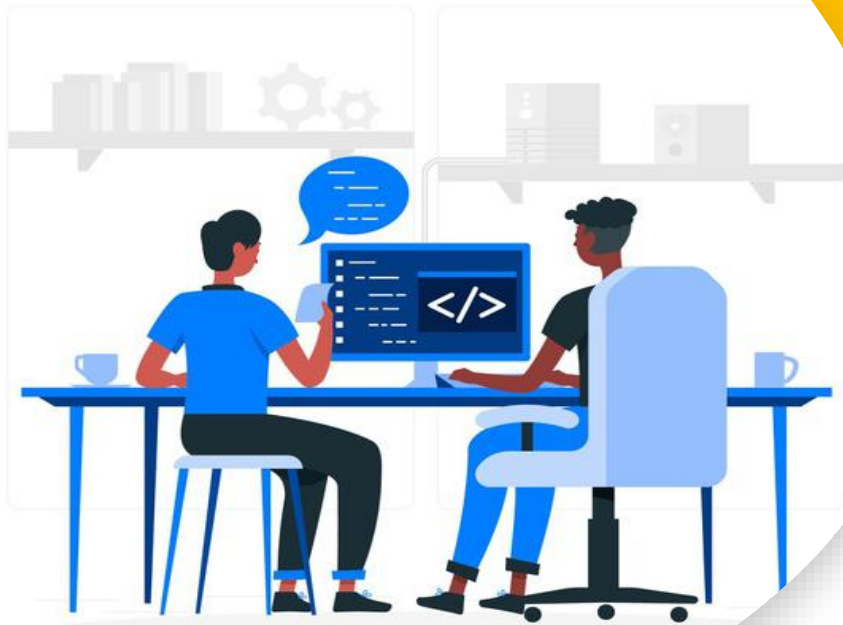
for of vs for in

➤ For ... in...

- Cách duyệt mảng đã được sử dụng trong ES5
- Cơ chế duyệt mảng theo index

CYBERLEARN
ĐÀO TẠO CHUYÊN GIA LẬP TRÌNH

```
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 of vs for in

➤ For ... of ...

- Cách duyệt mảng mới của ES6
- Cơ chế duyệt mảng theo từng phần tử

CYBERLEARN
ĐÀO TẠO CHUYÊN GIA LẬP TRÌNH

```
let currencies = ['VND', 'USD', 'SGN', 'AUD'];
for (let value of currencies) {
  console.log(value);
}
// VND
// USD
// SGN
// AUD
```

	For	For in	For of
Cú pháp	<ul style="list-style-type: none"> Dài 	<ul style="list-style-type: none"> Ngắn gọn 	<ul style="list-style-type: none"> Ngắn gọn
Giá trị và index	<ul style="list-style-type: none"> Cho phép truy cập vào index Sử dụng arr[i] để lấy giá trị 	<ul style="list-style-type: none"> Cho phép truy cập vào index Sử dụng arr[i] để lấy giá trị 	<ul style="list-style-type: none"> Truy cập trực tiếp đến giá trị của phần tử Không truy cập đến index của phần tử (*)
Phần tử rỗng	<ul style="list-style-type: none"> không qua các phần tử rỗng 	<ul style="list-style-type: none"> bỏ qua các phần tử rỗng 	<ul style="list-style-type: none"> không qua các phần tử rỗng

(*) Để lấy được index của phần tử đang được duyệt khi sử dụng for/of, sử dụng Array entries function

```
let currencies = ['VND', 'USD', 'SGN', 'AUD'];
for (let [i,v] of currencies.entries()) {
  console.log(i,v);
}
// 0 VND
// 1 USD
// 2 SGN
// 3 AUD
```

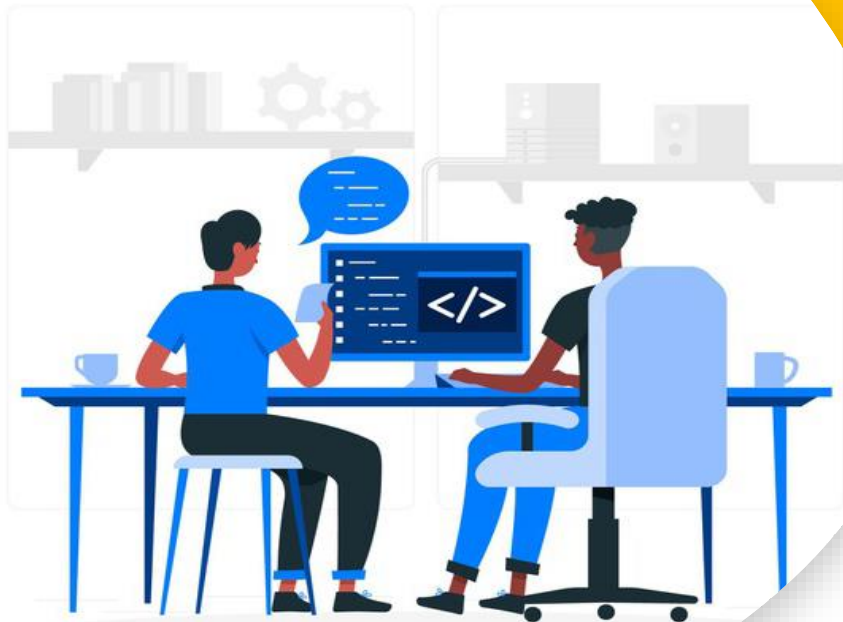


12



CYBERLEARN
ĐÀO TẠO CHUYÊN GIA LẬP TRÌNH

ES6 import và export



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 thành các phần khác nhau
- Trước ES6, các thư viện JavaScript dùng `require()` và `module.exports()` thay cho import và export (ví dụ: Node.js)

ES6 import và export

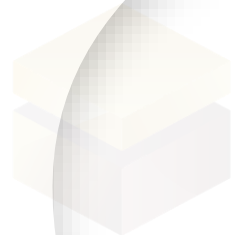
- Nếu export default, khi ta import có thể đặt tên biến tùy ý (không có dấu {})
- Nếu export không có từ khóa default, khi import ta phải đặt tên biến giống với tên đã export và có {}
- Có thể export nhiều biến bằng cách: export {...}

```
//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));
```



CYBERLEARN
ĐÀO TẠO CHUYÊN GIA LẬP TRÌNH

Thank You