# Phân biệt giữa Let và Const

## const

* const dùng để khai báo một hằng số - là một giá trị không thay đổi được trong suốt quá trình chạy.

const A = 5;

## let

let tạo ra một biến chỉ có thể truy cập được trong block bao quanh nó, khác với var - tạo ra một biến có phạm vi truy cập xuyên suốt function chứa nó.

function foo() {

let x = 10;

if (true) {

let x = 20; *// x này là x khác rồi đấy*

console.log(x); *// in ra 20*

}

console.log(x); *// in ra 10*

}

# Arrow functions (<https://www.codehub.vn/ES6-Co-Ban/ES6-Arrow-Function>)

Arrow function - là cú pháp được mượn từ CoffeeScript cú pháp này là cách ngắn gọn hơn dùng để viết function. Ở đây sử dụng kí tự =>, trông giống như một mũi tên . Arrow function là một hàm vô danh và nó thay đổi cách this bind đến function. Arrow function làm code của ta trông ngắn gọn hơn, giúp đơn giản hóa function scoping cũng như từ khóa this

Bằng cách sử dụng arrow function, chúng ta tránh được việc phải gõ từ khoá function, return và dấu ngoặc nhọn.

var multiply = (x, y) => { return x \* y };

# Imports and Exports(<https://www.codehub.vn/ES6-Co-Ban/ES6-import-va-export>)

## Import

Câu lệnh import trong ES6 dùng để *nhập* vào mô-đun từ một file cho trước (có thể là từ một thư viện hay mô-đun mà chúng ta tự định nghĩa).

Ví dụ để nhập vào mô-đun sayHello được tạo ra:

**import** { sayHello } **from** 'say\_hello.js';

Đoạn code trên sẽ *nhập* vào mô-đun sayHello từ tập tin say\_hello.js và gán giá trị của mô-đun này cho biến sayHello.

Bạn có thể tuỳ ý đặt tên biến khi import mô-đun sayHello sử dụng từ khoá as:

**import** { sayHello **as** sayHelloFunction } **from** 'say\_hello.js';

**Lưu ý:** Trong trường hợp có nhiều mô-đun được tạo ra trong cùng một file thì bạn có thể tuỳ ý lựa chọn số lượng mô-đun nhập vào chứ không cần thiết phải nhập tất cả.

## export

Câu lệnh export dùng để xuất ra một mô-đun:

**let** sayHello = **function** (name) {

console.log("Xin chào! Tên tôi là " + name);

}

**export** { sayHello };

có thể export nhiều mô-đun trong cùng một file:

**let** sayHello = **function** (name) {

console.log("Xin chào! Tên tôi là " + name);

}

**let** sayGoodbye = **function** () {

console.log("Chào tạm biệt!");

}

**export** { sayHello, sayGoodbye };

# Classes

## các class giống như các "function đặc biệt", và cũng giống như có thể định nghĩa hàm biểu thức  và khai báo hàm . Tạo class bằng function

Định nghĩ một class mới bằng từ khóa function và sử dụng các biến cho instance bằng từ khóa this

function **Person**(name,birthYear) {

this.name = name;

this.birthYear = birthYear;

this.getAge = function() {

return new **Date**().**getFullYear**() - birthYear;

}

}

var p = new Person("TH",1991);

p.getAge(); *// 24*

# Destructuring (<https://viblo.asia/p/destructuring-assignment-in-es6-xlbRBNQgRDM>)

một cú pháp cho phép tách dữ liệu được lưu trữ bên trong (nested) Objects hoặc Arrays (tổng quát hơn là các iterable values) và gán chúng cho các biến riêng biệt.

$domain = array(

    'freetuts.net',

    'qa.freetuts.net'

);

// Gán hai phần tử vào hai biến $main và $sub

list($main, $sub) = $domain;

echo $main; // kết quả freetuts.net

echo $sub; // kết quả qa.freetuts.net

# for...of

cú pháp

for (tên biến of đối tượng để loop) {

...câu lệnh...

}

**tên biến**

*Tên biến* đại diện cho một phần tử được loop.

**đối tượng để loop**

Array, Object,... các đối tượng để chạy vòng lặp

function\* foo(){

yield 1;

yield 2;

}

for (let o of foo()) {

console.log(o);

// expected output: 1

break; // closes iterator, triggers return

}

# Computed Properties

 là thuộc tính này được sinh ra sau khi qua một function xử lí. Bình thường với các phép tính toán đơn giản, ta có thể viết trực tiếp trong template bằng cách đặt trong cú pháp {{ }}. Tuy nhiên, khi phép tính toán trở nên phức tạp hơn, thì việc viết trong template sẽ rất cồng kềnh, gây khó khăn cho việc phát triển, bảo trì.

<script>

export default {

name: 'test',

data () {

return {

message: 'Welcome to my app'

}

},

computed: {

//

}

}

</script>

# Object Concise Method

Là phương pháp giúp code ngắn gọn, súc tích hơn

function () {}đếnrender()

vd ban đầu hay sd

var o = {

foo() { .. }

};

Cách viết ngắn hơn

var o = {

foo: function() { .. }

};

# Object Short Notation

# Rest Params (<https://www.codehub.vn/ES6-Co-Ban/ES6-Rest-Parameter>)

bạn có thể khai báo một hàm với số lượng tham số không xác định, đây là một tính năng mới khiến Javascript ngày càng trở nên mạnh mẽ hơn.

có thể định nghĩa một hàm với số lượng tham số có thể thay đổi tuỳ ý. Hay nói theo cách khác khi chúng ta không biết chắc chắn số lượng tham số cần có của một hàm chúng ta có thể sử dụng rest parameter.

Vđ

// Khai báo hàm

let domainList = (main, sub, ...other) =>

{

    console.log("Main: " + main);

    console.log("Sub: " + sub);

    console.log("Other");

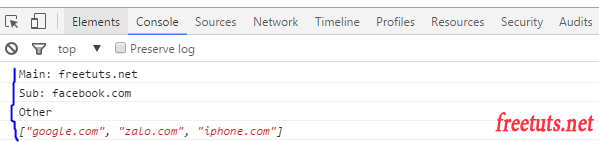
    console.log(other);

}

// Gọi hàm

domainList('freetuts.net', 'facebook.com', 'google.com', 'zalo.com', 'iphone.com');

Hàm này có 3 tham số truyền vào là main, sub và Rest Parameter other. Chạy lên bạn sẽ thấy giao diện như sau:



Dựa vào trong hình bạn thấy biến other sẽ lưu một mảng giá trị các tham số cuối cùng, vì vậy để lấy từng giá trị thì bạn chỉ việc sử dụng vòng lặp là được.

# Template Literals

ó thể **nhúng biểu thức** JavaScript (ví dụ như giá trị của biến, hoặc kết quả của một phép toán...) bên trong chuỗi string một cách dễ dàng.

Sử dụng phiên bản trước của ECMAScript thì việc nối chuỗi với giá trị của một biến hoặc biểu thức thường sử dụng toán tử + và chúng ta sẽ thường được thấy những đoạn code tương tự sau xuất hiện trong source code

**var** spiderMan = {

name: "Peter",

age: 25

};

**var** superMan = {

name: "Clark",

age: 200

};

console.log("Hello, " + superMan.name

+ ", my name is " + spiderMan.name

+ ", I'm " + spiderMan.age + " years old."

+ " And you're " + superMan.age + " years old."

+ " Do you want to have a fight!!!");

Đoạn code trên khi chạy sẽ hiển thị kết quả sau trong tab console:

Hello, Clark, my name **is** Peter, I'm 25 years old. And you're 200 years old. Do you want to have a fight!!!

Template literal có cú pháp như sau:

`Text ${javascript\_expression} other text...`

Trong đó javascript\_expression là một biểu thức JavaScript.

Ví dụ:

**let** firstName = "Peter";

**let** lastName = "Parker";

console.log(`Hello everyone! My name is ${firstName} ${lastName}.`);

Kết quả trả về:

Hello everyone! My name **is** Peter Parker.

# Function Trailing Comma (<https://ehkoo.com/bai-viet/why-trailing-comma-javascript>)

là việc để dư **một dấu phẩy** sau phần tử cuối cùng ở cuối một danh sách (có thể là phần tử trong Array literal, property trong Object literal, tham số của hàm…). Ví dụ:

// trailing comma trong Array literal

var categories = [

'men',

'women',

'accessories',

];

// trailing comma trong Object literal

var person = {

name: 'Harry Potter',

gender: 'male',

house: 'Gryffindor',

};

// trailing comma trong tham số của hàm (và trong lời gọi hàm)

function compare(

param1,

param2,

) { /\* ... \*/ }

# Async Functions (<https://viblo.asia/p/tim-hieu-ve-async-functions-LzD5dbbWZjY>)

một tính năng của JavaScript giúp chúng ta làm việc với các hàm bất đồng bộ theo cách thú vị hơn và dễ hiểu hơn. Nó được xây dựng trên Promises và tương thích với tất cả các Promise dựa trên API.

# Object Spread

cho phép chuyển đổi một chuỗi thành nhiều argument (trong trường hợp gọi với hàm) hoặc thành nhiều phần tử (cho array). Thêm vào nữa nó cũng cho phép làm nhiệm vụ destructure. Operator này có syntax là 3 dấu chấm ...,

tkhao <https://www.codehub.vn/ES6-Co-Ban/ES6-Spread-Operator>

# Optional Chaining (<https://viblo.asia/p/tim-hieu-ve-optional-chaining-trong-javascript-924lJEXmZPM>)

là một operator trong lập trình hướng đổi tượng thường access một object và trả về **các giá trị pointer được nested** hoặc **undefined/null nếu không tồn tại**.

b3