**ES6**

# [Khai báo biến trong **ES6** với **let var const:**](ES6-notes.docx)

### 1.Let:

* Dùng để khai báo biến (Thay thế và khác phục một số nhược điểm của var)
* Có thể gán giá trị nhìu lần.

2.Const:

* Là hằng số, đại lượng không đổi.
* Không thể gán lại = 1 giá trị khác.
* Nếu hằng số là 1 Object nó không thể gán = 1 object khác tuy nhiên nó có thể set lại giá trị thuộc tính.

-Phân biệt var, let, const với cơ chế **Hoisting** :

Trong JS cho phép khi khai báo biến kiểu **var**  :

Ta có:

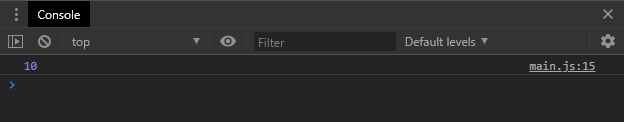
a= 10;

console.log(a);

var a=20;

biến **a** được sử dụng trước khi khai báo và JS cho phép điều đó khi khai báo kiểu **var** .

và kết quả:



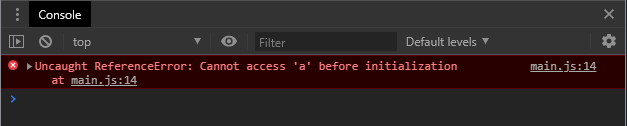
Nhưng khi khai báo kiểu **let** hoặc **const** :

a= 10;

console.log(a);

let a=20;

thì kết quả:



Vì biến **a** theo cơ chế của **let** , **const** khi biên dịch qua cơ chế **ES5**  thì nó sẽ biên dịch kiểu này :

/1/ var a;

/2/ console.log(a);

/3/ a=10;

Biến **a** ở dòng 1 dc khai báo chưa có giá trị nên khi thực hiện dòng lệch 2 **console.log(a);** sẽ **lỗi** ,đến dòng 3 thì biến **a** mới dc gán giá trị = **10**

**Vì** những gì khi khai báo biến let hay const trình biên dịch sẽ kéo lên đầu của **Function Scope** .

**Kết luận:**  *Mục đích của từ khóa* ***let****,* ***const*** *là để gàng buộc cơ chế là khai báo biến sau đó mới dc sử dụng .chứ ko phải như* ***var*** *có thể sử dụng tùy ý mặt dù chưa khai báo. “****khắc phục cơ chế sử dụng biến khi chưa được khai báo*** *.”*

[II. **Function Scope** và **Block Scope** :](ES6-notes.docx)

### Function Scope :

* + - Là phạm vi khai báo biến bên trong một hàm.
    - **Biến** bên trong **Scope** sẽ không lấy giá trị ra được từ bên ngoài.
    - **Bên ngoài** không sử dụng được **biến** bên trong NHƯNG **bên trong** sử dụng được **biến** bên ngoài **.**

### Block Scope :

* Là phạm vi khai báo **biến** bên trong **{...}** . biến bên trong scope sẽ không lấy dc giá trị từ bên ngoài . **Ngoại trừ Var**  nó sẽ không tuân thủ theo quy luật này .

if(true){

    var x= 10;

}

console.log(x);

// kết quả : 10

Khi dùng **let** :

if(true){

    let x= 10;

}

console.log(x);

// kết quả : Uncaught ReferenceError: x is not defined

**Kết luận:**

* *khi khai báo* ***let******phạm vi hoạt động*** *của nó nằm trong 1* ***scope*** *ứng với mỗi* ***scope*** *thì khai báo* ***let******x*** *sẽ khác nhau .*
* *còn đối với* ***var*** *khi khai báo trùng nó sẽ ảnh hưởng đến bên trong* ***scope*** *.*

[III. **Arrow Function** :](ES6-notes.docx)

**Arrow Function:** là một cách viết ngắn gọn của ES6. Là **function** được viết rút gọn từ khóa **function** thay bằng dấu mũi tên.

-Ngoài việc viết ngắn gọn **function**

### **Khi sử dụng Function của ES5:**

// no-arrow function

var hoTen ="abc";

// nó sẽ hỉu là

// window.hoTen="abc";

let hocVien={

    hoTen:'Nguyễn A',

    lop:'11dhpm',

    layThongTinHocVien: function(){

        function hienThiThongTin(){

            console.log('Họ Tên: '+*this*.hoTen+' Lớp: '+*this*.lop);

        }

        hienThiThongTin();

    }

}

hocVien.layThongTinHocVien();

// kq : 'Họ Tên: abc Lớp: undefined'

-vì khi sử dụng **function** của **ES5** **ngữ cảnh** của con trỏ **this** nó sẽ hỉu là của **window**

**Khi** sử dụng **arrow function** **ES6 :**

//use arrow function

window.hoTen="abc";

let hocVien = {

    hoTen: 'Nguyễn A',

    lop: '11dhpm',

    layThongTinHocVien: function () {

        let hienThiThongTin = () => {

            console.log('Họ Tên: ' +*this*.hoTen + ' Lớp: ' +*this*.lop);

        }

        hienThiThongTin();

    }

}

hocVien.layThongTinHocVien();

// kq : 'Họ Tên: Nguyễn A Lớp: 11dhpm'

-vì khi sử dụng **arrow function** nó sẽ hiểu ngữ cảnh của con trỏ **this** là của đối tượng **hocVien** mặc dùng t có khai báo biến hoTen với ngữ cảnh con trỏ là **window**.

Câu hỏi: v khi nào dùng **function** và khi nào dùng **arrow function** ?

-Ta sẽ dùng **arrow** **function** cho cách trường hợp dạng truyển **call back function** (nghĩa là :trong 1 **function** chúng ta cần sử dụng thêm 1 **function** nữa lòng vào trong và khi đó chúng ta muốn sử dụng đúng ngữ nghĩa con trỏ **this** chúng ta dùng **arrow function** ví dụ trong trường hợp trên).

[IV. **Rest Params** :](file:///C:\Git\front-end\ES6\ES6-notes.docx)

**Rest:** Các tham số truyền vào sẽ hợp thành 1 mảng, dùng khi không biết có bao nhiu tham số đầu vào của 1 hàm.

Vì trong JS **không có** khái niệm hàm chồng lên nhau

Ví dụ :

function tinhTong(a,b){

    console.log(a+b);

    return a+b;

}

function tinhTong(a,b,c){

    console.log(a+b+c);

    return a+b+c;

}

tinhTong(1,2);

tinhTong(1,2,3);

// kq : NaN 6

Nó sẽ không hàm **tinhTong** ở trên nên khi truyền 2 **param** thì nó trả về **NaN**

Khi sử dụng **…RestParams**

function tinhTong(...resParams){

    let tong=0;

    for(let i=0 ;i<resParams.length;i++){

        tong+=resParams[i];

    }

console.log(tong);

}

tinhTong(1,2);

tinhTong(6,7,2,6,3);

tinhTong(3,5,1,6);

// kq :3 24 15

**Ví dụ cho phép định nghĩa hàm chồng bằng Rest Param:**

let mangHocVien = [

    {

        maHV: 1,

        tenHV: 'Nguyễn Văn C',

    },

    {

        maHV: 2,

        tenHV: 'Nguyễn Văn D',

    },

]

function xuLyMangHocVien(...resParam) {

    if (resParam.length === 2) {

        resParam[1].push(resParam[0]);

        console.log(resParam[1]);

    }else if(resParam.length>2){

        switch(resParam[2]){

            case 'Delete':{

                let index=resParam[1].findIndex(hv => hv.maHV===resParam[0].maHV);

                resParam[1].splice(index,1);

                console.log(resParam[1]);

            }

            case 'Update':{

                let index=resParam[1].findIndex(hv => hv.maHV===resParam[0].maHV);

                resParam[1][index].tenHV='Kha đẹp trai';

                console.log(resParam[1]);

            }

        }

    }else{

        console.log(resParam[0]);

    }

}

let hv={

    maHV:3,

    tenHV:'Ngọc Kha',

}

xuLyMangHocVien(mangHocVien); // (2) [{…}, {…}]

xuLyMangHocVien(hv,mangHocVien); // (3) [{…}, {…}, {…}]

xuLyMangHocVien(hv,mangHocVien,'Update'); // (3) [{…}, {…}, {…}]

[V. **Spread Operator** :](file:///C:\Git\front-end\ES6\ES6-notes.docx)

**Spread Operator:**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**.

let mangA = [1, 2, 3, 4];

let mangB = mangA;

mangB.push(5, 6);

mangA.push(7,8);

console.log(mangB); // mình nghĩ : 1, 2, 3, 4

 kq: (8) [1, 2, 3, 4, 5, 6, 7, 8] !!!!

-khi mà gán 1 **mangB** = **mangA** thì theo tính chất của con trỏ trong lập trình thì vùng nhớ của **mangA** sẽ được trỏ cho **mangB**. khi **mangB** thay đổi thì **mangA** cũng thay đổi và tương tự ngược lại với khi thay đổi **mangA**.

Nếu muốn lấy dự liệu từ **mangA** ra xử lý và không muôn dữ liệu thay đổi thì **Spread Operator ES6** sẽ giúp chúng ta làm việc đó .

let mangA = [1, 2, 3, 4];

let mangB = [...mangA];

mangB.push(5, 6);

mangA.push(7,8);

console.log(mangB);

console.log(mangA);

/\* kq: (6) [1, 2, 3, 4, 5, 6]

       (6) [1, 2, 3, 4, 7, 8]

\*/

cú pháp [...mangA] tạo ra 1 mảng mới sẽ ôn hết giá trị mangA bỏ vào [] nên khi ta thay đổi mangB mangA ko bị ảnh hưởng .

**Điều này cũng xẩy ra tương tự với object:**

 let hs1={

    maHS:1, tenHS:'Nguyen Ngoc A',

 }

 let hs2=hs1;

 hs2.tenHS='Nguyễn Ngọc Kha';

 console.log(hs1);

//  kq : {maHS: 1, tenHS: "Nguyễn Ngọc Kha"}

Sử dụng **Spread operation** của **ES6** :

let hs1={

    maHS:1, tenHS:'Nguyen Ngoc A',

 }

 let hs2={...hs1};

 hs2.tenHS='Nguyễn Ngọc Kha';

 console.log(hs1);

//  kq: {maHS: 1, tenHS: "Nguyen Ngoc A"}

[VI. **Default params** :](file:///C:\Git\front-end\ES6\ES6-notes.docx)

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

function tinhTong(a = 5, b = 10, c = a + b) {

    console.log(a + b + c);

    return a + b + c;

}

tinhTong(); //nếu không truyền tham số gì thì nó sẽ lấy tham số mặc định của hàm

// kq : 15

tinhTong(10); //thì nó sẽ lấy tham số này thay vào giá trị đầu tiên và cộng tiếp cho tham số thứ 2

// kq : 20

tinhTong(2, 2);//thì nó sẽ lấy tham số 3 mặc định

// kq : 8

tinhTong(2,2,2);

// kq : 6

[VI. **For in For of**:](file:///C:\Git\front-end\ES6\ES6-notes.docx)

**For in** duyệt mảng theo chỉ số **index**

let arrName = ['Khai', 'Hùng', 'Tiên', 'Mỹ', 'Mọi'];

for(let index in arrName){

    console.log( 'index: '+index+' name: '+arrName[index]);

}

/\* kq:

index: 0 name: Khai

index: 1 name: Hùng

...

\*/

-**for in** sẽ lấy về **vị trí** của phần tử đó.

**for of** sẽ lấy về 1 **đối tưởng** của phần tử đó.

let arrName=[{name:'Tùng',age:20},{name:'Nhủ',age:21},

            {name:'Đại',age:22},{name:'Hoa',age:19}]

for (let item of arrName) {

    console.log(item);

}

/\* kq:

   {name: "Tùng", age: 20}

   {name: "Nhủ", age: 21}

   ....

\*/

[VII. **OOP:**](file:///C:\Git\front-end\ES6\ES6-notes.docx)