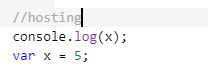
**Bài 1: Hoisting trong javascript**

Khai báo biến nằm ở đầu trương trình, khai báo biến nằm ở sau trương trình thì sẽ bị hoisting giá trị undifined.



**Bài 2: var, let , const**

**Let** và **const** không thể khai báo lại nhưng let vẫn gán được giá trị

Khai báo biến const thì phải gán giá trị cho nó;

Hoisting: Let và const đều không có thể hosting.

Block scope: những mở {} là khai báo scope. Trừ tạo object.

**Bài 3: function context & bind**

**Global context:** this mà không được gán vô đối tượng nào thì là global context.

**Function context:**

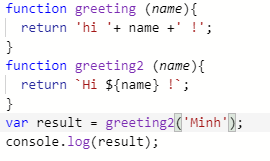
**Bind:** Tạo ra một function mới sẽ được đặt là giá trị cho trước. bind(biến) sẽ làm biến this.

**Bài 4: Arrow function expression**

**Arrow function: =>** hàm 1 tham số thì không cần ngoặc không có tham số có ngoặc không tham số. arrow fc không có biến this, context nên biến this này là biến this của hàm gần nhất.

**Fat arrow: ->**

**Bài 5: Template string.**

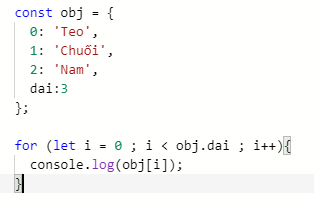


Bên trong $ có thê viết những gì liên quan đến javascript.

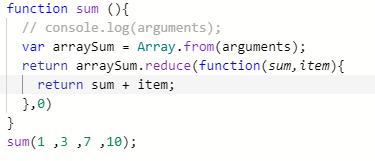
**Bài 7 : arguments.**

Biến nằm trong tất cả các hàm trừ arrow function.

**Array-like:** object giống như array thuộc tính của object đấy bắt đầu từ 0 và có một thuộc tính là chiều dài của object , # array không có method của array.

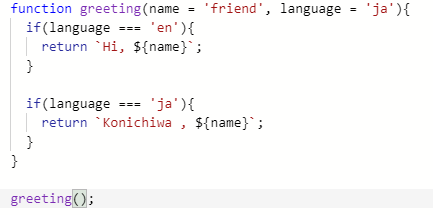


**Arguments:** không phải kiểu arrow function, hàm nhận n tham số đầu vào. Là một array.like object. Biến này không khai báo nhưng luôn tồn tại trong scope của hàm đấy.



**Bài 8: Default parameters**

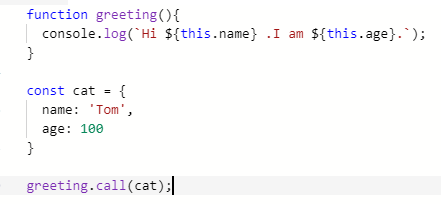
Tham số mặc định của một cái hàm. Mỗi tham số có một giá trị mặc định việc so sánh rất mất thời gian thay vào đó truyền vào default parameters.



**Bài 9: call**

**Method call:** nhận vào bao nhiều tham số cũng được. Khi hàm đó chạy tham số sẽ đại diện cho this, tham số thứ hai trở đi là tham số của hàm này.

* **Gọi luôn hàm tại lúc đó , còn bind trả về 1 hàm để gọi sau.**
* **Call: func.call(this, param1 , param2, …);**

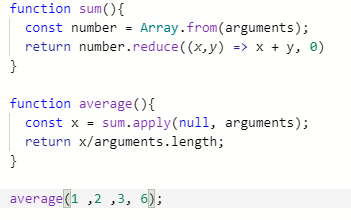


**Bài 9: Apply**

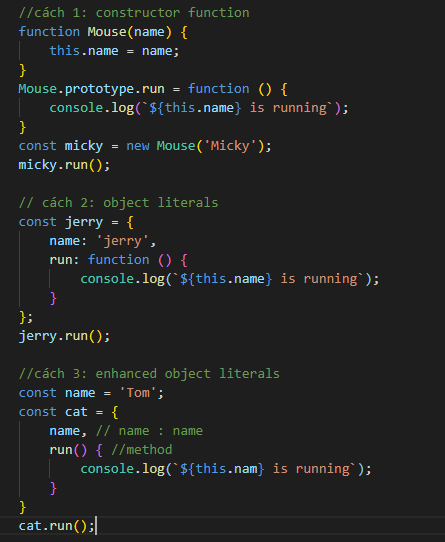
**Apply: function.apply(this, [param1 , param2, …]);**

**Nhận vào tham số là một array hoặc là một array like object.**

**Ứng dụng: gọi hàm mà mình không biết bao nhiêu đầu vào. Nó có thể kế thừa**



**Bài 10: Enhanced object literals**

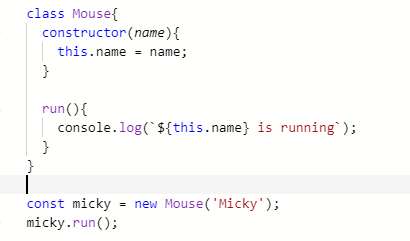


**Bài 12: class**

**Javascript không phải ngôn ngữ hướng đối tượng ngay từ đầu nên chúng ta sẽ không đi quá sâu vào các lý thuyết oop.**

**Cú pháp class được sinh ra để viết constructor function ngắn gọn hơn giống với ngôn ngữ oop hơn.**

**Bản chất của class là cũng sài prototype thôi.**

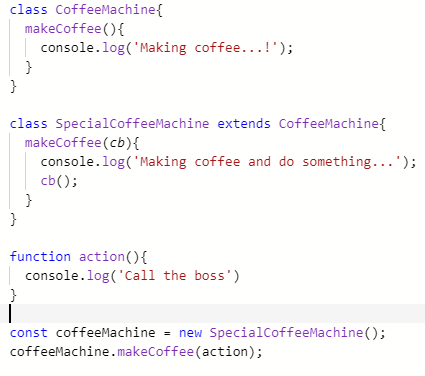


**Bài 12: class inheritance**

|  |  |
| --- | --- |
| **Class** | **Constructor** |
|  |  |

**Bài 14: method overriding**

**Ghi đè lên method của thang cha.**



**Bài 15: Supper:**

**Từ khóa supper đại diện cho thàng cha(Base class) kế thừa những gì của thằng cha và triển khai lên thêm.**

class RangedHero extends Hero{

constructor(name, hp, damege ,range){

super(name ,hp ,damege );

this.range = range;

}

attack(enemy){

super.attack(enemy);

this.hp += this.damege;

}

}

**Bài 16: Static**

**Static: Thuộc về class rồi chỉ có class mới gọi được nó.**

**Instance: một object được tạo ra từ class nào đây.**

**Ưng dụng: Không cần phải tạo object mới để sử dụng object đấy.**



**Bai 17: rest.( gom lại )**

**Toán tử rest của es6: không biết số lượng tham số không được gán là một mảng các tham số.**

**Khác với arguments: các biến truyền vào là một object chứ không phải là array.**

**Ứng dụng: Không biết bao nhiêu đầu vào thì sử dụng.**

// ví dụ 1

function sum(...nums){

return nums.reduce((a,b) => a+b,0);

}

sum(1,2,3,4,5,6)

// ví dụ 2

function concat(separator, ...string){

//rest lấy từ a b c

return string.join(separator);

}

concat('.','a','b','c');

**Bài 18: spread:**

**Toán tử spread (trải ra) bắt đầu băng …**

var b = [12, 13, 14, 15];

var c = [1,...b, 2];

**Bài 19: Value type vs Reference types:**

**Value type :lưu luôn giá trị vào biến đó.**

let a1 = 1;

let a2 = 1;

console.log(a1 === a2);

let a3 = a2;

console.log(a3 === a2);

a3 = 2;

console.log(a3 === a2)

**Reference types:kiểu array, function, object khi tạo ra trong bộ nhớ biến obj1 lưu giá trị tham chiếu tới giá thực tế của object đấy.**

* **Khi gán biến bằng một object hoặc một array gán tham chiếu tới object đấy. khi mà thay đổi thì thay đổi cùng một object.**
* let obj1 = { a: 1 };
* let obj2 = { a: 1 };
* console.log(obj1 === obj2);

let obj3 = obj2;

obj3.a = 2;

console.log(obj2 === obj3);

**Bài 20: spread Part 2.**

**Const obj1 ={ a: 1, b: 2, c:3 };**

**Cách 1:**

**Let obj2 = {…obj1};**

**Cách 2:**

**Let obj2 = Object.assign({}, obj1);**

**Đối với object lồng nhau tạo object mới từ object**

**Let obj2 = JSON.parse(JSON.stringify(a));**

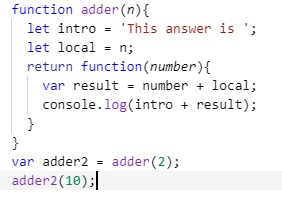
**Bài 21: Closure.**

**Hàm closure có thể truy cập biến số 3 phạm vi khác nhau là:**

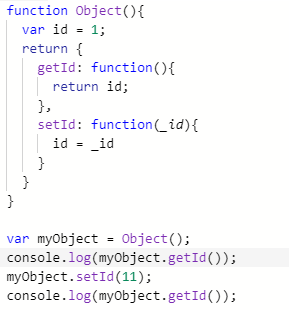
* **Biến toàn cục (global).**
* **Biến được khái báo ở hàm số chứa hàm closures(outer function).**
* **Biến ở bên trong closures.**



**Hàm closure có thể truy cập tới biến của hàm chứa nó, dù cho hàm đó đã return.**

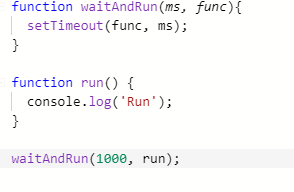


**Hàm closures lưu biến số của outer function theo kiểu tham chiếu**



**Bài 22: Higer order functions**

**Hàm mà có thể nhận một function làm tham số, hoặc trả về một function ( return ).**



**Bài 23: Destructuring**

**Rỡ bỏ cấu trúc. Có thể áp dụng cho array hoặc object**

const arr = [1, 2, 3, 4, 5, 6];

const [a, b] = arr;

const [a, b, c, d, e] = arr;

// a = 1; b = 2,...

const [a,,,d] = arr;

//mỗi dấu , bỏ qua một số.

const [a, ...d] = arr;

//...d 2 trở đi

const [a, b = 20] = arr;

console.log(b);

**Đối vớ Object:**

