1. IIFE trong Javascript.

Khai báo một hàm và thực thi ngay khi khởi tạo, như bên dưới khi chương trình start lên thì dòng console.log sẽ ghi ra tên là Tim Nguyen trên trình duyệt.

(*function* (*name*) {

*console*.log(`Hello ${name}`);

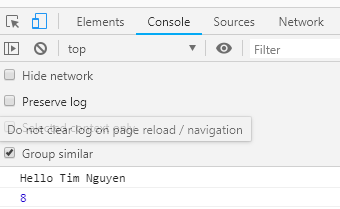
})('Tim Nguyen');

(*function* (*a*, *b*) {

*console*.log(a + b);

})(3, 5);

// Khi chạy lên thì kết quả là 8 sẽ được in ra màn hình console ngay lập tức.



1. Để cập nhật lại các dependencies thì ta gõ lệnh node update trong thư mục ta cần cập nhật.

Sau khi thành công sẽ có một thư mục là node\_modules chứa các dependencies mà được khai báo trong file package.json.

1. Getter và setter trong ES6.

Dấu \_ chỉ là một conventional, nó không ép buộc ta phải tuân theo.

1. *class* Vehicle {
2. *constructor*(*id*, *name*, *weight*) {
3. this.\_id = id;
4. this.\_name = name;
5. this.\_weight = weight;
6. }
7. //getter
8. *get* id() {
9. return this.\_id;
10. }
11. *get* name() {
12. return this.\_name;
13. }
14. *get* weight() {
15. return this.\_weight;
16. }
17. //setter
18. *set* id(*value*) {
19. if (value == 3333)
20. *console*.log("invalid id");
21. else
22. this.\_id = value;
23. }
24. *set* name(*value*) {
25. this.\_name = value;
26. }
27. *set* weight(*value*) {
28. this.\_weight = value;
29. }
30. }
31. *let* vehicle = new Vehicle(1, "truck", 333);
32. vehicle.id = 3333;
33. *console*.log(vehicle.id);
34. *console*.log(vehicle.name);
35. *console*.log(vehicle.weight);

4. Khởi tạo đối tượng cha từ đối tượng con.

// Khởi tạo đối tượng cha từ đối tượng con.

*class* Vehicle {

*constructor*(*licenseNum*) {

this.licenseNum = licenseNum;

}

}

*class* Drone extends *Vehicle* {

}

*class* Car extends *Vehicle* {

*constructor*(*licenseNum*) {

super(licenseNum);

}

}

*let* car = new Car("123");

*console*.log(car.licenseNum);

5. Kết thừa các properties.

// Khởi tạo đối tượng cha từ đối tượng con.

*class* Vehicle {

*constructor*(*licenseNum*) {

this.gpsEnable = true;

}

}

*class* Drone extends *Vehicle* {

}

*class* Car extends *Vehicle* {

*constructor*() {

super();

// Override lại property gpsEnable của base class.

// Một điều lưu ý là ta thực hiện việc override này phải sau việc gọi constructor của cha

// Tức là sau super(), nếu ta làm ngược lại sẽ bị lỗi.

this.gpsEnable = false;

}

}

*let* car = new Car();

// Ta có thể truy cập tới gpsEnable vì nó thuộc thằng cha mà car là kế thừa từ nó.

*console*.log(car.gpsEnable);

6. Việc kế thừa các methods.

*class* Vehicle {

start() {

*console*.log("staring vehicle");

}

}

*class* Drone extends *Vehicle* {

}

*class* Car extends *Vehicle* {

start() {

*console*.log("staring car");

}

}

// Nếu một method ở lớp cha không có ở lớp con thì thằng con sẽ lấy của thằng cha ra sài.

// Nếu một method đã có ở lớp cha mà thằng con cũng có luôn thì method ở thằng con sẽ overrde

// của thằng cha.

*let* c = new Car();

c.start();

*class* Vehicle {

start() {

*console*.log("staring vehicle");

}

}

*class* Drone extends *Vehicle* {

}

*class* Car extends *Vehicle* {

start() {

super.start();

*console*.log("staring car");

}

}

*let* c = new Car();

c.start();

// Kết quả ta sẽ có cả kết của hai methods start ở cả thằng con và thằng cha.

*class* Vehicle {

start() {

*console*.log("staring vehicle");

}

}

*class* Drone extends *Vehicle* {

}

*class* Car extends *Vehicle* {

start() {

super.start();

// Việc gọi đến super như trên không bắt buộc phải đặt vị trí ở đâu.

// nếu ta muốn gọi super ở trên hay ở dưới lời gọi của lớp con thì đặt theo cách ta mong muốn.

*console*.log("staring car");

}

}

*let* c = new Car();

c.start();

// Kết quả ta sẽ có cả kết của hai methods start ở cả thằng con và thằng cha.

// Demo việc sử dụng static method ở lớp con và lớp cha

*class* Vehicle {

start() {

*console*.log("staring vehicle");

}

static getCompanyName() {

*console*.log("My Company");

}

}

*class* Drone extends *Vehicle* {}

*class* Car extends *Vehicle* {

start() {

super.start();

*console*.log("staring car");

}

static getCompanyName() {

// Dòng lệnh bên dưới không phụ thuộc vào vị trí ta có thể đặt bất cư nơi đầu mà ta muốn trong methoid này

super.getCompanyName();

*console*.log("My other Company");

}

}

*let* c = new Car();

// Cũng tưng tự như method khác.

// Khi ta override một hàm ở con giống thằng cha thì thằng con sẽ lấy của.

Car.getCompanyName();

Để tách các lớp ra thành file riêng rẽ ta dùng từ khóa export và import.

// Ở file car.js

import {

Vehicle

} from '../classes/vehicle.js';

export *class* Car extends *Vehicle* {

start() {

super.start();

*console*.log("staring car");

}

static getCompanyName() {

// Dòng lệnh bên dưới không phụ thuộc vào vị trí ta có thể đặt bất cư nơi đầu mà ta muốn trong methoid này

super.getCompanyName();

*console*.log("My other Company");

}

}

// Ở file vehicle.js

export *class* Vehicle {

start() {

*console*.log("staring vehicle");

}

static getCompanyName() {

*console*.log("My Company");

}

}

// Ở file app.js

import {

Car

} from './classes/car.js';

import {

Drone

} from './classes/drone.js';

*let* c = new Car();

Car.getCompanyName();