**Module**

* Là một đoạn mã nguồn tồn tại độc lập có thể tải sử dụng được
* Khi tạo ra module thì ta dùng module.exports.
* Khi gọi module ta dùng require.

**Hàm trong javascript.**

Ngôn ngữ lập trình được coi là first-class function khi mà hàm trong ngôn ngữ dduso được sử dụng như một object.

Trong javascript object chính là function.

Trong js là first-class function

* Có thể đóng vai trò làm tham số của hàm.
* Có thể đóng vai trò là giá trị trả về từ một hàm
* Có thể gàn cho một biến.
* *function* f1(){
* *console*.log("f1");
* }
* *function* PassFunctionAsParams(*f*){
* f();
* *var* f2 = *function*(){
* *console*.log("f2");
* }
* return f2;
* }
* PassFunctionAsParams(f1);

Ví dụ

*function* sayHello(){

*console*.log("Hello World!");

}

sayHello();

//first-class function

*function* logSaying(*fn*) {

//Gọi một hàm khác, lưu ý định nghĩa gọi tới

//một hàm khác phải cùng tên với đối số khai báo bên trên

//cụ thể ở đây là fn

fn();

}

logSaying(sayHello);

//function expression

*var* sayGoodbye = *function*(){

*console*.log("Good bye!");

}

sayGoodbye();

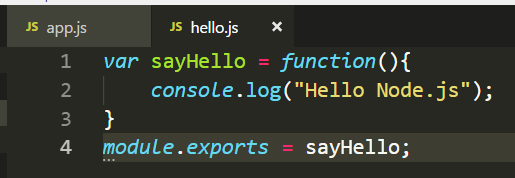
//Tryền hàm như một biểu thức.

logSaying(*function*(){

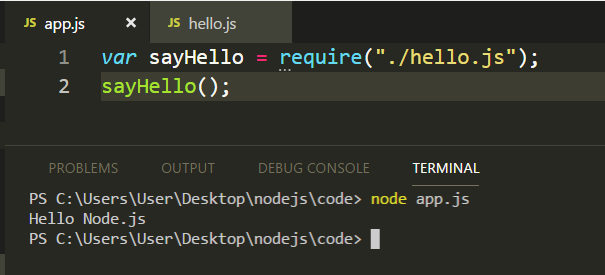
*console*.log("Hello Node.js !! This is expression");

});

**Tạo một module trong node.js**



//Ta không cần phải khai báo phần mở rộng .js, có thể gọi require(“./hello”)



**Các đối tượng Objects và mẫu khai báo Object Literal.**

Name/value

* Là cặp tên ánh xạ với một giá trị
* Trong cùng môt ngữ cảnh thì name có thể được định nghĩa nhiều hơn 1 lần, nhưng chỉ có một giá trị value
* Value thì có thể có nhiều cặp name/ value khác.

Object

* Là một tập hợp các cặp Name/Value
* Là cách định nghĩa đơn giản trong javascript.
* Các kiểu giá trị value có thể mang:
  + Primitive: string, Number…
  + Object
  + Function.

Object Literal

* Là cách khai báo nhanh đối tượng trong javascript.
* Gồm các gặp Name/ Value được ngăn cách nhau bởi dấu phải trong cặp ngoặc nhọn.
* {
* name: 'John',
* Age: 18,
* Address{
* City: 'New York',
* Country: 'US'
* }
* }

Demo

*var* person ={

firstName: "Hoa",

lastName: "Mai",

sayHello: *function*(){

*console*.log("Hello, "+this.firstName+" "+this.lastName);

}

}

person.sayHello();

//Ta cũng có thể truy xuất trực tiếp

*console*.log(person.firstName);

//Cũng có thể truy xuất thông tua Name

*console*.log(person["firstName"]);

**Hàm khởi tạo và kế thừa Prototype**

Sự kế thừa Prototype:

**Obj ==== > prop1**

**Obj ==== > prototype{} ==== > prop2**

**Obj ==== > prototype{} ==== > prototype{} ==== > prop3**

* **Obj.prop1, Obj.prop2, Obj.prop3 (đây gọi là prototype chain).**

Là việc môt Object truy cập đến các thuộc tính và phương thức của một object khác.

* Hơi khác với các ngôn ngữ lập trình khác, Javascript đưa thêm vào khái niệm kế thừa prototype.

Hàm khởi tạo:

* Được sử dụng để khởi tạo đối tượng.
* Sử dụng từ khóa this để truy cập đến các biến của đối tượng hoặc khởi tạo mới.

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

this.name = name;

this.password = password;

}

//Mở rộng đối tượng Person. Bằng cách thêm property getName

*Person*.prototype.getName = *function*(){

return this.name;

}

//Mở rộng đối tượng Person. Bằng cách thêm property getPassword

*Person*.prototype.getPassword = *function*(){

return this.password;

}

//Mở rộng đối tượng Person. Bằng cách thêm property getrLevel

*Person*.prototype.getLevel = *function*(){

return this.level;

}

*Person*.prototype.level = "admin";

*function* User(*name*){

this.name = name;

}

//Tạo một đối tượng User kế thừa thừa từ Person, User kế thừa toàn bộ các phương thức của Person.

*User*.prototype = new Person();

//Person có tên là Ti và Password là 3938

*var* Person = new Person("Ti","3938");

//User có tên là Teo và Password là 339393

*var* User = new User("Teo","339393");

//User kế thừa được các phương thức+ getName, getPassword, getLevel

//và field giá trị là level

//Các giá trị name, passworl thì User không được kế thừa vì nó được định nghĩa

//trong hàm khởi tạo của Person.

*console*.log(User.getName()); // Kết quả là Teo

*console*.log(User.getLevel()); // Kết quả là Admin

*console*.log(User.getPassword()); // Kết quả là undefied

*console*.log(Person.getName()); // Kết quả là Ti

//Mở rộng đối tượng Date trong javascript.

*Date*.prototype.vnFormat = *function*(){

*var* yyyy = this .getFullYear();

*var* mm = this.getMonth()+1; //Kiểu ngày tháng trong javascript tính từ 0.

*var* dd = this.getDate();

return dd+"/"+mm+"/"+yyyy;

}

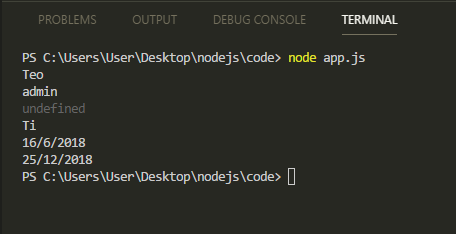
*var* now = new Date();

*var* xms = new Date(2018,11, 25);

*console*.log(now.vnFormat());

*console*.log(xms.vnFormat());

Ta được kết quả như sau:



**Biến tham trị và biến tham chiếu.**

Demo:

*function* changeByValue(*b*){

b = 2;

}

*var* a = 1;

changeByValue(a);

*console*.log(a);

*function* ChangeByRef(*obj*){

obj.prop1 = *function*(){

//

};

obj.newProp2 = {};

obj.d = 100;

}

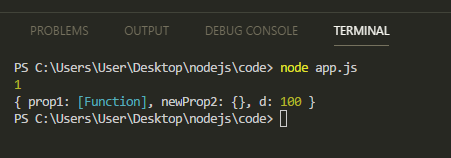
*var* c = {};

c.prop1 ={};

ChangeByRef(c);

*console*.log(c);

Kết quả.



**Thực thi hàm ngay khi được khởi tạo {scope}.**

Scope là phạm vi mà đoạn mã chương trình có thể truy cập đến biến hoặc function.

*function*(){} // Lúc thực thi sẽ báo lỗi.

(*function*(){}) // Lúc thực thi không báo lỗi.

(*function*(){

*var* firstName = "Hoa";

*console*.log(firstName);

}()) //truyền vào () để thực thi ngay lúc nó khởi tạo, vì nó là hàm không tên

//nên phải thực thi ngay lúc khởi t

*var* firstName = "Mai";

*console*.log(firstName);

//Đoạn chương trình xuất ra kết quả là Hoa rồi tới Mai.

*var* firstName = "Mai";

(*function*(){}) // Lúc thực thi không báo lỗi.

(*function*(){

*var* firstName = "Hoa"; //khi ta khai báo ở đây ta tạo ra một phạm vi truy cập mới.

*console*.log(firstName); //và bên ngoài không thể sử dụng hàm này, vì không thể

//truy cập vào các biến bên trong nó.

}())

//truyền vào () để thực thi ngay lúc nó khởi tạo, vì nó là hàm không tên

//nên phải thực thi ngay lúc khởi tạo

*console*.log(firstName);

//Đoạn chương trình trên vẫn xuất ra kết quả là Hoa rồi tới Mai.

*var* firstName = "Mai";

(*function*(*lastName*){

*var* firstName = "Hoa";

*console*.log(firstName);

*console*.log(lastName);

}("Phương"))// "Phượng" ở đây là giá trị truyền vào cho hàm, chính là lastName

//truyền vào () để thực thi ngay lúc nó khởi tạo, vì nó là hàm không tên

//nên phải thực thi ngay lúc khởi tạo

*console*.log(firstName);

//Đoạn chương trình trên vẫn xuất ra kết quả là Hoa rồi tới Mai.

//Đây là phong cách thiết kế bên trong các module của node js

**Thao tác với Json**

Cấu trúc chuỗi JSON

Chuỗi JSON được bọc lại bởi cặp dấu{}

Các key, value của JSON bắt buộc đặt trong dấu nháy kép.

Dùng dấu sổ dọc \ nếu giá trị chứa dấu nháy kép. Ví dụ: \”json”\

Nếu có nhiều cặp key, value trong chuỗi json thì dùng dấu phẩy để năng cách.

{

"firstName": "Hoa",

"lastName": "Mai",

"address": {

"street": "123 abc",

"city": "Hanoi",

"country": "Vietnam"

}

}

{

"sv001": {

"id\_mon\_toan": "Mon Toan",

"id\_mon\_ly": "Mon Ly"

},

"sv002": {

"id\_mon\_toan": "Mon Toan",

"id\_mon\_anh": "Mon Tieng Anh"

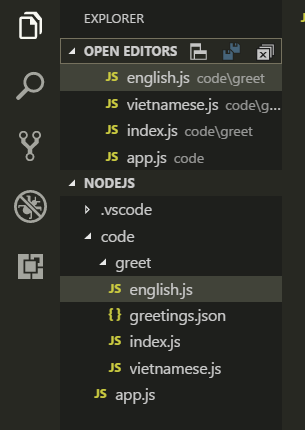
}

}

**Tìm hiểu thêm về module và require**

**Demo**

Tạo cấu trúc files và thư mục như sau:



Lần lượt tạo các file như sau:

**Greetings.json**

{

*"en"*: "Hello",

*"vi"*: "Xin chao"

}

**vietnamese.js**

*var* greetings = require("./greetings.json");

*var* sayHello = *function*(){

*console*.log(greetings.vi);

}

*module*.*exports* = sayHello;

**english.js**

*var* greeting = require("./greetings.json");

*var* sayHello = *function*(){

*console*.log(greeting.en);

}

*module*.*exports* = sayHello;

**index.js**

*var* english = require("./english");

*var* vietnamese = require("./vietnamese");

//Cho bên ngoài sử dụng english và vietnamese phía trên.

*module*.*exports* ={

english: english,

vietnamese: vietnamese

**app.js**

*var* greet = require("./greet");

greet.english();

greet.vietnamese();

**Các kỹ thuật khai báo và sử dụng module**