Notes de cours NodeJS

**Intro**

Le JavaScript est un langage interprété, tout comme le Java ( qui est compilé ET interprété ). cela lui permet d'être très modulable, étant donné que l'exécution ne plantera que lorsque l'exécution atteint une ligne erroné ( par opposition au C qui compile toutes les lignes au préalable). Le JS est orienté objet ( présence de classes) et non typé.

**Les fonctions**

Pas de type de retour, et on ne connait pas les types des paramètres. en dehors de ça la déclaration se fait avec le mot clé function. On peut également stocker des fonction dans des varaibles :

Function test() {

console.log(“Hello”);

}

var f = test;

f();// Hello

en cas de noms similaire, les variables sont prioritaires.

il est possible de passer une variable de type fonction à une fonction :

function exec(f){

f();

}

exec(function(){

console.log(“Hello ESGI”);

});

il existe de nombreux outils de manipulation de variable. par exemple

var a=”hello”;

var b=a.charAt(0);

var c=a.lenght;// ceci est un propriété, il n’y a donc pas de parenthèses.

b vaut ‘h’.

c vaut 5.

il existe de nombreux outils pour les tableau également, la liste complète est disponible sur w3c.

**Créer un projet**

Dans une console, aller dans le dossier voulue pour la création, et taper : npm init. suivez l’invité de commande en renseignat tout les champs voulue. npm vous creer alors un fichier de configuration de votre projet qui regroupe toutes les dépendances. Une fois ce fichier créé, il suffit de creer un fichier .js qui porte le nom du fichier d’entrée renseigné dans la console npm. le code se trouvera donc dans le fichier index.js ( si c’est ce que vous avez entrer dans la console de npm) et la configuration dans le fichier .json.

**Premier programme**

function say(val,fonction){// déclare une fonction. deux parametre : une variable de

//type string, et une variable de type fonction

 var a = fonction || console.log// la variable a contient la fonction 'fonction' sauf si elle

//contient NULL, dans ce cas la variable a vaudra

//'console.log'

 a(val);// on execute la fonction contenue dans 'a' avec la variable val passé en

//parametre

}

say("Hello",function(value){console.log("\*\*\*\*"+value+"\*\*\*\*")});//appel la fonction say,

//avec en parametre le

//string "Hello" et

                                                              // la fonction que l'ont

//déclare ici meme et

//qui apelle console.log

                                                              //avec des "\*" en plus.

**Les tableaux**

var s="azertyuiop";

say(s);

console.log("length : " + s.length);

console.log("bracket[2] : " + s[2]);

console.log("charAt(5) : "+s.charAt(5));

var arr = new Array();

arr2=new Array();

var arr3=["al1",'al2'];

arr4=[];

console.log("display arr3 : " + arr3);

console.log("length arr3 : "+arr3.length);

console.log("boucle for : ");

for(var i =0;i<arr3.length;i++){

 console.log(arr3[i]);

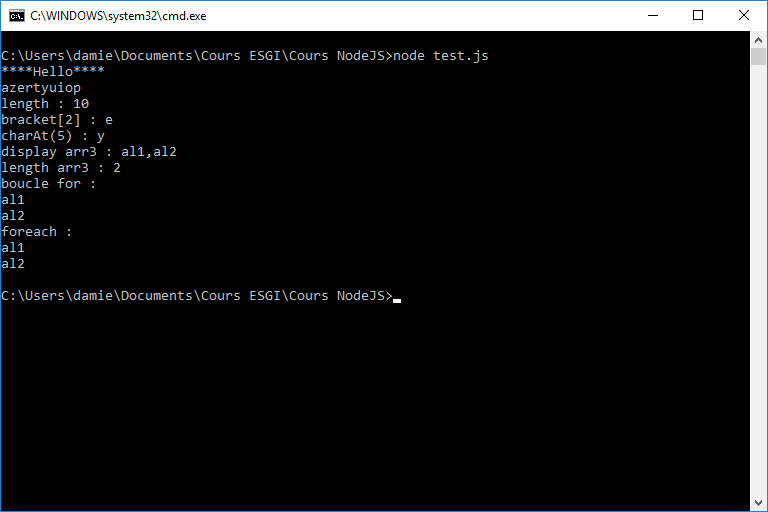
}

console.log("foreach : ");

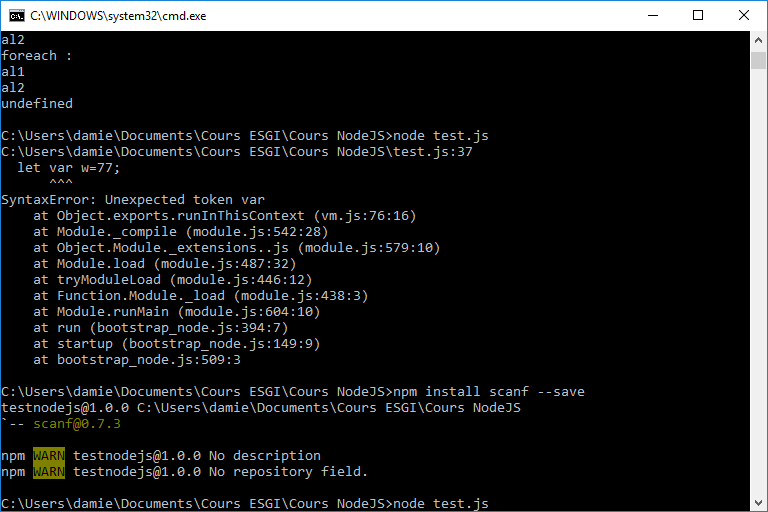
for(var s of arr3){

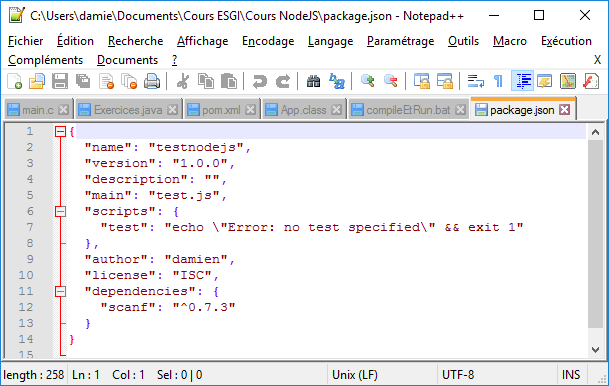
 console.log(s);

}



**Ajouter un module**

dans npm, taper “npm install scanf” pour que npm télécharge le module scanf. ajouter --save pour que npm modifie le fichier de config .json pour y rajouter la dépendance. si vous faites ça, il suffira au autres dev de taper npm install pour qu’il récupère toutes les dépendance sauvegarder dans ce fichier de config json.



Une fois le module ajouter, il faut le mettre dans une variable :

const ESGI = require("scanf");

pour exploiter le scanf, il faudra alors apeler la variable ESGI :

const ESGI = require("scanf");

messageUser=ESGI('%s');

console.log("\*\*\*"+messageUser+"\*\*\*");

**ajouter des mot clé**

il est possible de rajouter des mot clé en les définissant en haut du code comme par exemple l’ajout du mot clé ‘let’ qui empêche le compilateur de redéfinir les variable ne haut de leur porté :"use strict";

function leaveIt(){

 console.log(w);

 let var w=77;//plante

}

leaveIt();

function leaveIt(){

 console.log(w);

 var w=77;//undifined

}

leaveIt();

function leaveIt(){

 var w=77;

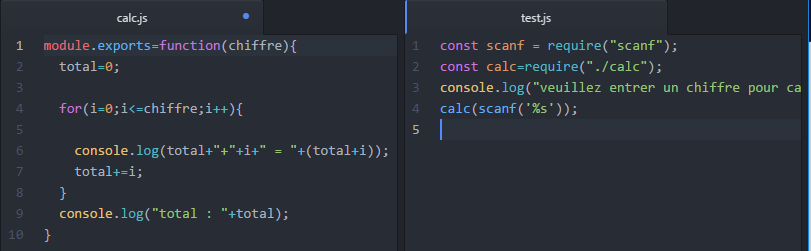
 console.log(w);//fonctionne

}

leaveIt();

**Programmation modulaire**

il est possible de faire appel à des fonction présente dans d’autre fichier. pour inclure un autre fichier dans le code, cela fonctionne comme pour un module : il faut mettre dans une variable le fichier cible :



**Création d’object inline**

Déclaration entre accolade dans une variable

Similaire à un dictionnaire clé-valeur

Intégration de fonction

|  |
| --- |
| var myObject = {  x : 0,  y : 0,  add : function() {  return this.x + this.y;  }  }  myObject.x = 10;  myObject.y = 50;  alert(myObject.add()); |

**Création de classes**

Exemple 1 :

|  |
| --- |
| var Person = function(lastName, firstName){  this.lastName = lastName;  this.firstName = firstName;  }  p1 = new Person("Jean Luc");  alert(p1.lastName); // Jean Luc  alert(p1.firstName); //undefined  p2 = new Person("Jean", "Luc");  alert(p1.lastName); // Jean  alert(p1.firstName); // Luc |

Exemple 2 :

|  |
| --- |
| var Person = function(lastName, firstName){  this.lastName = lastName;  this.firstName = firstName;  this.display = function() {  alert(this.lastName + " " + this.firstName);  }  }    p1 = new Person("Briatte","Benoit");  p2 = new Person("Briatte","Benoit");  alert (p1.display == p2.display); // affiche true |

Exemple 3 :

|  |
| --- |
| var Person = function(lastName, firstName){  this.lastName = lastName;  this.firstName = firstName;  }  Person.prototype.display = function() {  alert(this.lastName + " " + this.firstName);  }    p1 = new Person("Briatte","Benoit");  p2 = new Person("Briatte","Benoit");  alert (p1.display == p2.display); // affiche true |

Les prototypes et l’héritage :

il n’y a pas de “extends”

pour faire de l’heritage il faut créer un constructeur  et faire une fonction

|  |
| --- |
| var Person = function(lastName, firstName){     this.lastName = lastName;     this.firstName = firstName;  };  Person.prototype.hello = function(){     alert("Bonjour");  };  var Employee = function(){     Person.apply(this, arguments); //eq. to super  };  fextends(Employee, Person); //avant modification du prototye d’Employee  Employee.prototype.work = function(){     alert("Je travaille (ou pas)");  };  var Manager = function() {     Employee.apply(this, arguments); //eq. to super  };  fextends(Manager, Employee); // avant modification du Manager |