# TP1 : Initiation à Node.js

**Objectifs**
Apprentissage de Javascript côté serveur
**Section**
R5_10 : Département Informatique
IUT CAEN Campus 3
**Auteur**
E.Porcq
**Date**
04/09/2024 durée 3h00

http://nodejs.developpez.com/tutoriels/javascript/node-js-livre-debutant/
https://openclassrooms.com/courses/des-applications-ultra-rapides-avec-node-js

Node.js permet d'exécuter du javascript côté serveur. L'objectif de ce TP est d'apprendre les bases de Node et de javascript avant de développer des applications plus complexes .
Dans le TD, les fichiers XXX sont fournis et les fichiers YYY sont à créer

## 1 Introduction à node.js

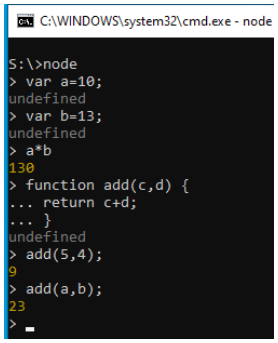
### 1.1 Installation

- Télécharger et installer node.js si nécessaire. Tester l'installation.
  - `c:\>node -v`
    `v18.17.0`
- Créer un lecteur pour réduire la longueur du chemin
  - `subst s: /D`
  - `Z:\fait_ici\M4101C_2_TP2>subst s: .`
  - `Z:\fait_ici\M4101C_2_TP2>s:`
  - `S:\>`
- Lancer node
- Réaliser le petit programme suivant et le tester (^D pour sortir)

```
console.log("Afficher les nombres de 1 à 10");
for (var i=1;i<=10;i++)
  console.log(i);
```

- Copier le code dans p11.js
- Exécuter : `node p11.js`

### 1.2 Utilisation de l'interprète de commande javascript (REPL)




L'Interprète fonctionne selon une boucle d'évaluation, appelée REPL, pour read, eval, print, loop. Dans cette boucle, l'interprète :

- R) lit une expression.
- E) évalue (calcule le résultat de) cette expression.
- P) imprime sur la sortie standard le résultat de l'évaluation.
- L) recommence en R).

Tester l'interprète avec quelques commandes
Rq : `console.log("\u001B[2J\u001B[0;0f");` - efface l'écran
`console.clear();` fonctionne aussi

### 1.3 Gestion des modules

Les modules sont des fichiers « Javascript » regroupés dans un répertoire.
On peut inclure un module dans une application avec l'instruction `require(<module>)`.
Dans une application, il est nécessaire de donner le chemin du module (sauf pour les modules situés dans `node_modules`).
D'un point de vue pratique, on peut inclure un module incluant lui-même les modules de son répertoire.
Tester p13.js et commenter les lignes.

### 1.4 Modules avec fonctions (voir node.js module.exports sur le WEB)

Réaliser un module (mod_14.js) contenant une fonction « addition » retournant la somme des deux paramètres qu'elle reçoit.
À partir d'un autre fichier (p14.js), appeler la méthode « addition ». Tester et conclure.

### 1.5 Modules avec fonction principale

Réaliser un module (mod_15.js) contenant une fonction principale appelant les 4 opérations. Le script p15.js appellera la fonction principale et la fonction multiplier.
Pour déclarer une fonction comme principale, la syntaxe est : `module.exports = main;`

```
S:\>node s:\prof\p15.js
les param�tres sont 4 et 5
Addition: 9
soustraction: -1
multiplication: 20
division: 0.8
R�sultat de multiplier : 20
{ [Function: main] multiplier: [Function: multiplier] }
```

### 1.6 Variables globales et locales

Tester précisément les scripts p16.js et mod_16.js et conclure

```
// p16.js
var module1 = require("./mod_16.js");

a=101;
b=102;
//var module1 = require("./mod_16.js"); // placer ici change
module1();
module1();
console.log("p16 : a vaut "+a);
console.log("p16 : b vaut "+b);
//console.log("p16 : c vaut "+c);
console.log("p16 : d vaut "+d);
console.log("p16 : e vaut "+e);

console.log(module1);
```

```
1  //mod p16.js
2  a=11;
3  var b=12;
4  e = 9;
5  var c=105;
6
7  function main()
8  {
9      let c=15;
10     let i=0;
11     var j=0;
12     d = 8;
13     console.log("dans le main a vaut "+a++);
14     console.log("dans le main b vaut "+b++);
15     console.log("dans le main c vaut "+c++);
16     console.log("dans le main d vaut "+d++);
17     console.log("dans le main e vaut "+e++);
18     console.log("------------------------");
19     while (i<10 && j<10)
20     {
21         let i=5;
22         i++;
23         j++;
24         console.log("dans la boucle i vaut "+i);
25         console.log("dans la boucle j vaut "+j);
26     }
27     console.log("------------------------");
28     console.log("dans le main i vaut "+i);
29     console.log("dans le main j vaut "+j);
30     console.log("------------------------");
31 }
32 module.exports = main;
```

### 1.7 tableaux d'octets et chaînes de caractères

- Tester p17.js
- Commenter les résultats

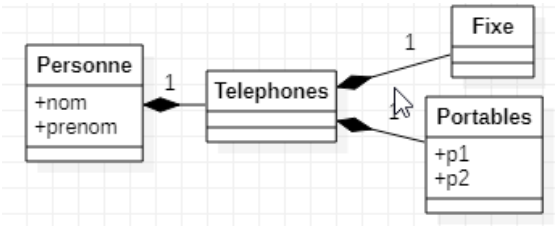
```
S:\prof>node p17.js
tab1 : Bonjour les etudiants
tab1 : Bonjour les etudiants
<Buffer 42 6f 6e 6a 6f 75 72 20 6c 65 73 20 65 74 75 64 69 61 6e 74 73>
<Buffer 42 6f 6e 6a 6f 75 72 20 6c 65 73 20 c3 a9 74 75 64 69 61 6e 74 73>
[String: 'Bonjour les etudiants']
tab3 : [String: 'Bonjour les etudiants']
tab3 : Bonjour les etudiants
tab3 : 21
------------------------------
tab1 : Boujour les etudiants
tab3 : BonXour les etudiants
tab3 : BonXour les etudiants
------------------------------
Buffer(21) [Uint8Array] [
   66, 111, 117, 106, 111, 117,
  114,  32, 108, 101, 115,  32,
  101, 116, 117, 100, 105,  97,
  110, 116, 115
]
Buffer(22) [Uint8Array] [
   66, 111, 110, 106, 111, 117,
  114,  32, 108, 101, 115,  32,
  195, 169, 116, 117, 100, 105,
   97, 110, 116, 115
]
'BonXour les etudiants'
------------------------------
```

### 1.8 Structures

- Déclarer une personne sous forme d'une structure et afficher ses caractéristiques .

```
var personne =
{
```

- En utilisant le module « util » et la méthode « inspect », afficher la personne comme dans la capture ci-contre (utiliser depth) p18_1.js.




```
S:\>node s:\prof\p18_1.js
--------------depth=0-------------------------
{ nom: 'Supormoi', prenom: 'Steven', 'téléphones': [Object] }
--------------depth=1-------------------------
{
  nom: 'Supormoi',
  prenom: 'Steven',
  'téléphones': { fixe: '02 31 48 49 50', portables: [Object] }
}
--------------depth=2-------------------------
{
  nom: 'Supormoi',
  prenom: 'Steven',
  'téléphones': {
    fixe: '02 31 48 49 50',
    portables: { p1: '06 31 48 49 66', p2: '06 31 48 68 66' }
  }
}
--------------depth=2-------------------------
{
  nom: 'Supormoi',
  prenom: 'Steven',
  'téléphones': {
    fixe: '02 31 48 49 50',
    portables: { p1: '06 31 48 49 66', p2: '06 31 48 68 66' }
  }
}
//////////////////////
```

- Compléter ce programme pour créer un monde contenant 2 personnes. p18_2.js.

```
S:\prof>node p18_2.js
--------------depth=0-------------------------------
[ [Object], [Object] ]
--------------depth=1-------------------------------
[
  { nom: 'Supormoi', prenom: 'Steven', 'téléphones': [Object] },
  { nom: 'Jarface', prenom: 'Sylvain', 'téléphone': [Object] }
]
--------------depth=2-------------------------------
[
  {
    nom: 'Supormoi',
    prenom: 'Steven',
    'téléphones': { fixe: '02 31 48 49 50', portables: [Object] }
  },
  {
    nom: 'Jarface',
    prenom: 'Sylvain',
    'téléphone': { fixe: '02 32 36 16 50', portables: [Object] }
  }
]
--------------depth=3-------------------------------
[
  {
    nom: 'Supormoi',
    prenom: 'Steven',
    'téléphones': {
      fixe: '02 31 48 49 50',
      portables: { p1: '06 31 48 49 66', p2: '06 31 48 68 66' }
    }
  },
  {
    nom: 'Jarface',
    prenom: 'Sylvain',
    'téléphone': {
      fixe: '02 32 36 16 50',
      portables: { p1: '06 38 48 49 32', p2: '06 45 48 68 31' }
    }
  }
]
```

### 1.9 Objets et tableaux d'objets

- Pour créer une classe, on peut créer une fonction et y initialiser des attributs qui peuvent eux-mêmes être des objets.
- En partant de l'exemple (" p18_2.js ") , compléter p19_1.js en créant un tableau de 2 personnes avec les fonctions Personne, Telephone et Portable (pas de classe).

```
S:\>node p19_1.js
-------------------------
[ '0', '1' ]
-------------------------
[ 'nom', 'prenom', 'telephone' ]
-------------------------
-------------------------
0
Personne {
  nom: 'Supormoi',
  prenom: 'Steven',
  telephone:
   Telephone {
     fixe: '02 31 48 49 50',
     portable: Portable { p1: '06 31 48 49 66', p2: '06 31 48 68 66' } } }
1
Personne {
  nom: 'Jarface',
  prenom: 'Sylvain',
  telephone:
   Telephone {
     fixe: '02 32 36 16 50',
     portable: Portable { p1: '06 38 48 49 32', p2: '06 45 48 68 31' } } }
-------------------------
02 32 36 16 50
```

- Même question mais en utilisant des classes p19_2.js .

```
S:\>node p19_2b.js
Individu { tel: [Telephone], nom: 'Boudu', prenom: 'Sylvie' }
-----------------------------------------------------------
Individu {
  tel: Telephone { fixe: '02 31 48 49 50', portable: [Portable] },
  nom: 'Boudu',
  prenom: 'Sylvie' }
-----------------------------------------------------------
Individu {
  tel:
   Telephone {
     fixe: '02 31 48 49 50',
     portable: Portable { p1: '06 31 48 49 66', p2: '06 31 48 68 66' } },
  nom: 'Boudu',
  prenom: 'Sylvie' }
-----------------------------------------------------------
Individu {
  tel:
   Telephone {
     fixe: '02 31 48 66 00',
     portable: Portable { p1: '07 31 48 49 66', p2: '06 30 40 60 66' } },
  nom: 'Gorin',
  prenom: 'Bernard' }
-----------------------------------------------------------
```

**peut-on annuler la suite et placer le 2.1 du TP3 ?**

## 2 Node.js avancé

### 2.1 Gestion des événements

- Tester le premier exemple (p21_11.js) et vérifier les informations sur le module events .
- Tester p21_12.js. (Créer une instance d'event.EventEmitter).

Une fois l'objet créé, il peut être utilisé pour gérer les événements qui lui sont adressés. On utilise pour cela les méthodes emit() et addListener().

| | |
|---|---|
| addListener(event, listener) | ajouter à un objet un gestionnaire d'événements pour "event". Lorsque cet événement se produit, la fonction callback "listener" se déclenche |
| defaultMaxListeners | positionne le nombre max d'écouteurs pour un événement (défaut10) |
| on(event, listener) | équivaux à addListener |
| once(event, listener) | idem à on mais le déclenchement ne pourra s'opérer qu'une seule fois |
| emit(event) | déclenche l'événement "event" de l'objet |
| removeListener(event, listener) | supprimer à un objet le gestionnaire d'événement |
| removeAllListener([event]) | supprimer à un objet tous les gestionnaires concernant event. Si event, n'est pas mentionné, tous les gestionnaires de tous les événements sont supprimés |
| setMaxListeners(n) | indiquer le nombre maximal de gestionnaires d'événements pouvant être positionnés sur un objet (10 par défaut). 0 = pas de limite |
| listeners(event) | retourner un tableau contenant les références vers les gestionnaires d'événements associés à event et positionnés pour cet objet |
| eventNames() | retourne un tableau contenant les événements enregistrés |
| getMaxListeners() | retourne le nombre d'écouteurs alloués |
| listenerCount() | retourne le numéro d'écouteur avec son nom |

- Compléter (p21_13.js) montrant le déclenchement d'événements.
  - Utiliser addListener pour créer **des** fonctions interceptant des événements
  - Déclencher l'événement avec emit
  - Résultat

```
S:\>node p21_13.js
Détection 1 de l'événement
Détection 2 de l'événement
Détection 3 de l'événement
Détection 3 de l'événement
---------------
EventEmitter {
  _events: [Object: null prototype] {
    eve1: [ [Function (anonymous)], [Function: f2], [Function: maFonction3] ],
    eve2: [Function: maFonction3]
  },
  _eventsCount: 2,
  _maxListeners: 4,
  defaultMaxListeners: 9,
  [Symbol(kCapture)]: false
}
---------------
```

- Compléter ce programme (p21_14.js) pour transmettre des paramètres textes au déclenchement.
- Modifier le programme précédent pour transmettre des paramètres objets (p21_15.js).
  - Déclenchement d' "eve1 " en passant l'objet déclenchant en paramètre
  - Dans le traitement d' "eve1 ",
    - affichage dans une fonction A1 d'un message et déclenchement d' "eve2 " en passant un texte en paramètre
    - affichage dans une fonction A2 d'un message
  - Dans le traitement d' "eve2 ", affichage du paramètre

```
S:\>node p21_14.js
Détection 1 Porcq Eric
Détection 2 Porcq Eric
Détection 3 Porcq Eric
[ 'eve1', 'eve2' ]
Détection 3 Porcq Eric
[ 'eve1', 'eve2' ]
```

### 2.2 Fonctions de Callback

https://o7planning.org/fr/11945/le-concept-de-callback-dans-nodejs_

- Tester les programmes p22_1 et p22_2 et en déduire l'intérêt des fonctions de callback