

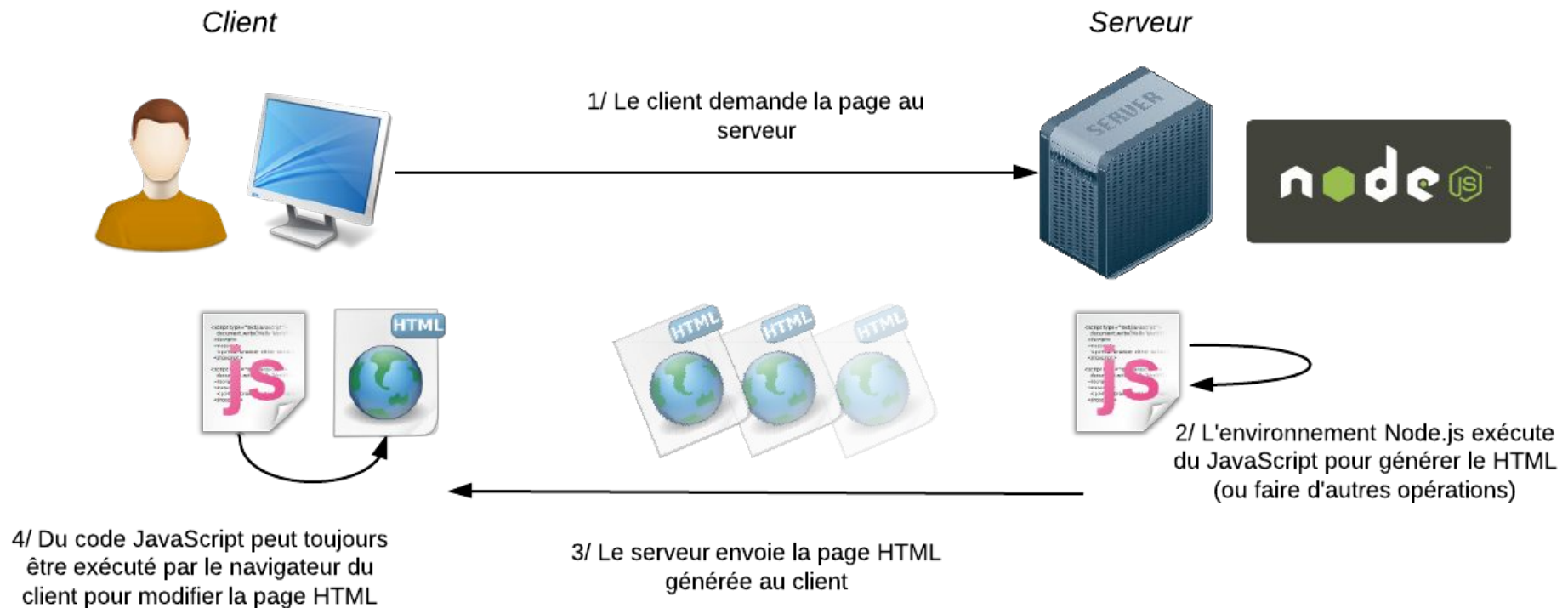
# Introduction

NodeJs

# Introduction

## Node.js : le JavaScript côté serveur

Node.js offre un environnement côté serveur qui nous permet d'utiliser le langage JavaScript pour générer des pages web tels les langages serveur comme ASP, PHP, Java EE, etc.



(crédit : OpenclassRoom)

# Introduction

- Node Js, comme Js est un **langage événementiels**
- Contrairement au php ou au Java, toute la programmation est à penser **orienté événements**
- Node Js est un environnement de **bas niveau** se rapprochant plus du C que du Php
- Node Js n'est **PAS un framework**! Des frameworks existent pour NJS ☐ Express
- Node Js est **Rapide**

# Introduction

## Le moteur V8



V8 JavaScript engine est un moteur JavaScript open source développé par Google au Danemark. Il est notamment utilisé dans les navigateurs Internet Google Chrome et Chromium. Il fonctionne sur les architectures x86 (32 bits et 64 bits) et ARM.

NodeJS utilise ce moteur qui transforme le code JavaScript très rapidement en code machine et l'optimise même grâce à des procédés complexes : code inlining, copy elision

Plus d'infos sur <http://blog.js-republic.com/v8-engine-comment-ca-marche/>

# Introduction

## **Le modèle non bloquant**

Comme JavaScript est un langage conçu autour de la notion d'événement, Node.js a utilisé ce principe pour mettre en place une architecture de code entièrement non bloquante.

Le mode non bloquant consiste tout simplement en l'utilisation de callback pour rappeler des parties de code une fois une instruction terminée plutôt que d'attendre que celle-ci finisse de s'exécuter.

### **Exemple:**

« Je souhaite télécharger un fichier Excel une fois celui-ci généré par le serveur. Ce fichier doit traiter une grosse quantité de données, l'opération va plus que certainement être trèèèèèè longue! »

# Introduction

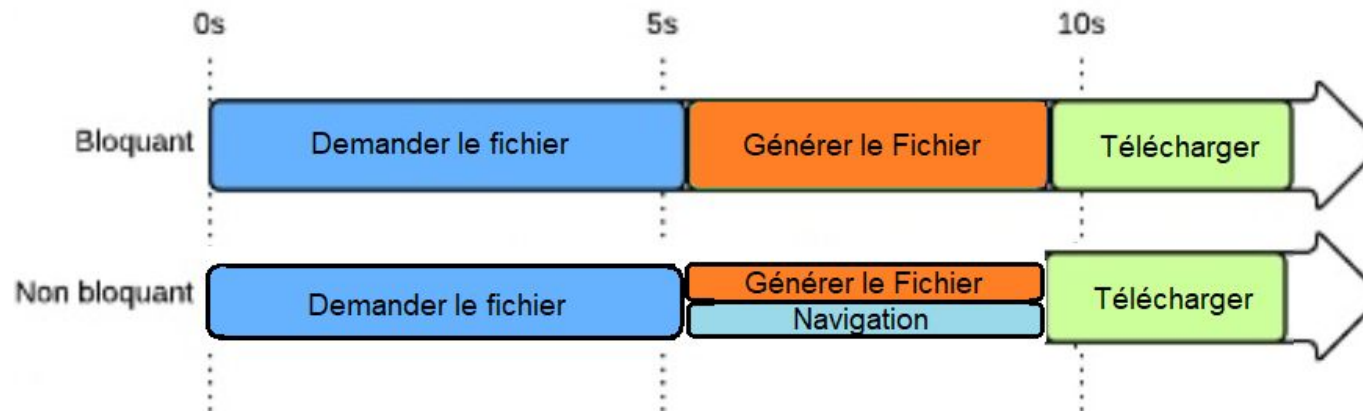
## Modèle bloquant

- J'appelle et je traite ma source de données
- Je génère mon fichier 🕒
- Je télécharge mon fichier
- Je continue ma navigation

## Modèle non-bloquant

- J'appelle et je traite ma source de données
  - Je génère mon fichier
  - Je télécharge mon fichier
- Je continue ma navigation

La navigation continue et les sous-tâches seront effectuées une fois le, les événements parents terminés.



# Installation

NodeJs



# Installation

L'installation est plutôt simple.

Depuis l'accueil du site <https://nodejs.org/>, vous devriez voir un bouton « Download for Windows (x64) » (si vous êtes sur Windows).

Cliquez pour télécharger automatiquement le zip le plus adapté à votre système.

Vous pouvez également choisir vous même votre zip dans [Download](#) pour obtenir au choix le *Windows Installer (.msi)* et/ou le *Windows Binary (.exe)* en 32/64 bit.



Node.js® is a JavaScript runtime built on [Chrome's V8 JavaScript engine](#).

## Download for Windows (x64)

**12.16.2 LTS**

Recommended For Most Users

**13.12.0 Current**

Latest Features

[Other Downloads](#) | [Changelog](#) | [API Docs](#)

[Other Downloads](#) | [Changelog](#) | [API Docs](#)

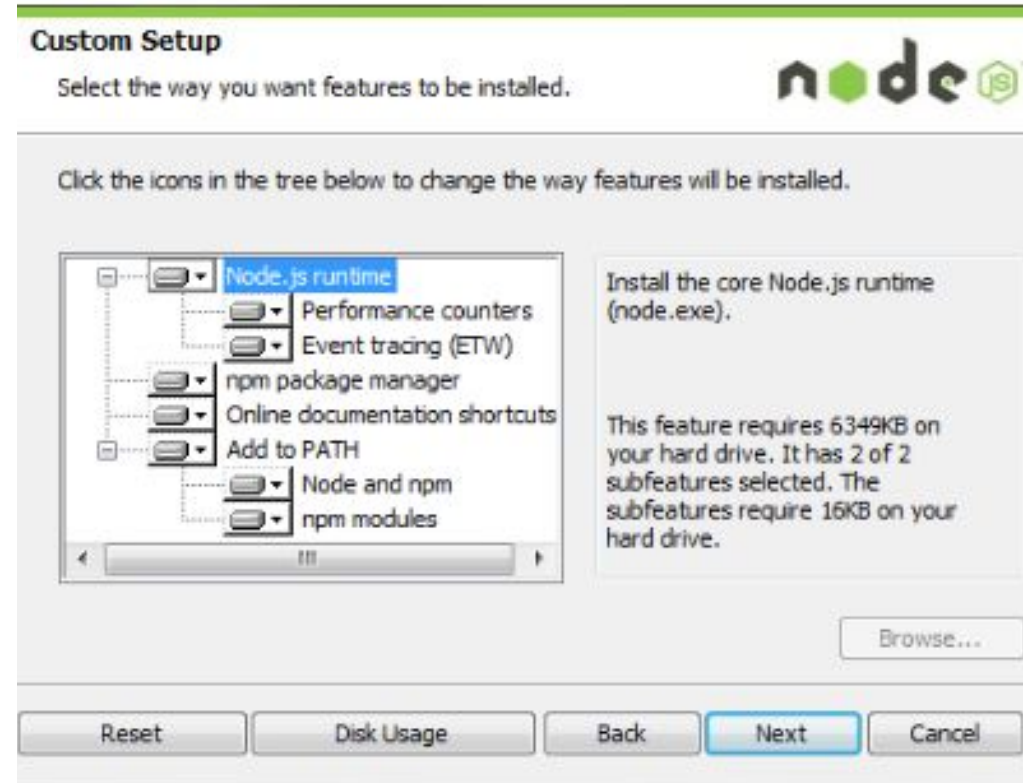
Or have a look at the [Long Term Support \(LTS\) schedule](#).

Sign up for [Node.js Everywhere](#), the official Node.js Monthly Newsletter.

# Installation

Exécutez votre fichier, acceptez la licence, choisissez le dossier "Program files" et ce package va vous installer :

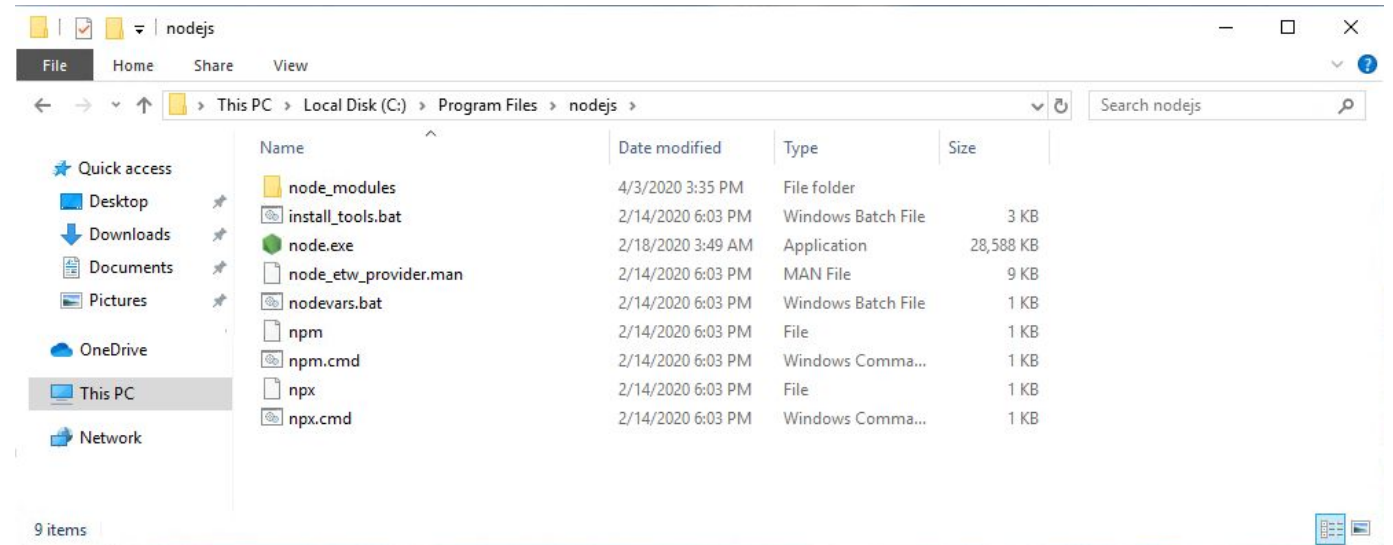
- **L'exécuteur node.js :**  
le programme permettant d'exécuter des fichiers .js (comme php.exe le ferait avec des .php).
- **Le module npm (Node Package Manager) :**  
un gestionnaire de modules qui va vous permettre simplement d'ajouter et retirer les librairies dont vous aurez besoin pour vos applications (pas de surplus, seulement le nécessaire donc).
- **Un raccourci vers la documentation en ligne.**
- **Des variables d'environnements :**  
Ainsi vous pourrez exécuter les commandes node et npm dans votre invité de commande.



# Installation

Quand l'installateur aura fini vous aurez un dossier d'environ 12Mo contenant entre autre node.js et npm.cmd

Ces éléments seront « appelables » depuis n'importe quel dossier avec les commandes *node* et *npm*.



# REPL

NodeJS

# REPL

NodeJs vient avec un environnement virtuel appelé

*Read*

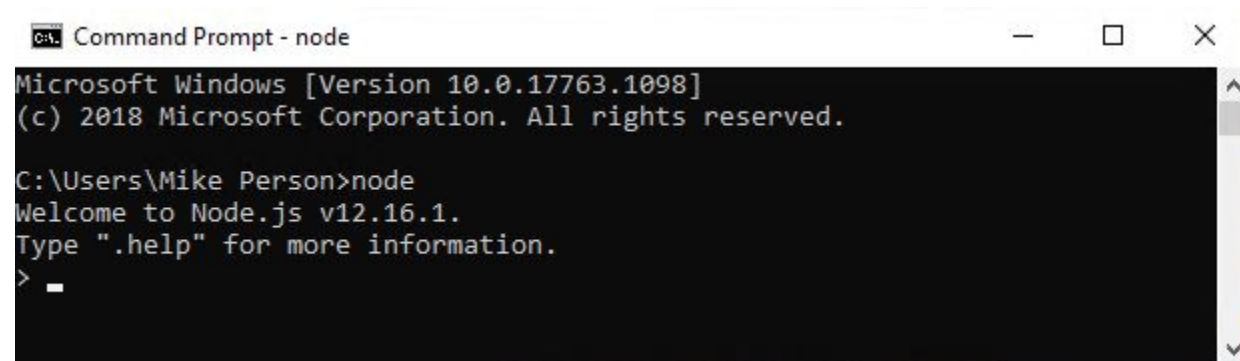
*Eval*

*Print*

*Loop.*

C'est la façon la plus simple et rapide de tester du NodeJS.

Pour lancer l'environnement, il suffit de se rendre dans une console (command prompt sous Windows ou Terminal sous Linux) et ensuite taper node.



```
C:\> Command Prompt - node
Microsoft Windows [Version 10.0.17763.1098]
(c) 2018 Microsoft Corporation. All rights reserved.

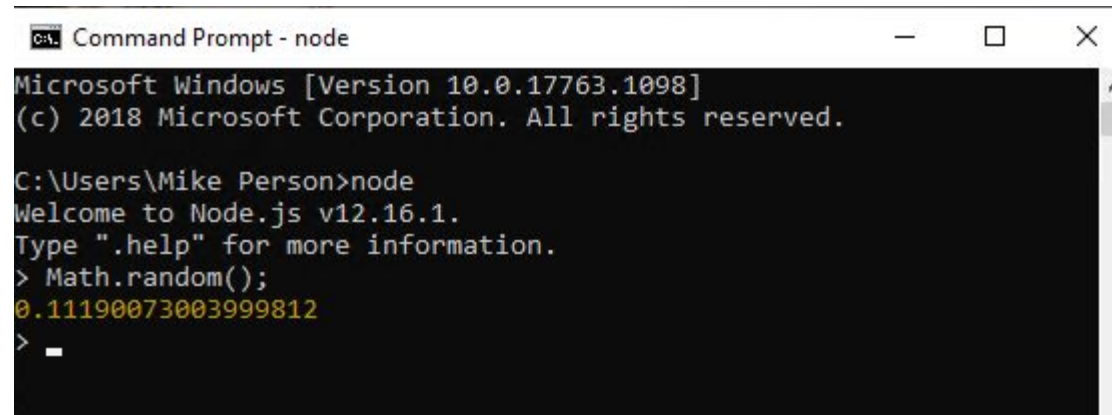
C:\Users\Mike Person>node
Welcome to Node.js v12.16.1.
Type ".help" for more information.
> _
```

# REPL

Commandes	Description
.help	Affiche la page d'aide des commandes
tab Keys	Affiche la liste des commandes
Up/Down Keys	Permet de remonter à la commande précédente ou aller vers la suivante
.save filename	Sauvegarde la session actuelle dans un fichier
.load filename	Charge un fichier contenant une sessions REPL
ctrl + c	Permet de terminer la session
ctrl + c (twice)	Permet de quitter l'environnement
ctrl + d	Permet de quitter l'environnement
.break	Permet de sortir d'une expression multi-ligne
.clear	Permet de sortir d'une expression multi-ligne

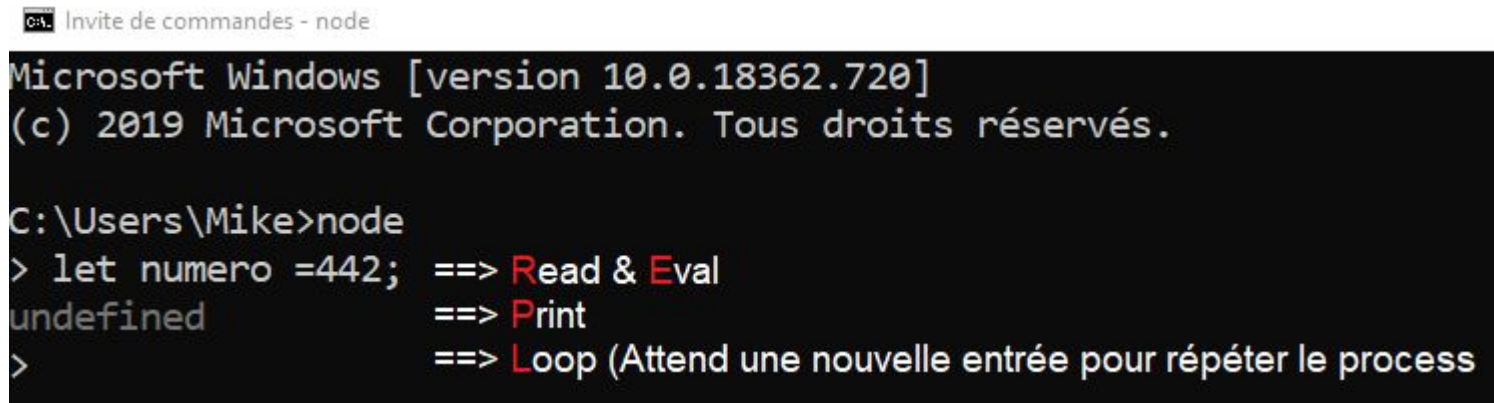
# REPL

Grâce à cet outils, vous pouvez écrire du javascript et tester facilement celui-ci.



```
Command Prompt - node
Microsoft Windows [Version 10.0.17763.1098]
(c) 2018 Microsoft Corporation. All rights reserved.

C:\Users\Mike Person>node
Welcome to Node.js v12.16.1.
Type ".help" for more information.
> Math.random();
0.11190073003999812
> _
```

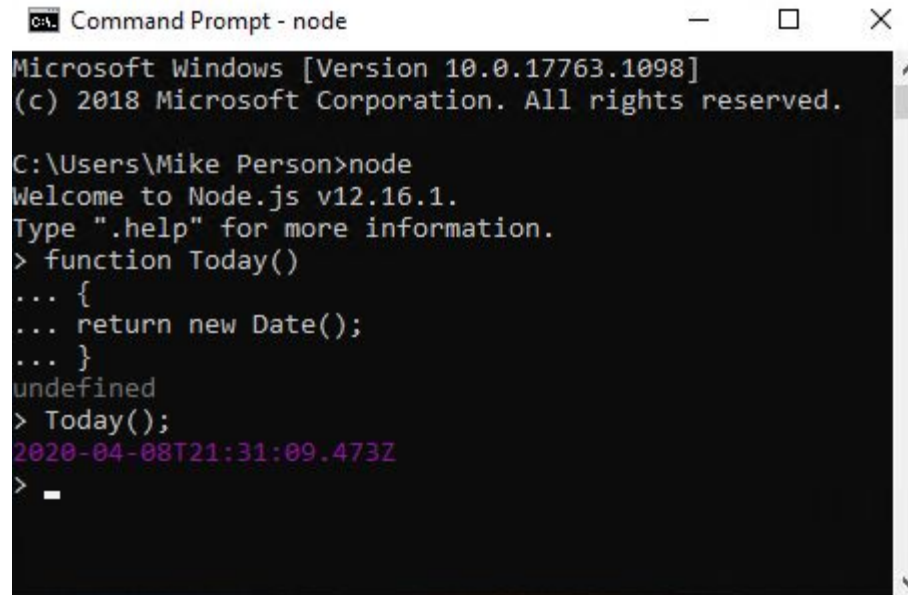


```
Invite de commandes - node
Microsoft Windows [version 10.0.18362.720]
(c) 2019 Microsoft Corporation. Tous droits réservés.

C:\Users\Mike>node
> let numero =442; ==> Read & Eval
undefined ==> Print
> ==> Loop (Attend une nouvelle entrée pour répéter le process
```

# REPL

L'outil est capable de détecter si nous sommes dans une instruction multilignes



```
Command Prompt - node
Microsoft Windows [Version 10.0.17763.1098]
(c) 2018 Microsoft Corporation. All rights reserved.

C:\Users\Mike Person>node
Welcome to Node.js v12.16.1.
Type ".help" for more information.
> function Today()
... {
...   return new Date();
... }
undefined
> Today();
2020-04-08T21:31:09.473Z
> _
```



# REPL

Le mode *editor* permet d'écrire de multiples instructions pour ensuite pouvoir les utiliser.

La commande `.editor` ouvre un éditeur et on utilise CTRL+D pour le fermer.

```
Microsoft Windows [Version 10.0.17763.1098]
(c) 2018 Microsoft Corporation. All rights reserved.

C:\Users\Mike Person>node
Welcome to Node.js v12.16.1.
Type ".help" for more information.
> .editor
// Entering editor mode (^D to finish, ^C to cancel)
function Add(x,y)
{
    return x+y;
}

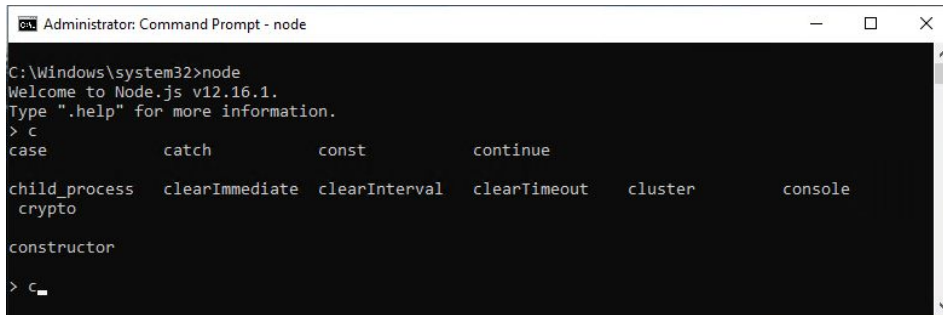
function Today()
{
    return new Date();
}

undefined
> Add(2,4)
6
> Today()
2020-04-08T21:34:33.106Z
> _
```

# REPL

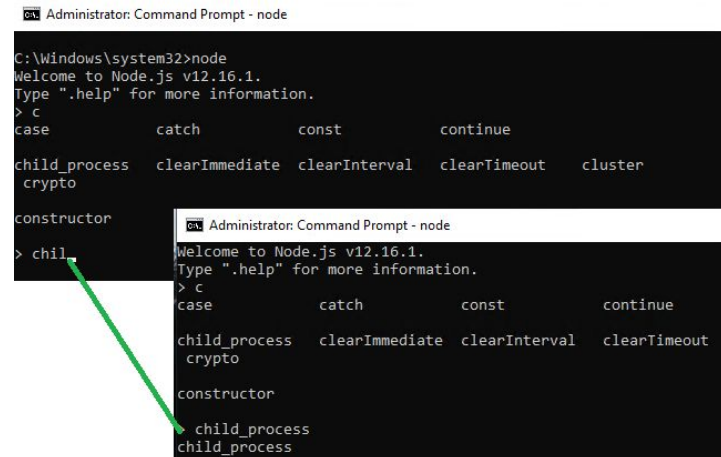
## La touche Tabulation : la touche magique.

La touche Tab dans l'outil permet de proposer des instructions (2 press)



```
Administrator: Command Prompt - node
C:\Windows\system32>node
Welcome to Node.js v12.16.1.
Type ".help" for more information.
> c
case          catch          const          continue
child_process clearImmediate clearInterval clearTimeout cluster    console
crypto
constructor
> c_
```

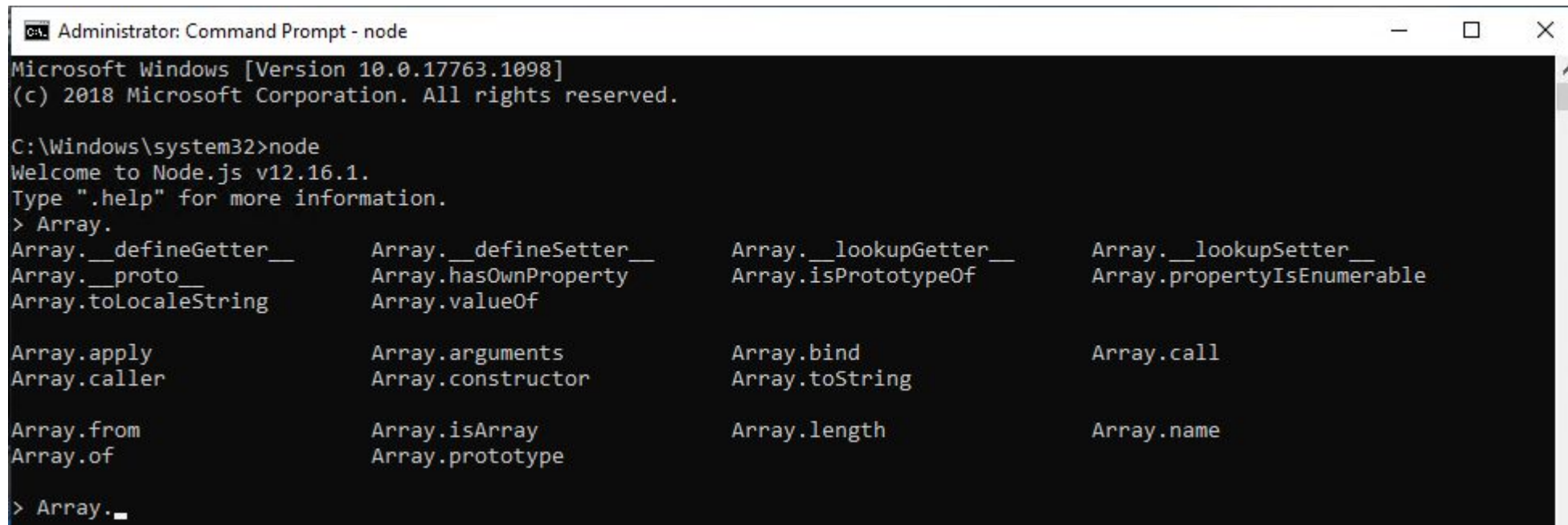
ou de compléter une instruction (1 press)



```
Administrator: Command Prompt - node
C:\Windows\system32>node
Welcome to Node.js v12.16.1.
Type ".help" for more information.
> c
case          catch          const          continue
child_process clearImmediate clearInterval clearTimeout cluster    co
crypto
constructor
> chil_
child_process
child_process
```

# REPL

Ce principe de tabulation fonctionne également pour avoir un « intellisense » à partir d'une classe



```
Administrator: Command Prompt - node
Microsoft Windows [Version 10.0.17763.1098]
(c) 2018 Microsoft Corporation. All rights reserved.

C:\Windows\system32>node
Welcome to Node.js v12.16.1.
Type ".help" for more information.
> Array.
Array.__defineGetter__      Array.__defineSetter__    Array.__lookupGetter__    Array.__lookupSetter__
Array.__proto__            Array.hasOwnProperty      Array.isPrototypeOf       Array.propertyIsEnumerable
Array.toLocaleString       Array.valueOf
Array.apply                Array.arguments           Array.bind                Array.call
Array.caller              Array.constructor        Array.toString
Array.from                 Array.isArray            Array.length              Array.name
Array.of                   Array.prototype
> Array._
```

# REPL

## Les Timers

Les fonctions existantes en JS se retrouvent bien entendu en NodeJs.

- Si on désire retarder l'exécution d'une fonction : **setTimeout(référence de la fonction à exécuter, temps en ms[, arg1,arg2,arg3,...])**

Exemple :

```
Administrator: Command Prompt - node
Microsoft Windows [Version 10.0.17763.1098]
(c) 2018 Microsoft Corporation. All rights reserved.

C:\Windows\system32>node
Welcome to Node.js v12.16.1.
Type ".help" for more information.
> const func = audience =>{ console.log(audience + ' is listening');}
undefined
> setTimeout(func,4*1000,'WebApps');
Timeout {
  _idleTimeout: 4000,
  _idlePrev: [TimersList],
  _idleNext: [TimersList],
  _idleStart: 90525,
  _onTimeout: [Function: func],
  _timerArgs: [Array],
  _repeat: null,
  _destroyed: false,
  [Symbol(refed)]: true,
  [Symbol(asyncId)]: 270,
  [Symbol(triggerId)]: 5
}
> WebApps is listening
```

# REPL

- Ou si on désire répéter une instruction dans le temps : **setInterval**

```
C:\Windows\system32>node
Welcome to Node.js v12.16.1.
Type ".help" for more information.
> const func = audience =>{ console.log('Coucou - ' + new Date().toString());}
undefined
> setInterval(func, 3*1000);
Timeout {
  _idleTimeout: 3000,
  _idlePrev: [TimersList],
  _idleNext: [TimersList],
  _idleStart: 27550,
  _onTimeout: [Function: func],
  _timerArgs: undefined,
  _repeat: 3000,
  _destroyed: false,
  [Symbol(refed)]: true,
  [Symbol(asyncId)]: 87,
  [Symbol(triggerId)]: 5
}
> Coucou - 10:48:44 GMT+0200 (Central European Summer Time)
Coucou - 10:48:47 GMT+0200 (Central European Summer Time)
Coucou - 10:48:50 GMT+0200 (Central European Summer Time)
Coucou - 10:48:53 GMT+0200 (Central European Summer Time)
```

# REPL

Il existe également les fonctions suivantes :

- **setImmediate** : équivalent à l'appel du setTimeout avec un délai de 0
- **clearTimeout** : permet d'annuler l'appel au setTimeout dont l'id est passé en paramètre
- **clearInterval** : permet d'annuler l'appel au setInterval dont l'id est passé en paramètre
- **clearImmediate** : permet d'annuler l'appel au setImmediate dont l'id est passé en paramètre

```
const timerId = setTimeout(  
  () => console.log('You will not see this one!'),  
  0  
);  
  
// setImmediate  
  
clearTimeout(timerId);  
// clearInterval  
// clearImmediate
```

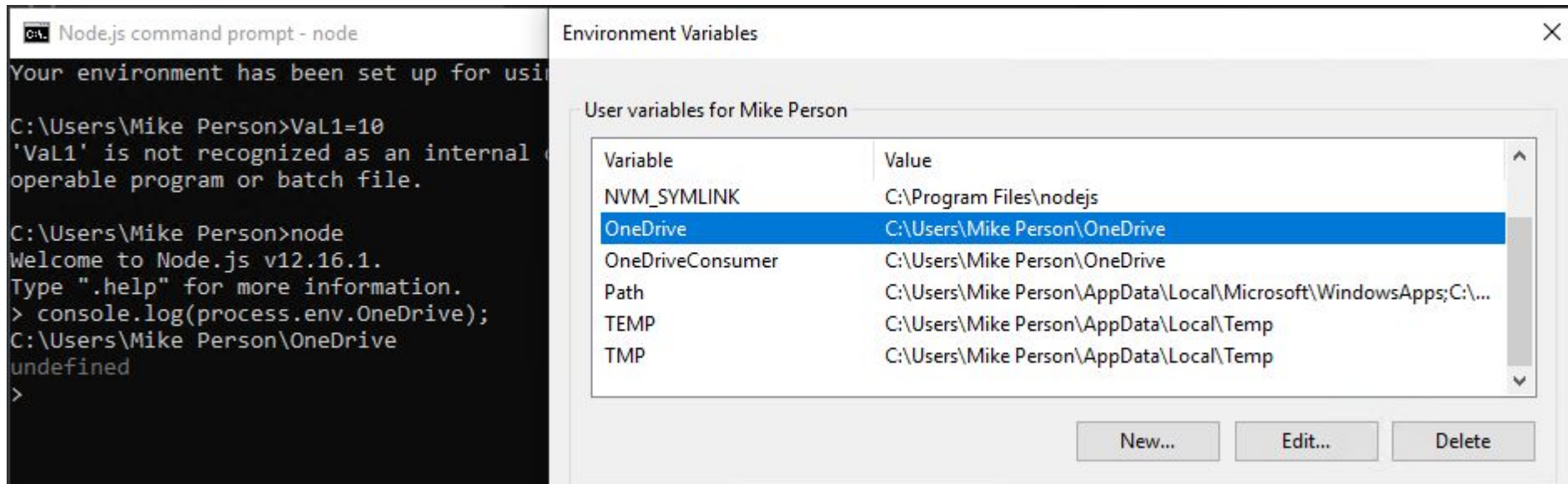
# REPL

## L'objet Process

Avec node, vous pouvez utiliser dans vos scripts les variables d'environnements définies en dehors du script.

Cela se fait via l'objet process et sa propriété env.

## Exemple :





# REPL

L'objet process contient également un attribut stdin qui nous permet de récupérer au clavier les infos encodées par l'utilisateur et stdout qui fait l'affichage dans la console

## Exemple :

```
1 process.stdin.on('readable',()=>
2 {
3   const info = process.stdin.read();
4   if(info!==null)
5   {
6     console.log("via le log : " + info);
7     //ou
8     process.stdout.write("via process :" + info);
9   }
10  });
```

DEBUG CONSOLE PROBLEMS OUTPUT TERMINAL 1: powershell v

```
PS C:\Cours\NodeJs\Exemples\REPL\ProcessObject> node .\app.js
Test
via le log : Test

via process :Test
PS C:\Cours\NodeJs\Exemples\REPL\ProcessObject> |
```



# NPM

NodeJs

# NPM

**Node Package Manager** est un outil gérant les bibliothèques de programmation **JavaScript** pour **Node.js**.



# NPM

## Npm init

Indispensable pour pouvoir utiliser npm avec votre projet .

Cette commande va générer un fichier [package.json](#) qui décrit la configuration de votre projet.

Chaque module de npm est configuré ainsi, ce qui fait que votre projet est par définition un paquet au même titre que les autres et donc potentiellement publiable sur npm .

```
PS C:\Cours\NodeJs\Exemples\NPMTEST> npm init
This utility will walk you through creating a package.json file.
It only covers the most common items, and tries to guess sensible defaults.
```

```
See `npm help json` for definitive documentation on these fields
and exactly what they do.
```

```
Use `npm install <pkg>` afterwards to install a package and
save it as a dependency in the package.json file.
```

```
Press ^C at any time to quit.
package name: (npmtest) npmtest
version: (1.0.0) 1.0.0
description: Ceci est un test
entry point: (index.js) app.js
test command:
git repository: http://gitlab.com/mperson/npmtest.git
keywords: npm, command, samples
author: Mike Person
license: (ISC)
About to write to C:\Cours\NodeJs\Exemples\NPMTEST\package.json:
```

```
{
  "name": "npmtest",
  "version": "1.0.0",
  "description": "Ceci est un test",
  "main": "app.js",
  "scripts": {
    "test": "echo \"Error: no test specified\" && exit 1"
  },
  "repository": {
    "type": "git",
    "url": "http://gitlab.com/mperson/npmtest.git"
  },
  "keywords": [
    "npm",
    "command",
    "samples"
  ],
  "author": "Mike Person",
  "license": "ISC"
}
```

```
Is this OK? (yes) █
```

# NPM

## npm -v

Permet d'obtenir la version installée

```
PS C:\Cours\NodeJs\Exemples\NPMTEST> npm -v  
6.13.4
```

## Npm install <nom du paquet>

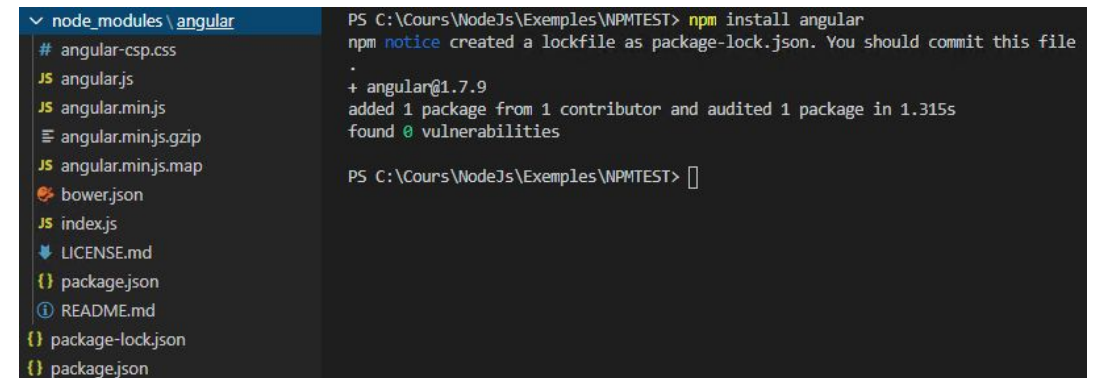
Cette commande permet l'installation d'un paquet et de ses dépendances.

Npm considère comme paquet valide :

- Un dossier contenant un fichier package.json
- Une archive gzip (contenant un dossier et le package.json associé )
- Une url vers une archive gzip
- Un paquet publié sur npm ( <nom\_du\_paquet>@<version>  
ou  
<nom\_du\_paquet>@<tag> ou <nom\_du\_paquet> )
- Une adresse git qui fournit une archive

## Npm uninstall <nom du paquet>

Cette commande permet de désinstaller un paquet



```
node_modules\angular  
# angular-csp.css  
JS angular.js  
JS angular.min.js  
angular.min.js.gzip  
JS angular.min.js.map  
bower.json  
JS index.js  
LICENSE.md  
package.json  
package-lock.json  
package.json  
package-lock.json
```

```
PS C:\Cours\NodeJs\Exemples\NPMTEST> npm install angular  
npm notice created a lockfile as package-lock.json. You should commit this file  
.  
+ angular@1.7.9  
added 1 package from 1 contributor and audited 1 package in 1.315s  
found 0 vulnerabilities  
  
PS C:\Cours\NodeJs\Exemples\NPMTEST>
```

# NPM

## Remarque :

Un paquet peut être installé globalement avec l'option -g ce qui permet une utilisation à travers la ligne de commande de n'importe où.

## **Attention à l'usage de l'installation globale !**

Si il peut sembler pratique d'avoir accès à des paquets de n'importe où sur le système sans devoir les réinstaller, ***la portabilité de votre module/programme ne sera plus assurée.***

# NPM

## Déclarer ses dépendances

NPM nous propose un gestionnaire de dépendance qui nous permet facilement de passer notre projet d'un environnement à un autre tout en gardant les dépendances à jour et de manière propre.

Npm distingue principalement deux types de dépendances renseignées sous forme d'objet dans le fichier package.json :

- Production
- Développement

```
"dependencies":  
{  
  "angular": "1.7.9"  
},  
"devDependencies":  
{  
  "angular": "1.7.9",  
  "webpack-dev-server": "3.10.3"  
}
```

Grâce à ce fichier, lors de la distribution de notre projet, un simple **npm install** et les dépendances s'installeront.

```
PS C:\Cours\NodeJs\Exemples\NPMTEST> npm install  
npm WARN deprecated resolve-url@0.2.1: https://github.com/lydell/resolve-url#de  
precated  
npm WARN deprecated urix@0.1.0: Please see https://github.com/lydell/urix#depre  
cated  
npm WARN optional SKIPPING OPTIONAL DEPENDENCY: fsevents@^1.2.7 (node_modules\  
hokidar\node_modules\fsevents):  
npm WARN optional SKIPPING OPTIONAL DEPENDENCY: Unsupported platform for fsevents
```

# NPM

Pour mettre à jour ses dépendances :

- `npm update <nom du paquet>` ☐ Mets à jour le paquet cité
- `npm update` ☐ Mets à jour toutes les dépendances

# NPM

## Semantic Versioning (SemVer)

L'écosystème nodejs utilise majoritairement la méthode **SemVer** pour numéroter les versions des paquets.

4 . 2 . 0

Major Minor Patch

Npm introduit un certain nombre de patterns permettant de gérer plus finement les versions qu'avec une numérotation exacte :

- Les opérateurs `<` `<=` `>` `>=` `=` peuvent être combinés avec `||` ou un espace pour former une expression définissant un ensemble de versions

`1.2.7 || >=1.2.9 <2.0.0` correspond aux versions 1.2.7 , 1.2.9 et 1.4.6 mais pas 1.2.8 ni 2.0.0

- `*` ( ou `X` ou `x` ) prendra toujours la dernière version disponible

`1*` équivaut à la dernière version de la branche 1 soit `>=1.0.0 <2.0.0`

- `–` détermine un intervalle inclusif

`1.2.3 – 2.3.4` équivaut à `>=1.2.3 <=2.3.4`



# NPM

- Les opérateurs ~ ^

- ~ inclus la version mineure la plus récente mais sans changer de branche

~1.2.3 équivaut à  $\geq 1.2.3 < 1.3.0$

- ^ inclus la version majeur la plus récente

^1.2.3 équivaut à  $\geq 1.2.3 < 2.0.0$

Il existe un site permettant de s'y retrouver lors de la recherche d'une version pour un package particulier : <https://semver.npmjs.com/>

Ce site mettra en surbrillances les versions incluent suite aux wildcard utilisés

Pour aller plus loin : <https://docs.npmjs.com/misc/semver> et <https://maxlab.fr/javascript/comprendre-npm-astuces-et-configuration/>

# Modules

NodeJs

# Modules

L'utilisation **Module** permet de simplifier l'écriture de code et de le gérer dans votre application.

Normalement, chaque **module** sera écrit dans un fichier séparé.

**NodeJS** intègre de nombreux **Modules** (Bibliothèques standards) dont les plus important ci-après :

Core Module	Description
<a href="#">http</a>	http module includes classes, methods and events to create Node.js http server.
<a href="#">url</a>	url module includes methods for URL resolution and parsing.
<a href="#">querystring</a>	querystring module includes methods to deal with query string.
<a href="#">path</a>	path module includes methods to deal with file paths.
<a href="#">fs</a>	fs module includes classes, methods, and events to work with file I/O.
<a href="#">util</a>	util module includes utility functions useful for programmers.

# Modules

## Charger un module

Pour pouvoir utiliser un module dans NodeJS, il faut utiliser la fonction *Require*

```
Var module = require('module_name');
```

```
var http = require('http');  
  
var server = http.createServer(function(req, res){  
    //write code here  
});  
  
server.listen(5000); |
```

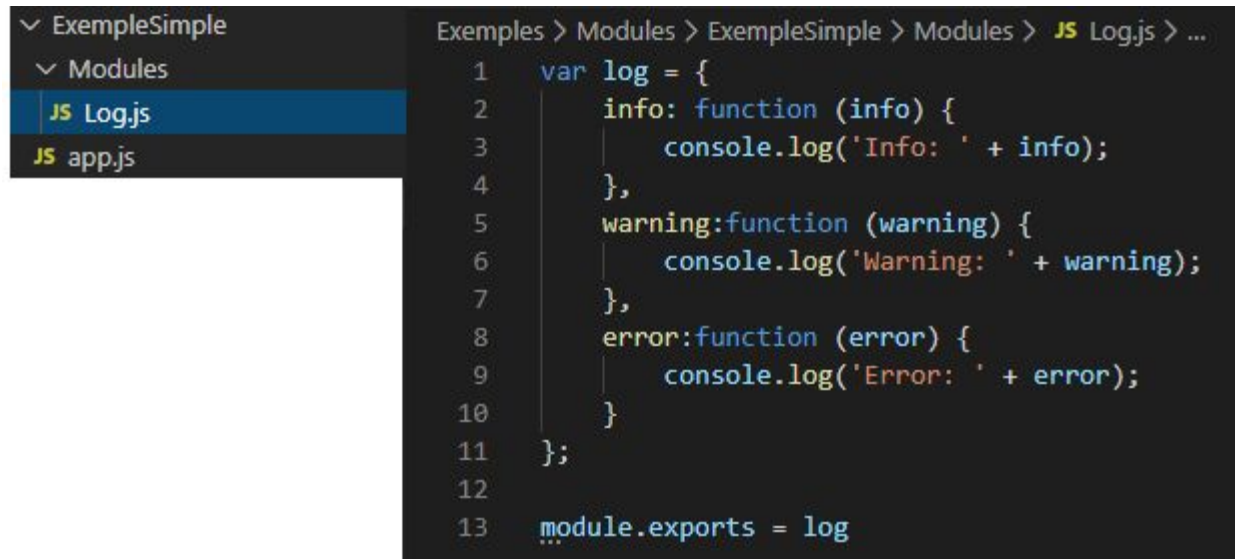
La fonction *require* renvoie un objet qui peut donc être utilisé par la suite dans le script.

# Modules

## Module local

Un module local est donc un module directement créé dans notre application NodeJs.

Ces modules locaux vont donc être placés dans différents dossier/fichiers pour fournir différentes fonctionnalités récurrentes.



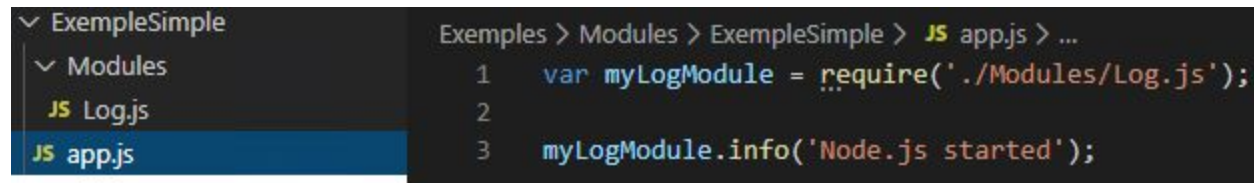
```
1  var log = {
2      info: function (info) {
3          console.log('Info: ' + info);
4      },
5      warning: function (warning) {
6          console.log('Warning: ' + warning);
7      },
8      error: function (error) {
9          console.log('Error: ' + error);
10     }
11 };
12
13 module.exports = log
```

Dans cet exemple, nous déclarons un objet contenant différentes fonctions et c'est via **module.exports** que nous définissons celui-ci comme étant un module.

# Modules

## Module local

Pour charger le module local dans notre app, nous utilisons la même fonction *require*



```
ExempleSimple
├── Modules
│   ├── Log.js
│   └── app.js
└── ...

Exemples > Modules > ExempleSimple > JS app.js > ...
1  var myLogModule = require('./Modules/Log.js');
2
3  myLogModule.info('Node.js started');
```

Ce qui nous donne le résultat suivant :

```
PS C:\Cours\NodeJs\Exemples\Modules\ExempleSimple> node app.js
Info: Node.js started
PS C:\Cours\NodeJs\Exemples\Modules\ExempleSimple> 
```

# Modules

## Export Module

Module.exports ou exports est un objet permettant inclus dans n'importe quel fichier js sous NodeJs.

*Module* est une variable qui représente le module courant et *exports* est un objet qui est exposé en tant que Module.

En résumé, tout ce que vous assignez à Module.exports ou exports sera considéré comme un module par NodeJs.

### - Export literals

Nous pouvons exporter directement une valeur.

```
module.exports = 'Hello world'; //ou exports = 'Hello world';
```

### - Export Object/function/class

```
module.exports = function (msg) {  
  console.log(msg);  
};
```

Exemples > Modules > ExempleFunctionExport > JS app.js > ...

```
1 var msg = require('./Modules/Log.js');  
2  
3 msg('Hello World');
```

```
module.exports = function (firstName, lastName) {  
  this.firstName = firstName;  
  this.lastName = lastName;  
  this.fullName = function () {  
    return this.firstName + ' ' + this.lastName;  
  }  
}
```

Exemples > Modules > ExempleClassExport > JS app.js > ...

```
1 var person = require('./Modules/Person.js');  
2  
3 var person1 = new person('James', 'Bond');  
4  
5 console.log(person1.fullName());
```

# File System

NodeJS



# File System

NodeJS implémente la gestion des I/O via le module *fs*.

```
var fs = require("fs")
```

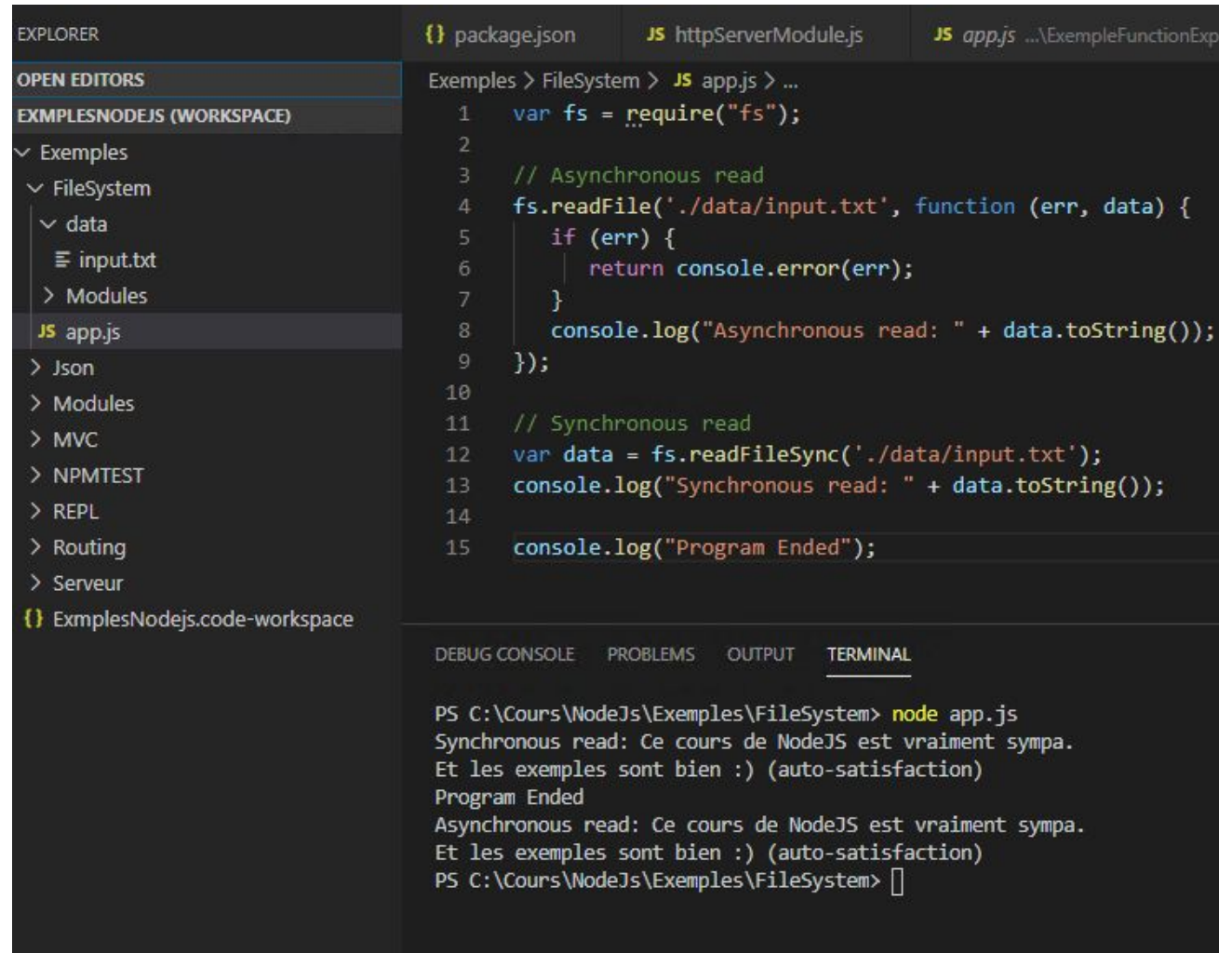
## Synchrone et Asynchrone

Nous avons la possibilité de travailler des deux manières avec *fs*

*fs.readFile(path, callback);*

*Ou*

*Let data = fs.readFileSync(path);*



The screenshot shows the Visual Studio Code interface. On the left, the Explorer sidebar shows a workspace named 'ExemplesNodejs.code-workspace' with a folder 'Exemples' containing a subfolder 'FileSystem'. Inside 'FileSystem', there is a file 'app.js' which is currently open in the editor. The code in 'app.js' demonstrates both asynchronous and synchronous file reading using the 'fs' module. The asynchronous part uses 'fs.readFile' with a callback function, and the synchronous part uses 'fs.readFileSync'. The terminal at the bottom shows the command 'node app.js' being executed, resulting in two lines of output: 'Synchronous read: Ce cours de NodeJS est vraiment sympa. Et les exemples sont bien :) (auto-satisfaction)' and 'Asynchronous read: Ce cours de NodeJS est vraiment sympa. Et les exemples sont bien :) (auto-satisfaction)', followed by 'Program Ended'.

```
package.json  JS httpServerModule.js  JS app.js ...\ExempleFunctionExp

EXPLORER
OPEN EDITORS
EXMPLESNODEJS (WORKSPACE)
  Examples
    FileSystem
      data
        input.txt
      Modules
      JS app.js
      Json
      Modules
      MVC
      NPMTEST
      REPL
      Routing
      Serveur
  ExmplesNodejs.code-workspace

Examples > FileSystem > JS app.js > ...
1  var fs = require("fs");
2
3  // Asynchronous read
4  fs.readFile('./data/input.txt', function (err, data) {
5    if (err) {
6      return console.error(err);
7    }
8    console.log("Asynchronous read: " + data.toString());
9  });
10
11 // Synchronous read
12 var data = fs.readFileSync('./data/input.txt');
13 console.log("Synchronous read: " + data.toString());
14
15 console.log("Program Ended");

DEBUG CONSOLE  PROBLEMS  OUTPUT  TERMINAL
PS C:\Cours\NodeJs\Exemples\FileSystem> node app.js
Synchronous read: Ce cours de NodeJS est vraiment sympa.
Et les exemples sont bien :) (auto-satisfaction)
Program Ended
Asynchronous read: Ce cours de NodeJS est vraiment sympa.
Et les exemples sont bien :) (auto-satisfaction)
PS C:\Cours\NodeJs\Exemples\FileSystem>
```

# File System

## Ouverture de fichier

Nous pouvons préciser des *flags* pour l'ouverture du fichier afin de le traiter en lecture seule par exemple

```
fs.open(path, flags[, mode], callback)
```

Exemples > FileSystem > JS appOpen.js > ...

```
1  var fs = require("fs");
2
3  // Asynchronous - Ouverture du fichier
4  console.log("Nous allons ouvrir le fichier");
5  fs.open('./data/input.txt', 'r+', function(err, fd) {
6      if (err) {
7          return console.error(err);
8      }
9      console.log("Le fichier est ouvert!");
10 });
```

Flags	
<b>r</b>	Ouverture en lecture seule (exception si le fichier n'existe pas)
<b>r+</b>	Ouverture en lecture/écriture (exception si le fichier n'existe pas)
<b>rs</b>	Ouverture en lecture seule en mode synchrone
<b>rs+</b>	Ouverture en lecture/écriture en mode synchrone
<b>w</b>	Ouverture en écriture (le fichier est créé si il n'existe pas ou vidé si il existe)
<b>wx</b>	Ouverture en écriture (le fichier est créé si il n'existe pas ou échec si il existe)
<b>w+</b>	Ouverture en écriture/lecture (le fichier est créé si il n'existe pas ou vidé si il existe)
<b>wx+</b>	Ouverture en écriture/lecture (le fichier est créé si il n'existe pas ou échec si il existe)
<b>a</b>	Permet d'ajouter du contenu à un fichier (créé si le fichier n'existe pas)
<b>ax</b>	Permet d'ajouter du contenu à un fichier (Exception si le fichier n'existe pas)
<b>a+</b>	Permet de lire et d'ajouter du contenu à un fichier (créé si le fichier n'existe pas)
<b>ax+</b>	Permet lire et d'ajouter du contenu à un fichier (Exception si le fichier n'existe pas)

# File System

## Obtenir des informations sur le fichier

*fs* nous propose la fonction *stat*

```
fs.stat(path, callback);
```

### Methodes

**stats.isFile()** renvoie true si c'est un fichier

**stats.isDirectory()** renvoie true si c'est un dossier

**stats.isBlockDevice()** renvoie true si c'est un *block device* (cdrom, disquette,...)

**stats.isCharacterDevice()** renvoie true si c'est une *character device* (port serie, carte son,...)

**stats.isSymbolicLink()** renvoie true si c'est un lien symbolic (pointer vers un autre fichier ☐ linux)

**stats.isSocket()** renvoie true si c'est un type socket

```
Exemples > FileSystem > JS appStats.js > ...
1  var fs = require("fs");
2
3  console.log("Récupération des infos du fichier");
4  fs.stat('./data/input.txt', function (err, stats) {
5      if (err) {
6          return console.error(err);
7      }
8      console.log(stats);
9      console.log("Récupération ok!");
10
11     // Check file type
12     console.log("Est-ce un fichier ? " + stats.isFile());
13     console.log("Est-ce un dossier ? " + stats.isDirectory());
14 });
```

DEBUG CONSOLE PROBLEMS OUTPUT TERMINAL

Windows PowerShell

Copyright (C) Microsoft Corporation. All rights reserved.

PS C:\Cours\NodeJs\Exemples> cd .\FileSystem\

PS C:\Cours\NodeJs\Exemples\FileSystem> node .\appStats.js

Récupération des infos du fichier

```
Stats {
  dev: 4038446957,
  mode: 33206,
  nlink: 1,
  uid: 0,
  gid: 0,
  rdev: 0,
  blksize: 4096,
  ino: 1125899907166597,
  size: 88,
  blocks: 0,
  atimeMs: 1586942568794.74,
  mtimeMs: 1586942568794.74,
  ctimeMs: 1586942568794.74,
  birthtimeMs: 1586942540297.8015,
  atime: 2020-04-15T09:22:48.795Z,
  mtime: 2020-04-15T09:22:48.795Z,
  ctime: 2020-04-15T09:22:48.795Z,
  birthtime: 2020-04-15T09:22:20.298Z
}
```

Récupération ok!

Est-ce un fichier ? true

Est-ce un dossier ? false

# File System

## Ecrire dans un fichier

Pour écrire (écraser le contenu aussi) d'un fichier :

```
fs.writeFile(filename, data[,options],callback);
```

- Data : string ou buffer
- Options : un objet json

```
{  
  encoding (UTF8),  
  mode (0666),  
  flag (w)  
}
```

Exemples > FileSystem > JS appWrite.js > ...

```
1  var fs = require("fs");  
2  
3  fs.writeFile('./data/Nouveau.txt', 'C'est écrit (Francis Cabrel)', function(err) {  
4    if (err) {  
5      return console.error(err);  
6    }  
7  
8    console.log("Ecriture OK");  
9    console.log("Lecture du fichier");  
10  
11    fs.readFile('./data/Nouveau.txt', function (err, data) {  
12      if (err) {  
13        return console.error(err);  
14      }  
15      console.log("Lecture async: " + data.toString());  
16    });  
17  });
```

DEBUG CONSOLE PROBLEMS OUTPUT TERMINAL

```
PS C:\Cours\NodeJs\Exemples\FileSystem> node .\appwrite.js  
Ecriture OK  
Lecture du fichier  
Lecture async: C'est écrit (Francis Cabrel)  
PS C:\Cours\NodeJs\Exemples\FileSystem> 
```

# File System

## Lire un fichier

Pour lire via un fichier

```
fs.read(fd, buffer, offset, length, position, callback)
```

- fd : la valeur retournée par un open
- Buffer : contiendra les données
- Offset : position dans le buffer pour l'écriture
- Length : nombre de bytes à lire
- Position : int définissant le début de la lecture.  
Si null, lecture à la position courante

```
Exemples > FileSystem > JS appReadBuffer.js > fs.open('./data/input.txt', 'r+') callback
1  var fs = require("fs");
2  var buf = Buffer.alloc(1024);
3
4  console.log("Ouverture du fichier existant");
5  fs.open('./data/input.txt', 'r+', function(err, fd) {
6      if (err) {
7          return console.error(err);
8      }
9      console.log("Ouverture ok!");
10     console.log("Lecture");
11
12     fs.read(fd, buf, 0, buf.length, 0, function(err, bytes){
13         if (err){
14             console.log(err);
15         }
16         console.log(bytes + " => Nb bytes lus");
17
18         if(bytes > 0){
19             console.log("-");
20             console.log(buf.slice(0, bytes).toString());
21         }
22     });
23 }
24 };
```

DEBUG CONSOLE PROBLEMS OUTPUT TERMINAL

```
PS C:\Cours\NodeJs\Exemples\FileSystem> node .\appReadBuffer.js
Ouverture du fichier existant
Ouverture ok!
Lecture
88 => Nb bytes lu
-
Ce cours de NodeJS est vraiment sympa.
Et les exemples sont bien :) (auto-satisfaction)
PS C:\Cours\NodeJs\Exemples\FileSystem>
```



# File System

## Fermer un fichier

```
// Close the opened file.
fs.close(fd, function(err) {
  if (err) {
    console.log(err);
  }
  console.log("Fichier fermé.");
});
```

## Créer un dossier

```
var fs = require("fs");

console.log("On crée le dossier ./test");
fs.mkdir('./test',function(err) {
  if (err) {
    return console.error(err);
  }
  console.log("Dossier créé!");
});
```

## Supprimer un fichier

```
var fs = require("fs");

fs.unlink('./data/Nouveau.txt', function(err) {
  if (err) {
    return console.error(err);
  }
  console.log("Fichier supprimé!");
});
```

## Parcourir un dossier

```
var fs = require("fs");

console.log("lire le dossier ./data/");
fs.readdir("./data",function(err, files) {
  if (err) {
    return console.error(err);
  }
  files.forEach( function (file) {
    console.log( file );
  });
});
```

# File System

## Supprimer un dossier

```
var fs = require("fs");

console.log("supprimer ./test");
fs.rmdir("./test",function(err, files) {
  if (err) {
    return console.error(err);
  }
  console.log("Dossier supprimé!")
});
```

# WebServer

NodeJS



# WebServer

NodeJS est généralement utilisé pour créer des applications Clients/Server.

Il existe plusieurs modules tels *http* ou *request* qui peuvent nous aider à mettre en place un WebServer ou à effectuer une requête http

## Utilisation du module *http*

Voici un simple exemple de mise en place d'un serveur Web



# WebServer

## Digression pratique

Lorsque nous changeons notre code, nous devons stopper et relancer la commande node...

Cela peut s'avérer « énervant ».

Il est possible d'utiliser le package *nodemon* qui est un simple wrapper permettant de relancer pour nous nodejs lorsqu'un changement est détecté sur un fichier

Pour l'utiliser :

- 1- npm install -g nodemon
- 2- on remplace node par nodemon pour lancer notre app.

```
Exemples > Serveur > JS ServerTest.js > ...
1  var http = require('http');
2
3  var server = http.createServer(function(req,res){
4
5      res.writeHead(200);
6      res.end('<DOCTYPE!><html><h1>Hello à Tous !</h1></html>');
7  });
8
9  server.listen(8001,()=>{console.log("Server is running ...");});
```

DEBUG CONSOLE PROBLEMS OUTPUT TERMINAL

```
PS C:\Cours\NodeJs\Exemples\Serveur> nodemon .\ServerTest.js
[nodemon] 2.0.3
[nodemon] to restart at any time, enter `rs`
[nodemon] watching path(s): *.*
[nodemon] watching extensions: js,mjs,json
[nodemon] starting `node .\ServerTest.js`
Server is running ...
```

# WebServer

## Request

le premier argument de la fonction associée au *createServer* est l'http Request qui est de type *IncommingMessage* sous NodeJS.

Cet Objet *IncommingMessage* permet d'accéder aux différentes propriétés de la requête tel le statuscode, le headers,...

Il est intéressant de voir que lorsqu'on demande l'affichage d'une page simple, la fonction callback du *http.createServer* est appelé plus d'une fois.

Cela s'explique par la demande du Navigateur.

```
Exemples > Serveur > IncommingMessageSample > JS ServerTest.js > server > http.createServer() callback
1  var http = require('http');
2  let cpt =1;
3  var server = http.createServer(function(req,res){
4      console.log(cpt +" - " +req.url);
5      res.writeHead(200);
6
7      res.end('<DOCTYPE!><html><h1>Hello à Tous !</h1></html>');
8      cpt++;
9  });
10
11  server.listen(8001,()=>{console.log("Server is running ... http://127.0.0.1:8001");});
```

DEBUG CONSOLE PROBLEMS OUTPUT TERMINAL

```
PS C:\Cours\NodeJs\Exemples\Serveur\IncommingMessageSample> nodemon .\ServerTest.js
[nodemon] 2.0.3
[nodemon] to restart at any time, enter `rs`
[nodemon] watching path(s): *.*
[nodemon] watching extensions: js,mjs,json
[nodemon] starting `node .\ServerTest.js`
Server is running ... http://127.0.0.1:8001
1 - /
2 - /favicon.ico
```

# WebServer

## GET

Si nous désirons récupérer les paramètres se trouvant dans l'url, nous utiliserons *req.url* comme dans l'exemple précédent.

Cependant, pour nous faciliter la vie, nous utiliserons un module supplémentaire appelé *url* (<https://nodejs.org/api/url.html>)

Grâce à cette librairie, nous pourrions facilement « parser » notre querystring et obtenir les paramètres transmis

```
Exemples > Serveur > QueryStringSample > JS ServerTest.js > server > http.createServer() callback
1  var http = require('http');
2  var url = require('url');
3  let cpt =1;
4  var server = http.createServer(function(req,res){
5      if(cpt==1)
6      {
7          console.log(url.parse(req.url,true).query);
8          console.log("Host : " + url.parse(req.url,true).host);
9          console.log("Pathname : " + url.parse(req.url,true).pathname);
10         console.log("search : " + url.parse(req.url,true).search );
11     }
12
13     res.writeHead(200);
14
15     res.end('<DOCTYPE!><html><h1>Hello à Tous !</h1></html>');
16     cpt++;
17 });
18
19 server.listen(8001,()=>{console.log("Server is running ... http://127.0.0.1:8001");});
```

DEBUG CONSOLE PROBLEMS OUTPUT TERMINAL

```
PS C:\Cours\NodeJs\Exemples\Serveur\QueryStringSample> nodemon .\ServerTest.js
[nodemon] 2.0.3
[nodemon] to restart at any time, enter `rs`
[nodemon] watching path(s): *.*
[nodemon] watching extensions: js,mjs,json
[nodemon] starting `node .\ServerTest.js`
Server is running ... http://127.0.0.1:8001
[Object: null prototype] { id: '2', titre: 'Bonjour' }
Host : null
Pathname : /app.js
search : ?id=2&titre=Bonjour
```



# WebServer

## POST

En premier lieu, nous devons vérifier si nous sommes bien face à une requête POST.

Ensuite nous devons gérer deux événements :

- data □ on reçoit les données du formulaire
- end □ les données ont été transférées

Enfin, nous devons lire un *ReadableStream* qui contient les informations du POST et pour cela, nous utilisons la méthode *parse* de la librairie *querystring*.

```
1  const http= require('http');
2  const { parse } = require('querystring');
3  const requestListener = function(req,res)
4  {
5      if (req.method === 'POST')
6      {
7          let body = '';
8          req.on('data', form => { body += form.toString(); }); //On récupère le corps de la requête
9          req.on('end', () => { //Tout le stream a été transmis
10             console.log(parse(body)); //on utilise la méthode parse de la librairie querystring
11             res.writeHead(204);
12             res.end();
13         });
14     }
15     else
16     {
17         let body = `<!doctype>
18             <html><body>
19                 <form action="/" method="post">
20                     <input type="text" name="fname" /><br />
21                     <input type="number" name="age" /><br />
22                     <input type="file" name="photo" /><br />
23                     <button>Save</button>
24                 </form>
25             </body></html>`;
26         res.writeHead(200,{
27             'Content-Length': Buffer.byteLength(body),
28             'Content-Type': 'text/html'
29         });
30         res.end(body);
31     }
32 }
33 var server = http.createServer();
34 server.on('request',requestListener);
35 server.listen(8001,()=>{console.log("Server is running on http://localhost:8001/");});
```

DEBUG CONSOLE PROBLEMS OUTPUT TERMINAL 1: node

```
Server is running on http://localhost:8001/
[Object: null prototype] {
  fname: 'Mike',
  age: '5',
  photo: 'package.json'
}
```

# WebServer

## Tips :

- Si nous désirons envoyer de l'html plutôt que du simple texte à notre navigateur, nous devons définir le *content-type* dans le header de la response.

```
res.writeHead(200,{  
  'Content-Length': Buffer.byteLength(body),  
  'Content-Type': 'text/html'  
});
```

- Si nous voulons écrire un string sur plusieurs lignes (pour une question de lisibilité), écrivez

`<h1>...</h1>`

plutôt que

`<h1>...</h1>`

# Templating

NodeJS

# Templating

NodeJS seul n'est pas très pratique pour afficher de belle pages Html car nous devons encoder tout notre html dans un string.

```
let body = `<!doctype>
<html><body>
  <form action="/" method="post">
    <input type="text" name="fname" /><br />
    <input type="number" name="age" /><br />
    <input type="file" name="photo" /><br />
    <button>Save</button>
  </form>
</body></html>`;
```

Nous avons bien entendu la possibilité d'utiliser une Framework comme express (<https://expressjs.com/fr/>) qui nous faciliterais la vie pour le templating mais également pour la gestion du serveur web.

Mais si nous désirons simplement utiliser un moteur de Template sans Framework, c'est possible !!!

Il existe plusieurs moteurs de template : Vash, Jade, Haml, Mustache, Handlebar, Underscore, Pure, ...

Pour notre exemple, nous utiliserons EJS (<https://ejs.co/>)



# Templating

## EJS

Pourquoi EJS ?

Car c'est un langage de template simple qui permet de générer un balisage Html via le javascript.

### Quelques infos de syntaxe :

Tag	Utilisation
<%	Juste pour inclure du script, aucune sortie html
<%=	Affiche la valeur du modèle envoyé (Html escaped)
<% -	Affiche la valeur Html non escaped
<%%	Affiche un littéral
<% #	Commentaire
%>	Balise de fin

# EJS

## Include

Il est possible d'inclure un morceau de template dans un autre (~= partial view).

Pour cela, nous utiliserons la syntaxe suivante :

```
<%- include('header'); -%>
<h1>
  Title
</h1>
<p>
  My page
</p>
<%- include('footer'); -%>
```

Le paramètre de la fonction *include* est une chemin relatif vers le fichier ejs à inclure

# Templating

## Ejs – Utilisation

Première étape, installer le package via npm

```
npm install ejs
```

Ensuite, nous créons notre template avec l'extension *ejs*

```
<!doctype>
<html>
  <body>
    <h1><%= titreform %></h1>
    <form action="/" method="post">
      <input type="text" name="fname" /><br />
      <input type="number" name="age" /><br />
      <input type="file" name="photo" /><br />
      <button>Save</button>
    </form>
  </body>
</html>
```

# Templating

Nous pouvons finalement utiliser ce template dans notre script,

Pour cela, nous devons charger la librairie :

```
const ejs = require('ejs');
```

Grâce à cette librairie, nous avons accès à la fonction *render(template, Model)*.

- Template : un string qui contient l'html avec les séquence <% permettant l'injection des valeurs du Model
- Model : un objet avec les différents clé/valeurs devant être injectées dans le template

Pour créer le string qui sera associé au template, nous utiliserons *fs.readFileSync* pour récupérer le contenu de notre View

```
let body = fs.readFileSync(__dirname+'/views/index.ejs','utf8');
let bodyRendered = ejs.render(body,{titreform : 'Contactez-nous rapidement '});
res.writeHead(200,{
  'Content-Length': Buffer.byteLength(body),
  'Content-Type': 'text/html'
});
res.end(bodyRendered);
```

# Templating

## Remarque :

si *ejs.render* ne se trouve pas dans le fichier d'entrée principale (app.js par exemple), les includes ne fonctionneront pas car *ejs* ne pourra pas retrouver le chemin relatif au document.

Il faut alors utiliser

```
ejs.renderFile(filename, data, options, function(err, str){ // str => Rendered HTML string });
```

(<https://github.com/mde/ejs>)