

NODE.JS

JOUR 1 : DECOUVERTE DE NODE.JS



PRESENTE PAR NATACHA DESSE

AFEC



DEROULE SEMAINE



Découverte de Node.JS



Développer une API REST avec Express



Gestion d'erreurs et middleware



Authentification et JWT





DECOUVERTE DE NODE.JS



LES OBJECTIFS

Comprendre ce qu'est Node.js et à quoi il sert

Écrire ses premières instructions en JavaScript côté serveur

Manipuler les modules natifs de Node.js

Savoir ce qu'est npm et installer un package externe





NUAGE DE MOTS

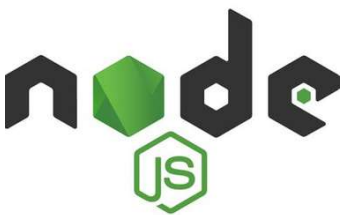
Que vous évoque Node.JS?





QU'EST CE QUE NODE.JS

C'EST QUOI NODE ?



NodeJS est un **runtime** javascript.

Un runtime est un environnement d'exécution.
C'est à dire un couple constitué d'un moteur et d'un contexte global.

Il a été créé par Ryan Dahl en 2009



Moteur

Pour javascript, c'est un interpréteur (= lit et exécute le code à la volée, sans compilation).

Il existe de nombreux moteurs javascript.

Node utilise « V8 », un moteur créé par Google, et présent dans Chrome (et tous ses dérivés Chromium, Brave, Vivaldi, ...)



UN NOUVEAU CONTEXTE

Contexte navigateur



- L'objet global s'appelle **window**.
- Il contient des fonctions pour interagir avec l'utilisateur (alert, prompt, confirm...) ou avec le navigateur lui-même (history, location, ...), mais aussi le DOM (window.document !)

Contexte Node

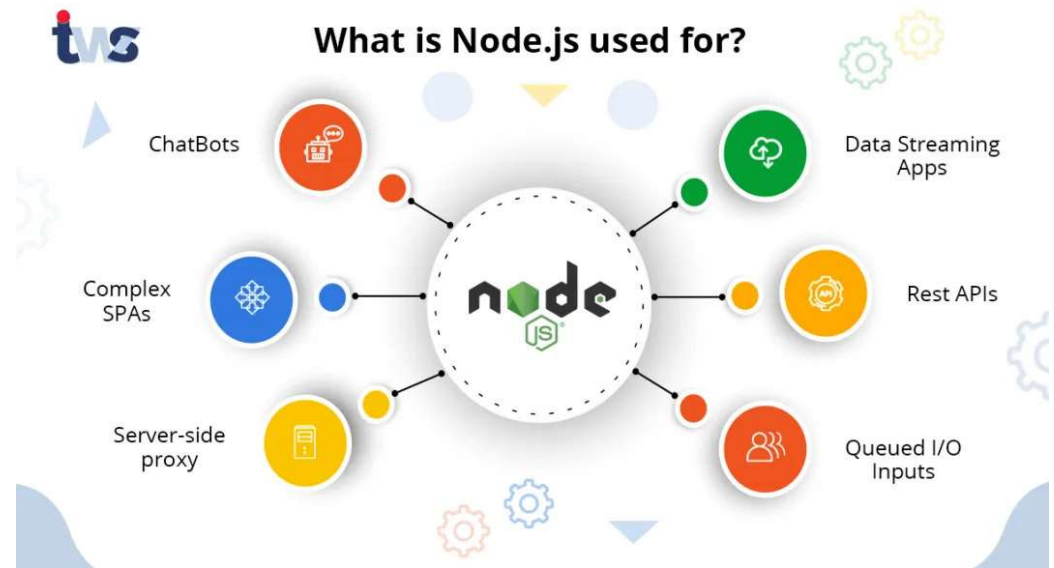


- L'objet global s'appelle **global**.
- Il contient des informations sur le processus Node (global.process) ainsi que la fonction require qui permet d'importer d'autres fonctionnalités (cf. modules)

POURQUOI UTILISER NODE.JS

Concrètement, pour faire du JS partout

- Utilisé par des grandes entreprises comme PayPal, Uber ou Netflix
- Permet de créer des services backend et des API
- Permet aux développeurs JavaScript de créer des applications full-stack
- Très évolutif et efficace
- Node.js est gratuit
- Node.js fonctionne sur différentes plateformes (Windows, Linux, Unix, Mac OS X, etc.).
- Facile de prise en main

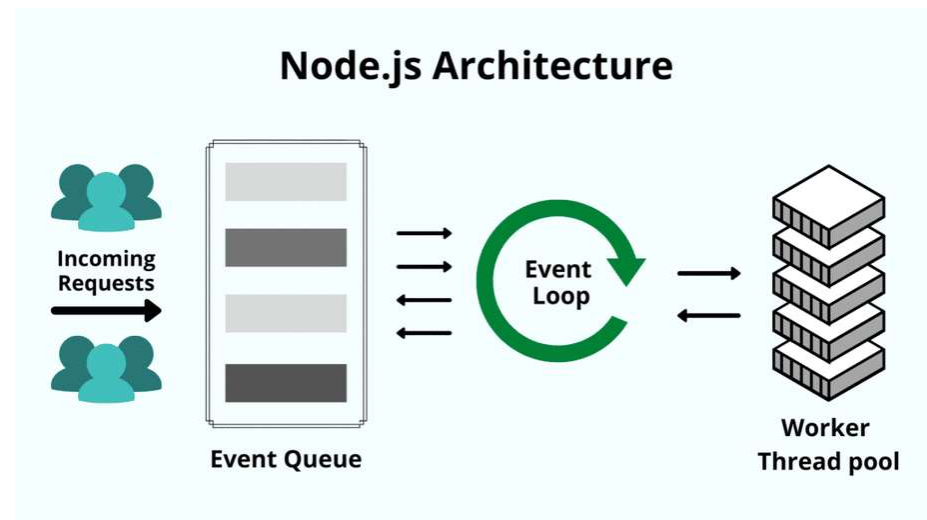


LE FONCTIONNEMENT DE NODE.JS

Node.js fonctionne en programmation :

- **évènementielle**
- non bloquante (**asynchrone**)
- à fil unique (**mono-thread**), ce qui est très efficace en terme de mémoire.

De ce fait, il est très rapide.



DANS QUEL CAS CE N'EST PAS PERTINENT

Calculs intensifs ou traitements CPU-bound

- Exemple : cryptographie lourde, compression vidéo, calculs scientifiques
- Pourquoi ? Node.js est mono-thread par défaut (il repose sur un thread principal), ce qui le rend moins adapté pour des calculs bloquants qui monopolisent le CPU.
- Préférer : des langages compilés comme C++, Java, Go ou Python avec NumPy pour les calculs scientifiques.

Applications fortement typées avec logique métier complexe

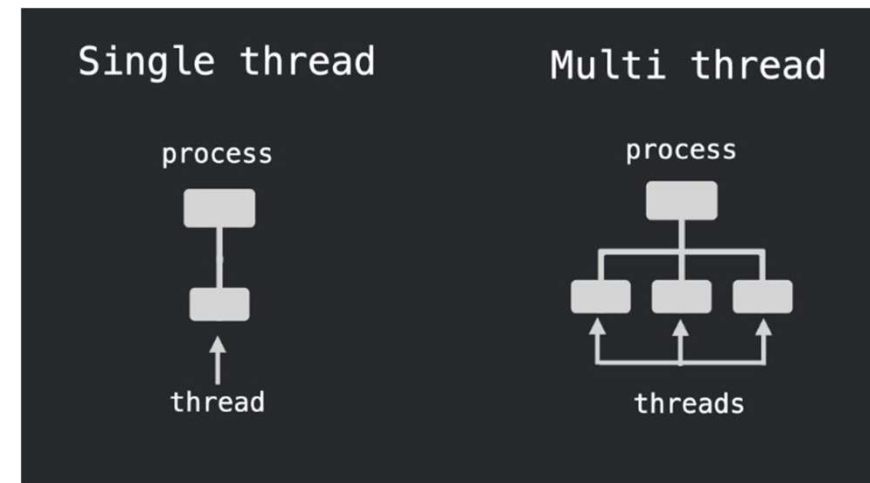
- Exemple : ERP, logiciels embarqués, systèmes bancaires internes
- Pourquoi ? L'absence de typage strict en JavaScript peut être un inconvénient. Même avec TypeScript, certains préféreront des langages comme C#, Java, Rust pour leur robustesse.

UN SEUL THREAD JAVASCRIPT

Comment un **serveur classique** (Apache-PHP, IIS-ASP, Tomcat-JavaEE,) traite une demande de fichier :

1. Il envoie la tâche au système de fichiers de l'ordinateur.
2. Il attend que la réponse du système de fichiers.
3. Quand il a obtenu la réponse, il renvoie le contenu au client.
4. Il est prêt à traiter la demande suivante.

Si l'ordinateur est multi-thread, il peut traiter plusieurs tâches 1 en parallèle.



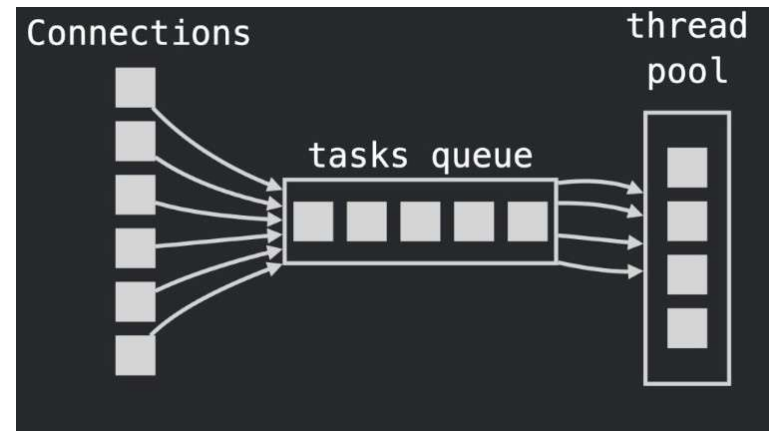
MAIS IL NE BLOQUE PAS... CAR IL DELEGUE

Comment **Node.js** traite une demande de fichier :

1. Il **envoie** la **tâche** au système de fichiers de l'ordinateur.
2. Il **traite** la **demande suivante sans attendre** la réponse du système de fichier : la demande 1 est une demande « asynchrone ».
3. Lorsque le système de fichiers renvoie la réponse au serveur, le serveur **met en pause**, les actions en cours, renvoie le contenu au client puis reprend ses actions en cours.

Quand tu fais une opération lente (comme lire un fichier ou interroger une base de données), Node.js ne **la fait pas lui-même dans le thread principal**.

Il délègue cette tâche à un thread du système (géré par la bibliothèque native **libuv**) ou à une API système non bloquante, et continue.



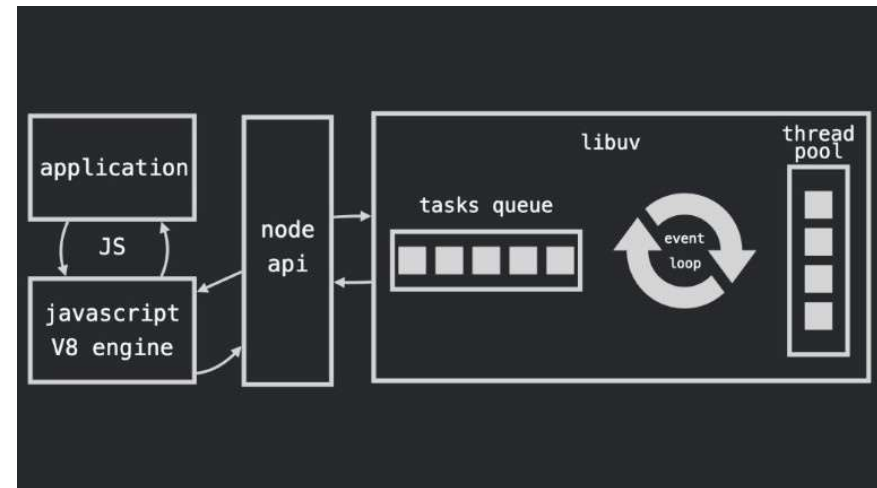
Exemples de tâches déléguées :

- Accès disque (fs.readFile)
- Requêtes réseau (http)
- Timers (setTimeout)
- DNS, etc.

LA BOUCLE D'ÉVÉNEMENTS (EVENT LOOP)

Node.js utilise une event loop qui fonctionne comme un chef d'orchestre :

- Elle regarde si des callbacks sont prêts à être exécutés.
- Elle les place dans la file d'exécution.
- Le thread principal exécute ensuite ces callbacks un à un, dans l'ordre.



EXEMPLE SIMPLE

```
1 console.log('Début');  
2  
3 fs.readFile('fichier.txt',  
4   'utf8', (err, data) => {  
5     console.log('Fichier lu');  
6   });  
7 console.log('Fin');
```

Ce qui se passe :

1. console.log('Début') s'exécute → affiche Début.
2. fs.readFile est délégué à **libuv** → Node continue sans attendre.
3. console.log('Fin') s'exécute → affiche Fin.
4. Quand le fichier est lu → la **callback** est mise dans la **queue**.
5. La boucle d'événements l'exécute → affiche Fichier lu.

DONC : SINGLE- THREADED ≠ SYNCHRONE

Node.js exécute JavaScript dans un seul thread, mais grâce à :

- libuv (threads en C++)
- la boucle d'événements
- les callbacks ou async/await

... il gère de nombreuses tâches en parallèle sans les bloquer.



INSTALLATION ET MODULES

INSTALLER NODE.JS

<https://nodejs.org>

- Choisir la version LTS
- Installer via l'assistant en ouvrant le .msi téléchargé
- Vérification dans le terminal :

```
C:\Users\Surface>node -v  
v20.12.2
```

```
C:\Users\Surface>npm -v  
10.6.0
```



LES MODULES

Les modules sont un concept **fondamental** en Node.js. Ils permettent de séparer ton code en fichiers **réutilisables** — comme des briques de construction pour ton application

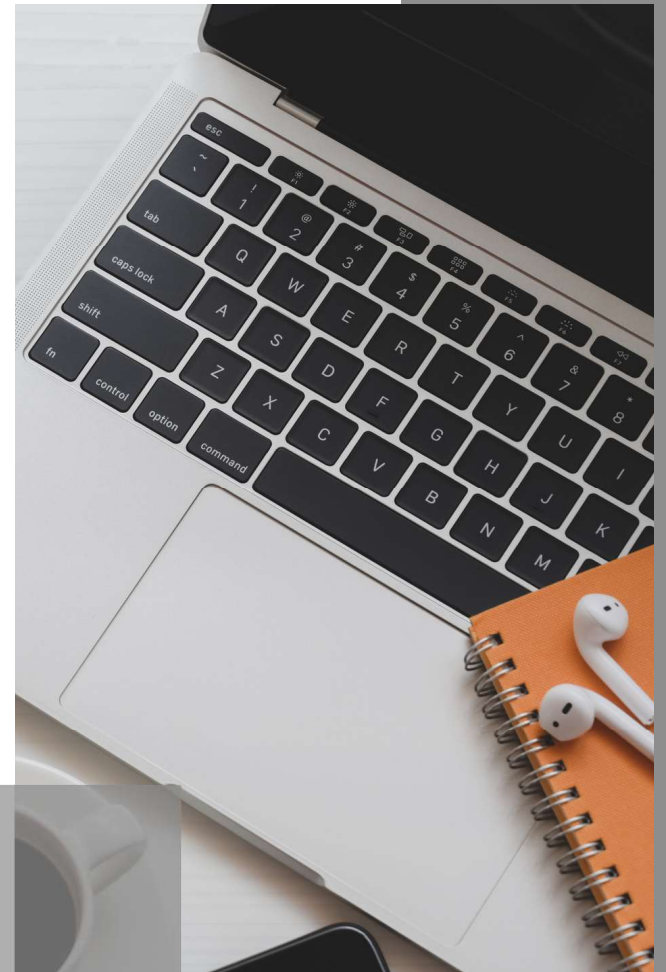
Les types de modules :

- Modules **natifs** (built-in) : inclus dans Node.js
- Modules **externes** npm
- Modules **utilisateurs** : personnels

Nous allons voir comment exploiter tous ces modules



LES MODULES NATIFS



LES MODULES NATIFS BUILT-IN

Node.js est livré avec un ensemble de modules prêts à l'emploi.

- fs → pour accéder au système de fichiers
- http → pour créer un serveur HTTP
- path → pour manipuler les chemins de fichiers
- os → pour obtenir des infos sur le système d'exploitation

Module

fs (File System)

http

path

os

Utilité principale

Lire, écrire, supprimer ou modifier des fichiers et dossiers.

Créer un serveur HTTP pour répondre aux requêtes.

Travailler avec les chemins de fichiers, de manière multiplateforme.

Obtenir des infos sur le système (CPU, RAM, etc.).

Exemples d'utilisation

fs.readFile(),
fs.writeFile(),
fs.readdir()

http.createServer(),
res.write(), req.url

path.join(),
path.basename(),
path.resolve()

os.platform(), os.cpus(),
os.freemem()

EXEMPLE D'UTILISATION

```
1 const fs = require('fs');
2 const http = require('http');
3 const path = require('path');
4 const os = require('os');
5
6 console.log('OS platform:', os.platform());
7 console.log('CPU count:', os.cpus().length);
8
9 const server = http.createServer((req, res) => {
10   const filePath = path.join(__dirname, 'hello.txt');
11   fs.writeFileSync(filePath, 'Hello Node!');
12   res.end('Fichier créé à: ' + filePath);
13 });
14
15 server.listen(3000, () => {
16   console.log('Serveur démarré sur http://localhost:3000');
17 });
18
```


ATELIER

Journal de bord



PREMIER PROGRAMME : JOURNAL DE BORD

 <https://github.com/dessna12/01-Decouverte-Node>

Enoncé

Nous allons créer un petit programme Node.js qui permet d'écrire des messages dans un fichier journal.txt, de les lire et d'ajouter éventuellement une heure.

1. Créer un fichier app.js contenant le script Node.js.

2. À chaque exécution :

- Demander un message à l'utilisateur (via readline).
- Ajouter ce message dans le fichier journal.txt
- Ensuite, afficher tous les messages enregistrés.

Bonus (optionnel)

- Ajouter la date et l'heure devant chaque message.
- Ne pas écrire de message vide.

Aides et commandes utiles

```
1 const readline =
  require('readline');
2
3 const rl =
  readline.createInterface({
4   input: process.stdin,
5   output: process.stdout
6 });
7
8 rl.question("Votre message :
  ", (answer) => {
9   console.log("Vous avez écrit
    :", answer);
10  rl.close();
11 });
```


1. IMPORTER LES MODULES NECESSAIRES

On commence par importer les modules natifs :

```
1 const fs = require('fs');  
2 const readline = require('readline');  
3 const path = require('path');  
4
```

2. DEFINIR LE CHEMIN VERS LE FICHIER JOURNAL

Utilisez `path.join()` pour générer un chemin compatible tous systèmes :

```
1 const journalPath = path.join(__dirname, 'journal.txt');
```

2. DEFINIR LE CHEMIN VERS LE FICHIER JOURNAL

Utilisez `path.join()` pour générer un chemin compatible tous systèmes :

```
1 const journalPath = path.join(__dirname, 'journal.txt');
```

3. CREER L'INTERFACE UTILISATEUR

Utilisez readline pour poser une question :

```
1 const rl = readline.createInterface({  
2   input: process.stdin,  
3   output: process.stdout  
4 });
```

4. SAUVEGARDER LE MESSAGE DANS LE FICHER

Création d'une fonction qui sauvegarde le message

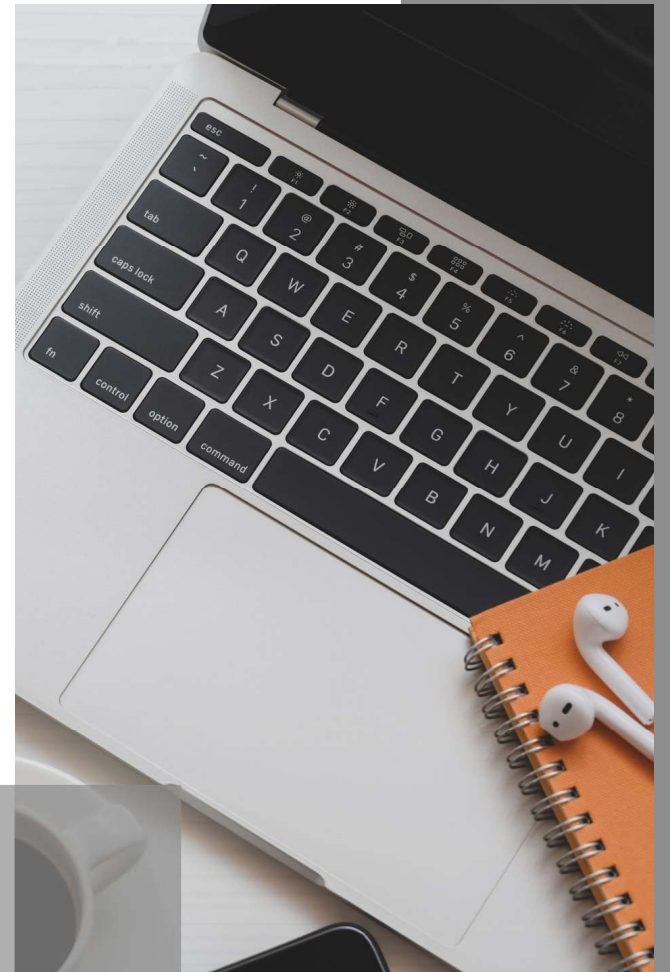
```
1 function sauvegarderMessage(message) {  
2   const ligne = `${message}\n`;  
3  
4   fs.appendFile(journalPath, ligne, (err) => {  
5     if (err) {  
6       console.error("Erreur :", err);  
7     } else {  
8       console.log(" Message enregistré.");  
9       afficherJournal();  
10    }  
11  });  
12 }
```

5. LIRE ET AFFICHER LE FICHIER

Création d'une fonction qui lis le fichier

```
1 function afficherJournal() {  
2   fs.readFile(journalPath, 'utf8', (err, data) => {  
3     if (err) {  
4       console.error("Erreur de lecture :", err);  
5     } else {  
6       console.log("\n Journal actuel :\n");  
7       console.log(data);  
8     }  
9     rl.close();  
10  });  
11 }
```

LES MODULES EXTERNES



LES MODULES EXTERNES ET NPM

Qu'est-ce que npm ?

npm signifie Node Package Manager

C'est l'outil qui permet :

- d'installer des librairies (ou packages) JavaScript
- de gérer les dépendances d'un projet Node.js
- de publier et partager son propre code

Il est installé automatiquement avec Node.JS



FICHER PACKAGE.JSON

C'est le cœur de tout projet Node.js

Il contient :

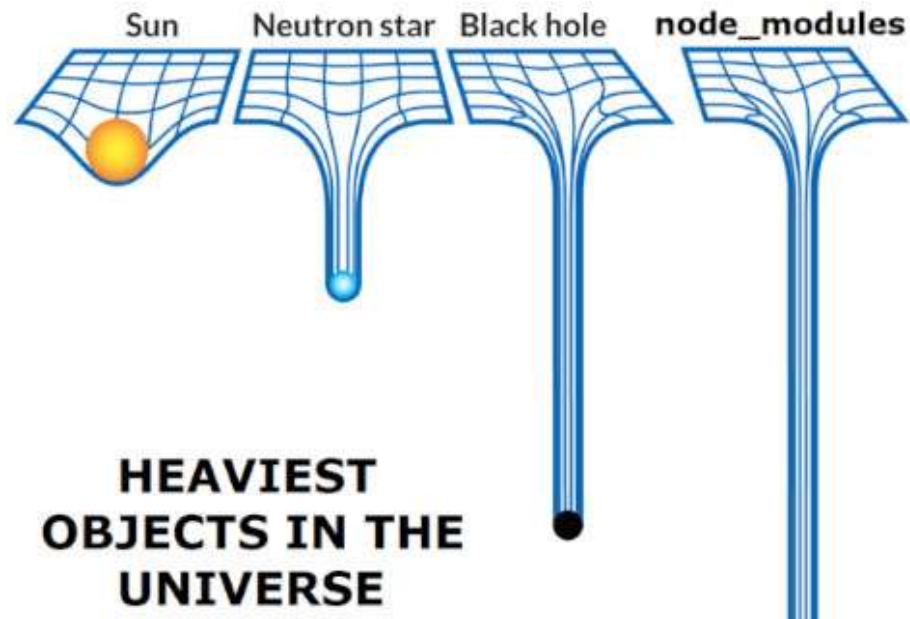
- Le nom et la version du projet
- Les scripts personnalisés (start, test, etc.)
- La liste des dépendances (librairies installées)

```
{
  "name": "package.json-mastery",
  "version": "1.0.0",
  "description": "Mastery of the package.json file",
  "private": false,
  "main": "index.js",
  "scripts": {
    "start": "node index",
    "dev": "nodemon index",
    "test": "jest"
  },
  "repository": {
    "type": "git",
    "url": "git+https://github.com/Easybuoy/package.json-mastery.git"
  },
  "keywords": [
    "node",
    "javascript",
    "npm",
    "yarn"
  ],
  "author": "Ezekiel Ekunola",
  "license": "ISC",
  "bugs": {
    "url": "https://github.com/Easybuoy/package.json-mastery/issues"
  }
}
```

MEMO NPM

Commande	Description
<i>npm init</i>	Initialise un projet Node
<i>npm install</i>	Installe un package ou tous les packages
<i>npm uninstall</i>	Supprime un package
<i>npm update</i>	Met à jour les packages
<i>npm run</i>	Exécute des scripts

BONNES PRATIQUES



- Ne jamais versionner le dossier node_modules
- Toujours committer le fichier package.json et package-lock.json
- Utiliser un .gitignore pour éviter d'envoyer les fichiers inutiles

ATELIER

Prise en main de npm



ASCII ART



<https://github.com/dessna12/01-Decouverte-Node>

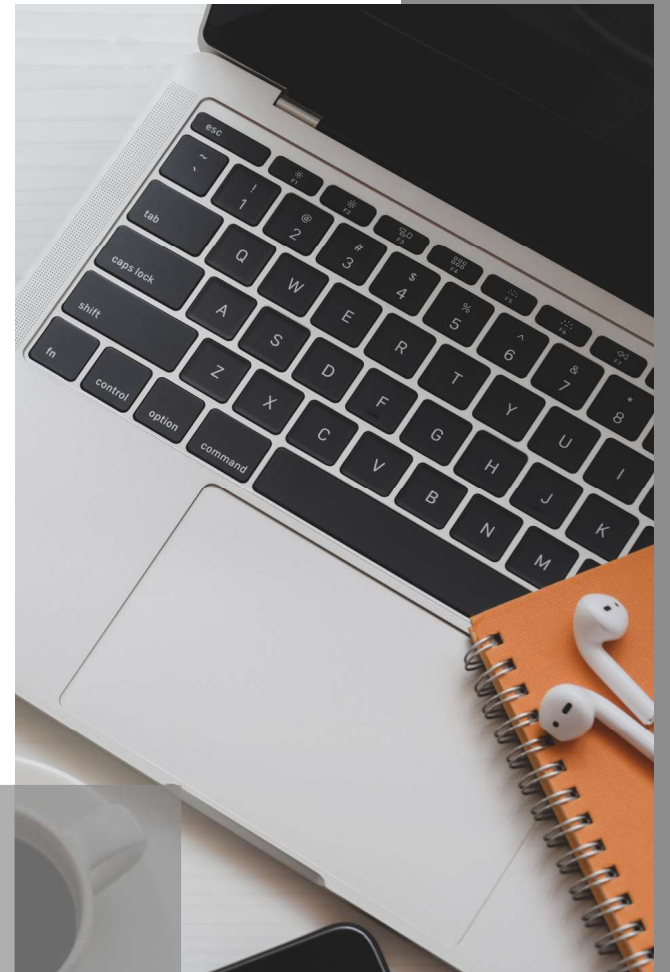
On se met sur la branche 02

Enoncé

Vous aimez le ASCII Art ? Ca va être l'occasion d'en faire ou de découvrir

1. Initialisez votre projet avec `npm init`
2. Importer le package `figlet`
3. Réalisez votre ASCII Art

LES MODULES UTILISATEURS



CUSTOM MODULES

Pourquoi utiliser des modules utilisateurs ?

- Eviter les fichiers trop longs.
- Réutiliser des fonctions dans plusieurs scripts.
- Organiser le code selon sa logique métier (fichiers, maths, réseau, etc.).

```
1 // math.js
2
3 // Fonction pour additionner deux nombres
4 function addition(a, b) {
5   return a + b;
6 }
7
8 // Fonction pour soustraire deux nombres
9 function soustraction(a, b) {
10  return a - b;
11 }
12
13 // On exporte les fonctions
14 module.exports = {
15   addition,
16   soustraction
17 };
```

UTILISATION DU MODULE

```
1 // app.js
2
3 // Import du module utilisateur
4 const math = require('./math');
5
6 console.log('Addition :', math.addition(3, 5)); // 8
7 console.log('Soustraction :', math.soustraction(10, 4)); // 6
```

On peut ensuite importer le module en précisant son emplacement

SYNTAXE MODERNE : REQUIRE VS IMPORT

- require() → syntaxe CommonJS, standard dans Node.js
- import / export → syntaxe ES Module, plus moderne (comme en front-end)

Tu dois :

- ajouter "type": "module" dans package.json pour utiliser import
- ou utiliser .mjs comme extension de fichier

```
1 import { addition } from './math.js';  
2 console.log('Soustraction :', math.soustraction(10, 4)); // 6
```

QUIZZ

