

1. Introduction

On me dit que JS est un langage super important et je n'arrive pas, comme dans Python¹, à faire un prompt dans VS.



Nous allons pour réponse à cette remarque, découvrir ce qu'est une application en node.js. Je parle d'une application et non d'un programme test de quelques lignes !

Mais, revenons au début de notre histoire et voyons comment écrire un code en nodeJS.

2. Nodejs



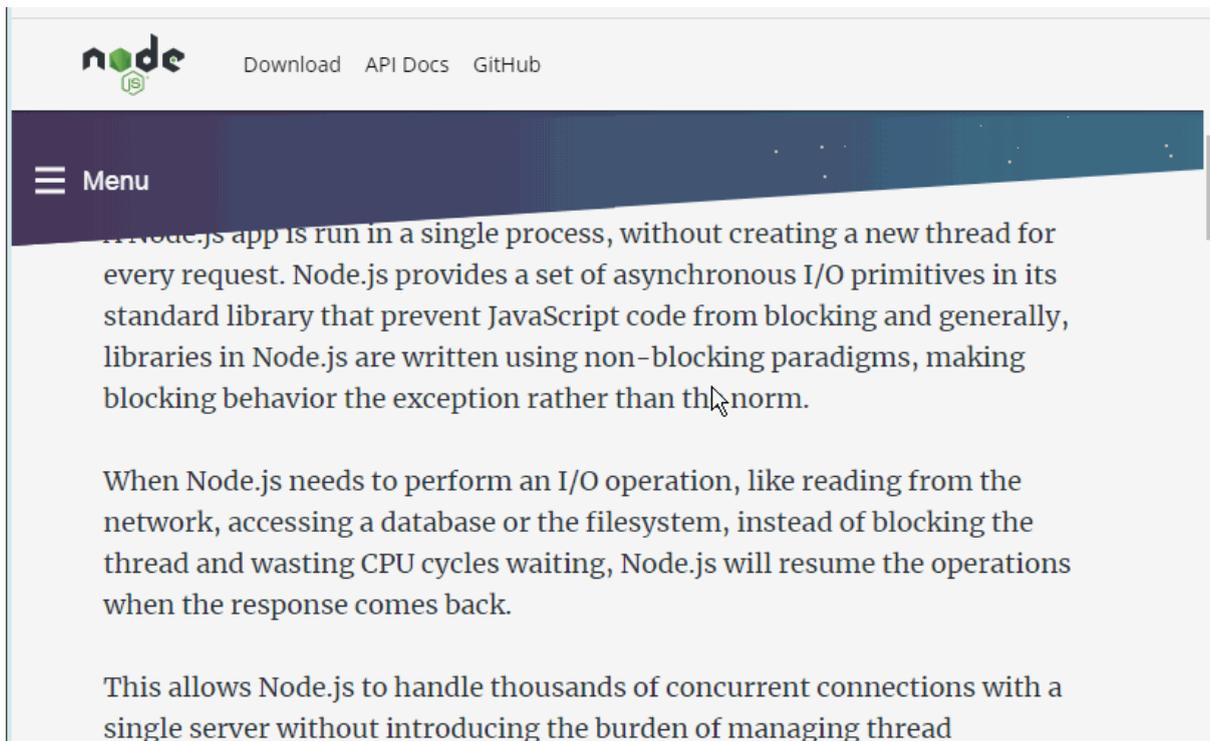
¹ Pas de prompt dans PythonTuor pour le langage JS

Comprendre nodejs est difficile pour le moment. Mais, on doit être très impressionné par sa puissance.

Voici, d'ailleurs, la preuve en code. Quelques lignes seulement² sont nécessaires pour créer un serveur ! ([code](#))

3. NPM

Dès l'introduction à nodejs, on se rend compte rapidement de l'importance de npm. ([doc](#))



Imaginez nodejs comme une cuisine quasiment vide. Vous pouvez ajouter plus de 1000 options ! Nous verrons que certaines options nécessitent d'autres options qui elles même nécessitent d'autres options. Pas de panique, c'est [npm](#) qui va gérer ce que nous appelons des dépendances.

Qu'est-ce que Node.js³ ?

Node.js est un environnement d'exécution JavaScript côté serveur qui exécute JavaScript côté serveur.

² Nous avons la même puissance en Python !

³

<https://docs.microsoft.com/fr-fr/visualstudio/javascript/tutorial-nodejs?view=vs-2019#before-you-begin>

Qu'est-ce que npm ?

npm est le gestionnaire de package par défaut de Node.js. Le Gestionnaire de package permet aux programmeurs de publier et partager plus facilement le code source des bibliothèques Node.js. Il est conçu pour simplifier l'installation, la mise à jour et la désinstallation des bibliothèques.

4. Prompt

Quelle ne fût pas ma surprise, de ne pas pouvoir utiliser prompt dans un fichier js.

Nous allons donc voir comment démarrer une application sereinement avec l'environnement Node.js dans Visual Studio. VS est un environnement de développement Node.js puissant.

Voici les étapes à suivre pour créer une application⁴.



De façon classique, tapez dans l'invite de commande

```
1. cmd
```

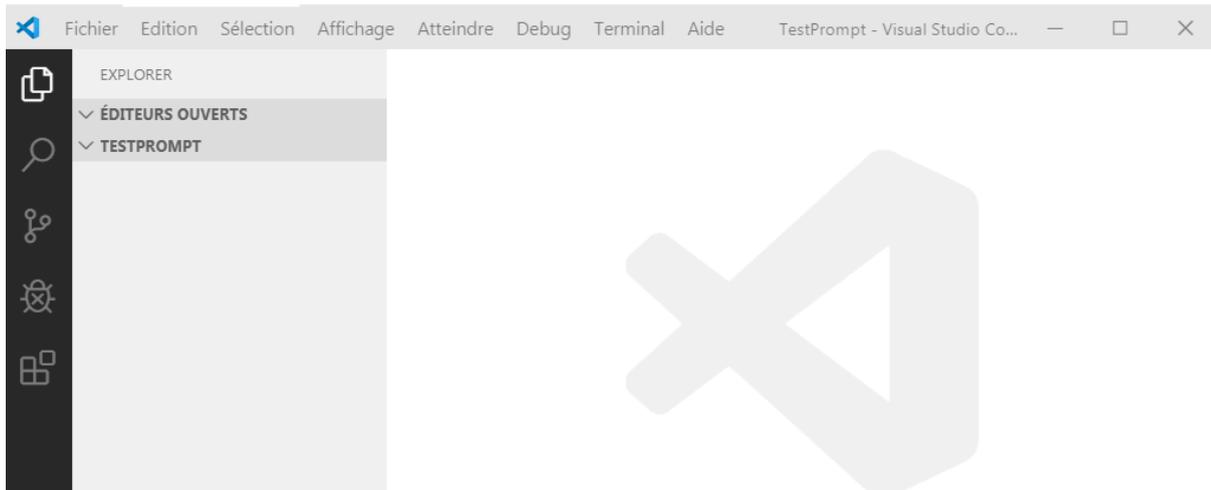
puis dans la nouvelle fenêtre :

```
1. mkdir 📁 TestPrompt
2. cd TestPrompt
3. code .
```

1. Vous créez un 📁 répertoire,
2. Vous allez dans ce répertoire et
3. Vous lancez VS.

Voici le résultat.

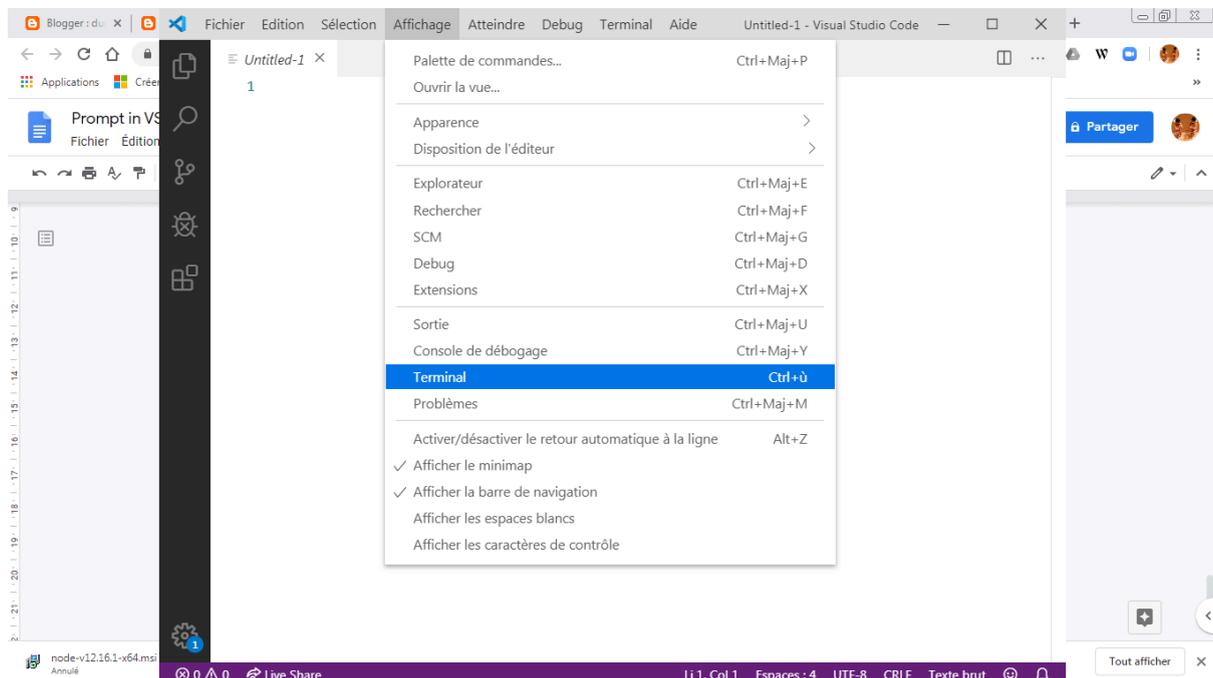
⁴ https://code.visualstudio.com/docs/nodejs/nodejs-tutorial#_hello-world



Autre méthode : l'éditeur est ouvert.

Notez que vous pouvez, si l'éditeur est déjà ouvert, utiliser directement un terminal dans lequel vous tapez les lignes 1-2.

Lancer le terminal Affichage->Terminal



Dans le terminal vous pouvez taper les commandes

1. mkdir 📁 TestPrompt
2. cd TestPrompt

```

denis.dupont@dd-HP MINGW64 ~
$ pwd
/c/Users/dd

denis.dupont@dd-HP MINGW64 ~
$ mkdir TestPrompt

denis.dupont@dd-HP MINGW64 ~
$ cd TestPrompt

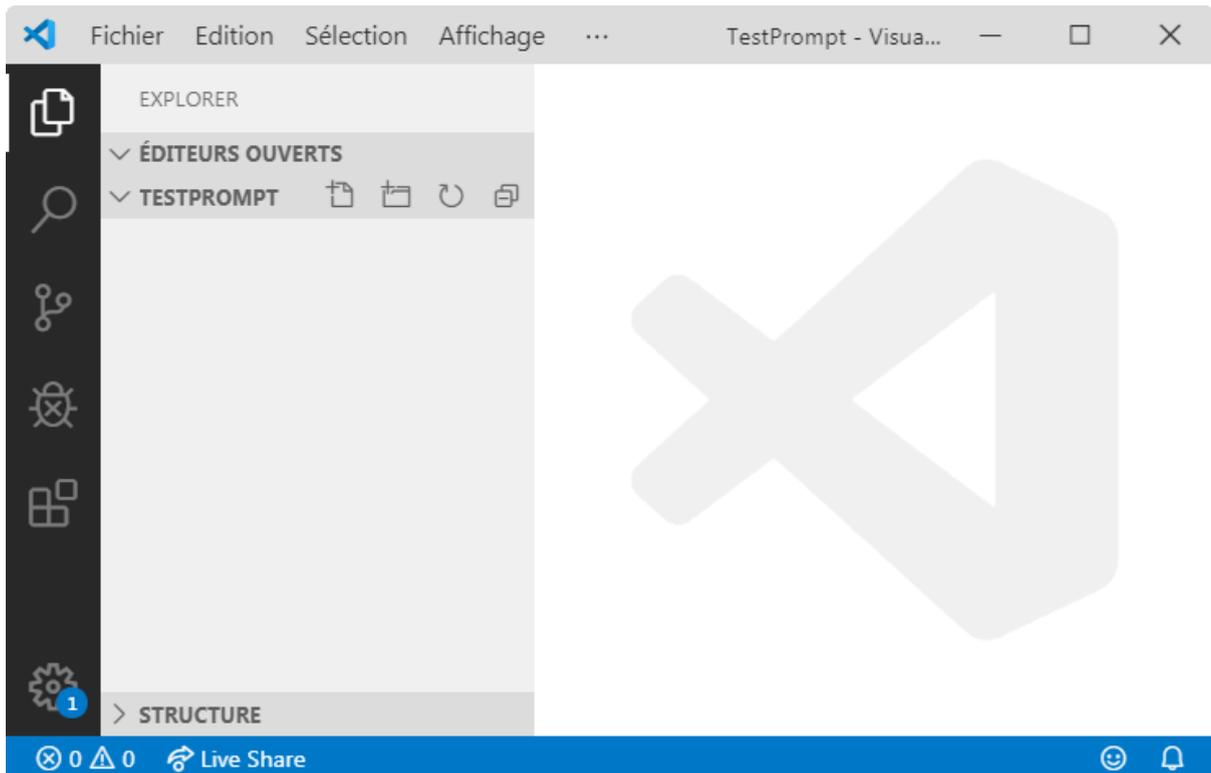
denis.dupont@dd-HP MINGW64 ~/TestPrompt
$ █
    
```

Maintenant, ouvrez le dossier 📁 dans lequel vous tapez le code (Fichier->Ouvrir le dossier) et recherchez le dossier TestPrompt.

```

Fichier  Edition  Sélection  Affichage  Atteindre  Debug  Terminal  Aide  Untitled-1 - Visual Studio Code
┌─ Nouveau fichier          Ctrl+N
├─ Nouvelle fenêtre       Ctrl+Maj+N
├─ Ouvrir le fichier...   Ctrl+O
├─ Ouvrir le dossier...   Ctrl+K Ctrl+O
├─ Ouvrir l'espace de travail...
├─ Ouvrir les éléments récents >
├─ Ajouter un dossier à l'espace de travail...
├─ Enregistrer l'espace de travail sous...
├─ Enregistrer            Ctrl+S
├─ Enregistrer sous...   Ctrl+Maj+S
├─ Enregistrer tout      Ctrl+K S
├─ ✓ Enregistrement automatique
├─ Préférences           >
├─ Rétablir le fichier
├─ Fermer l'éditeur      Ctrl+F4
├─ Fermer le dossier     Ctrl+K F
├─ Fermer la fenêtre     Ctrl+W
└─ Quitter
    
```

Vous serez dans la situation suivante :



L'aventure va maintenant commencer. Nous allons mettre en place une application. Et commençons par la création d'un fichier essentiel.

5. Création d'un fichier `package.json`

Mais c'est quoi un package ?

package ?

Une recherche rapide, montre que "prompt" existe dans nodejs sous forme de package ([lien](#)). Mais la documentation est pour l'instant très peu compréhensible (cela viendra).

Finalement, je comprends vite que la commande magique est : `npm install prompt`

Mais, pas si vite, il nous faut créer un fichier de configuration⁵. Ce fichier a pour nom : `package.json`.

⁵ Nous avons vu dans le cas du débogger la nécessité de créer également un fichier de configuration.

package.json ?

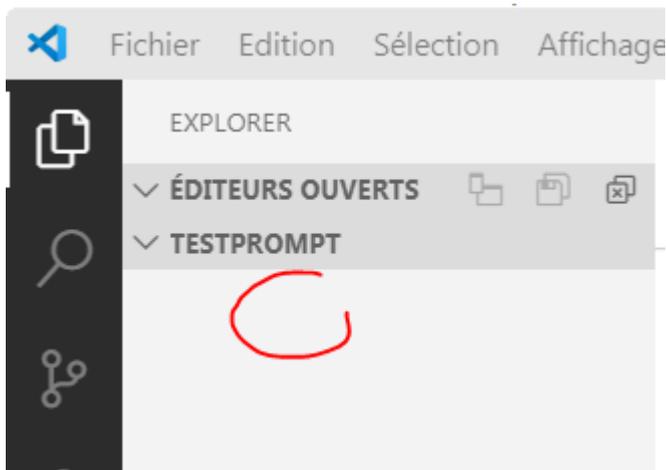
Voici une réponse à la question : "ça sert à quoi ce fichier json, un extrait de la documentation"⁶.

"You can add a package.json file to your package to make it easy for others to manage and install. Packages published to the registry must contain a package.json file."

Ce fichier magique va indiquer les dépendances entre les autres packages pour que npm puisse gérer l'ensemble des dépendances.

Initialisation

Avant de commencer le processus d'initialisation, observez bien le contenu vide du répertoire que vous venez de créer. A l'issue de la phase d'initialisation, il contiendra le fichier de configuration.

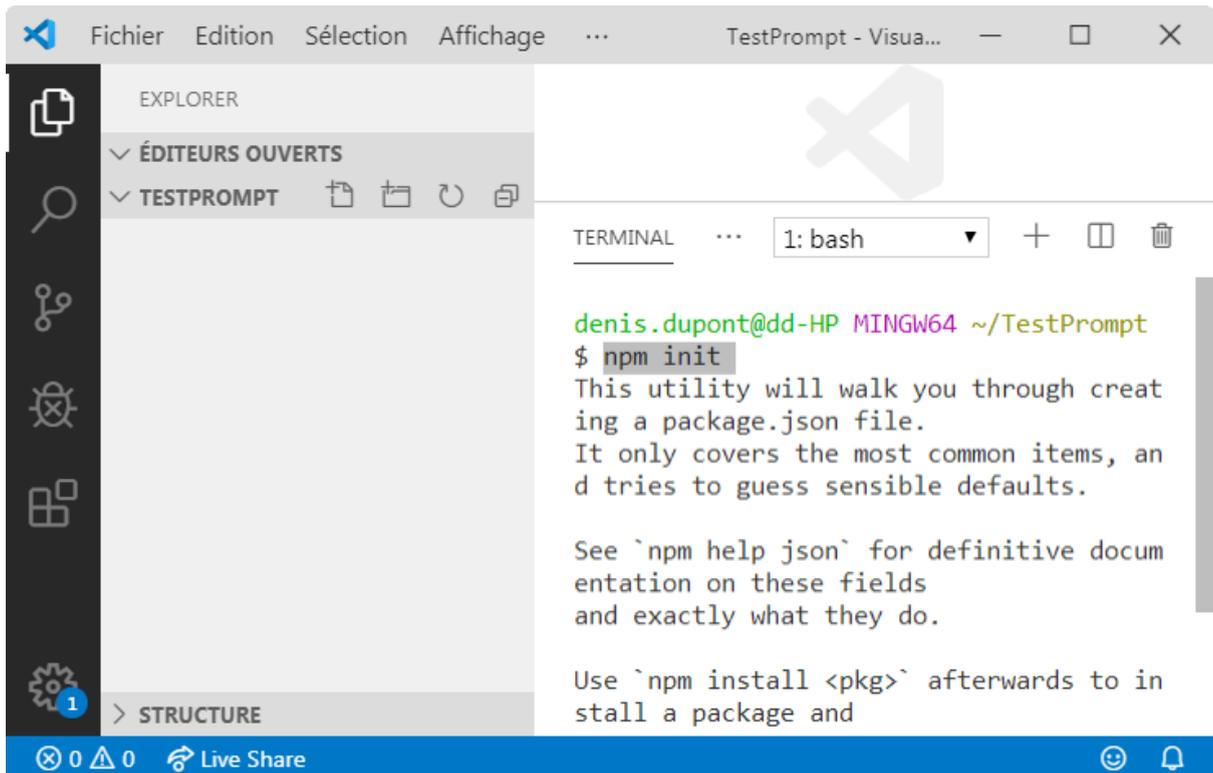


Pour créer le fichier package.json, nous facilite la tâche.

npm monte la structure.

Dans le terminal taper : **npm init**

⁶ <https://docs.npmjs.com/creating-a-package-json-file>



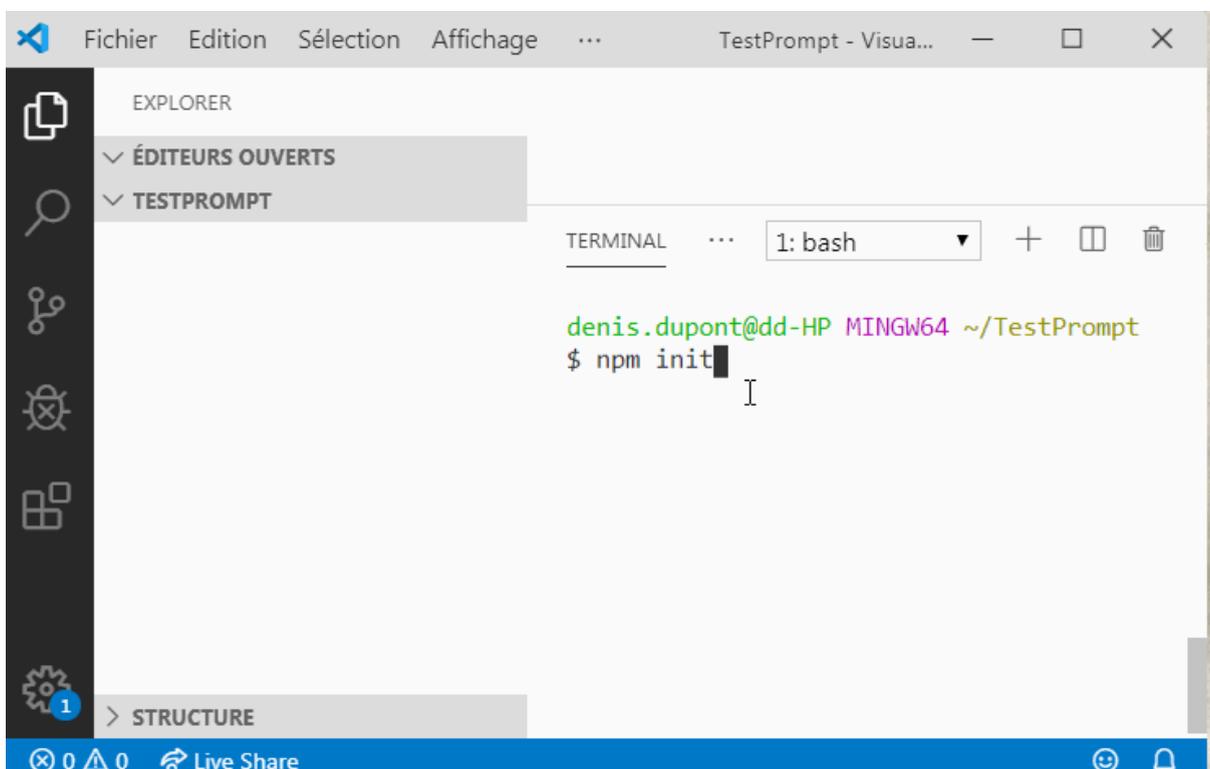
The screenshot shows the Visual Studio Code interface with a terminal window open. The terminal title is '1: bash'. The output of the 'npm init' command is displayed, including the prompt 'denis.dupont@dd-HP MINGW64 ~/TestPrompt', the command '\$ npm init', and the following text: 'This utility will 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'.

```
denis.dupont@dd-HP MINGW64 ~/TestPrompt
$ 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
```

Vous répondez entrée ↵ à chaque question posée. Nous reviendrons sur la valeur des attributs.



The screenshot shows the Visual Studio Code interface with a terminal window open. The terminal title is '1: bash'. The output of the 'npm init' command is displayed, including the prompt 'denis.dupont@dd-HP MINGW64 ~/TestPrompt' and the command '\$ npm init'. The cursor is positioned at the end of the command, ready for input.

```
denis.dupont@dd-HP MINGW64 ~/TestPrompt
$ npm init
```

Observez bien sur la partie de gauche en dessous de TESTPROMPT, l'apparition de votre fichier  package.json.

Vous pouvez éditer le fichier⁷ :

 package.json.

```
{  
  "name": "testprompt",  
  "version": "1.0.0",  
  "description": "",  
  "main": "index.js",  
  "scripts": {  
    "test": "echo \"Error: no test specified\" && exit 1"  
  },  
  "author": "",  
  "license": "ISC"  
}
```

Après l'initialisation, vient le temps de l'installation de notre package.

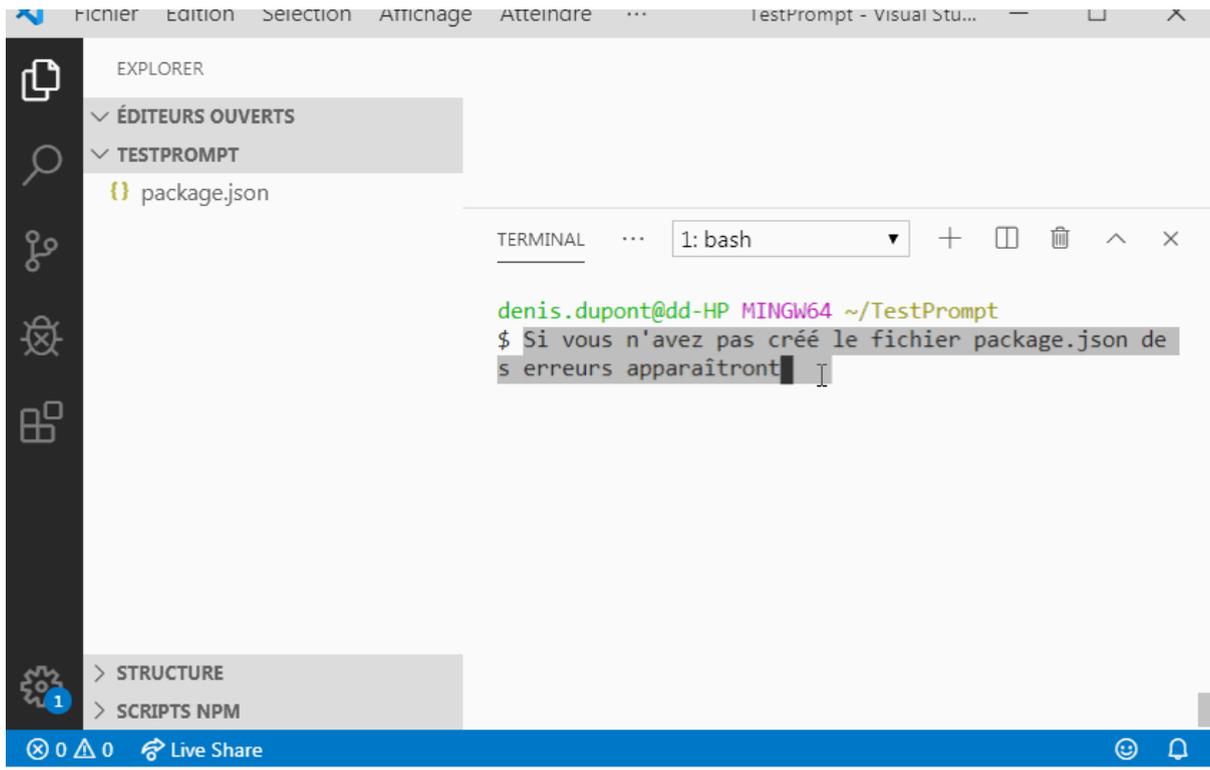
6. Installation du package prompt

Nous allons simplement taper dans le terminal⁸ :

```
npm install prompt
```

⁷ Sa lecture reste encore floue.

⁸ Si vous n'avez pas créé le fichier package.json des erreurs apparaîtront.



Je n'arrive pas encore à comprendre ce qui se passe, mais je vois que mon

fichier  package.json est devenu

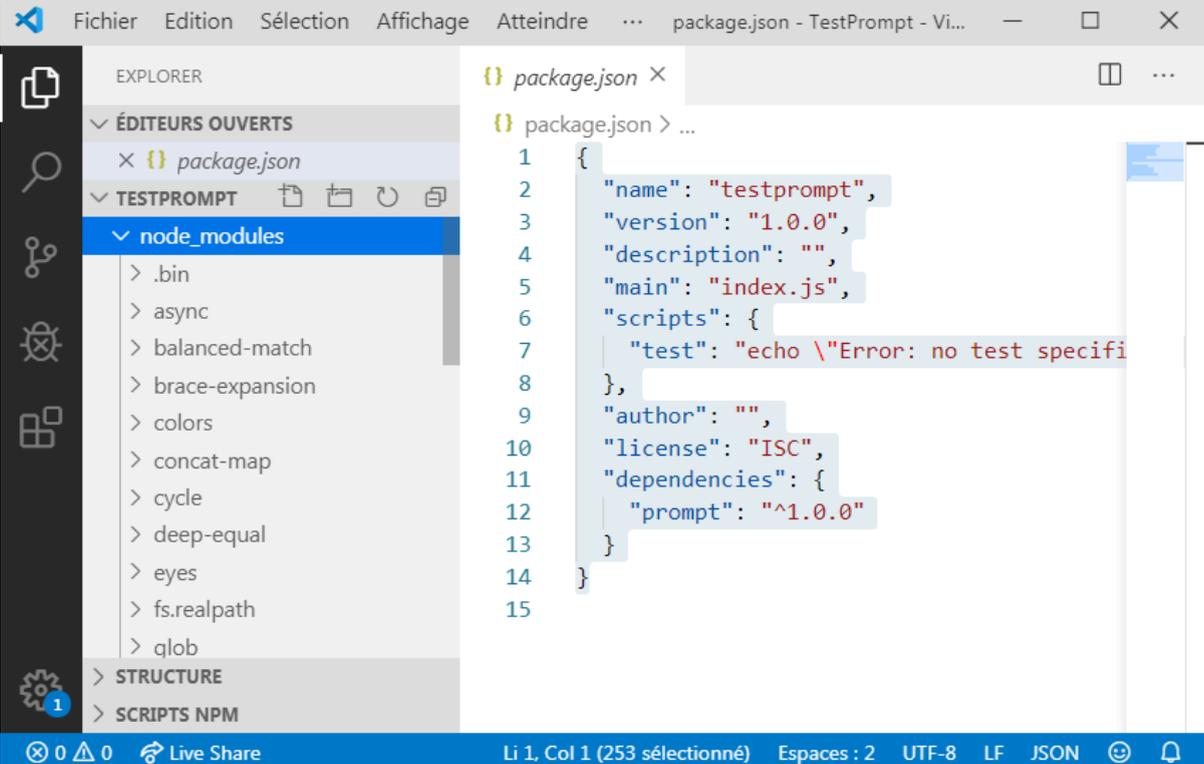
 package.json.

```
{  
  "name": "testprompt",  
  "version": "1.0.0",  
  "description": "",  
  "main": "index.js",  
  "scripts": {  
    "test": "echo \"Error: no test specified\" && exit 1"  
  },  
  "author": "",  
  "license": "ISC",  
  "dependencies": {  
    "prompt": "^1.0.0"
```

```
}
}
```

On a bien ajouté une dépendance au package "prompt".

Je vois également qu'un répertoire  node_modules a été créé⁹.



The screenshot shows the Visual Studio Code interface. On the left, the Explorer view shows the 'node_modules' directory expanded, listing various packages like '.bin', 'async', 'balanced-match', 'brace-expansion', 'colors', 'concat-map', 'cycle', 'deep-equal', 'eyes', 'fs.realpath', 'qlob', 'STRUCTURE', and 'SCRIPTS NPM'. The main editor shows the 'package.json' file with the following content:

```
1 {
2   "name": "testprompt",
3   "version": "1.0.0",
4   "description": "",
5   "main": "index.js",
6   "scripts": {
7     "test": "echo \\\"Error: no test speci
8   },
9   "author": "",
10  "license": "ISC",
11  "dependencies": {
12    "prompt": "^1.0.0"
13  }
14 }
15
```

Il contient plus de trente packages. Ces packages sont nécessaires au fonctionnement de prompt ! Impressionnant.

Je vois le package **prompt**.

⁹ Je reviendrai sur l'apparition d'un fichier package-lock.json

```

node_modules > prompt > lib > JS prompt.js > ...
14     winston = require('winston'),
15     colors = require('colors/safe');
16
17     //
18     // Monkey-punch readline.Interface to work
19     // https://github.com/joyent/node/issues/3
20     //
21     readline.Interface.prototype.setPrompt = function(prompt) {
22         this._prompt = prompt;
23         if (length) {
24             this._promptLength = length;
25         } else {
26             var lines = prompt.split(/\r\n/);
27             var lastLine = lines[lines.length - 1];
28             this._promptLength = lastLine.replace(
29             );
30     };
31
32     //

```

Le fichier  prompt.js ne fait que 780 lignes de code sans compter les dépendances.

Les dépendances sont écrites de la ligne 8-15.

 prompt.js

```

var events = require('events'),
    readline = require('readline'),
    utile = require('utile'),
    async = utile.async,
    read = require('read'),
    validate = require('revalidator').validate,
    winston = require('winston'),
    colors = require('colors/safe');

```

Et chacun de ces packages (events, readline, ...) dépend d'autres packages qui eux mêmes ... Au secours¹⁰ !

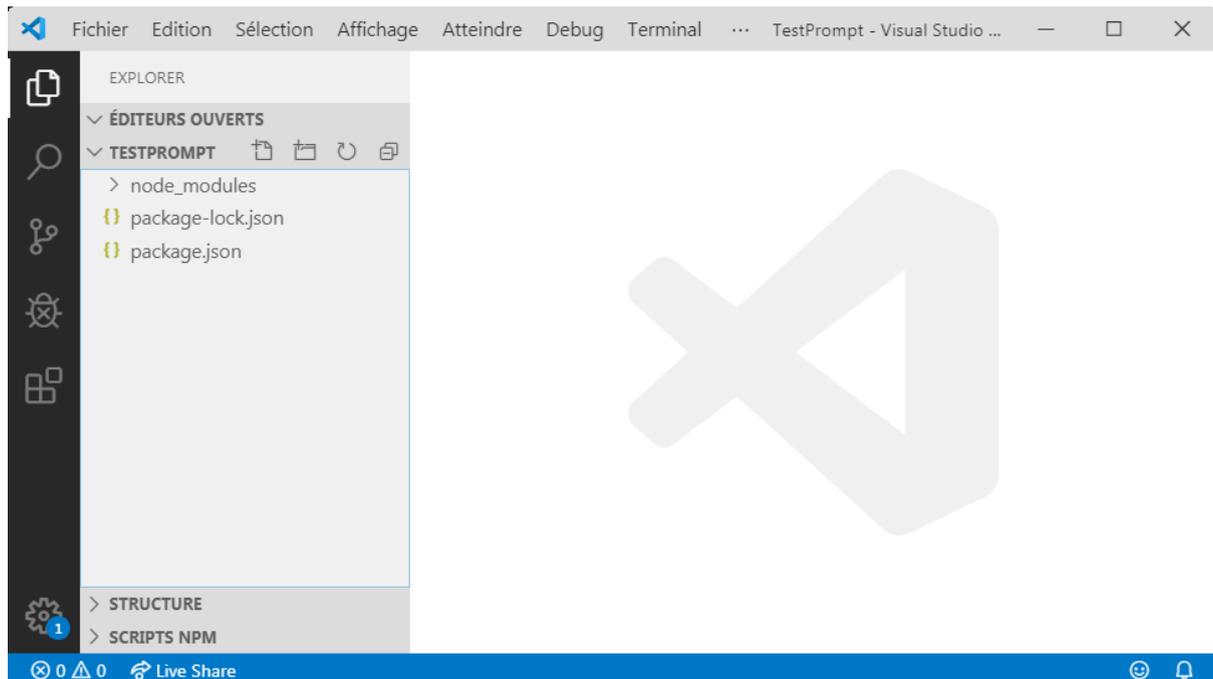
¹⁰ C'est dans tous les cas très impressionnant ! *Moi qui râlais de n'avoir pas prompt dans pythontutor !*

7. Ecriture de notre code

En fait, dans quelques temps ... nous ne nous poserons plus de questions et l'installation des packages sera très naturelle.

Nous allons maintenant écrire notre code. Pour cela, ouvrez un terminal.

La structure doit être celle de la figure suivante.



8. Edition d'un fichier

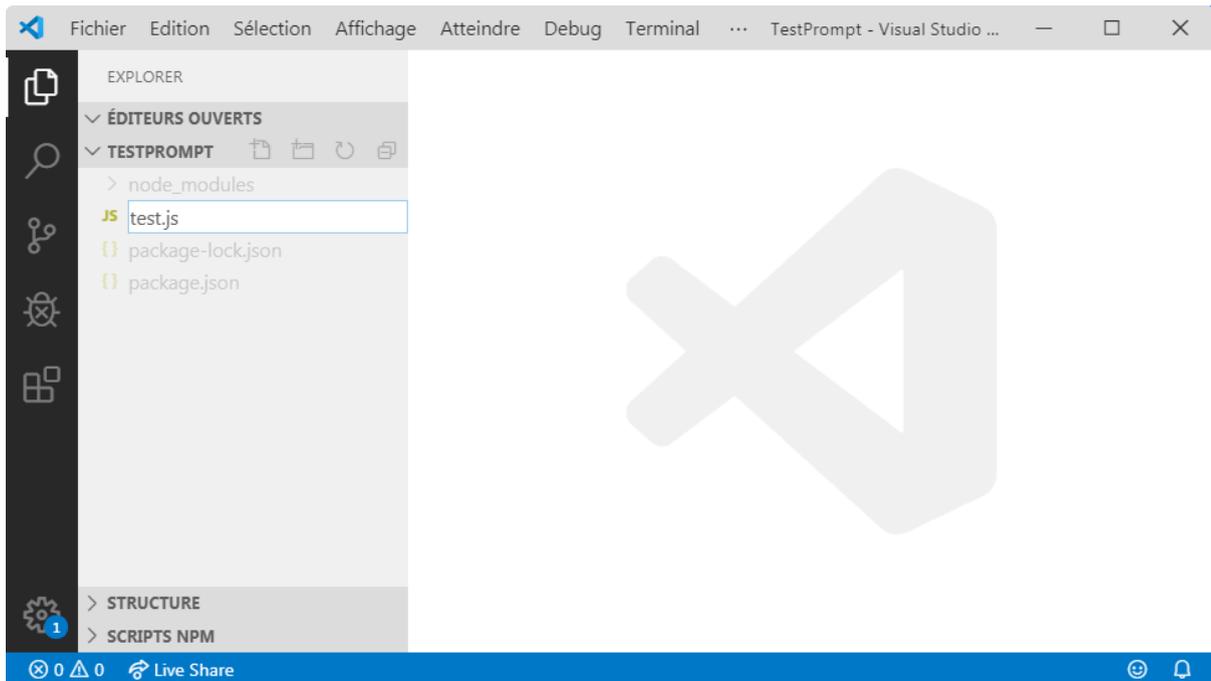
Nous allons créer un premier fichier .

Vous pouvez soit

- cliquer sur l'icône  Nouveau fichier ou
- tapez `ctrl-n` ou
- dans l'interface faire **Fichier->Nouveau fichier**

Donnez au fichier le nom (par exemple `test`) que vous voulez avec

l'extension `.js` .



Dans le fichier  test.js coller le code trouvé dans le fichier [\(lien\)](#)

 test.js

```

1. const prompt = require('prompt');
2.
3. prompt.start();
4.
5. prompt.get(['username', 'email'], function (err, result) {
6.     if (err) { return onErr(err); }
7.     console.log('Command-line input received:');
8.     console.log('  Username: ' + result.username);
9.     console.log('  Email: ' + result.email);
10. });
11.
12. function onErr(err) {
13.     console.log(err);
14.     return 1;
15. }

```

Aie ! C'est compliqué pour le moment.

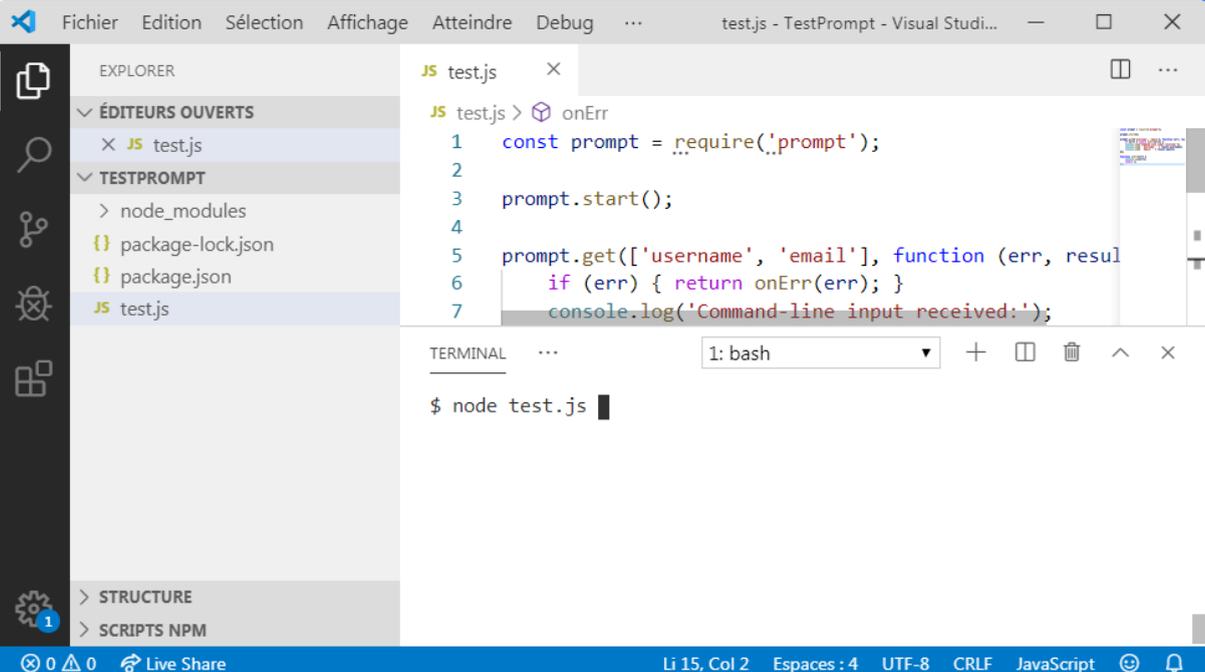
Disons simplement que

lig. 1 : on veut utiliser le package.

lig. 3 : on lance le prompt

Passons à l'exécution du code. Ouvrons un terminal (Ctrl+ù) et lançons le test avec la commande **node test.js**.

> **node test.js**



```
Fichier Edition Sélection Affichage Atteindre Debug ... test.js - TestPrompt - Visual Studi...  
EXPLORER  
ÉDITEURS OUVERTS  
  JS test.js  
TESTPROMPT  
  > node_modules  
  {} package-lock.json  
  {} package.json  
  JS test.js  
STRUCTURE  
SCRIPTS NPM  
0 0 Live Share Li 15, Col 2 Espaces : 4 UTF-8 CRLF JavaScript  
JS test.js  
JS test.js > onErr  
1 const prompt = require('prompt');  
2  
3 prompt.start();  
4  
5 prompt.get(['username', 'email'], function (err, resul  
6     if (err) { return onErr(err); }  
7     console.log('Command-line input received:');
```

Une fois lancé, il vous reste à répondre aux questions posées.

```

Fichier  Edition  Sélection  Affichage  Atteindre  Debug  ...  test.js - TestPrompt - Visual Studi...
EXPLORER
ÉDITEURS OUVERTS
  X JS test.js
TESTPROMPT
  > node_modules
  {} package-lock.json
  {} package.json
  JS test.js
STRUCTURE
  > SCRIPTS NPM
TERMINAL
1: bash
$ node test.js
prompt: username: Dupont
prompt: email: univ@evry
Command-line input received:
  Username: Dupont
  Email: univ@evry
denis.dupont@dd-HP MINGW64 ~/TestPrompt
$

```

Le test est concluant. Je peux le modifier pour tenter de mieux comprendre le code.

Voici quelques exemples de code

test-1.js

```

1. const prompt = require('prompt');
2.
3. const MAX = 100;
4.
5. prompt.start();
6.
7. let texte = 'Entrez un nombre entre 0 et ${MAX}`
8. console.log(texte);
9.
10. prompt.get([\`val\`], function (err, result) {
11.   if (err) { return onErr(err); }
12.   console.log(\`Votre choix est ${result.val}\`);
13.
14. });
15.
16. function onErr(err) {
17.   console.log(err);

```

```
18.     return 1;
19. }
```

Voici un autre exemple de code :



test-2.js

```
1. const prompt = require('prompt');
2.
3. //
4. // Start the prompt
5. //
6. prompt.start();
7.
8. //
9. // Get one property from the user: Guess
10. //
11. prompt.get([
12.   {
13.     name: 'guess',
14.     validator: /^[0-9]$/,
15.     warning: 'Guess should consist only one [0-9]',
16.     empty: false
17.   }
18. ], function (err, result) {
19.   //
20.   // Log the results.
21.   //
22.   console.log('Command-line input received:');
23.   console.log('  Guess: ' + result.guess);
24. });
```

Parfait, mais mon ignorance¹¹, fait que j'ai du mal à comprendre quelque chose de très important !

AIE ...

¹¹ le mode asynchrone qui fait la force de nodejs !

Je vais écrire un exemple pour vous montrer le comportement asynchrone du code en nodejs !

 test-3.js

```

1. console.log("- A")
2. prompt.get(['val'], function (err, result) {
3.     if (err) { return onErr(err); }
4.     console.log(`Votre choix est ${result.val}`);
5.     console.log("-- B")
6. });
7.
8. console.log("--- C");

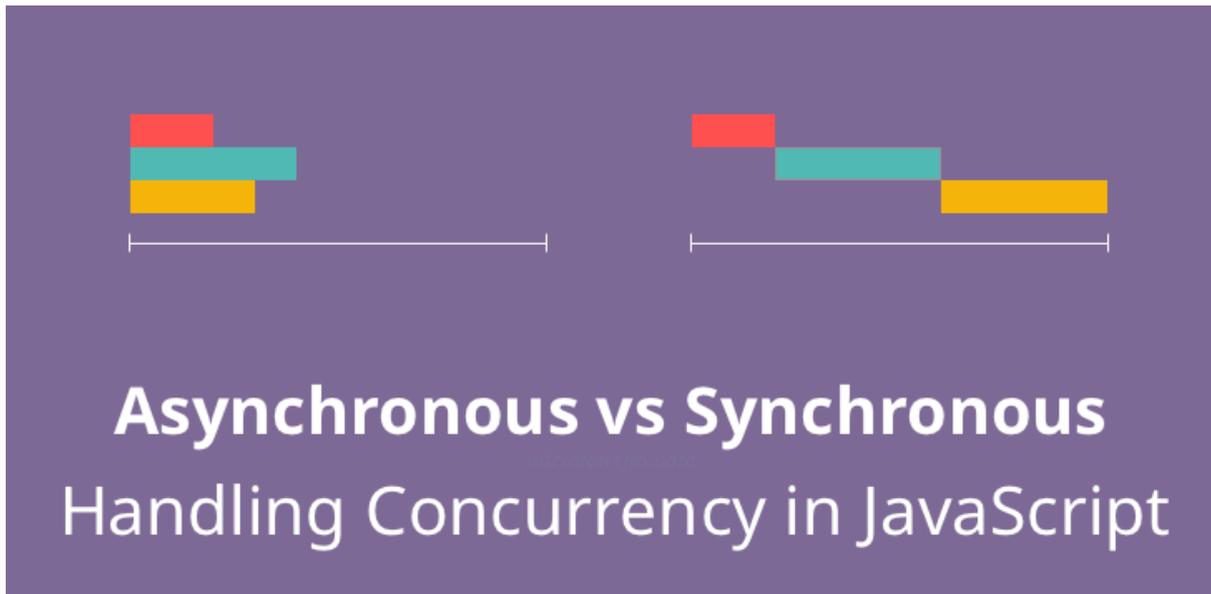
```

Voici l'affichage obtenu en fonction du style de programmation :

Synchrone	Asynchrone
<pre> 1. - A 2. prompt: val:2 3. Votre choix est 2 4. -- B 5. --- C </pre>	<pre> 1. - A 2. prompt: val: --- C 3. 2 4. Votre choix est 2 5. -- B </pre>

On peut admettre que dans un comportement asynchrone la ligne 8 n'attend pas le retour du prompt pour s'exécuter. C'est très très utile en programmation.

En image, cela peut ressembler à cela. [\(ref\)](#)



La puissance de ce type de code est incontestable.

Nous n'attendons pas le résultat, nous pouvons faire autre chose.

Mais, j'y pense dans le cas de [notre jeu GUESS](#), c'est embêtant. Car si on n'attend pas que l'utilisateur donne son chiffre pour le tester, cela ne va pas fonctionner ! Et bien, vous avez tout compris, le style non bloquant rend le problème ~~quasi-impossible~~ plus difficile à résoudre.

9. sync-prompt

Nous allons reprendre l'ensemble du code en installant un package non bloquant ¹².

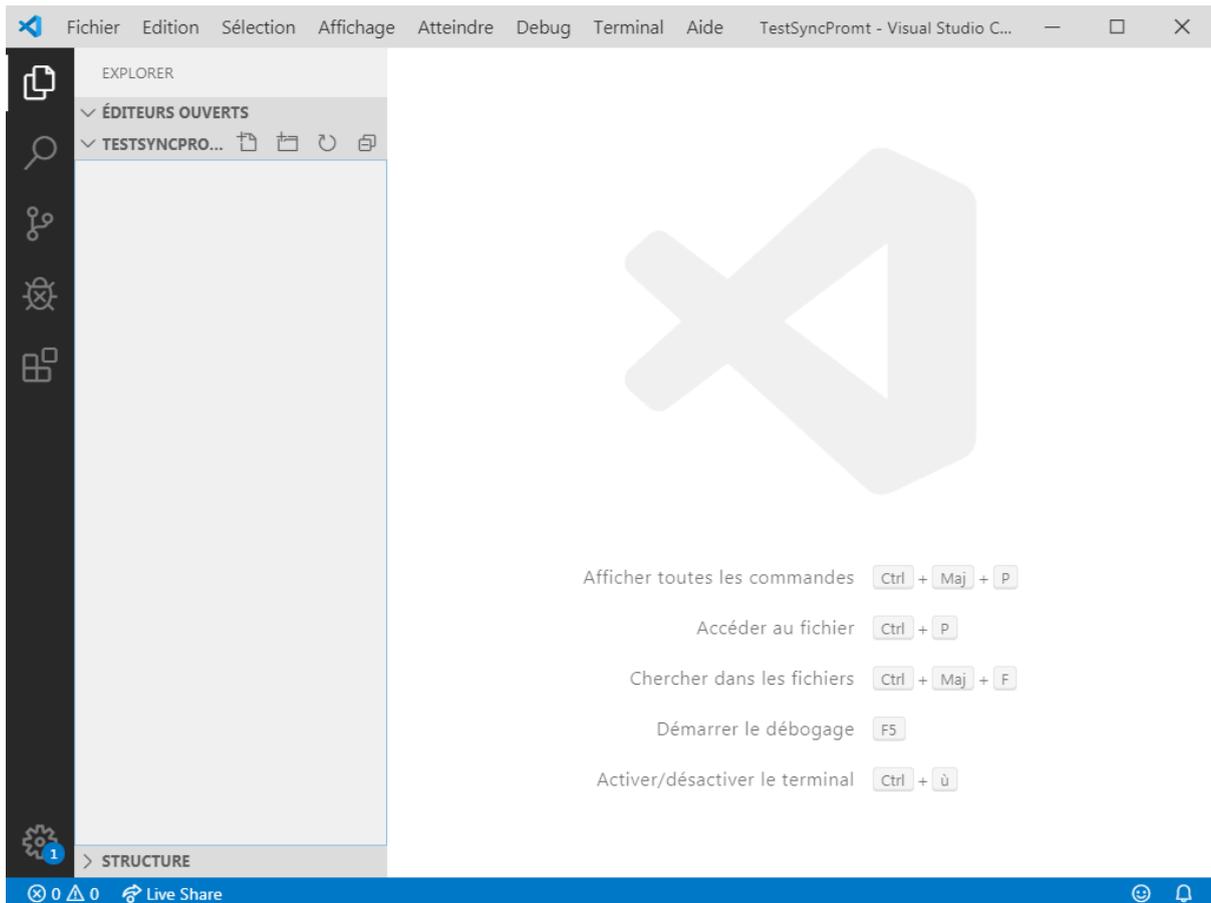
Après avoir ouvert un terminal tapez

```
$ mkdir 📁 TestSyncPromt
```

```
$ cd TestSyncPromt/
```

```
$ code .
```

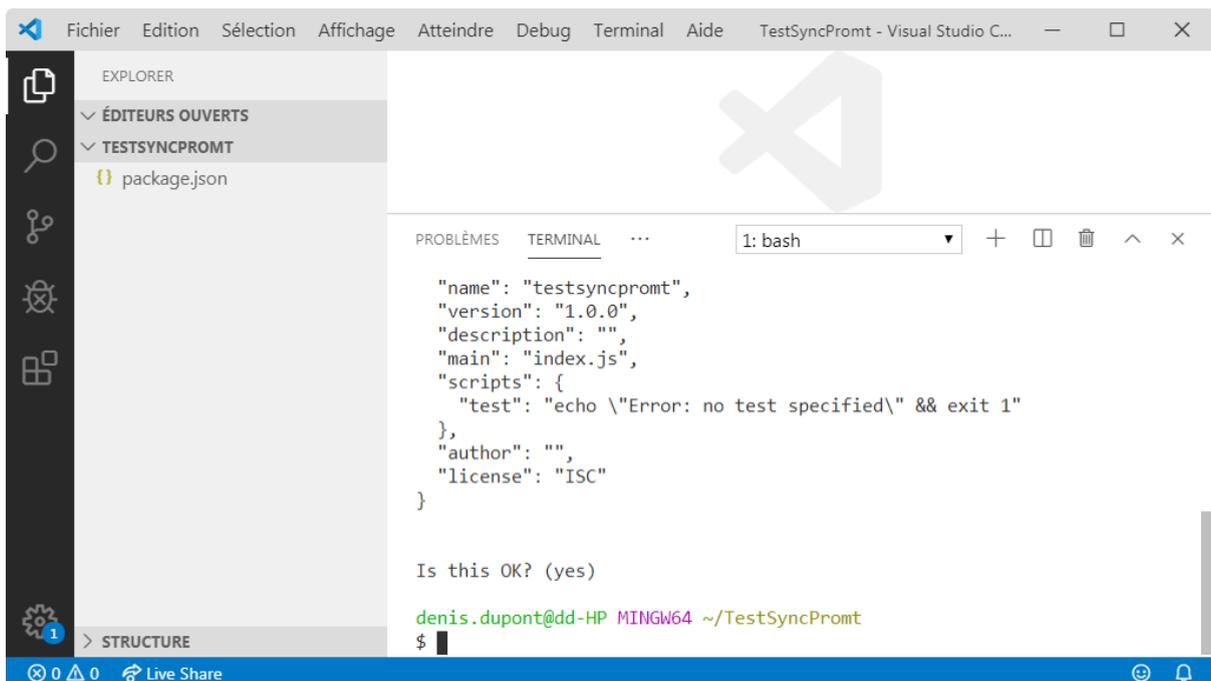
¹² Nous pourrions désinstaller le package précédent avec `> npm uninstall prompt`



tapez

`$ npm init`

et `↵` pour l'ensemble des questions.



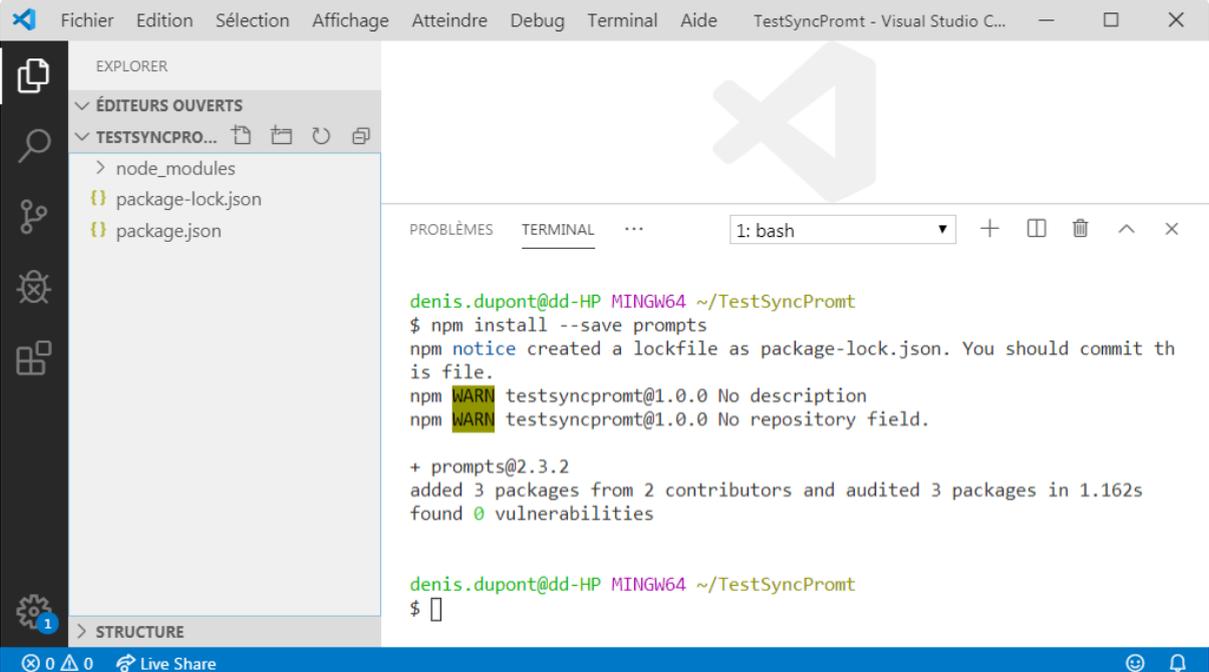
Vérifiez bien que le fichier  package.json est présent.

Nous allons charger le package suivant :

<https://www.npmjs.com/package/prompts>

Dans le terminal

```
$ npm install --save prompts
```



The screenshot shows the Visual Studio Code interface with a terminal window open. The Explorer sidebar on the left shows the file structure of a project named 'TESTSYNCPRO...', including 'node_modules', 'package-lock.json', and 'package.json'. The terminal window shows the following output:

```
denis.dupont@dd-HP MINGW64 ~/TestSyncPromt
$ npm install --save prompts
npm notice created a lockfile as package-lock.json. You should commit this file.
npm WARN testsyncprompt@1.0.0 No description
npm WARN testsyncprompt@1.0.0 No repository field.

+ prompts@2.3.2
added 3 packages from 2 contributors and audited 3 packages in 1.162s
found 0 vulnerabilities

denis.dupont@dd-HP MINGW64 ~/TestSyncPromt
$
```

Dans un nouveau fichier  test.js tapez le code suivant :

 test.js

```
1. const prompts = require('prompts');
2.
3. (async () => {
4.   const response = await prompts({
5.     type: 'number',
6.     name: 'value',
7.     message: 'How old are you?',
8.     validate: value => value < 18 ? `Nightclub is 18+ only` : true
9.   });
10.
11. console.log(response);
12.})();
```

Sans rentrer dans le détail, remarquons les deux mots clefs en gras `async` et `await`¹³. Cela oblige à attendre la saisie pour continuer.

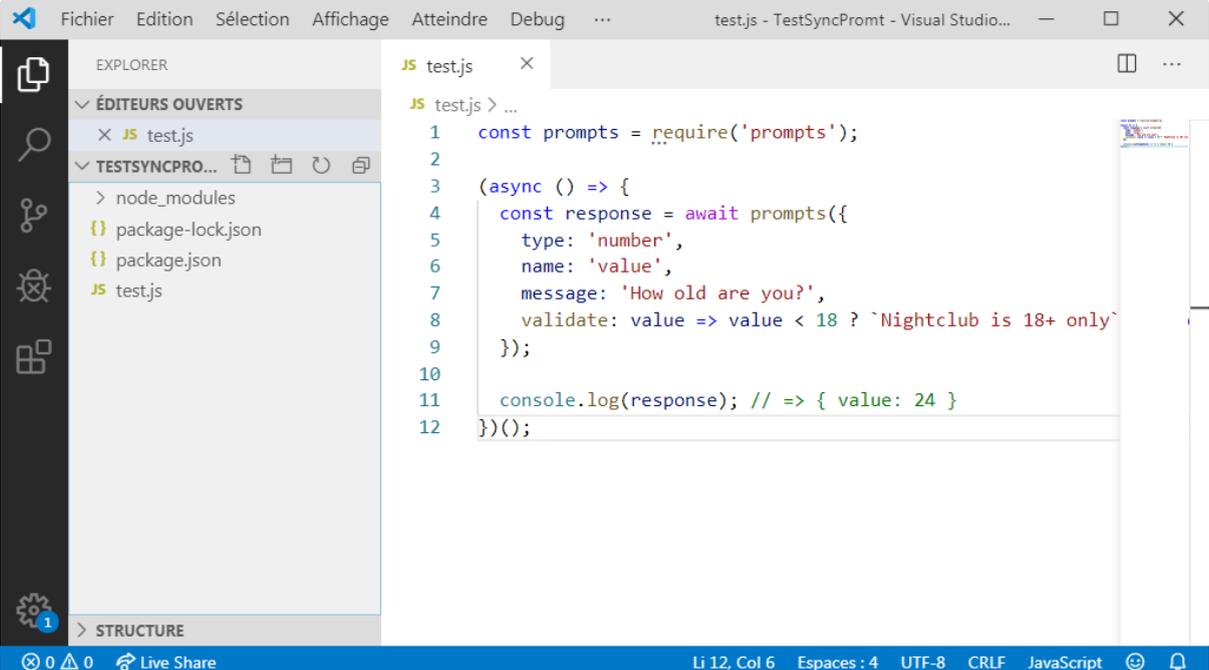
Ainsi l'exécution de

```

13. const prompts = require('prompts');
14. console.log('A');
15. (async () => {
16.   const response = await prompts({});
17.
18.   console.log('C');
19. })();

```

donnera à affichage dans cet ordre : A -> response -> C



```

Fichier Edition Sélection Affichage Atteindre Debug ... test.js - TestSyncPromt - Visual Studio...
EXPLORER
ÉDITEURS OUVERTS
  X JS test.js
TESTSYNCPRO...
  > node_modules
  {} package-lock.json
  {} package.json
  JS test.js
STRUCTURE
  > STRUCTURE

JS test.js
1 const prompts = require('prompts');
2
3 (async () => {
4   const response = await prompts({
5     type: 'number',
6     name: 'value',
7     message: 'How old are you?',
8     validate: value => value < 18 ? `Nightclub is 18+ only`
9   });
10
11   console.log(response); // => { value: 24 }
12 })();

```

Vous pouvez tester le code avec :

```
$ node test.js
```

```
√ How old are you? ... 24
```

¹³

<https://developer.mozilla.org/fr/docs/Web/JavaScript/Reference/Op%C3%A9rateurs/await>

```
{ value: 24 }
```

Nous allons revenir au jeu Guess ([lien](#))

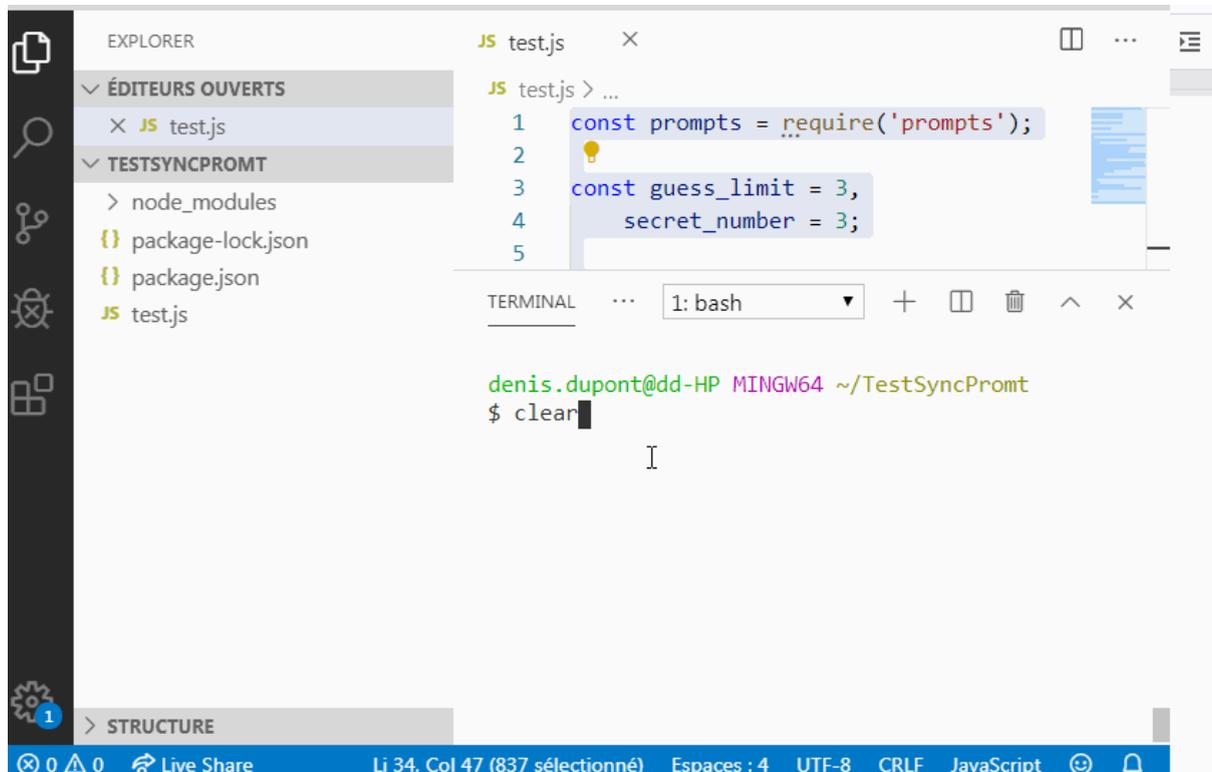
Voici le code permettant de jouer dans VS.

 guess.js

```
1. const prompts = require('prompts');
2.
3. const guess_limit = 3,
4.     guess_secret = 3;
5.
6. let game_over = false;
7.
8. (async () => {
9.
10.     for (let guess_turns=0; guess_turns < guess_limit; guess_turns++){
11.
12.         game_over = true;
13.
14.         const {guess} = await prompts({
15.             type: 'number',
16.             name: 'guess',
17.             message: 'Quel est votre choix ?',
18.             validate: guess => guess > 10 ? `Un chiffre SVP [0-9]` : true
19.         });
20.         if (guess == guess_secret){
21.             console.log(`YOU WIN with : ${++guess_turns} turns `);
22.             game_over = false;
23.             break;
24.         }
25.         console.log(`Guess_turns: ${guess_turns+1}`);
26.
27.
28.     } if ( game_over ){
29.         console.log("GAME OVER");
30.     }
31.
```

```
32.})();  
33.  
34.console.log("vous êtes en zone asynchrone ! ")
```

Remarquez bien la paire (async, await) en lig. 8 et 14.



Nous reviendrons dans les années futures sur quelques caractéristiques du code précédent.

Finalement, nous avons vu que les callbacks et autres promesses sont la base de l'asynchrone en JavaScript.

EOF