



# FORMATION FULL STACK

הרווחה, משרד העבודה  
והשירותים החברתיים



## Node.js

By Charles Cohen



Charles Cohen  
Senior Full Stack Developer  
[charlesc@edandweb.co.il](mailto:charlesc@edandweb.co.il)  
050-4449764



# Programme du cours Node.js

1. Introduction to Node.js
  - What is Node and what is it not ?
  - Node.js Features ?
  - Our first Node.js script: Hello World
  - Building a web server in Node.js
  - Debugging node applications
2. Building your Stack
  - Pulling in other libraries
  - Building custom libraries
  - A-synchronicity and callbacks
  - Blocking vs. non-blocking I/O
  - Working within the event loop
3. Modular JavaScript with Node.js
  - Writing Modular JavaScript with Node.js
  - Core Modules
  - Installing Packages
  - Publishing packages
4. Avoiding common pitfalls with Async.js
  - Introducing the Asynchronous problem
  - Async.js Library to the rescue
  - Collections
  - Flow Controllers
5. Working with the file system
  - Files manipulations
  - Folder manipulations
  - Putting the file-system module together Async.js
6. Building Web applications with the Express Framework
  - Introduction to Express, installation and basic setup
  - Application configuration
  - Routing
  - Views and Templating options
  - Persistence with Cookies, In-Memory Sessions and session-stores
  - Social Authentication with Passport.js
7. Connecting MySQL Server
  - Database connection
  - A-synchronicity Queries from node.js



# Plan du module Node.js

Cours	Date	Cours
1	Lun. 20/05	<ul style="list-style-type: none"><li>• Introduction to Node.js (1)</li></ul>
2	Mer. 29/05	<ul style="list-style-type: none"><li>• Building your Stack (2)</li><li>• Command Line</li><li>• File System</li><li>• Arrow Functions</li><li>• Modular JavaScript with Node.js</li></ul>
3	Mer. 05/06	<ul style="list-style-type: none"><li>• Asynchronous JS</li><li>• Consuming API – HTTP requests</li></ul>
4	Lun. 10/06	<ul style="list-style-type: none"><li>• Consuming API – HTTP requests</li><li>• Building Web applications with the Express Framework</li></ul>
5	Lun. 17/06	<ul style="list-style-type: none"><li>• Express framework – templates with Handlebars</li><li>• Building a get API entry point based on previous exercices (get coordinates and weather from mapbox.com and darksky.net) and returning data as JSON</li><li>• Implement a search address box that consuming the GET API</li></ul>
6	Lun. 24/06	<ul style="list-style-type: none"><li>• Promise, await, async, Async module ...</li><li>• Avoiding common pitfalls with Async.js</li></ul>
7	Lun. 01/07	<ul style="list-style-type: none"><li>• Working with MongoDB 1/2</li></ul>
8	Lun 08/07	<ul style="list-style-type: none"><li>• Working with MongoDB 1/2</li></ul>



# Environnement de travail

Editeur de code:

- Visual Studio Code
  - Extensions:

Liens Utiles:

- Node.js: <https://nodejs.org/>
- Moteur V8 Javascript: <https://v8.dev/>



# Résumé du cours précédent

- Node.js is a Javascript **runtime** built on [Chrome's V8](#) Javascript engine.
- Node.js uses an **event-driven, non-blocking**
- Node.js' package ecosystem, **npm**, is the largest ecosystem of open source libraries
- Node.js => JS Code => V8 (C++) => Result
- Le nom de l'objet global est **global** et l'objet équivalent à Document dans le browser est **process**
- Pour utiliser la librairie des fichiers (FileSystem) on utilisera la commande `const fs = require('fs');`
- Pour écrire on utilisera la méthode `WriteFileSync`
- Système de modules de Node.js (FileSystem ...), [API: https://nodejs.org/dist/latest-v10.x/docs/api/](https://nodejs.org/dist/latest-v10.x/docs/api/)
- Nos scripts `require('./utils.js');` ➔ `module.exports`
- `npm init`
- Packets npm (`npm install validator`)
- Projet existant: `npm install`



# Questions/Sujets à enrichir

- Files: If Node.js reading all the file or do we have a buffer ?
- How can we run our code outside the command line ?  
Write our own server and set an entry point (API: POST, GET, PUT, PATCH)



# Cours 2

Command Line

File System

Notes Application

Arrow Functions





# Plan du cours

Parties	Horaires	Cours
1	19:00-20:30	<ul style="list-style-type: none"><li>• NPM modules</li><li>• Utilisation de d'un packet NPM validator</li><li>• Challenge chalk</li><li>• Live reload avec nodemon</li><li>• Ligne de commande avec Node.js</li></ul>
2	20:30-21:00	<ul style="list-style-type: none"><li>• Pause</li></ul>
3	21:00-23:00	<ul style="list-style-type: none"><li>• Ajout d'une commande</li><li>• Passage d'arguments</li><li>• Installation et utilisation de Yargs</li><li>• Application Notes</li><li>• Application Notes</li><li>• Arrow functions</li></ul>

# Serveur Node.js

## (routes)

```
var http = require('http'); // Import Node.js core module
//create web server
var server = http.createServer(function (req, res) {
  if (req.url == '/') { //check the URL of the current request
    // set response header
    res.writeHead(200, { 'Content-Type': 'text/html' });
    // set response content
    res.write('<html><body><p>This is home Page.</p></body></html>');
    res.end();
  } else if (req.url == "/student") {
    res.writeHead(200, { 'Content-Type': 'text/html' });
    res.write('<html><body><p>This is student Page.</p></body></html>');
    res.end();
  } else if (req.url == "/admin") {
    res.writeHead(200, { 'Content-Type': 'text/html' });
    res.write('<html><body><p>This is admin Page.</p></body></html>');
    res.end();
  } else if (req.url == '/data') { //check the URL of the current request
    res.writeHead(200, { 'Content-Type': 'application/json' });
    res.write(JSON.stringify({ message: "Hello World" }));
    res.end();
  } else {
    res.end('Invalid Request!');
  }
});

server.listen(5000); //6 - listen for any incoming requests
console.log('Node.js web server at port 5000 is running..');
```

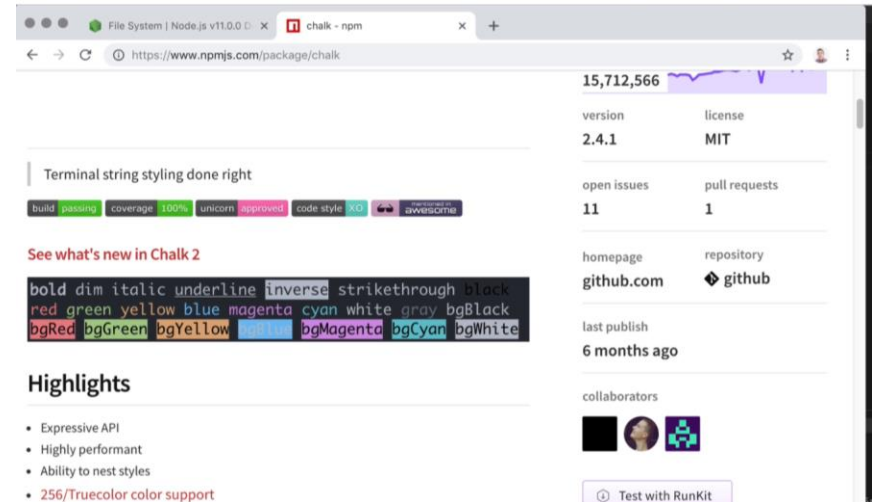
# Install a npm package

Ecrire un webserver à l'aide de node.js avec un point d'entrée (entry point) en GET.

Ce script devra enregistré dans un fichier logs ( les logs d'accès à l'application) avec les informations provenant du module os (libre choix).

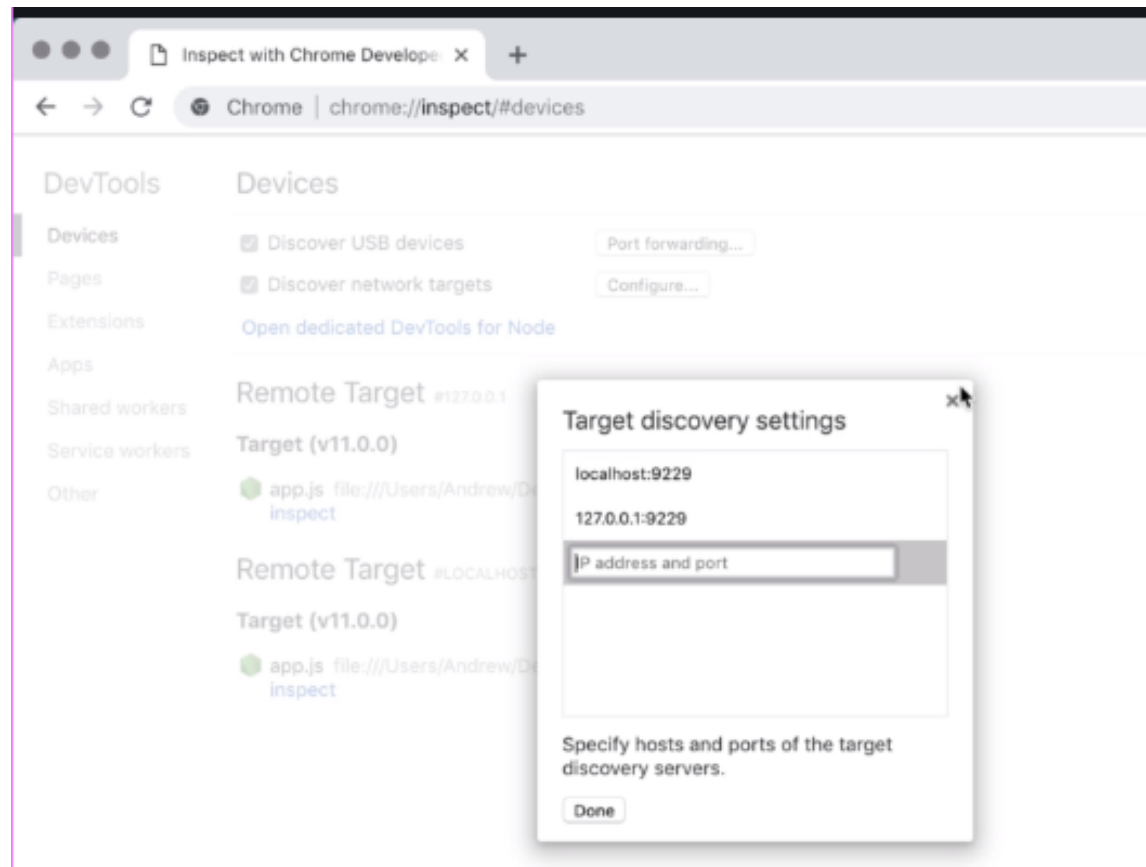
Utiliser le module chalk pour coloriser les message de la console.

1. Go to npmjs.com
2. Search for chalk
3. Install chalk on your project
4. Read the doc quickly
5. Open utils.js, use chalk to log a message with green background and write in yellow

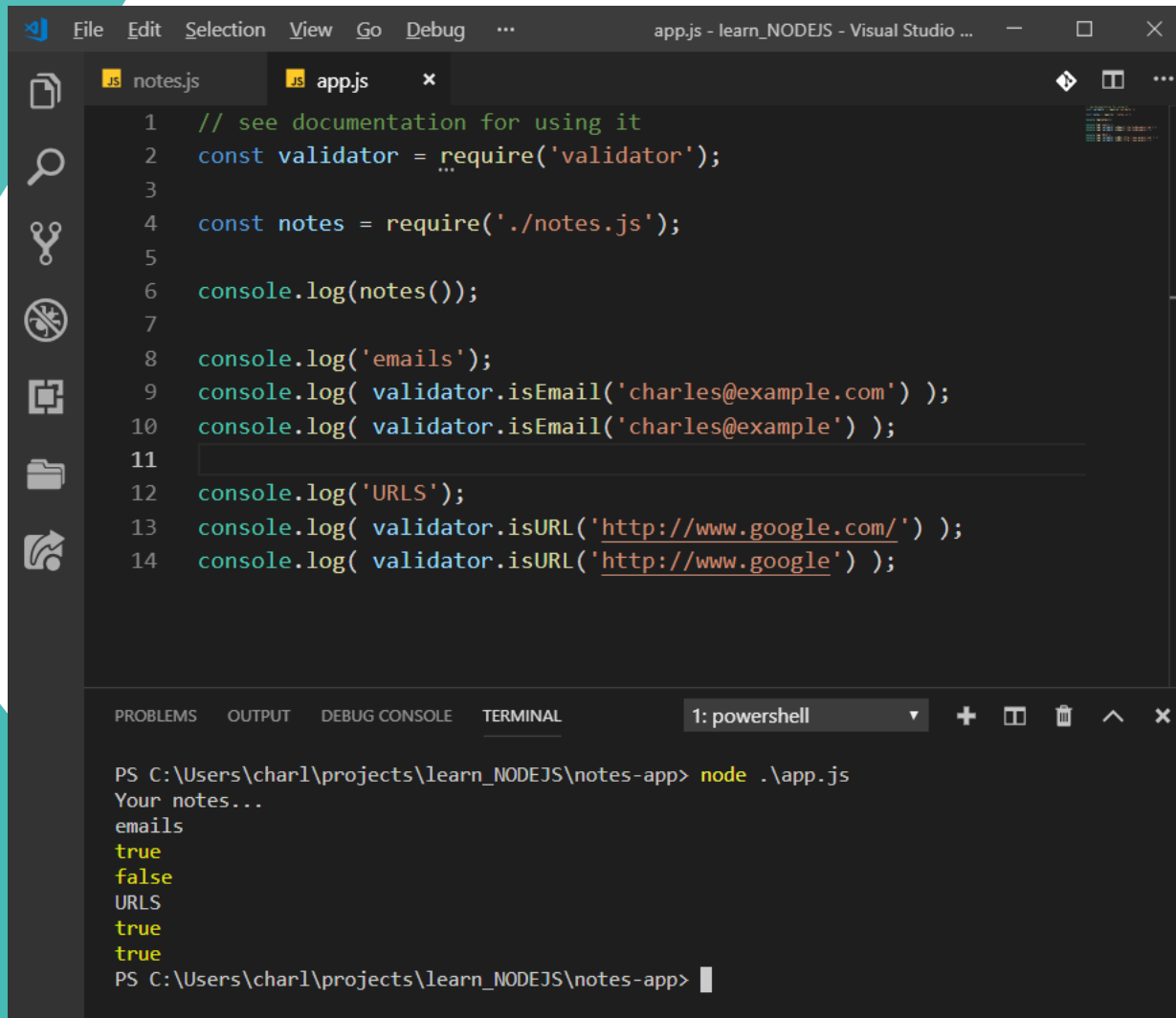


# Debugger avec Node.js

- console.log
- Node inspect script.js



# Working with npm package



The screenshot shows the Visual Studio Code editor interface. The main editor window displays the file `app.js` with the following JavaScript code:

```
1 // see documentation for using it
2 const validator = require('validator');
3
4 const notes = require('./notes.js');
5
6 console.log(notes());
7
8 console.log('emails');
9 console.log( validator.isEmail('charles@example.com') );
10 console.log( validator.isEmail('charles@example') );
11
12 console.log('URLS');
13 console.log( validator.isURL('http://www.google.com/') );
14 console.log( validator.isURL('http://www.google') );
```

The bottom panel of the editor shows the **TERMINAL** tab with the following output:

```
PS C:\Users\charl\projects\learn_NODEJS\notes-app> node .\app.js
Your notes...
emails
true
false
URLS
true
true
PS C:\Users\charl\projects\learn_NODEJS\notes-app>
```



# Travail avec un projets existant

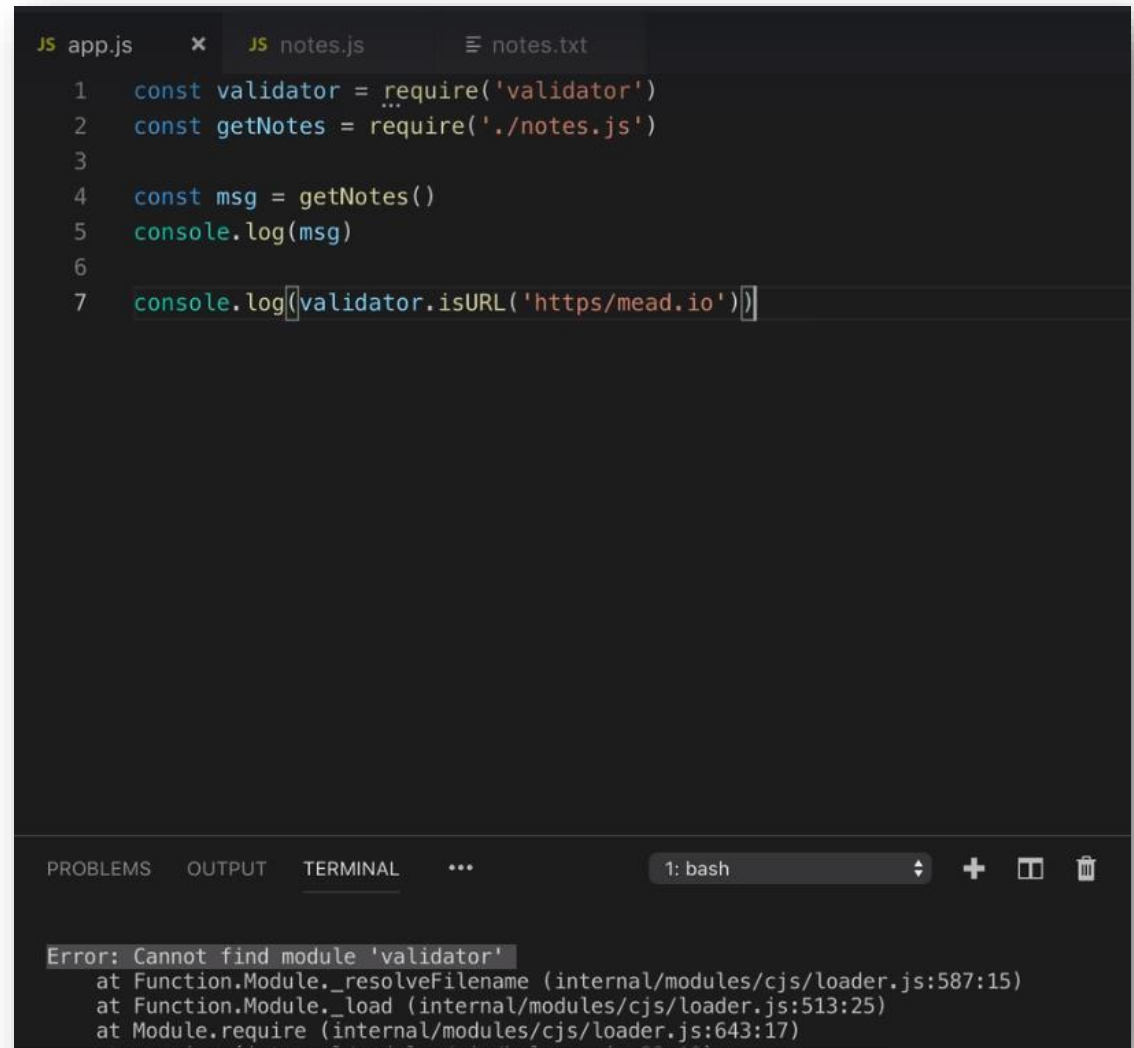
Si vous travaillez sur un projet existant (ex: projet sur git), les modules n'existent pas.

Supprimer votre dossier `node_modules` et relancer votre script.

Une erreur surviendra.

Utiliser la commande `npm install` et tous le dossiers des modules sera généré à nouveau.

Il ne faudra pas mettre à jours ce dossier manuellement car ces changements seront écrasés par la commande `npm`.



```
JS app.js x JS notes.js ≡ notes.txt
1  const validator = require('validator')
2  const getNotes = require('./notes.js')
3
4  const msg = getNotes()
5  console.log(msg)
6
7  console.log([validator.isURL('https://mead.io')])
```

PROBLEMS OUTPUT TERMINAL ... 1: bash

```
Error: Cannot find module 'validator'
    at Function.Module._resolveFilename (internal/modules/cjs/loader.js:587:15)
    at Function.Module._load (internal/modules/cjs/loader.js:513:25)
    at Module.require (internal/modules/cjs/loader.js:643:17)
```

# Install nodemon globally

## nodemon

1.18.10 • Public • Published 2 months ago

Readme

10 Dependencies

1,427 Dependents

201 Versions



## nodemon

nodemon is a tool that helps develop node.js based applications by automatically restarting the node application when file changes in the directory are detected.

nodemon does **not** require *any* additional changes to your code or method of development. nodemon is a replacement wrapper for `node`, to use `nodemon` replace the word `node` on the command line when executing your script.

npm package: 1.18.10 build: passing backers: 65 sponsors: 16

## Installation

Either through cloning with git or by using `npm` (the recommended way):

```
npm install -g nodemon
```

install

```
> npm i nodemon
```

± weekly downloads

1,390,860

version

1.18.10

license

MIT

open issues

10

pull requests

0

homepage

nodemon.io

repository

github

last publish

2 months ago

collaborators



Test with RunKit

Report a vulnerability

Pour lancer nos scripts jusqu'à maintenant nous avons utilisé la commande: `node`.

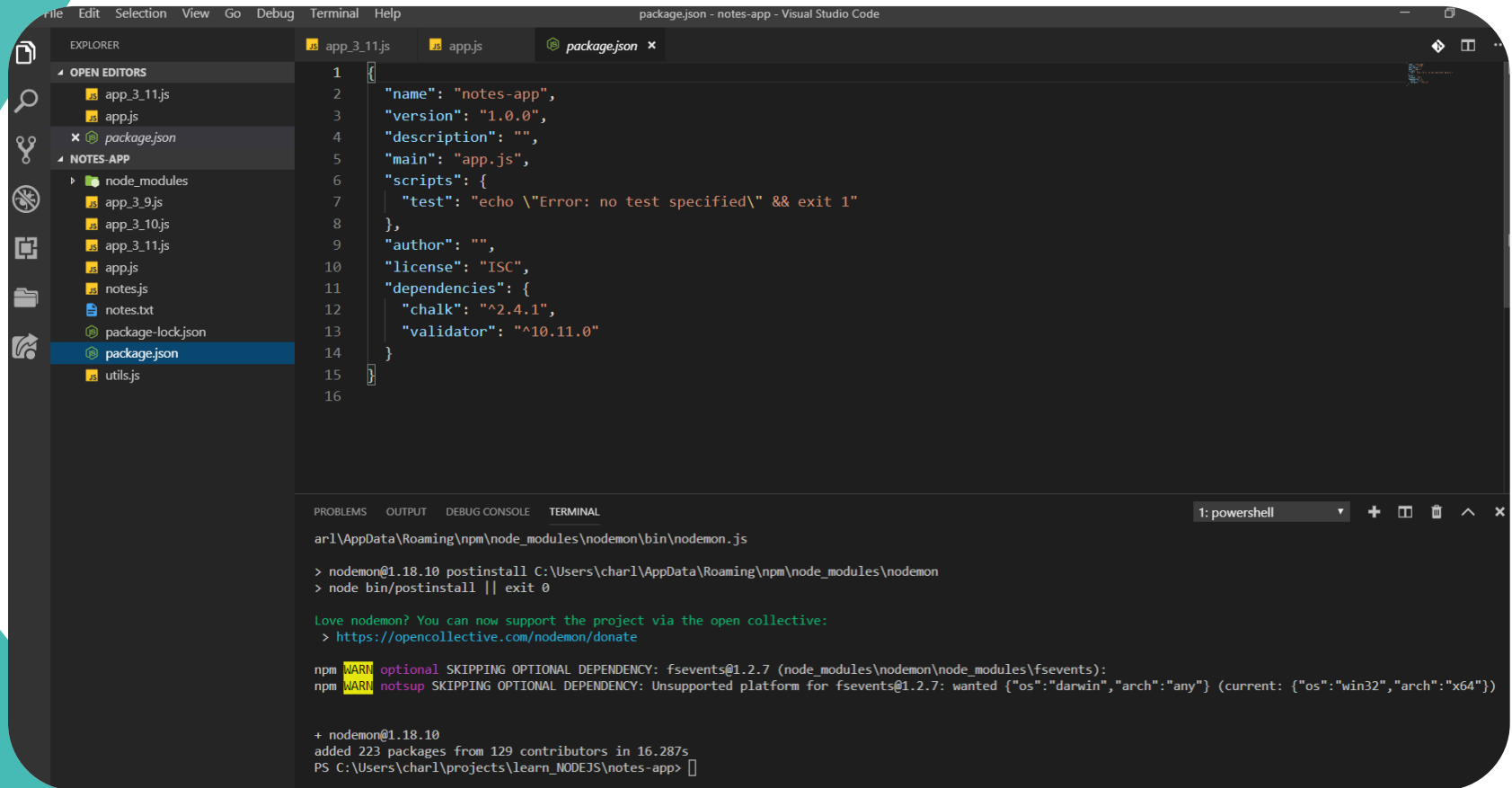
Si nous souhaitons laisser notre script tourné sans avoir besoin de relancer la commande à chaque changement nous utiliserons le packet `nodemon`.

```
npm install -g nodemon
```

If error on unix system use `sudo` before.



# Impact de la commande



The screenshot shows the Visual Studio Code interface. The Explorer sidebar on the left displays the file structure of a project named 'NOTES-APP', including files like app\_3\_9.js, app\_3\_10.js, app\_3\_11.js, app.js, notes.js, notes.txt, package-lock.json, package.json, and utils.js. The package.json file is selected and its content is displayed in the main editor. The terminal window at the bottom shows the execution of the 'nodemon' command, which has installed the 'nodemon' package and its dependencies. The terminal output includes a warning about an optional dependency 'fsevents' and a message about the number of packages added from contributors.

```
1 {
2   "name": "notes-app",
3   "version": "1.0.0",
4   "description": "",
5   "main": "app.js",
6   "scripts": {
7     "test": "echo \"Error: no test specified\" && exit 1"
8   },
9   "author": "",
10  "license": "ISC",
11  "dependencies": {
12    "chalk": "^2.4.1",
13    "validator": "^10.11.0"
14  }
15 }
16
```

```
ar1\AppData\Roaming\npm\node_modules\nodemon\bin\nodemon.js
> nodemon@1.18.10 postinstall C:\Users\charl\AppData\Roaming\npm\node_modules\nodemon
> node bin/postinstall || exit 0

Love nodemon? You can now support the project via the open collective:
> https://opencollective.com/nodemon/donate

npm WARN optional SKIPPING OPTIONAL DEPENDENCY: fsevents@1.2.7 (node_modules\nodemon\node_modules\fsevents):
npm WARN notsup SKIPPING OPTIONAL DEPENDENCY: Unsupported platform for fsevents@1.2.7: wanted {"os":"darwin","arch":"any"} (current: {"os":"win32","arch":"x64"})

+ nodemon@1.18.10
added 223 packages from 129 contributors in 16.287s
PS C:\Users\charl\projects\learn_NODEJS\notes-app>
```



# Live reload with nodemon

The screenshot shows the Visual Studio Code interface with a project named 'notes-app'. The Explorer sidebar on the left shows the file structure, including 'app\_3\_11.js', 'app.js', 'package.json', 'node\_modules', 'notes.js', 'notes.txt', 'package-lock.json', and 'utils.js'. The main editor displays 'app.js' with the following code:

```
1 const chalk = require('chalk');
2
3 const getNotes = require('./notes.js');
4
5 const msg = getNotes();
6 console.log(msg);
7
8 console.log(chalk.bgRed.yellow('Error!'));
9
10
```

The TERMINAL panel at the bottom shows the command prompt output for installing and running nodemon:

```
PS C:\Users\charl\projects\learn_NODEJS\notes-app> npm install -g nodemon
stall -g nodemon
C:\Users\charl\AppData\Roaming\npm\nodemon -> C:\Users\charl\AppData\Roaming\npm\node_modules\nodemon\bin\nodemon.js
> nodemon@1.18.10 postinstall C:\Users\charl\AppData\Roaming\npm\node_modules\nodemon
> node bin/postinstall || exit 0

Love nodemon? You can now support the project via the open collective:
> https://opencollective.com/nodemon/donate

npm WARN optional SKIPPING OPTIONAL DEPENDENCY: fsevents@1.2.7 (node_modules\nodemon\node_modules\fsevents):
npm WARN notsup SKIPPING OPTIONAL DEPENDENCY: Unsupported platform for fsevents@1.2.7: wanted {"os":"darwin","arch":"any"} (current: {"os":"win32","arch":"x64"})

+ nodemon@1.18.10
added 223 packages from 129 contributors in 16.287s
PS C:\Users\charl\projects\learn_NODEJS\notes-app> nodemon app.js
[nodemon] 1.18.10
[nodemon] to restart at any time, enter `rs`
[nodemon] watching: *.*
[nodemon] starting `node app.js`
Your notes...
Success!
[nodemon] clean exit - waiting for changes before restart
[nodemon] restarting due to changes...
[nodemon] starting `node app.js`
[nodemon] restarting due to changes...
Your notes...
Error!
[nodemon] starting `node app.js`
Your notes...
```

The status bar at the bottom indicates the 'Live Share' extension is active, and the 'Go Live' button is visible. The bottom right corner features a 'webschool' logo.

# Ligne de commande



# Ligne de commande

Lorsque nous utilisons un terminal pour réaliser nos opérations nous appelons cela la ligne de commande.

Si vous essayez de lancer votre script avec un argument exemple:

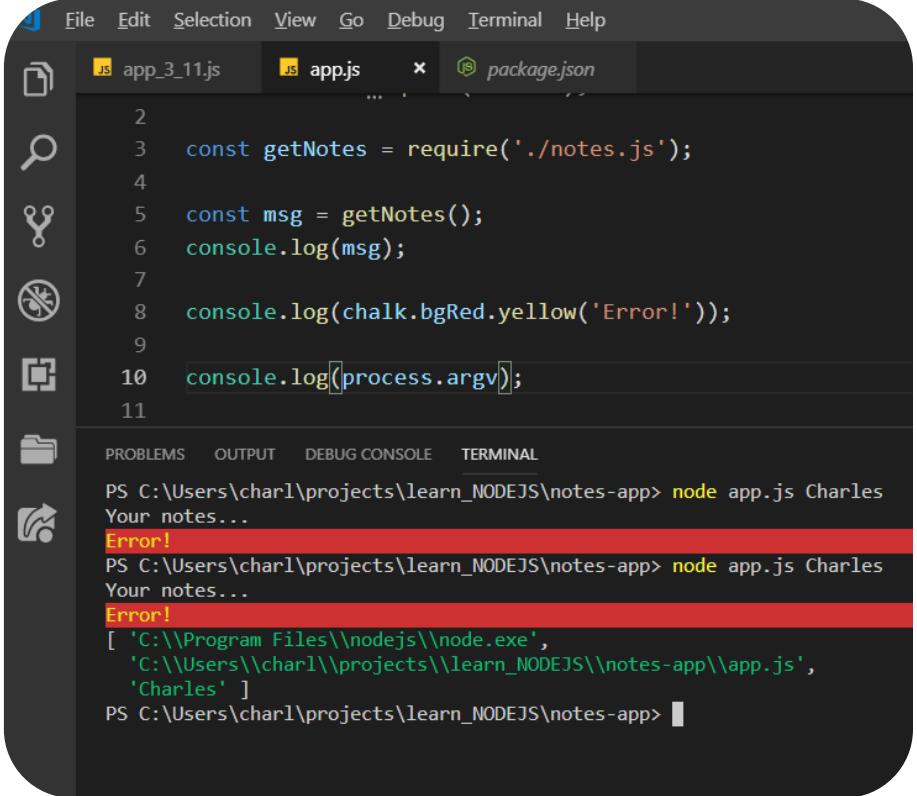
```
node app.js Charles
```

Vous verrez que cela n'affecte pas l'exécution du script. Si nous souhaitons accéder à cette variable:

```
process.argv
```

Notre variable contient un array:

1. Path vers l'exécutable de Node.js sur votre machine
2. Path vers le script en cours d'exécution
3. L'argument

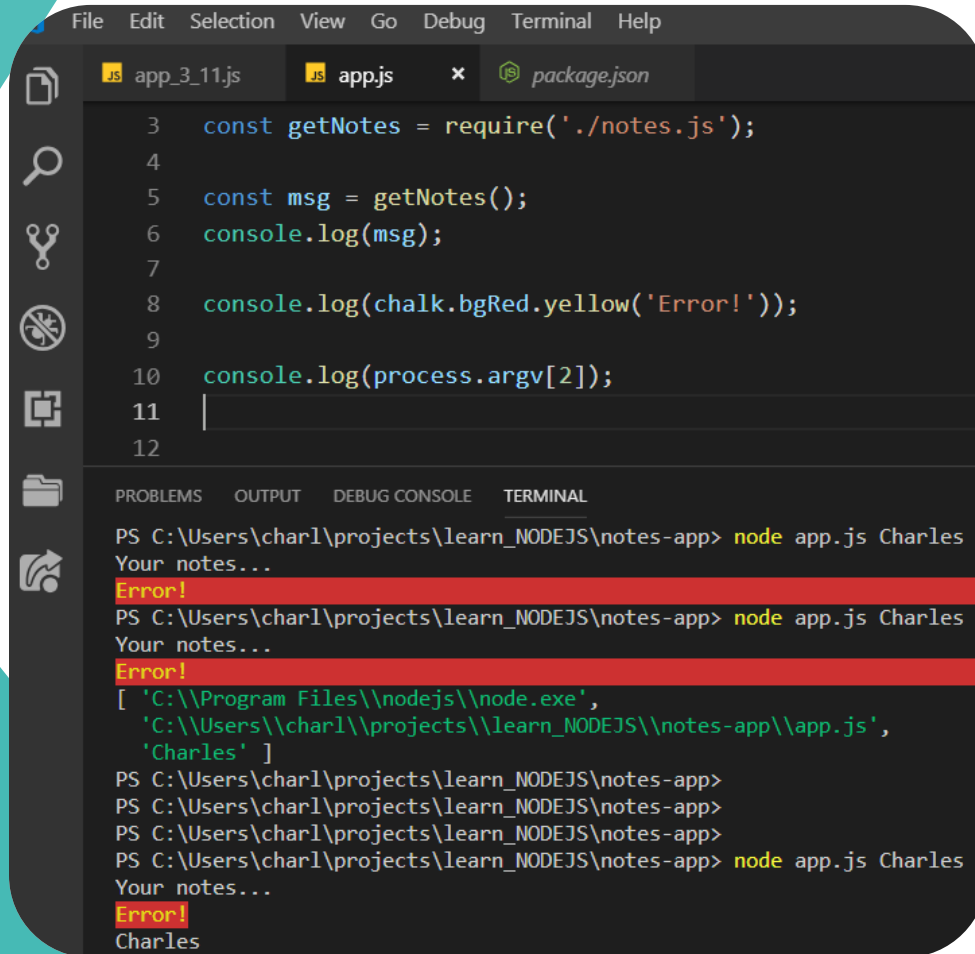


The screenshot shows a VS Code editor with a file named `app.js` open. The code in the editor is as follows:

```
2
3 const getNotes = require('./notes.js');
4
5 const msg = getNotes();
6 console.log(msg);
7
8 console.log(chalk.bgRed.yellow('Error!'));
9
10 console.log(process.argv);
11
```

Below the editor, the TERMINAL panel is visible, showing the command `node app.js Charles` being executed. The output shows the array `['C:\\Program Files\\nodejs\\node.exe', 'C:\\Users\\charl\\projects\\learn_NODEJS\\notes-app\\app.js', 'Charles']`.

# Afficher l'argument passer à la commande



The screenshot shows the Visual Studio Code editor with a file named `app.js` open. The code in the editor is as follows:

```
3 const getNotes = require('./notes.js');
4
5 const msg = getNotes();
6 console.log(msg);
7
8 console.log(chalk.bgRed.yellow('Error!'));
9
10 console.log(process.argv[2]);
11
12
```

Below the editor, the terminal window is visible, showing the command `node app.js Charles` being executed. The output shows the message "Your notes..." followed by "Error!" (highlighted in red) and then the argument "Charles".

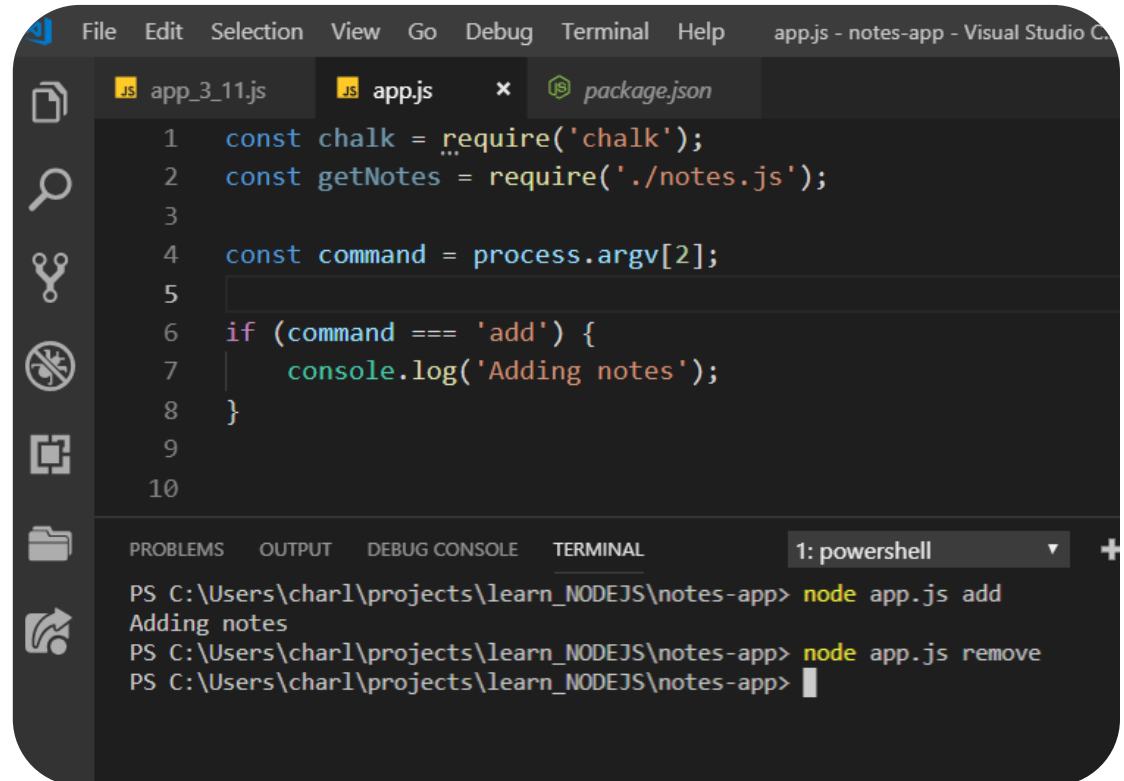
Notre argument se situe à la 3ème position dans le tableau.

Pour y accéder:

```
console.log(process.argv[2]);
```

# Créer une commande add

Pour écrire notre commande add il nous suffit de réaliser une simple condition et de comparer le texte passer en argument.

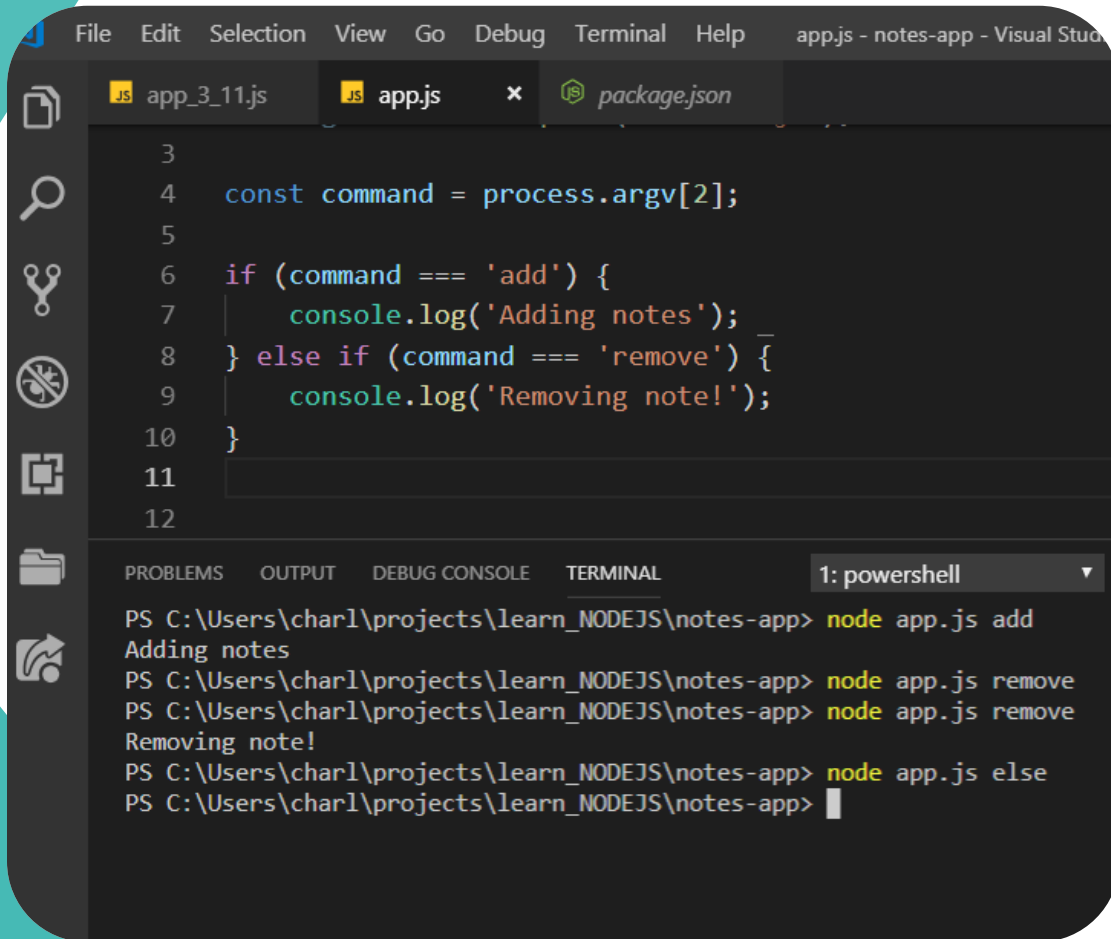


```
File Edit Selection View Go Debug Terminal Help app.js - notes-app - Visual Studio C
app_3_11.js app.js x package.json
1 const chalk = require('chalk');
2 const getNotes = require('./notes.js');
3
4 const command = process.argv[2];
5
6 if (command === 'add') {
7   console.log('Adding notes');
8 }
9
10
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL 1: powershell +

```
PS C:\Users\charl\projects\learn_NODEJS\notes-app> node app.js add
Adding notes
PS C:\Users\charl\projects\learn_NODEJS\notes-app> node app.js remove
PS C:\Users\charl\projects\learn_NODEJS\notes-app> |
```

# Ajouter une commande remove



```
3
4  const command = process.argv[2];
5
6  if (command === 'add') {
7      console.log('Adding notes');
8  } else if (command === 'remove') {
9      console.log('Removing note!');
10 }
11
12
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL 1: powershell

```
PS C:\Users\charl\projects\learn_NODEJS\notes-app> node app.js add
Adding notes
PS C:\Users\charl\projects\learn_NODEJS\notes-app> node app.js remove
PS C:\Users\charl\projects\learn_NODEJS\notes-app> node app.js remove
Removing note!
PS C:\Users\charl\projects\learn_NODEJS\notes-app> node app.js else
PS C:\Users\charl\projects\learn_NODEJS\notes-app>
```

Pour ajouter une commande remove nous utiliserons une autre condition avec if else if.

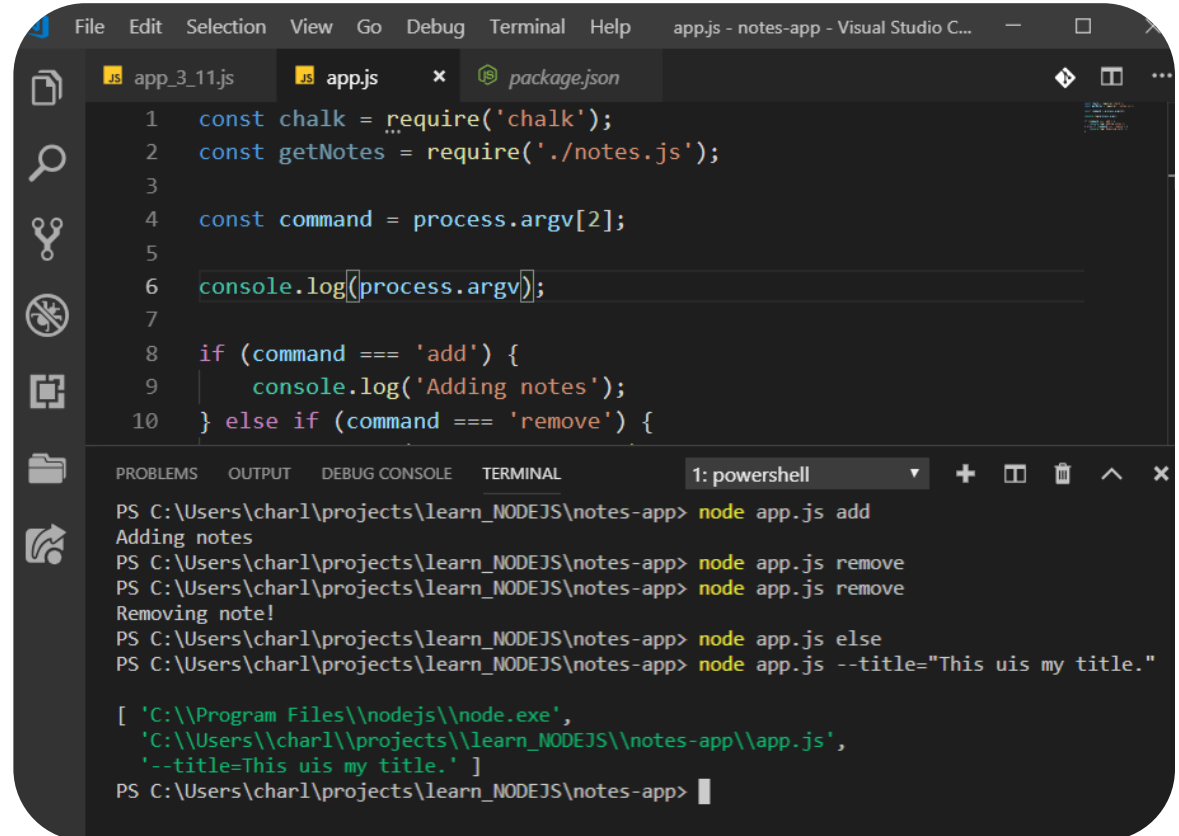
# Passer un argument à notre commande

En ligne de commande, les arguments que l'on souhaite nommer sont précédés par un double tiret ex:

```
--title="This is my title"
```

Nous pouvons commencer à écrire des fonctions pour parser les arguments.

Au lieu de cela nous allons utiliser yargs un paquet npm.



The screenshot shows the Visual Studio Code editor with a file named `app.js` open. The code in the file is as follows:

```
1 const chalk = require('chalk');
2 const getNotes = require('./notes.js');
3
4 const command = process.argv[2];
5
6 console.log(process.argv);
7
8 if (command === 'add') {
9   console.log('Adding notes');
10 } else if (command === 'remove') {
```

Below the code editor, the TERMINAL panel is active, showing a PowerShell session. The commands and their outputs are:

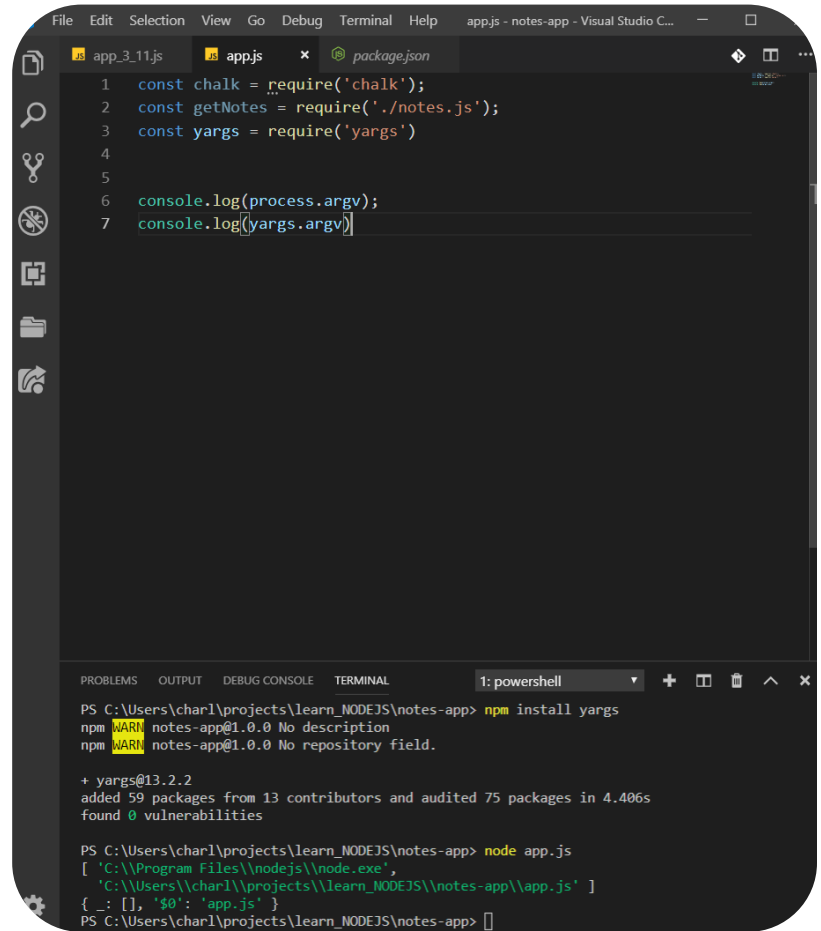
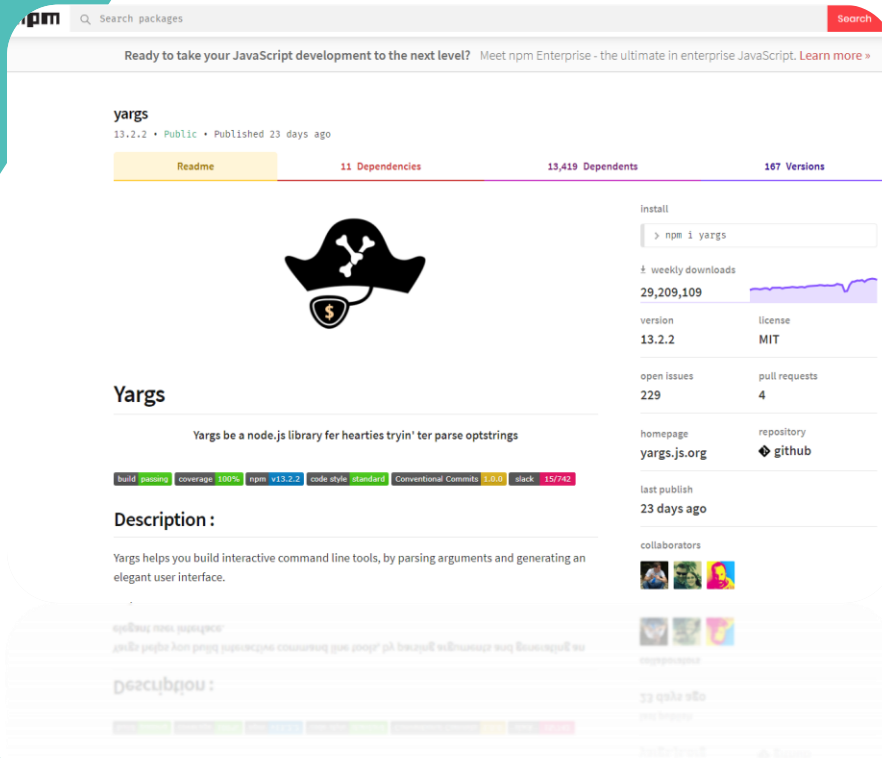
```
PS C:\Users\charl\projects\learn_NODEJS\notes-app> node app.js add
Adding notes
PS C:\Users\charl\projects\learn_NODEJS\notes-app> node app.js remove
Removing note!
PS C:\Users\charl\projects\learn_NODEJS\notes-app> node app.js else
PS C:\Users\charl\projects\learn_NODEJS\notes-app> node app.js --title="This uis my title."

[ 'C:\Program Files\nodejs\node.exe',
  'C:\Users\charl\projects\learn_NODEJS\notes-app\app.js',
  '--title=This uis my title.' ]
PS C:\Users\charl\projects\learn_NODEJS\notes-app>
```

```
b2 C:\Program Files\nodejs\node.exe
--title=This uis my title." ]
C:\Program Files\nodejs\node.exe
[ 'C:\Program Files\nodejs\node.exe',
  'C:\Users\charl\projects\learn_NODEJS\notes-app\app.js',
  '--title=This uis my title.' ]
b2 C:\Program Files\nodejs\node.exe
C:\Program Files\nodejs\node.exe
```



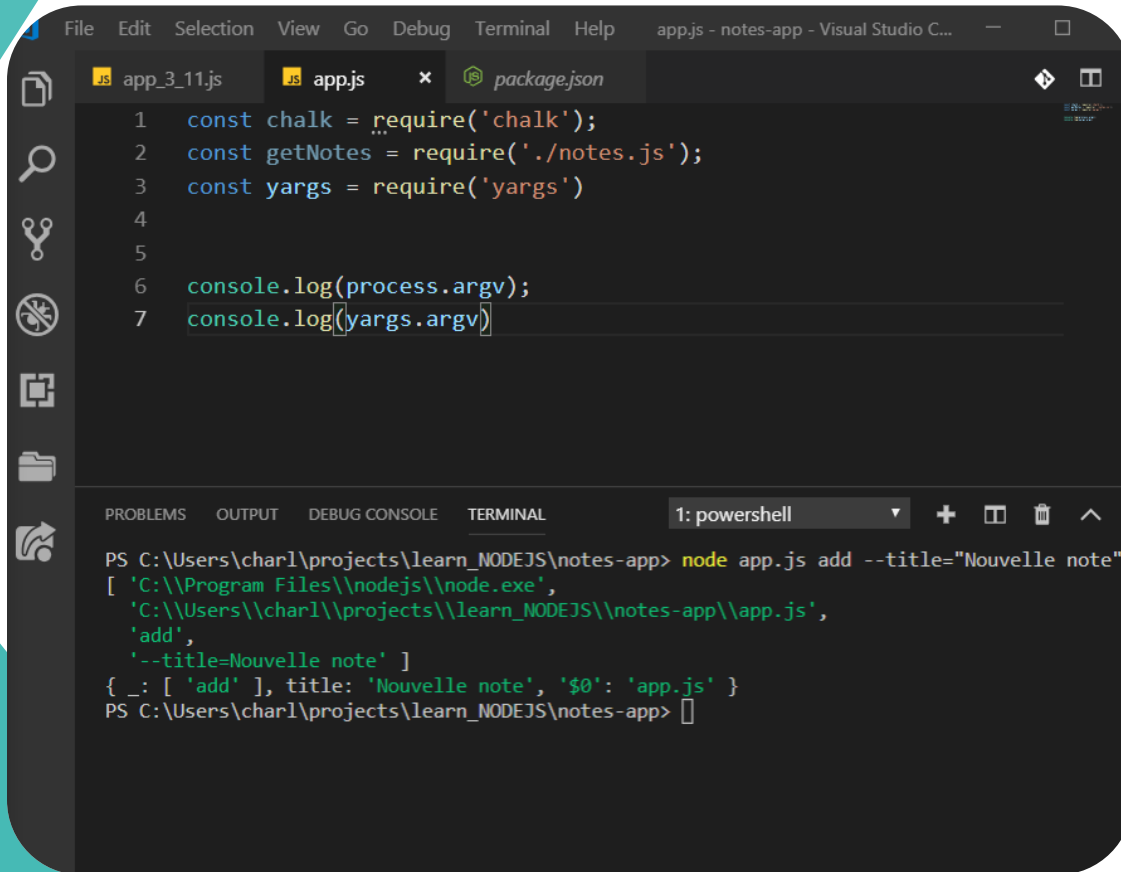
# Parse arguments with yargs



Yargs nous retourne un objet dans lequel nous aurons les arguments passer à la commande.



# Yargs – récupération des arguments



The screenshot shows the Visual Studio Code editor with three open files: `app_3_11.js`, `app.js`, and `package.json`. The `app.js` file contains the following code:

```
1 const chalk = require('chalk');
2 const getNotes = require('./notes.js');
3 const yargs = require('yargs')
4
5
6 console.log(process.argv);
7 console.log(yargs.argv)
```

Below the editor, the terminal window is open, showing the command prompt and the execution of the script:

```
PS C:\Users\charl\projects\learn_NODEJS\notes-app> node app.js add --title="Nouvelle note"
[ 'C:\Program Files\nodejs\node.exe',
  'C:\Users\charl\projects\learn_NODEJS\notes-app\app.js',
  'add',
  '--title=Nouvelle note' ]
{ _: [ 'add' ], title: 'Nouvelle note', '$0': 'app.js' }
PS C:\Users\charl\projects\learn_NODEJS\notes-app>
```

Pour accéder à notre argument title il nous suffit d'écrire:

```
yargs.argv.title
```

Pour obtenir des informations sur la commande utiliser:

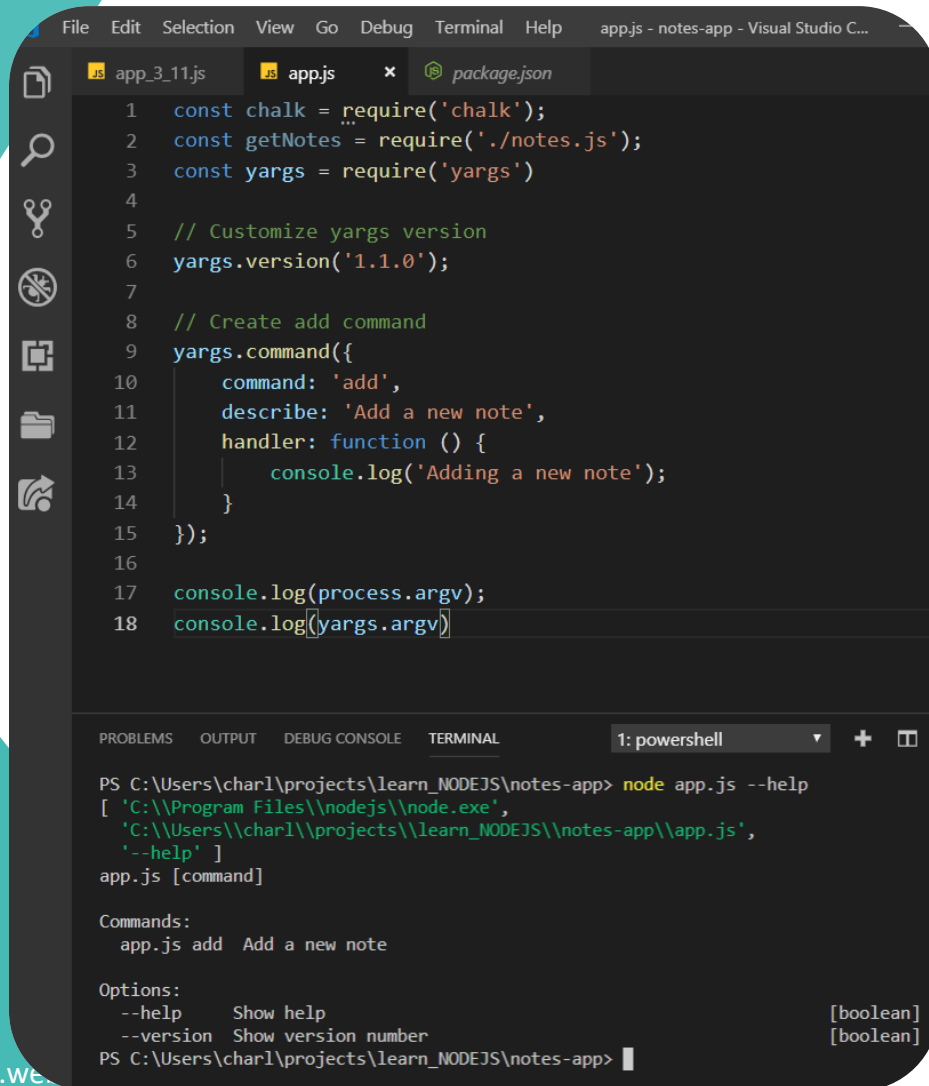
```
--help
```

# Ecriture d'une application notes

Nous allons écrire une application pour gérer des notes. Pour les sauvegarder nous utiliserons un fichier notes.json .



# Yargs – écrire une commande



```
1 const chalk = require('chalk');
2 const getNotes = require('./notes.js');
3 const yargs = require('yargs')
4
5 // Customize yargs version
6 yargs.version('1.1.0');
7
8 // Create add command
9 yargs.command({
10   command: 'add',
11   describe: 'Add a new note',
12   handler: function () {
13     console.log('Adding a new note');
14   }
15 });
16
17 console.log(process.argv);
18 console.log(yargs.argv)
```

```
PS C:\Users\charl\projects\learn_NODEJS\notes-app> node app.js --help
[ 'C:\\Program Files\\nodejs\\node.exe',
  'C:\\Users\\charl\\projects\\learn_NODEJS\\notes-app\\app.js',
  '--help' ]
app.js [command]

Commands:
  app.js add  Add a new note

Options:
  --help      Show help                                [boolean]
  --version   Show version number                       [boolean]
PS C:\Users\charl\projects\learn_NODEJS\notes-app>
```

Pour écrire une commande avec Yargs, utiliser la méthode `command`. Celle-ci reçoit un objet avec 3 propriétés:

- **command** : le nom de la commande
- **describe**: la description de la commande
- **handler**: la fonction qui sera appelée lors du lancement de la commande.

Nous pouvons obtenir de l'aide grâce à `--help`

Une fois les `console.log()` en fin de script supprimé le `console.log` dans le handler n'affichera rien.

Ajouter le code à la fin de votre script:

```
yargs.parse();
```



# Ajouter des commandes

Ajouter les commandes suivantes:

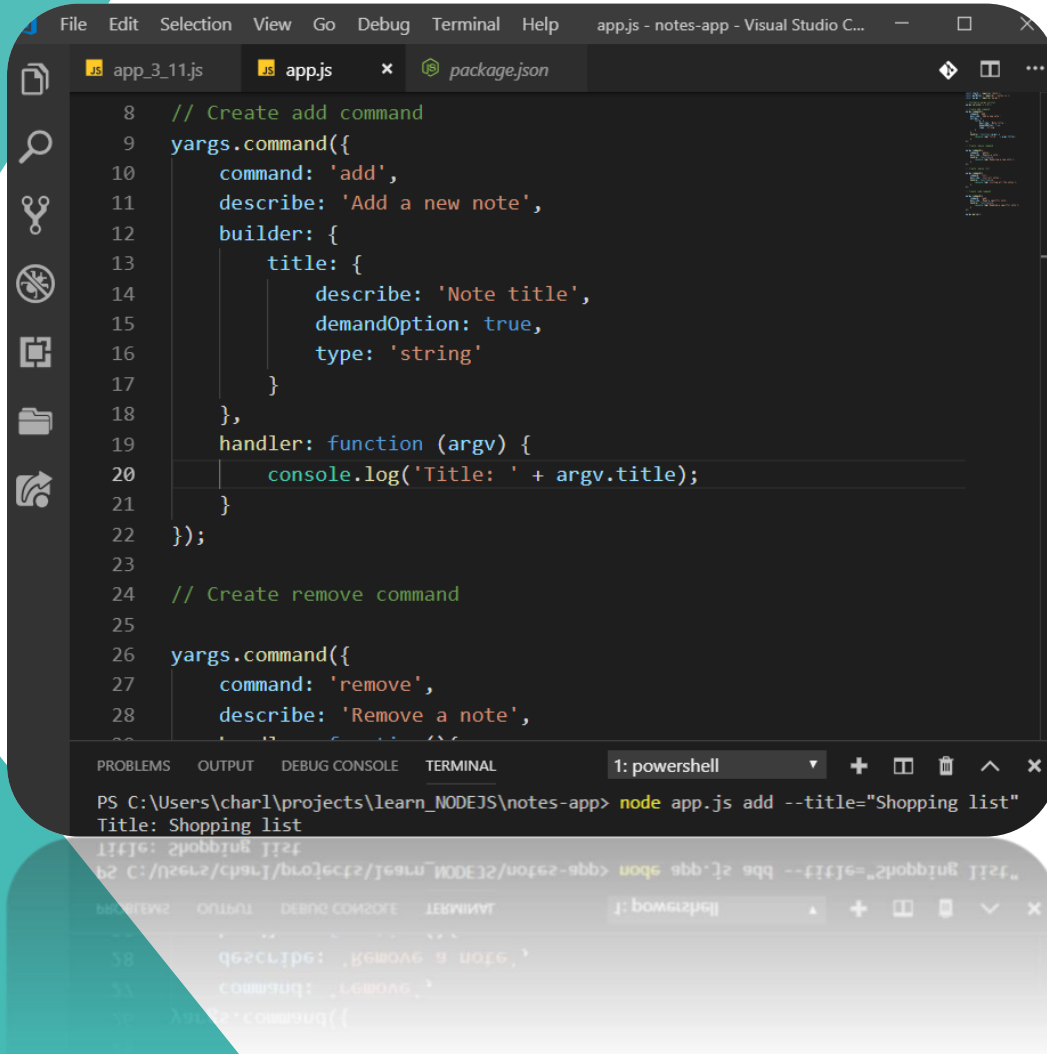
- add : ajoute une nouvelle note
- remove: supprime une note
- read: lire une note
- list: lister toutes les notes

Pour le handler de chaque commande logger un message texte avec le nom de la commande.

Tester toutes les commandes .

Utiliser chalk pour coloriser vos messages.

# Yargs – ajout d'un argument à une commande



```
8 // Create add command
9 yargs.command({
10   command: 'add',
11   describe: 'Add a new note',
12   builder: {
13     title: {
14       describe: 'Note title',
15       demandOption: true,
16       type: 'string'
17     }
18   },
19   handler: function (argv) {
20     console.log('Title: ' + argv.title);
21   }
22 });
23
24 // Create remove command
25
26 yargs.command({
27   command: 'remove',
28   describe: 'Remove a note',
29   handler: function (argv) {
30     console.log('Removing note');
31   }
32 });
33
34 module.exports = yargs;
35
```

Terminal output:

```
PS C:\Users\charl\projects\learn_NODEJS\notes-app> node app.js add --title="Shopping list"
Title: Shopping list
```

Pour ajouter un argument à notre commande nous utiliserons la propriété **builder**, qui recoit un objet (**clé:valeur**).

La **clé** de l'objet est le nom que vous souhaitez donner à l'argument.

La **valeur** est un objet définissant l'argument lui même:

- **describe:**  
description de l'argument
- **demandOption:**  
booléen pour définir l'argument comme obligatoire
- **type:**  
le type de l'argument



# Ajouter des arguments à vos commandes

Ajouter les arguments suivants:

- add : title et body
- remove: title
- read: title

Tester toutes les commandes avec les paramètres et afficher les dans la console.

Utiliser chalk pour coloriser vos messages.

Regarder l'utilisation de la commande `–help` sur vos commandes.

# Implémentation de la fonction add

Notre commande **add** a pour but d'implémenter une fonction ajoutant une note (**title**, **body**) dans un **fichier texte**.

Nous allons devoir stocker dans ce fichier texte (**notes.json**) un **tableau** contenant tout nos **objets** (les notes) au format JSON (string).

Avant d'ajouter une note nous devons **vérifier qu'elle n'existe pas (title comme référence)**. Il faudra **récupérer les notes existantes**.

Créons une fonction **loadNotes** qui retournera les notes:

1. `readFileSync`
2. `toString()`
3. `JSON.parse`

Lancer le script et essayer de logger les notes.



The image shows a Visual Studio Code editor window with a dark theme. The editor is open to a file named `notes.js` in the `app.js` folder. The code in `notes.js` is as follows:

```
1 const fs = require('fs');
2 const NOTES_FILE = 'notes.json';
3
4 const getNotes = function(){
5   return "Your notes...";
6 };
7
8 const addNote = function(title, body){
9   const notesData = loadNotes(NOTES_FILE);
10 };
11
12 const loadNotes = function(file){
13   const dataBuffer = fs.readFileSync(file);
14   const dataJSON = data.toString();
15   return JSON.parse(dataJSON);
16 };
17
18 module.exports = {
19   getNotes: getNotes,
20   addNote: addNote
21 }
```

The terminal window at the bottom shows the command `node app.js add --title="t" --body="b"` being executed. The output shows the command running successfully, but then an error is thrown:

```
PS C:\Users\charl\projects\learn_NODEJS\notes-app> node app.js add --title="t" --body="b"
C:\Users\charl\projects\learn_NODEJS\notes-app\node_modules\yargs\yargs.js:1163
    else throw err
           ^
Error: ENOENT: no such file or directory, open 'notes.json'
    at Object.openSync (fs.js:438:3)
    at Object.readFileSync (fs.js:343:35)
    at loadNotes (C:\Users\charl\projects\learn_NODEJS\notes-app\notes.js:13:27)
    at Object.addNote (C:\Users\charl\projects\learn_NODEJS\notes-app\notes.js:9:23)
    at Object.handler (C:\Users\charl\projects\learn_NODEJS\notes-app\app.js:26:15)
    at Object.runCommand (C:\Users\charl\projects\learn_NODEJS\notes-app\node_modules\yargs\lib\command.js:242:26)
    at Object.parseArgs [as parseArgs] (C:\Users\charl\projects\learn_NODEJS\notes-app\node_modules\yargs\yargs.js:1078:30)
    at Object.parse (C:\Users\charl\projects\learn_NODEJS\notes-app\node_modules\yargs\yargs.js:542:19)
    at Object.<anonymous> (C:\Users\charl\projects\learn_NODEJS\notes-app\app.js:60:7)
    at Module._compile (internal/modules/cjs/loader.js:701:30)
PS C:\Users\charl\projects\learn_NODEJS\notes-app>
```

The error message indicates that the file `notes.json` does not exist, which is why the `loadNotes` function fails when it tries to read the file.

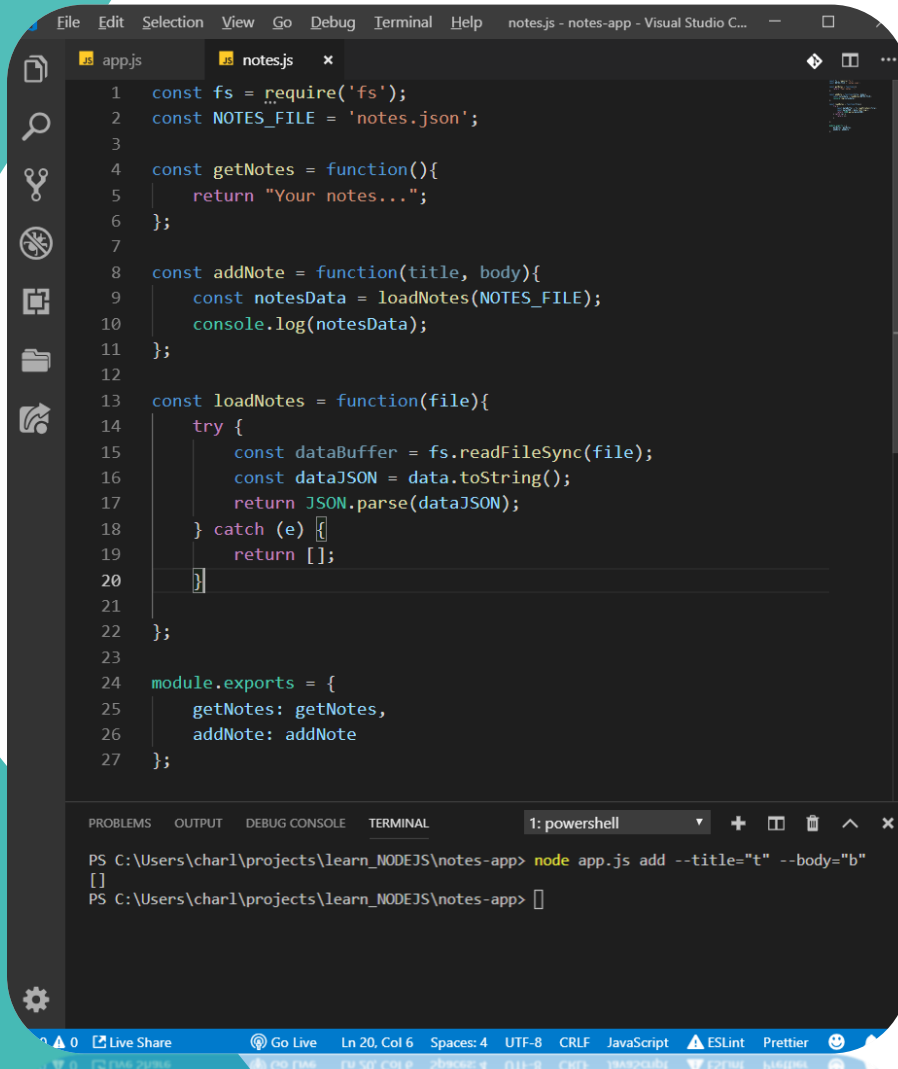
# Gestion des erreurs

## try {...} catch (e) {...}

Une fois le try catch ajouté, tester la commande à nouveau.

Notre script fonctionne, créer un fichier notes.json:

```
[{"title": "Webschool", "body": "Cours Node.js"}]
```



The screenshot shows the Visual Studio Code editor with a file named `notes.js` open. The code defines a `getNotes` function that returns a string, an `addNote` function that reads a file and logs its contents, and a `loadNotes` function that uses a `try...catch` block to handle file reading errors. The `try` block contains code to read a file synchronously, convert it to a string, and parse it as JSON. The `catch` block returns an empty array. The `module.exports` object exports the `getNotes` and `addNote` functions. Below the editor, a terminal window shows the command `node app.js add --title="t" --body="b"` being executed, which results in an empty array output.

```
1 const fs = require('fs');
2 const NOTES_FILE = 'notes.json';
3
4 const getNotes = function(){
5   return "Your notes...";
6 };
7
8 const addNote = function(title, body){
9   const notesData = loadNotes(NOTES_FILE);
10  console.log(notesData);
11 };
12
13 const loadNotes = function(file){
14   try {
15     const dataBuffer = fs.readFileSync(file);
16     const dataJSON = data.toString();
17     return JSON.parse(dataJSON);
18   } catch (e) {
19     return [];
20   }
21 };
22
23
24 module.exports = {
25   getNotes: getNotes,
26   addNote: addNote
27 };
28
```

```
PS C:\Users\charl\projects\learn_NODEJS\notes-app> node app.js add --title="t" --body="b"
[]
PS C:\Users\charl\projects\learn_NODEJS\notes-app>
```



# Implémentation de la fonction addNote

```
8  const addNote = function(title, body){
9    let notes = loadNotes(NOTES_FILE);
10
11    notes.push({
12      title: title,
13      body: body
14    });
15
16    saveNotes(notes);
17  };
18
19  const saveNotes = function(notes){
20    const dataJSON = JSON.stringify(notes);
21    fs.writeFileSync(NOTES_FILE, dataJSON);
22  };
23
```

Maintenant que nous avons réussi à charger les notes nous allons ajouter à chaque appel de la commande la nouvelle note.

La variable notes contient un array et pour y ajouter une nouvelle note nous pouvons utiliser la méthode **push** .

Créons une fonction saveNote qui sera responsable de la sauvegarde des notes:

1. JSON.stringify
2. writeFileSync

Relancer la commande add et vérifier que le fichier notes.json se met à jour.



# RAPPEL - Array - Méthodes Itératives

5 méthodes itératives existent pour les Array.

Elle reçoit en arguments 2 paramètres le premier obligatoire et le second optionnel.

Le premier est **une fonction qui sera exécuté sur chacun des éléments de l'Array**.

Le second est le contexte dit le scope (influant sur la valeur de **this**) on verra plus tard.

## Méthodes itératives

.every()	Exécute la fonction sur tous les éléments de l'Array et retourne true si la fonction à retourné true pour tous les éléments de l'Array .
.filter()	Exécute la fonction sur tous les éléments de l'Array et retourne un Array de tous les éléments pour lesquels la fonction à retourné true.
.forEach()	Exécute la fonction sur tous les éléments de l'Array et ne retourne aucune valeur.
.map()	Exécute la fonction sur tous les éléments de l'Array et retourne un Array avec les résultats.
.some()	Exécute la fonction sur tous les éléments de l'Array et retourne true si la fonction à retourné true pour au moins un élément de l'Array .



# Vérification de note dupliquée

Pour vérifier que le titre de la note n'existe pas, nous utiliserons filter sur notre array.

Le but est de créer un array avec les notes dupliquées et de vérifier que sa propriété length est bien à 0.

```
const addNote = function(title, body){
  const notes = loadNotes(FILE_NOTES);
  const duplicatNotes = notes.filter(function(note){
    return note.title === title;
  });

  if (duplicatNotes.length === 0) {
    notes.push({
      title: title,
      body: body
    });

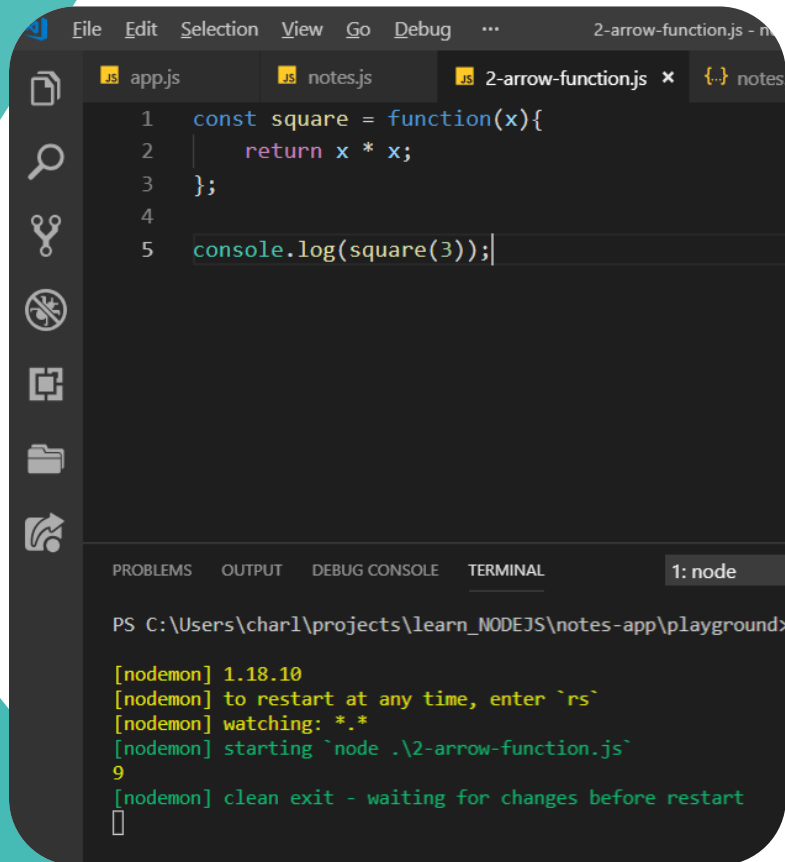
    saveNotes(notes);
    console.log(chalk.green.inverse('New note added!'));
  } else {
    console.log(chalk.blue.inverse('Note title taken!'));
  }
};
```



# Les fonctions Arrow => en JS



# Arrow function (x) => { }

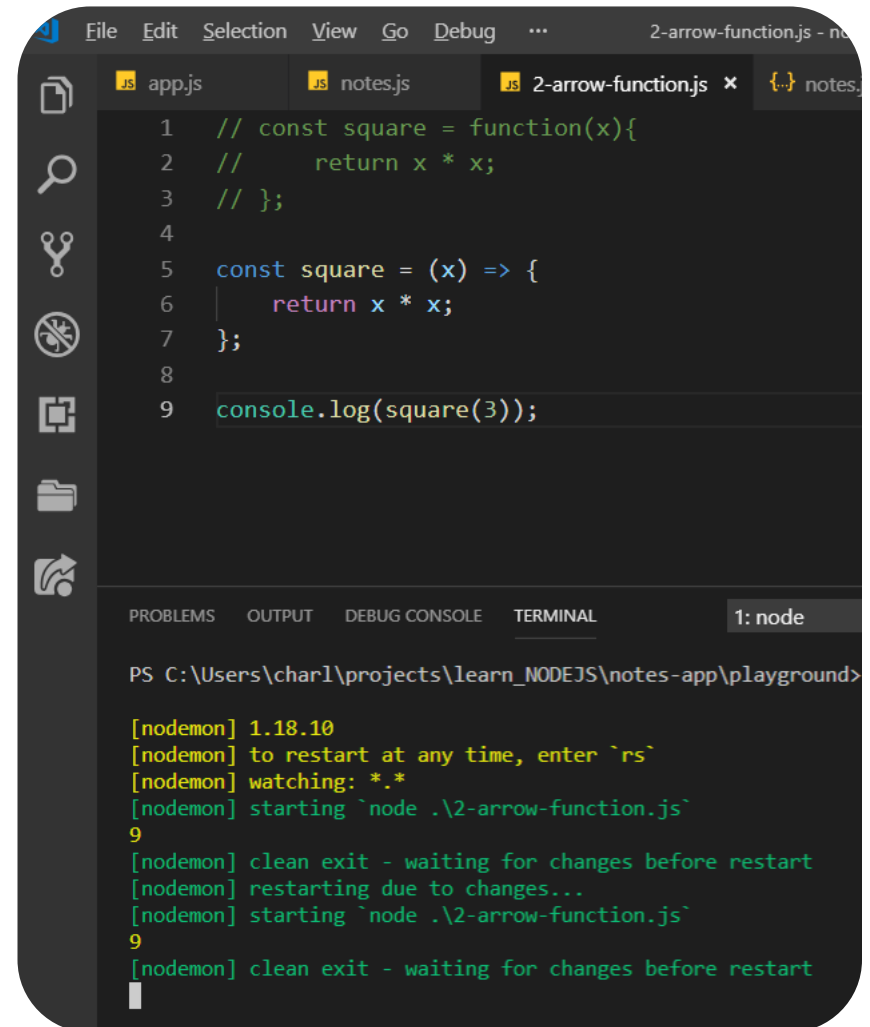


```
1 const square = function(x){
2   return x * x;
3 };
4
5 console.log(square(3));
```

1: node

```
PS C:\Users\charl\projects\learn_NODEJS\notes-app\playground>

[nodemon] 1.18.10
[nodemon] to restart at any time, enter `rs`
[nodemon] watching: *.*
[nodemon] starting `node .\2-arrow-function.js`
9
[nodemon] clean exit - waiting for changes before restart
```



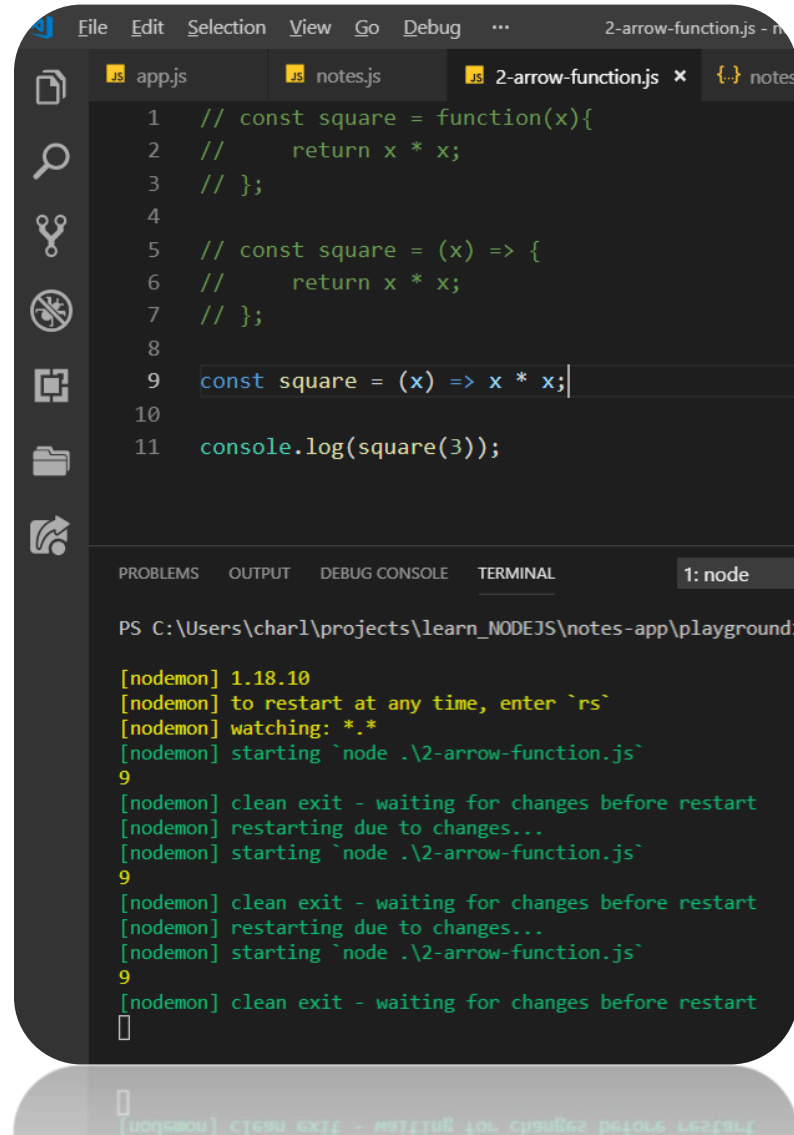
```
1 // const square = function(x){
2 //   return x * x;
3 // };
4
5 const square = (x) => {
6   return x * x;
7 };
8
9 console.log(square(3));
```

1: node

```
PS C:\Users\charl\projects\learn_NODEJS\notes-app\playground>

[nodemon] 1.18.10
[nodemon] to restart at any time, enter `rs`
[nodemon] watching: *.*
[nodemon] starting `node .\2-arrow-function.js`
9
[nodemon] clean exit - waiting for changes before restart
[nodemon] restarting due to changes...
[nodemon] starting `node .\2-arrow-function.js`
9
[nodemon] clean exit - waiting for changes before restart
```

# Shorthand arrow function



```
1 // const square = function(x){
2 //   return x * x;
3 // };
4
5 // const square = (x) => {
6 //   return x * x;
7 // };
8
9 const square = (x) => x * x;
10
11 console.log(square(3));
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL 1: node

```
PS C:\Users\charl\projects\learn_NODEJS\notes-app\playground>
[nodemon] 1.18.10
[nodemon] to restart at any time, enter `rs`
[nodemon] watching: *.*
[nodemon] starting `node .\2-arrow-function.js`
9
[nodemon] clean exit - waiting for changes before restart
[nodemon] restarting due to changes...
[nodemon] starting `node .\2-arrow-function.js`
9
[nodemon] clean exit - waiting for changes before restart
[nodemon] restarting due to changes...
[nodemon] starting `node .\2-arrow-function.js`
9
[nodemon] clean exit - waiting for changes before restart
```

# Arrow function for methods

Le contexte d'une fonction arrow depend de l'endroit ou elle est créée

```
File Edit Selection View Go Debug ...
app.js notes.js 2-arrow-function.js x notes.json
9 // const square = (x) => x * x;
10
11 // console.log(square(3));
12
13 const event = {
14   name: 'Birthday Party',
15   printGuestList: function(){
16     console.log('Guest List for ' + this.name);
17   }
18 };
19
20 event.printGuestList();
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL 1: node

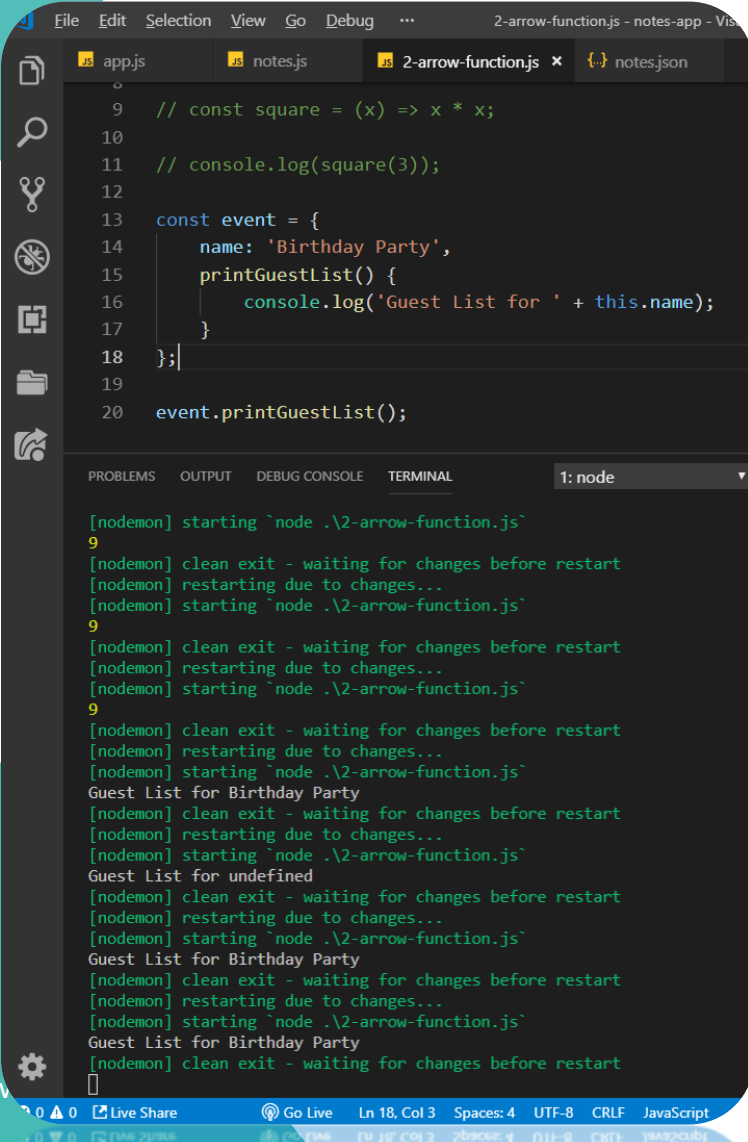
```
PS C:\Users\charl\projects\learn_NODEJS\notes-app\playground> nodemon
[nodemon] 1.18.10
[nodemon] to restart at any time, enter `rs`
[nodemon] watching: *.*
[nodemon] starting `node .\2-arrow-function.js`
9
[nodemon] clean exit - waiting for changes before restart
[nodemon] restarting due to changes...
[nodemon] starting `node .\2-arrow-function.js`
9
[nodemon] clean exit - waiting for changes before restart
[nodemon] restarting due to changes...
[nodemon] starting `node .\2-arrow-function.js`
9
[nodemon] clean exit - waiting for changes before restart
[nodemon] restarting due to changes...
[nodemon] starting `node .\2-arrow-function.js`
Guest List for Birthday Party
[nodemon] clean exit - waiting for changes before restart
```

```
File Edit Selection View Go Debug ...
app.js notes.js 2-arrow-function.js x notes.json
9 // const square = (x) => x * x;
10
11 // console.log(square(3));
12
13 const event = {
14   name: 'Birthday Party',
15   printGuestList: () => {
16     console.log('Guest List for ' + this.name);
17   }
18 };
19
20 event.printGuestList();
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL 1: node

```
PS C:\Users\charl\projects\learn_NODEJS\notes-app\playground> nodemon .\
[nodemon] 1.18.10
[nodemon] to restart at any time, enter `rs`
[nodemon] watching: *.*
[nodemon] starting `node .\2-arrow-function.js`
9
[nodemon] clean exit - waiting for changes before restart
[nodemon] restarting due to changes...
[nodemon] starting `node .\2-arrow-function.js`
9
[nodemon] clean exit - waiting for changes before restart
[nodemon] restarting due to changes...
[nodemon] starting `node .\2-arrow-function.js`
9
[nodemon] clean exit - waiting for changes before restart
[nodemon] restarting due to changes...
[nodemon] starting `node .\2-arrow-function.js`
Guest List for Birthday Party
[nodemon] clean exit - waiting for changes before restart
[nodemon] restarting due to changes...
[nodemon] starting `node .\2-arrow-function.js`
Guest List for undefined
[nodemon] clean exit - waiting for changes before restart
```

# Solution - arrow functions for methods



```
File Edit Selection View Go Debug ... 2-arrow-function.js - notes-app - Vis
app.js notes.js 2-arrow-function.js x notes.json
9 // const square = (x) => x * x;
10
11 // console.log(square(3));
12
13 const event = {
14   name: 'Birthday Party',
15   printGuestList() {
16     console.log('Guest List for ' + this.name);
17   }
18 };
19
20 event.printGuestList();
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL 1: node

```
[nodemon] starting `node .\2-arrow-function.js`
9
[nodemon] clean exit - waiting for changes before restart
[nodemon] restarting due to changes...
[nodemon] starting `node .\2-arrow-function.js`
9
[nodemon] clean exit - waiting for changes before restart
[nodemon] restarting due to changes...
[nodemon] starting `node .\2-arrow-function.js`
Guest List for Birthday Party
[nodemon] clean exit - waiting for changes before restart
[nodemon] restarting due to changes...
[nodemon] starting `node .\2-arrow-function.js`
Guest List for undefined
[nodemon] clean exit - waiting for changes before restart
[nodemon] restarting due to changes...
[nodemon] starting `node .\2-arrow-function.js`
Guest List for Birthday Party
[nodemon] clean exit - waiting for changes before restart
[nodemon] restarting due to changes...
[nodemon] starting `node .\2-arrow-function.js`
Guest List for Birthday Party
[nodemon] clean exit - waiting for changes before restart
```



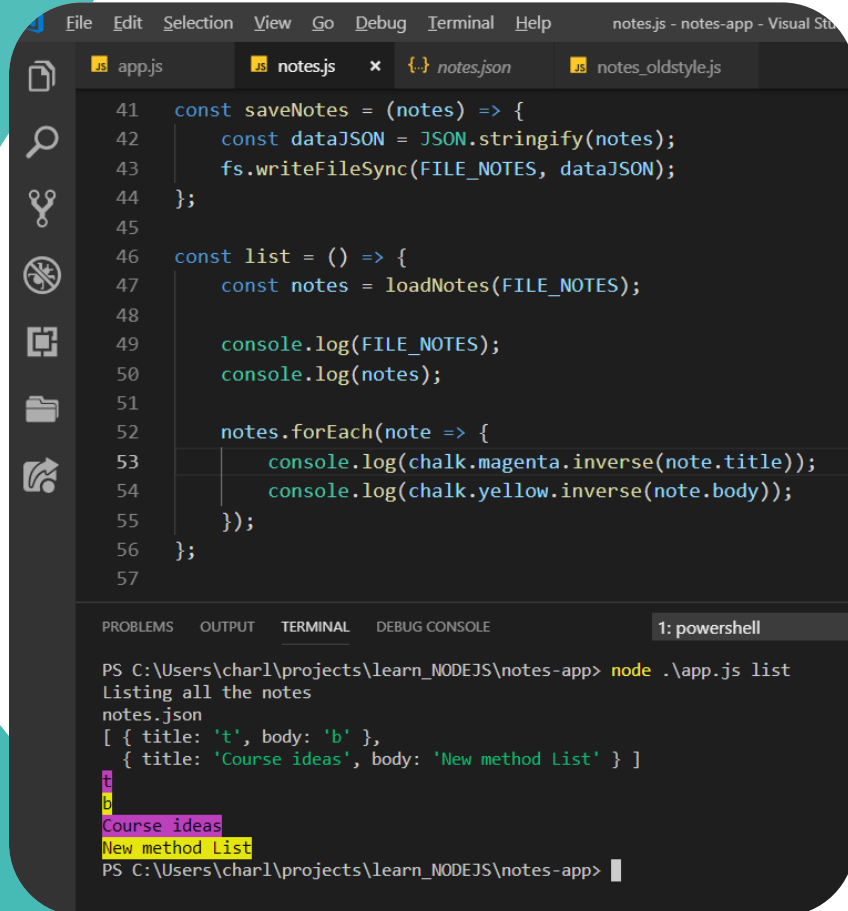
```
app.js notes.js notes.json package.json notes_oldstyle.js 2-a
10
11 // console.log(square(3));
12
13 const event = {
14   name: 'Birthday Party',
15   guestList: ['Charles', 'Sarah', 'Michelle'],
16   printGuestList() {
17     console.log('Guest List for ' + this.name);
18     this.guestList.forEach( (guest) => {
19       console.log(guest + ' is attending ' + this.name);
20     });
21   }
22 };
23
24 event.printGuestList();
```

PROBLEMS OUTPUT TERMINAL DEBUG CONSOLE

```
PS C:\Users\charl\projects\learn_NODEJS\notes-app\playground> node .\2-arrow-function.js
Guest List for Birthday Party
Charles is attending Birthday Party
Sarah is attending Birthday Party
Michelle is attending Birthday Party
```



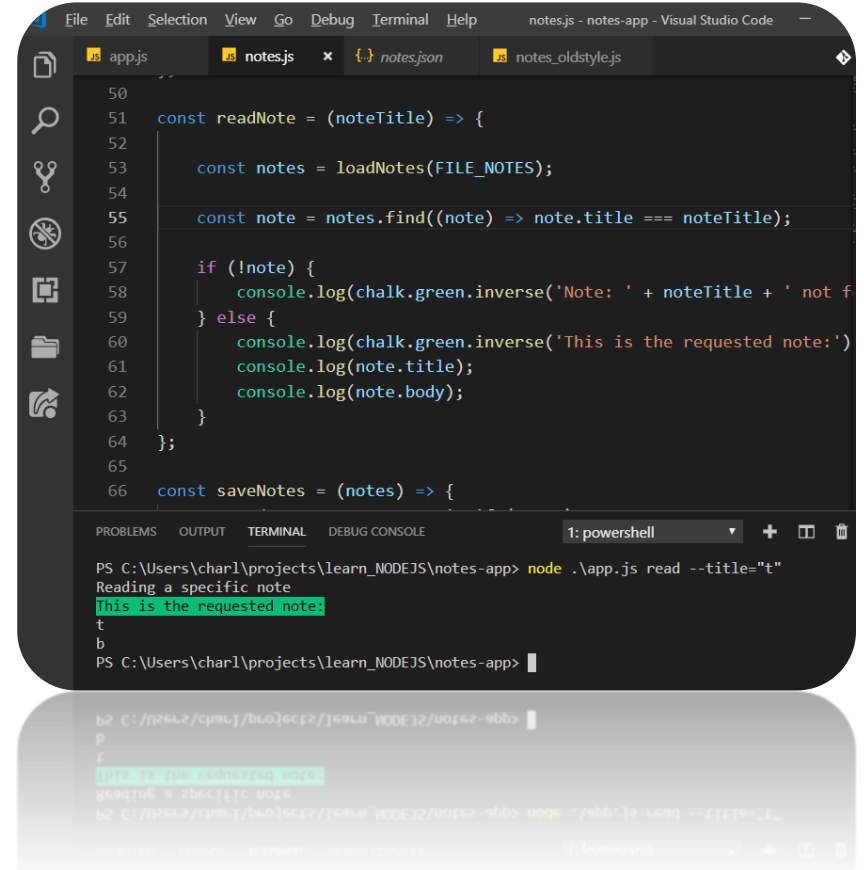
# Transformation de notre application avec des fonctions arrow



```
41 const saveNotes = (notes) => {
42   const dataJSON = JSON.stringify(notes);
43   fs.writeFileSync(FILE_NOTES, dataJSON);
44 };
45
46 const list = () => {
47   const notes = loadNotes(FILE_NOTES);
48
49   console.log(FILE_NOTES);
50   console.log(notes);
51
52   notes.forEach(note => {
53     console.log(chalk.magenta.inverse(note.title));
54     console.log(chalk.yellow.inverse(note.body));
55   });
56 };
57
```

PROBLEMS OUTPUT TERMINAL DEBUG CONSOLE 1: powershell

```
PS C:\Users\charl\projects\learn_NODEJS\notes-app> node .\app.js list
Listing all the notes
notes.json
[ { title: 't', body: 'b' },
  { title: 'Course ideas', body: 'New method List' } ]
t
b
Course ideas
New method List
PS C:\Users\charl\projects\learn_NODEJS\notes-app>
```



```
50
51 const readNote = (noteTitle) => {
52
53   const notes = loadNotes(FILE_NOTES);
54
55   const note = notes.find((note) => note.title === noteTitle);
56
57   if (!note) {
58     console.log(chalk.green.inverse('Note: ' + noteTitle + ' not f
59   } else {
60     console.log(chalk.green.inverse('This is the requested note:'))
61     console.log(note.title);
62     console.log(note.body);
63   }
64 };
65
66 const saveNotes = (notes) => {
```

PROBLEMS OUTPUT TERMINAL DEBUG CONSOLE 1: powershell

```
PS C:\Users\charl\projects\learn_NODEJS\notes-app> node .\app.js read --title="t"
Reading a specific note
This is the requested note:
t
b
PS C:\Users\charl\projects\learn_NODEJS\notes-app>
```

# Exercise



# Utiliser filter avec les fonctions arrow

Ecrire une fonction getTodo qui devra récupérer les taches non complétées (false) à l'aide de filter.

Utiliser une arrow fonction pour les objets avec ES6.

```
const todo = {  
  todos: [  
    {  
      text: 'Achats pour la maison',  
      completed: true  
    },  
    {  
      text: 'Rendre le livre au voisin',  
      completed: false  
    },  
    {  
      text: 'Aller chercher mon costume',  
      completed: false  
    }  
  ]  
};
```