Hands-on NodeJS

+ JavaScript
+ Debug(WebStorm)
+ TypeScript

arnaud.nauwynck@gmail.com

Esilv 2023

This document:
https://github.com/Arnaud-Nauwynck/presentations/tree/main/web/tp1-nodejs.pdf

Objectives

- Discover NodeJs using practical « Hello World » developper code
- Using Simple baby steps, step by step

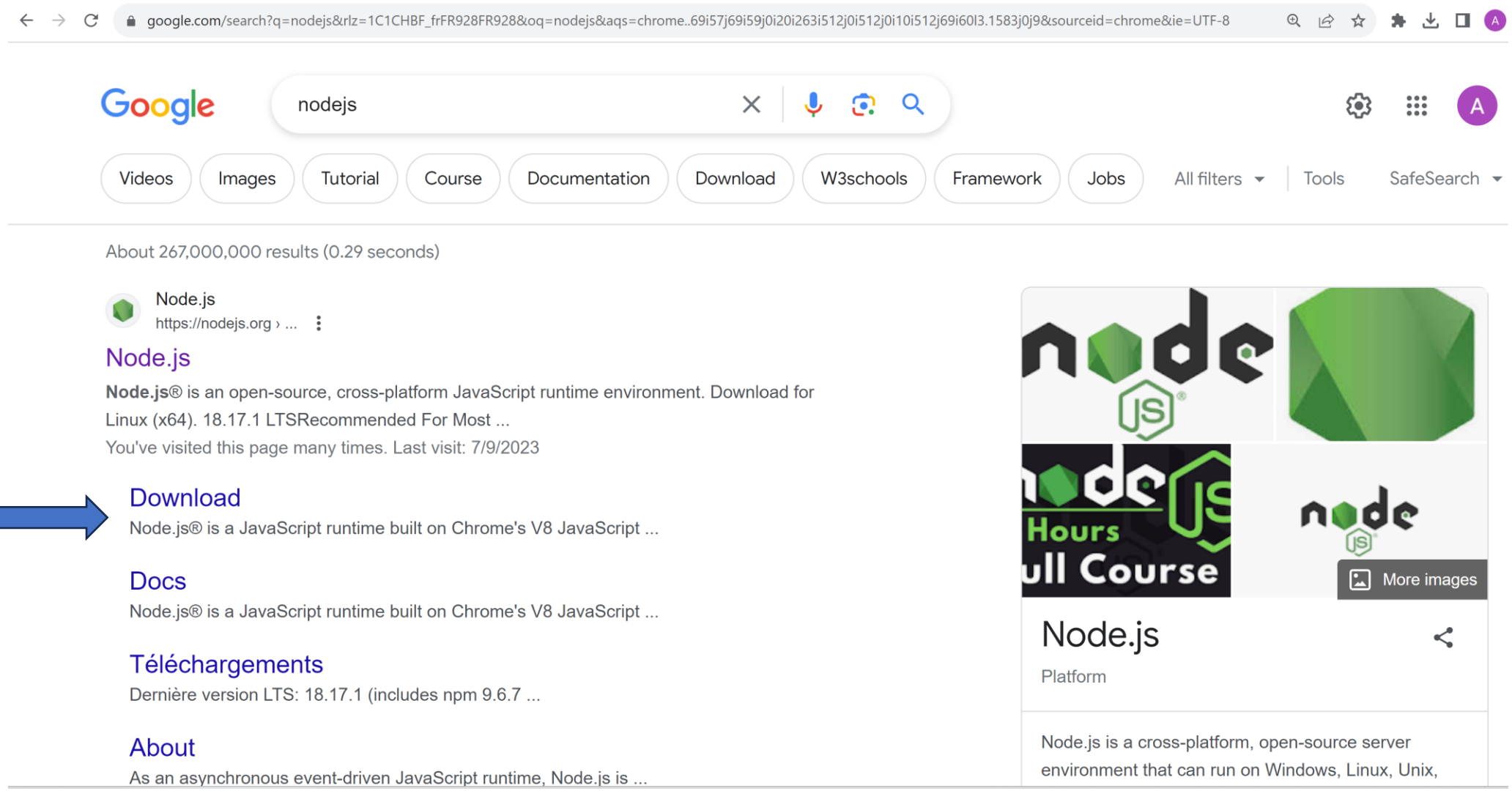
- Setup your web environment

- Become familiar with the Web FrontEnd ecosystem

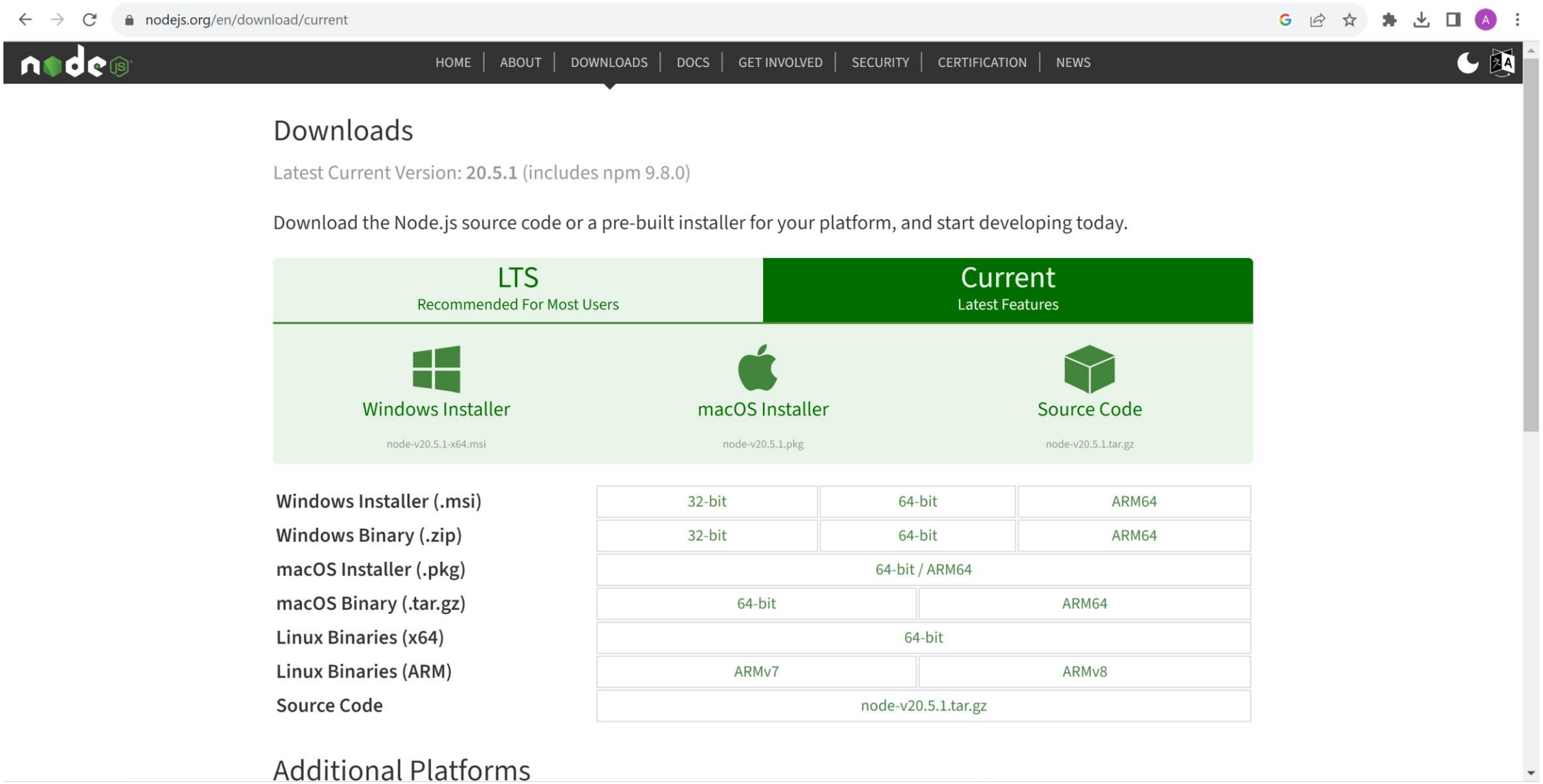
Outline

- Download + Install NodeJs
- Run NodeJS from the command line
- Discover and run basic javascript program
- Discover NPM Node Package Manager
- Install package.json dependencies
- Install WebStorm IDE
  (students may choose any IDE, VisualStudio Code)
- Debug Javascript program
- TypeScript to JavaScript from the IDE
- Tsc compiler
- Debug TypeScript

Google NodeJs



Download NodeJs



Check NodeJs exe (check version)

C:> node -v

System Diagnostic

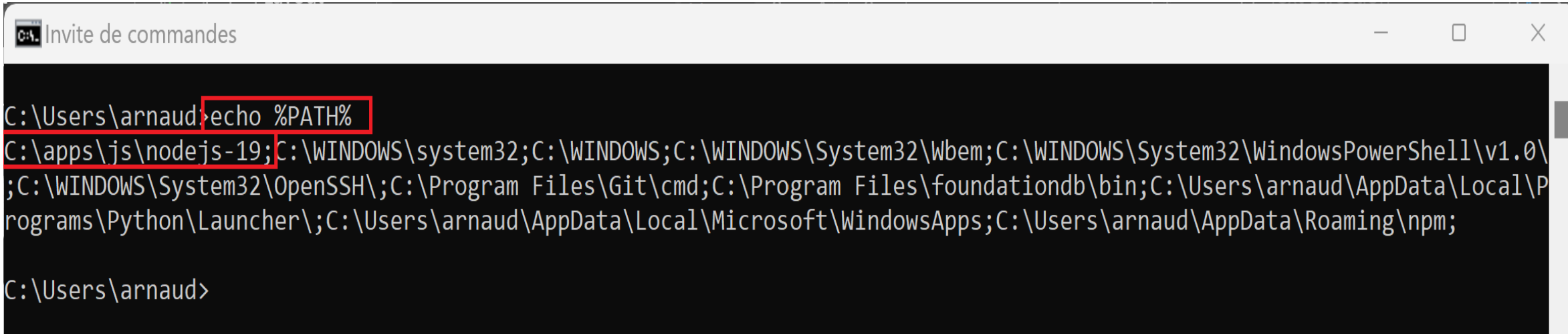
Several installed ?
… which one will be resolved

C:> where node

System Diagnostic

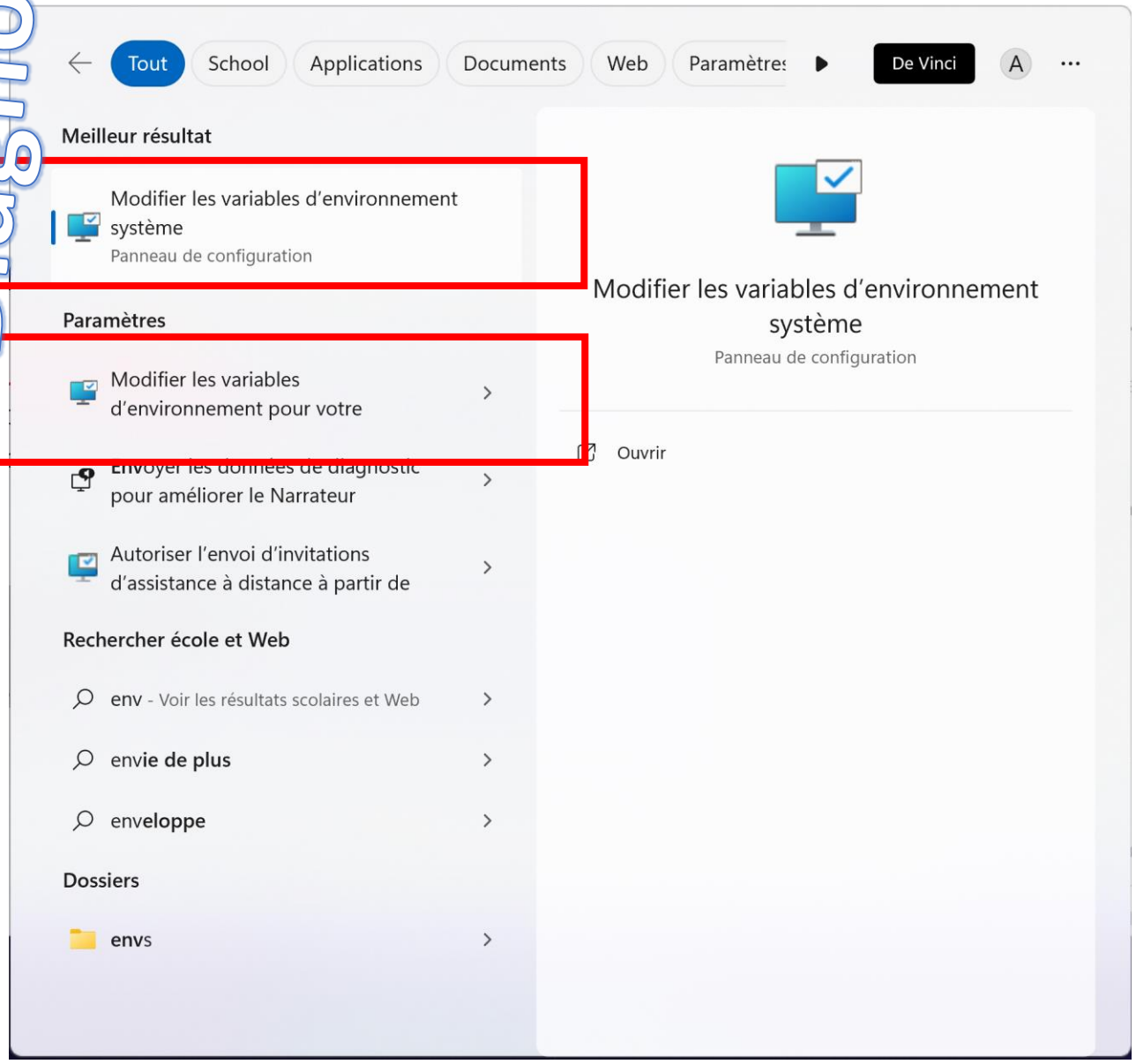
Check NodeJS is in « %PATH% »(win cmd terminal)

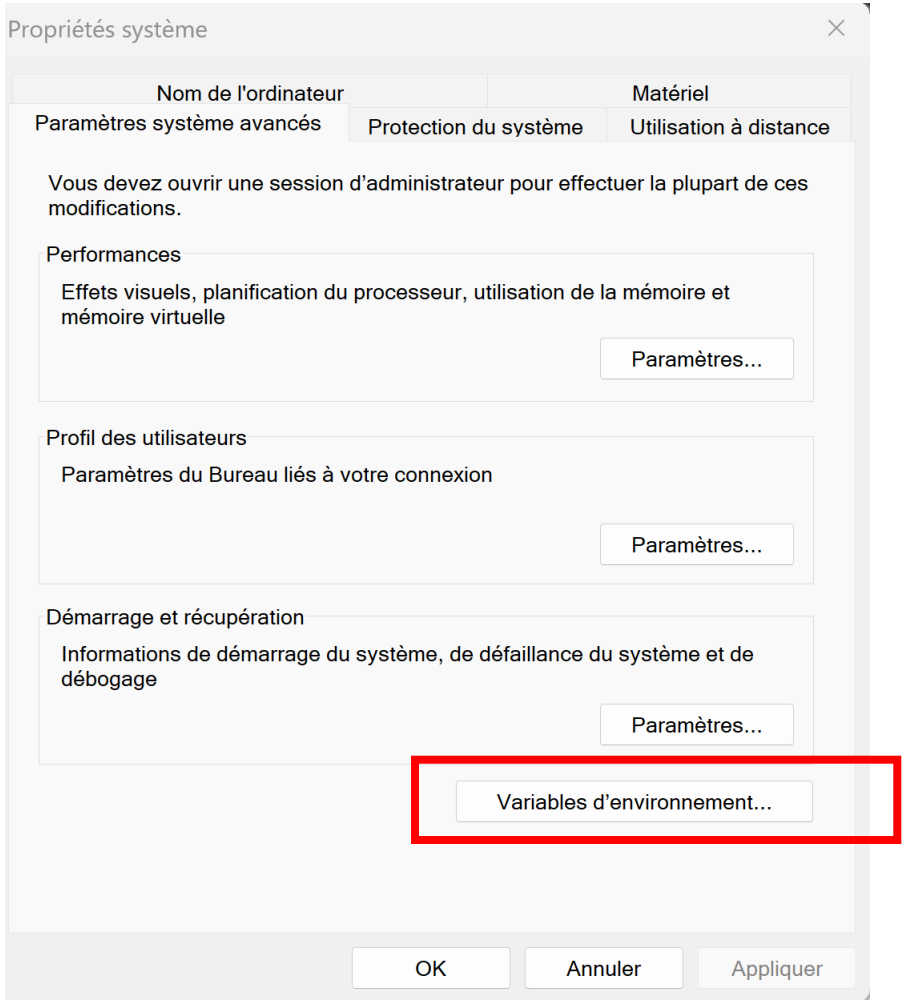
echo %PATH%



System Diagnostic

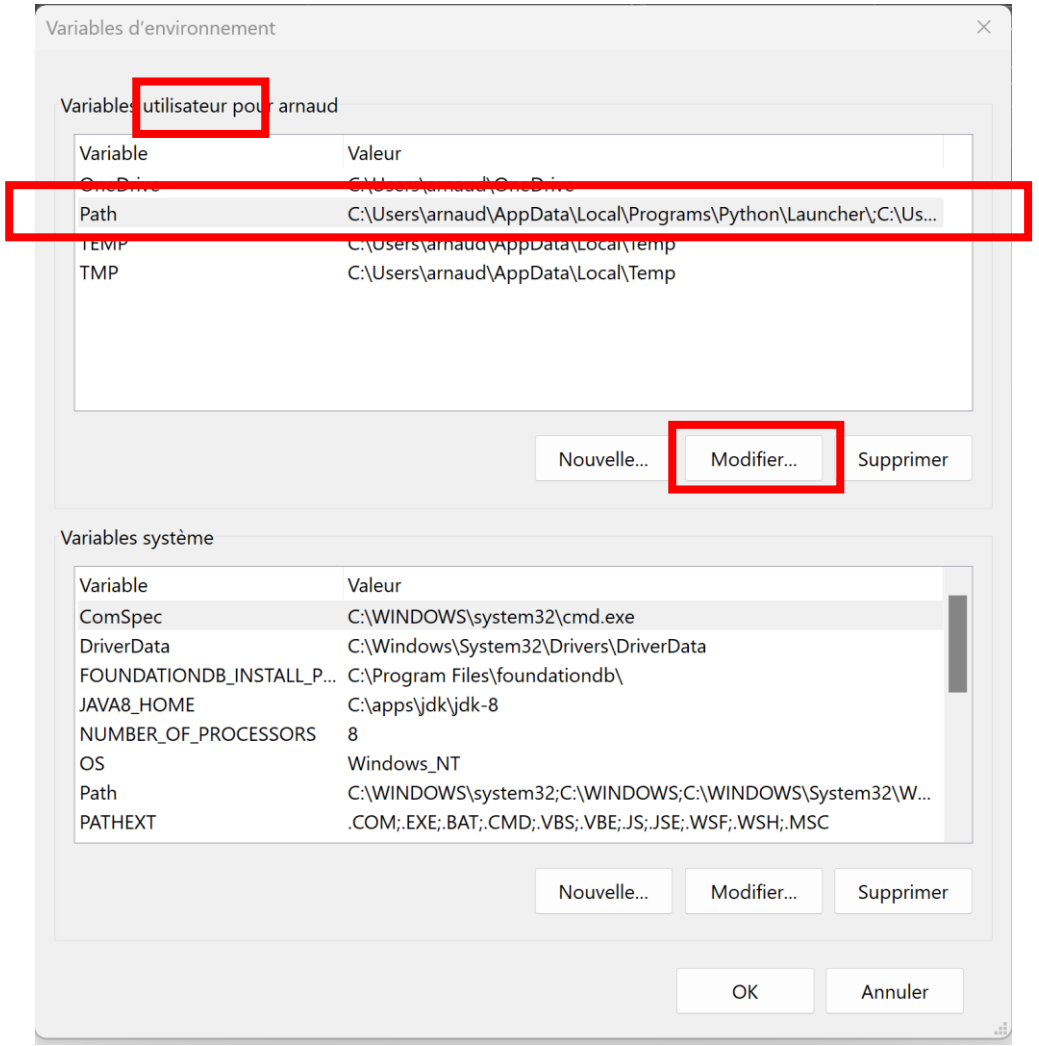
Edit System or User Environment Variables

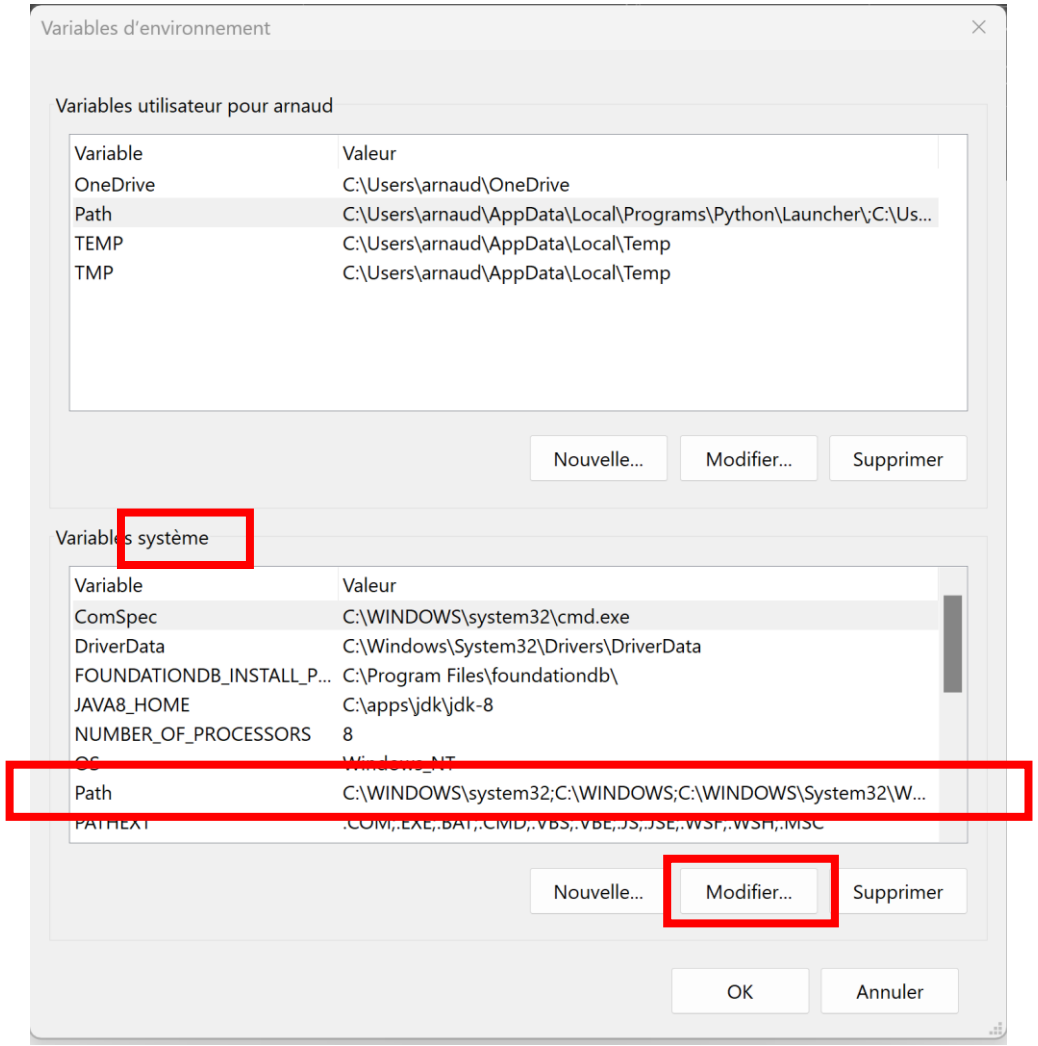




System Diagnostic

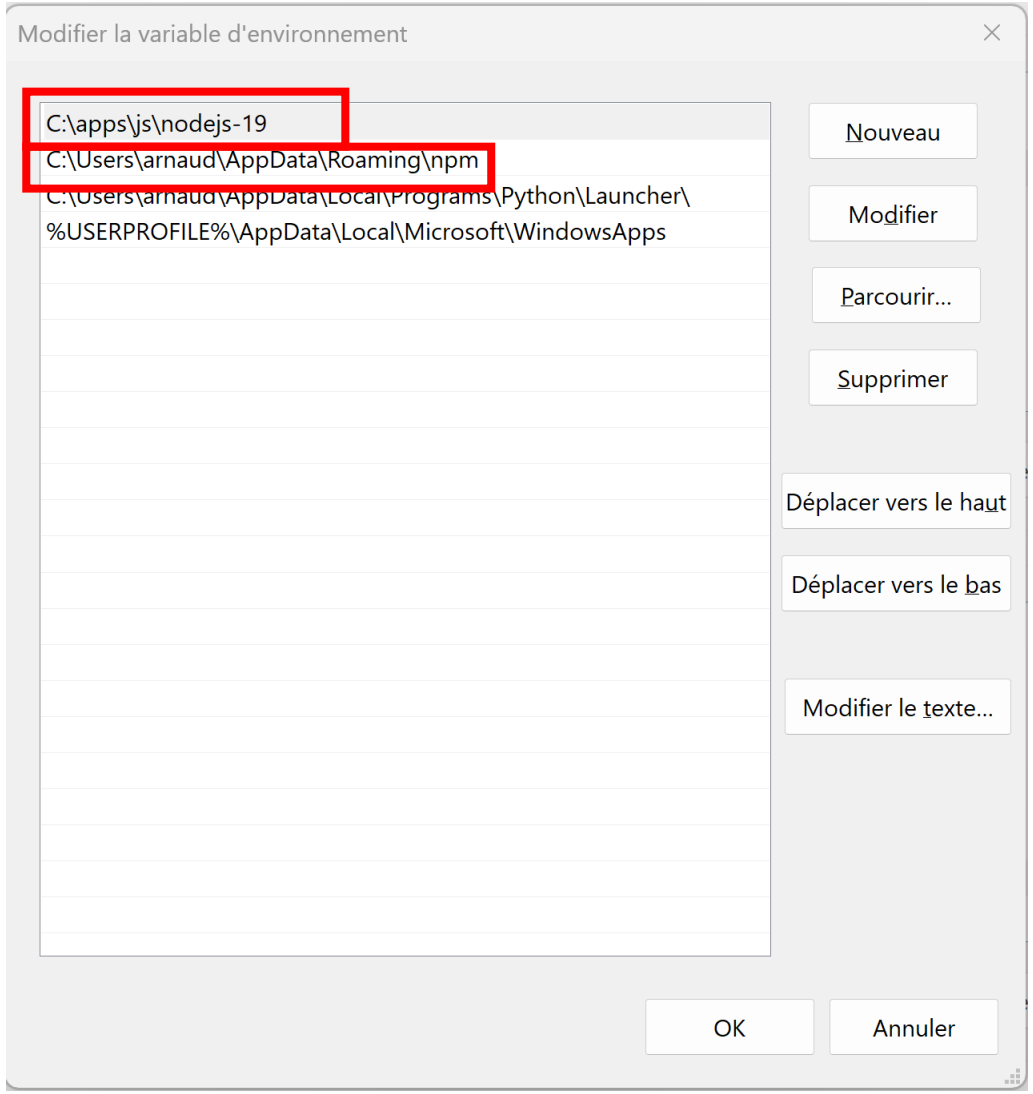
Edit {user|system} PATH environment variable





System Diagnostic

Prepend(=max priority) or Append to PATH



Then **restart**

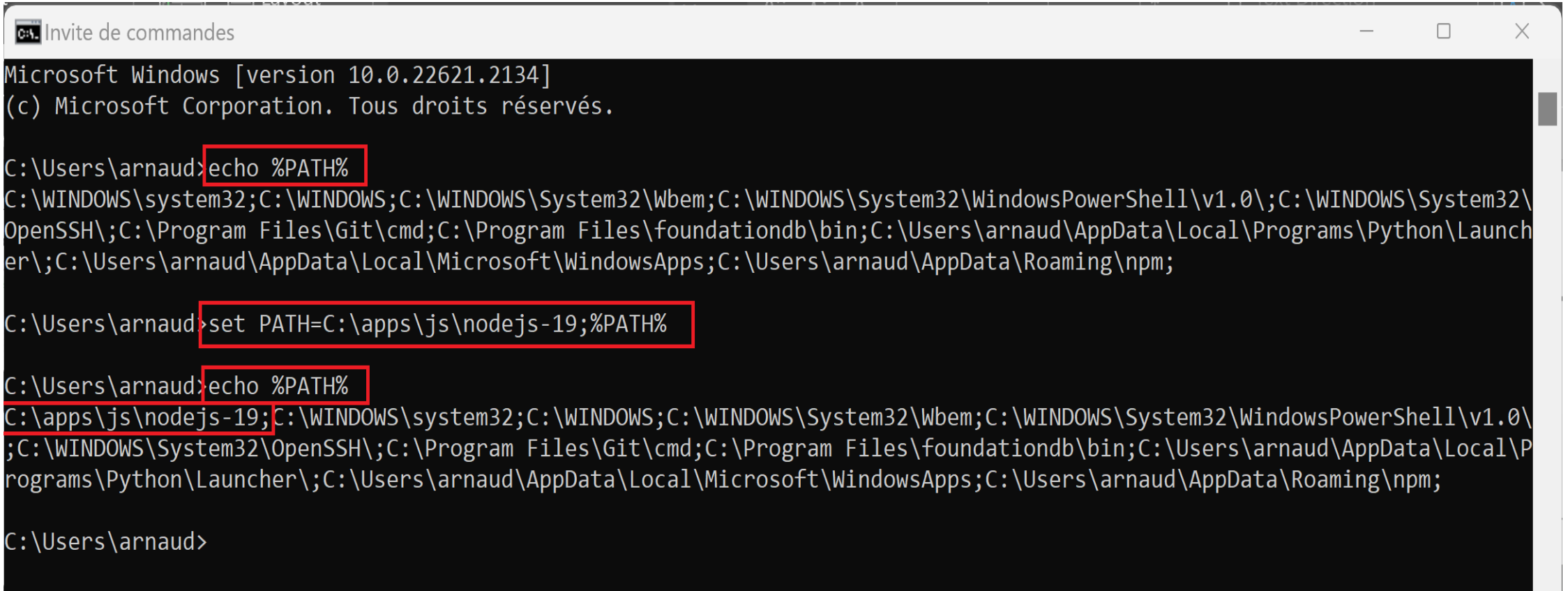
your terminals & apps !!

(env vars are inherited during program starts)

System Diagnostic

Override in terminal to add in « %PATH% » (may edit + call setenv.cmd)

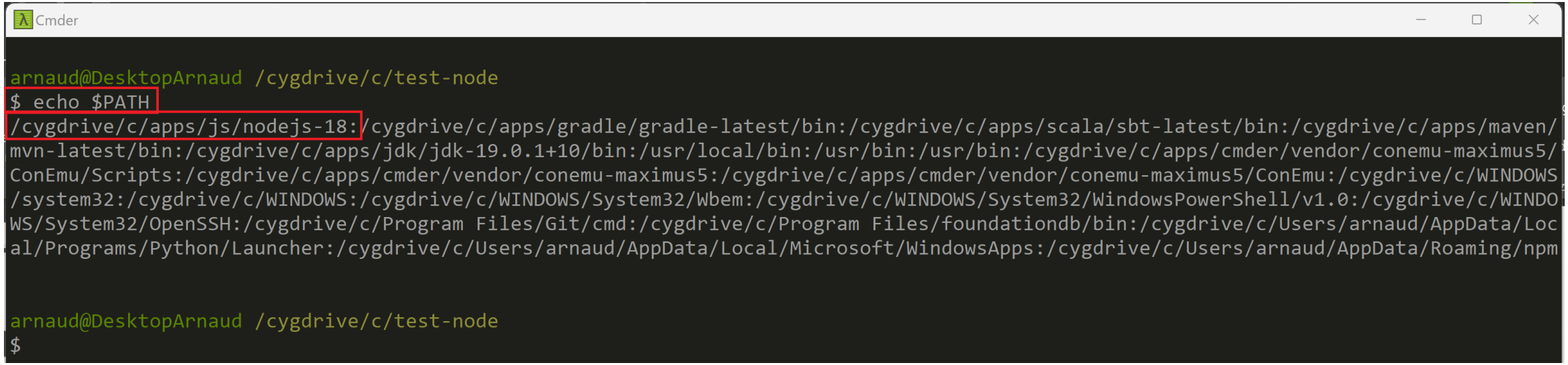
set PATH=c:\your-node-dir;%PATH%



System Diagnostic

Case 2... using Cmder > Bash$
Check NodeJS « $PATH »

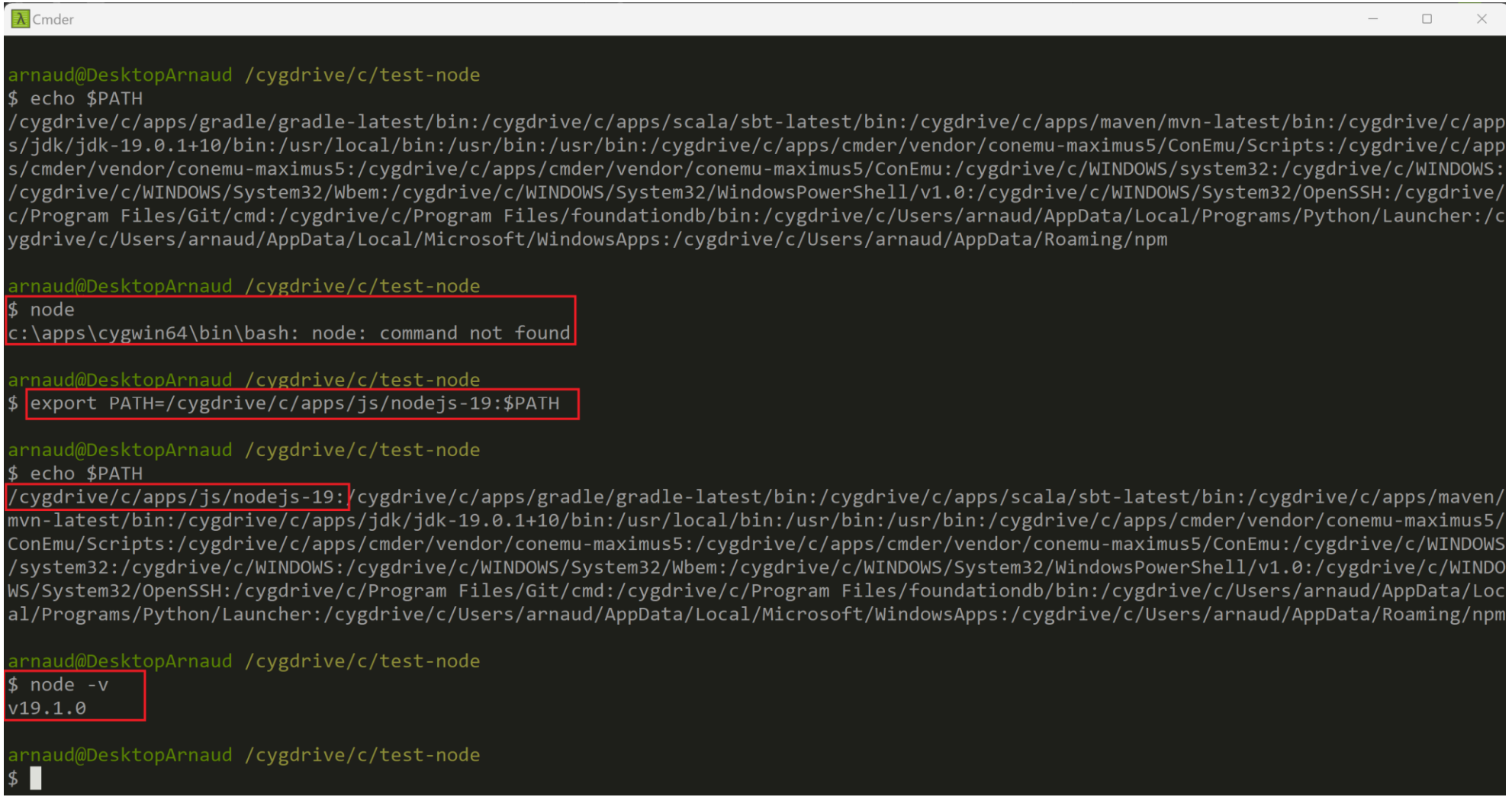
echo $PATH



System Diagnostic

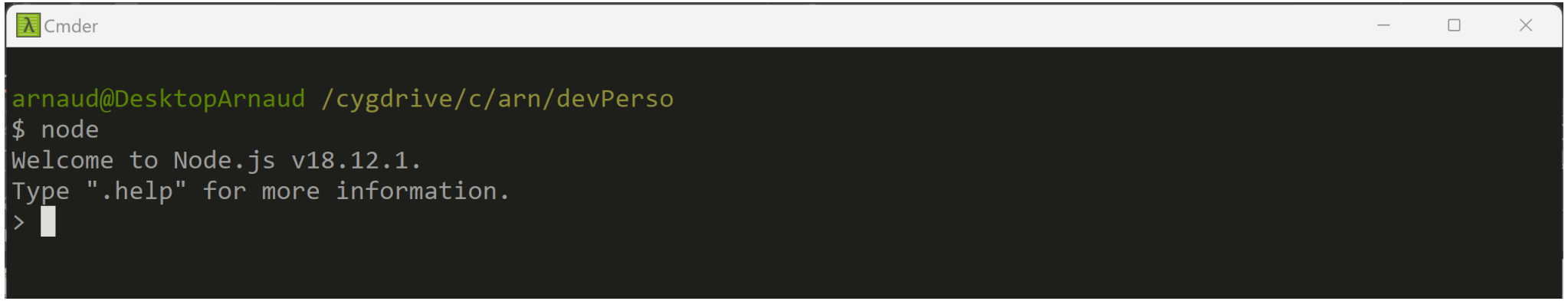
Case 2 … using Cmder Bash, Adding in $PATH (may edit in your $HOME/.profile or .bashrc)

export PATH=/c/your-node-dir:$PATH



Start NodeJs interactive prompt

C:> node

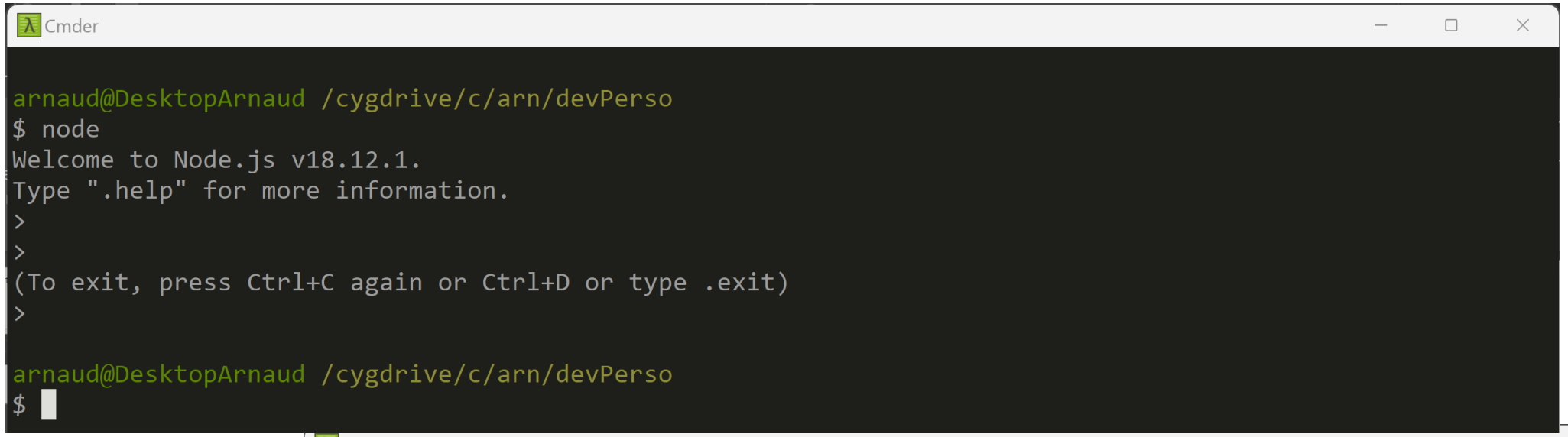


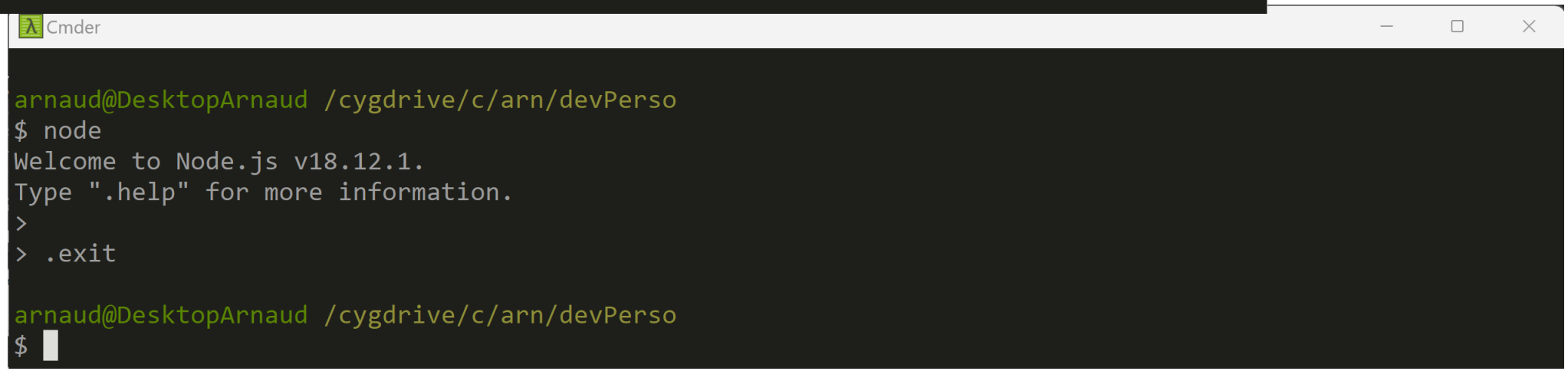
Play with interactive Javascript eval

1+1
console.log(‘Hello’)

Stop Prompt …

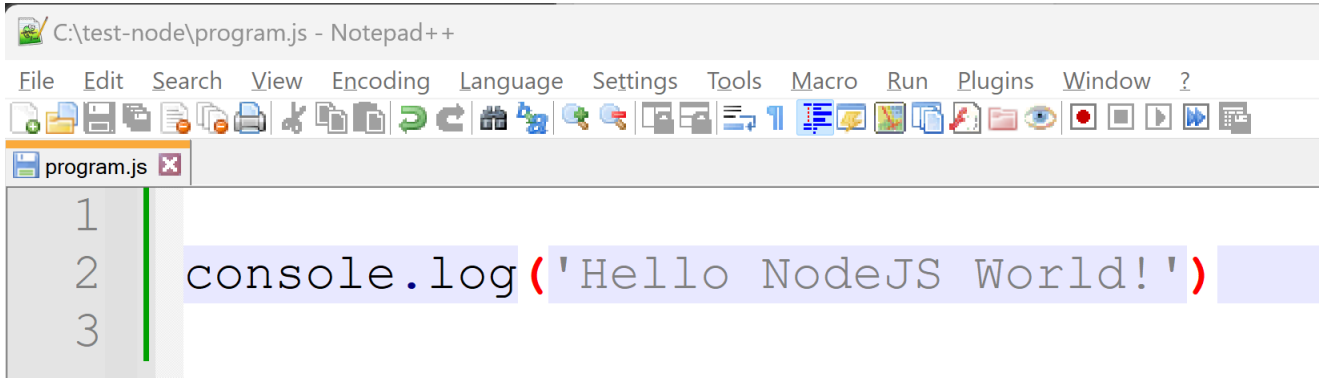
Ctrl+C Ctrl+C or .exit





Start NodeJs on a « program.js » file

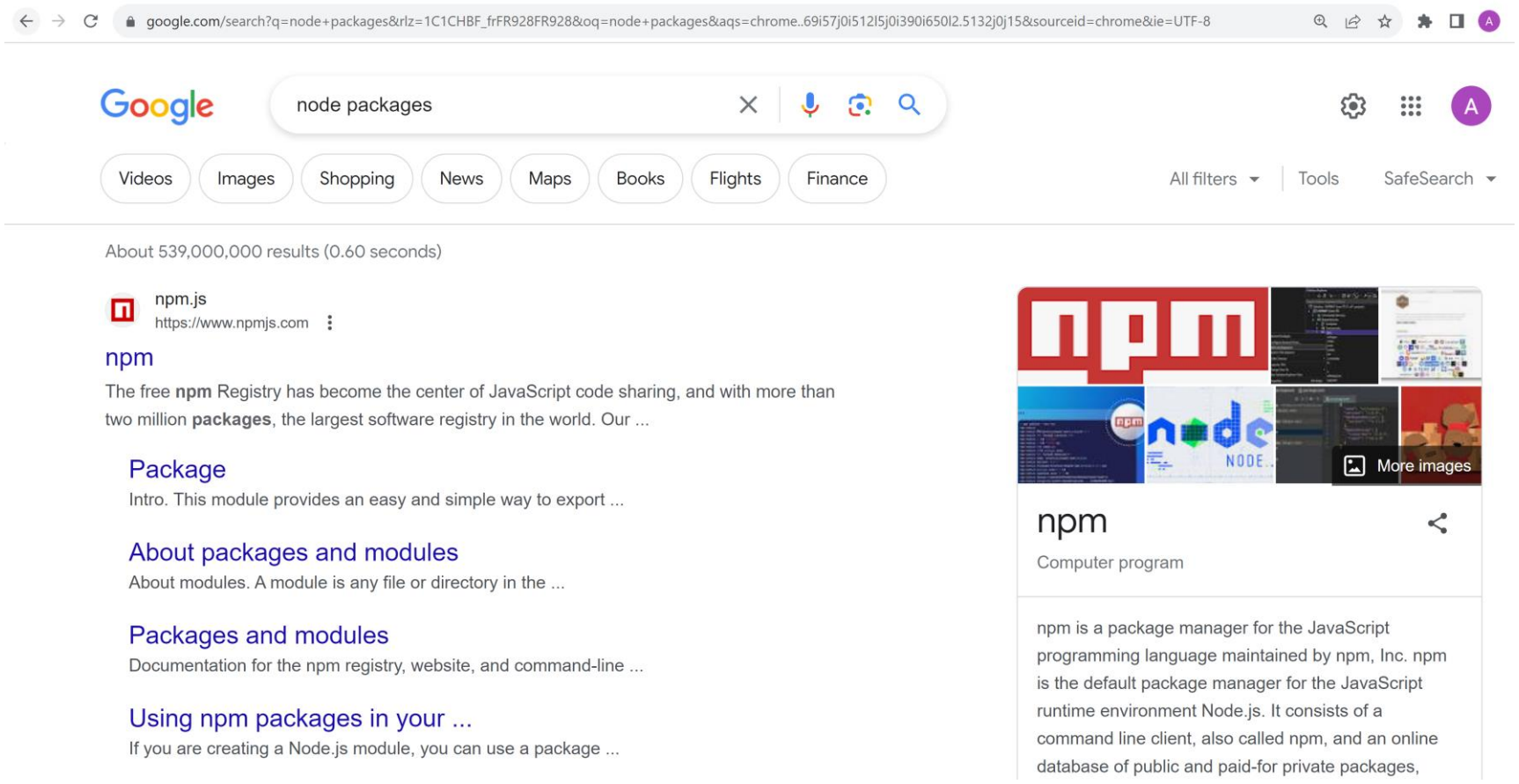
C:> node program.js



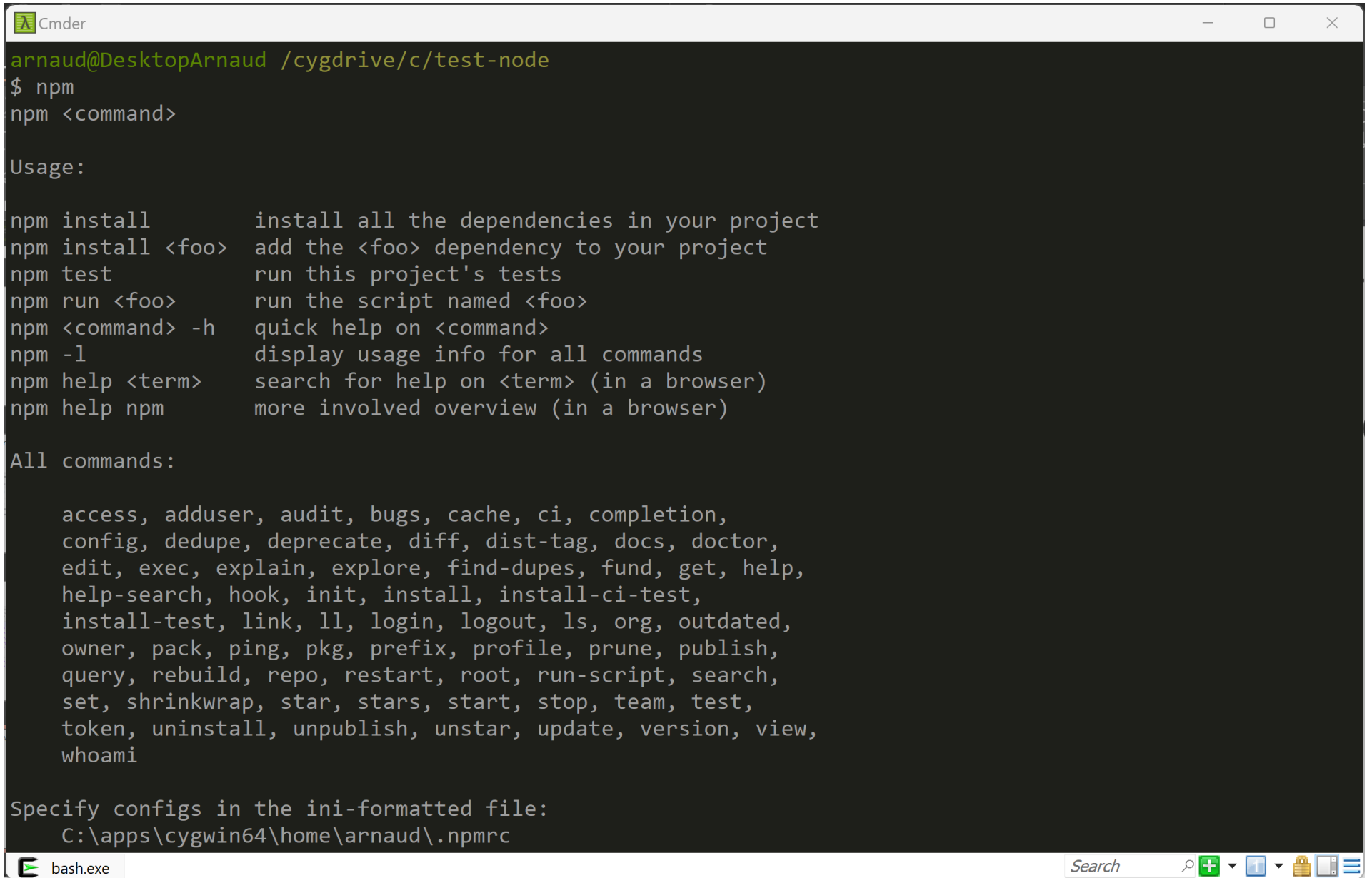
Coffee Break (?)

Node … need Package !!!

Npm = Node Package Manager



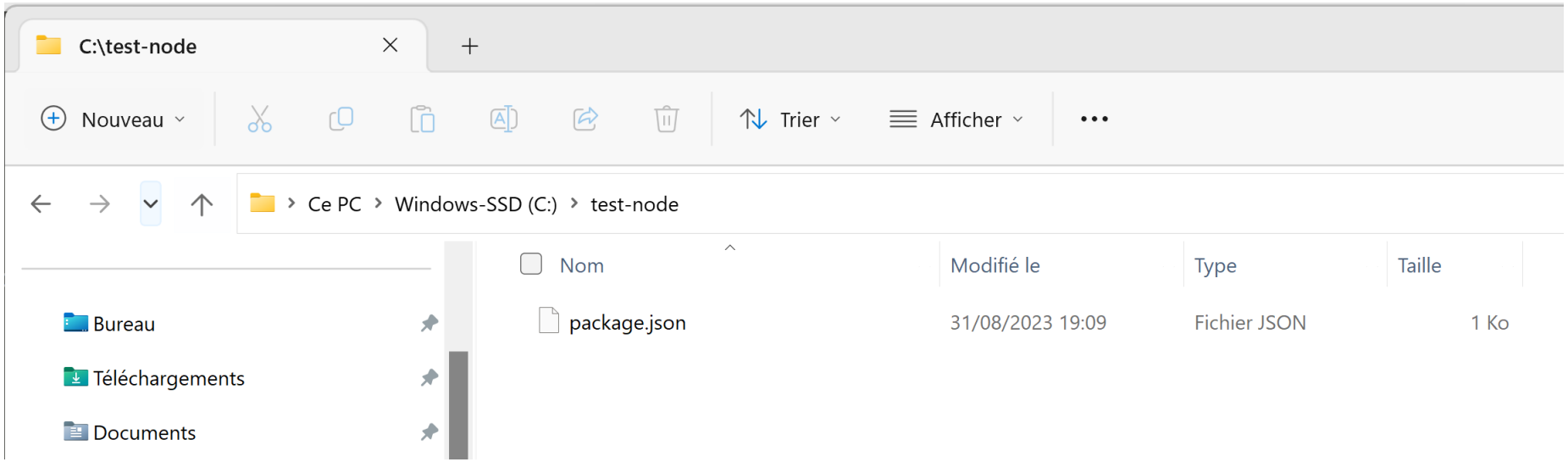
C:> npm


Cmder
arnaud@DesktopArnaud /cygdrive/c/test-node
$ npm
npm <command>
Usage:
npm install install all the dependencies in your project
npm install <foo> add the <foo> dependency to your project
npm test run this project's tests
npm run <foo> run the script named <foo>
npm <command> -h quick help on <command>
npm -l display usage info for all commands
npm help <term> search for help on <term> (in a browser)
npm help npm more involved overview (in a browser)
All commands:
access, adduser, audit, bugs, cache, ci, completion,
config, dedupe, deprecate, diff, dist-tag, docs, doctor,
edit, exec, explain, explore, find-dupes, fund, get, help,
help-search, hook, init, install, install-ci-test,
install-test, link, ll, login, logout, ls, org, outdated,
owner, pack, ping, pkg, prefix, profile, prune, publish,
query, rebuild, repo, restart, root, run-script, search,
set, shrinkwrap, star, stars, start, stop, team, test,
token, uninstall, unpublish, unstar, update, version, view,
whoami
Specify configs in the ini-formatted file:
C:\apps\cygwin64\home\arnaud\.npmrc
bash.exe
Search

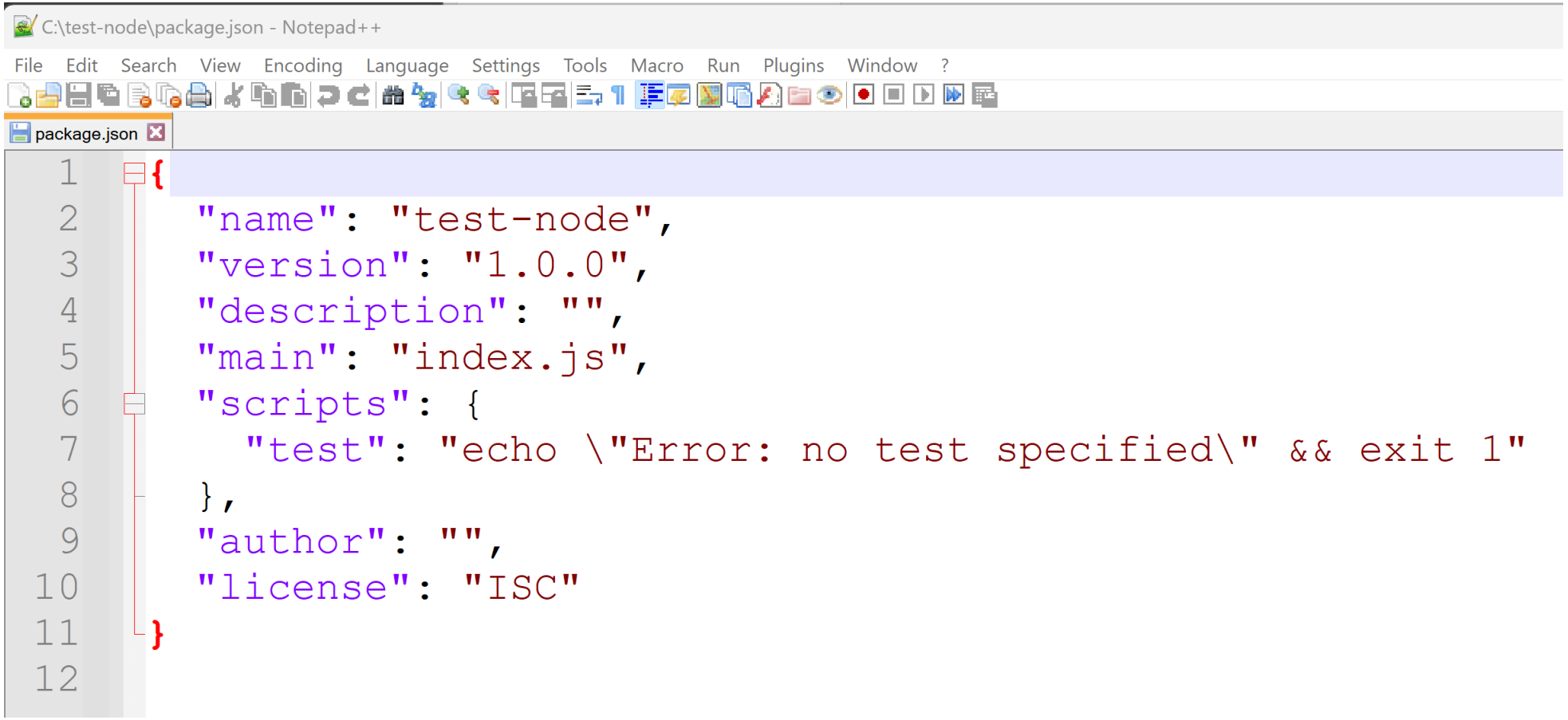
$ npm init

```
$ 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 init` 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: (test-node)
version: (1.0.0)
description:
entry point: (index.js)
test command:
git repository:
keywords:
author:
license: (ISC)
About to write to C:\arn\devPerso\Presentations\web\test-node\package.json:

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


Is this OK? (yes)
```

npm init => open File explorer
=> 1 File « package.json »

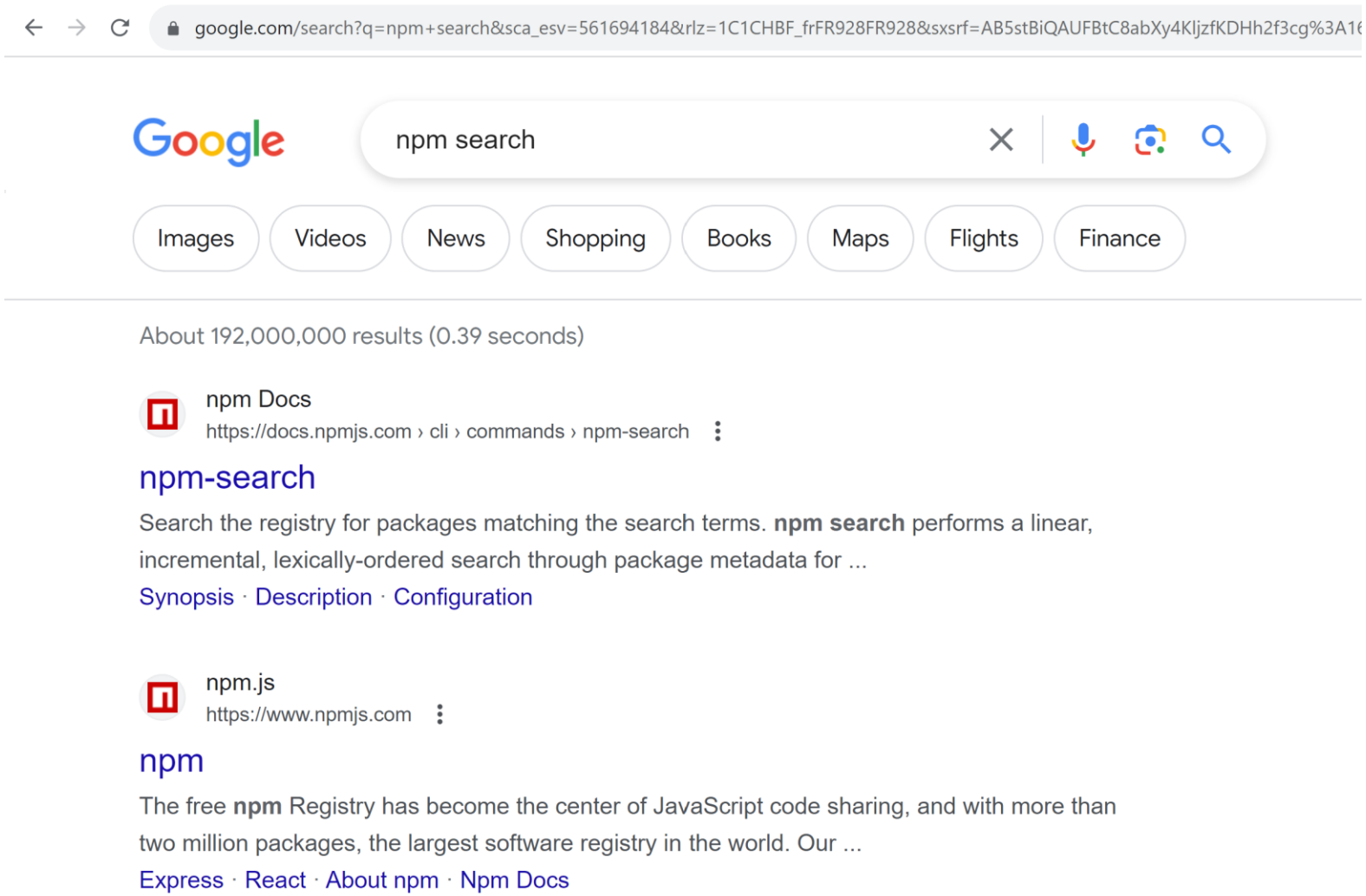


package.json

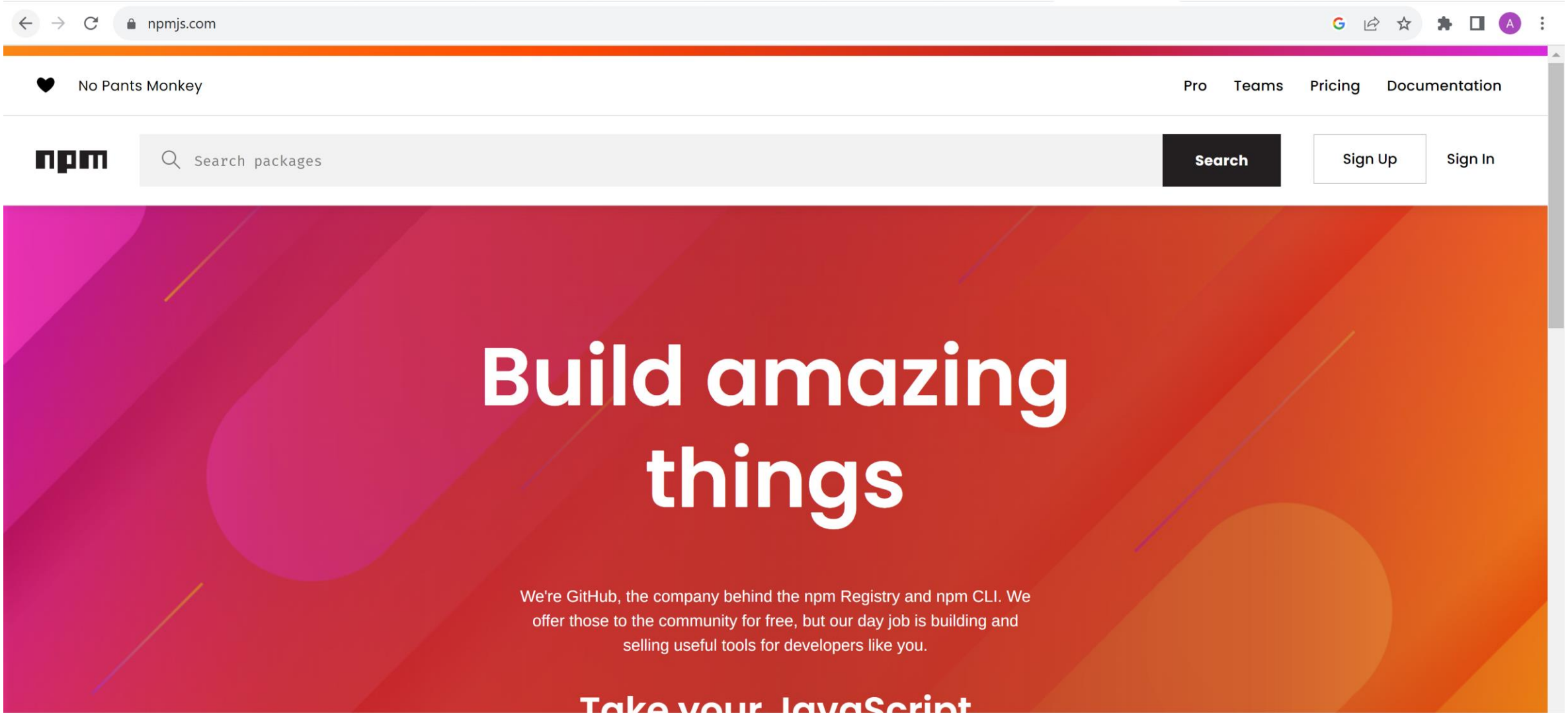


Search Package dependencies ? ...

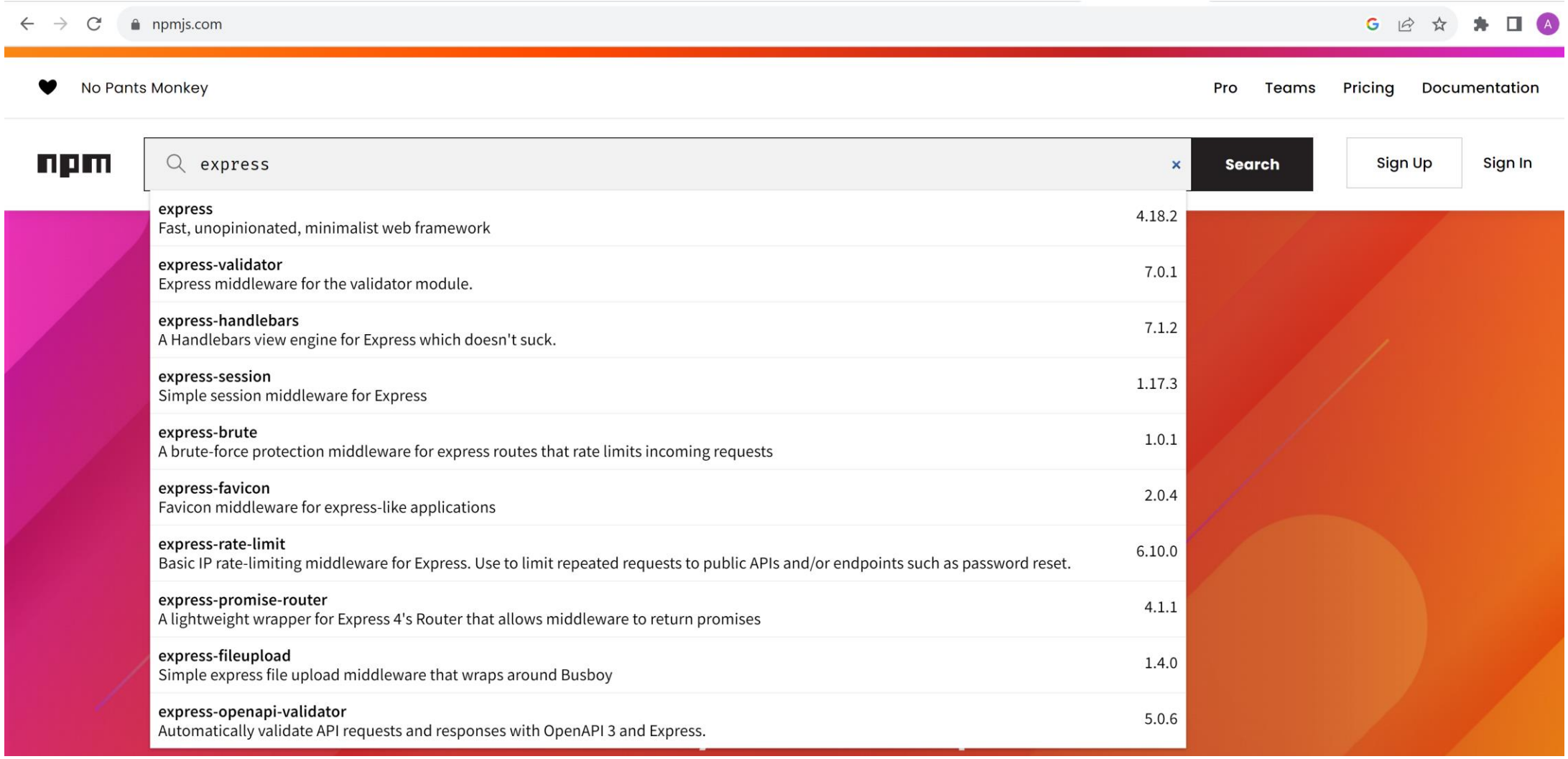
Npm search … Registry (=Repository)



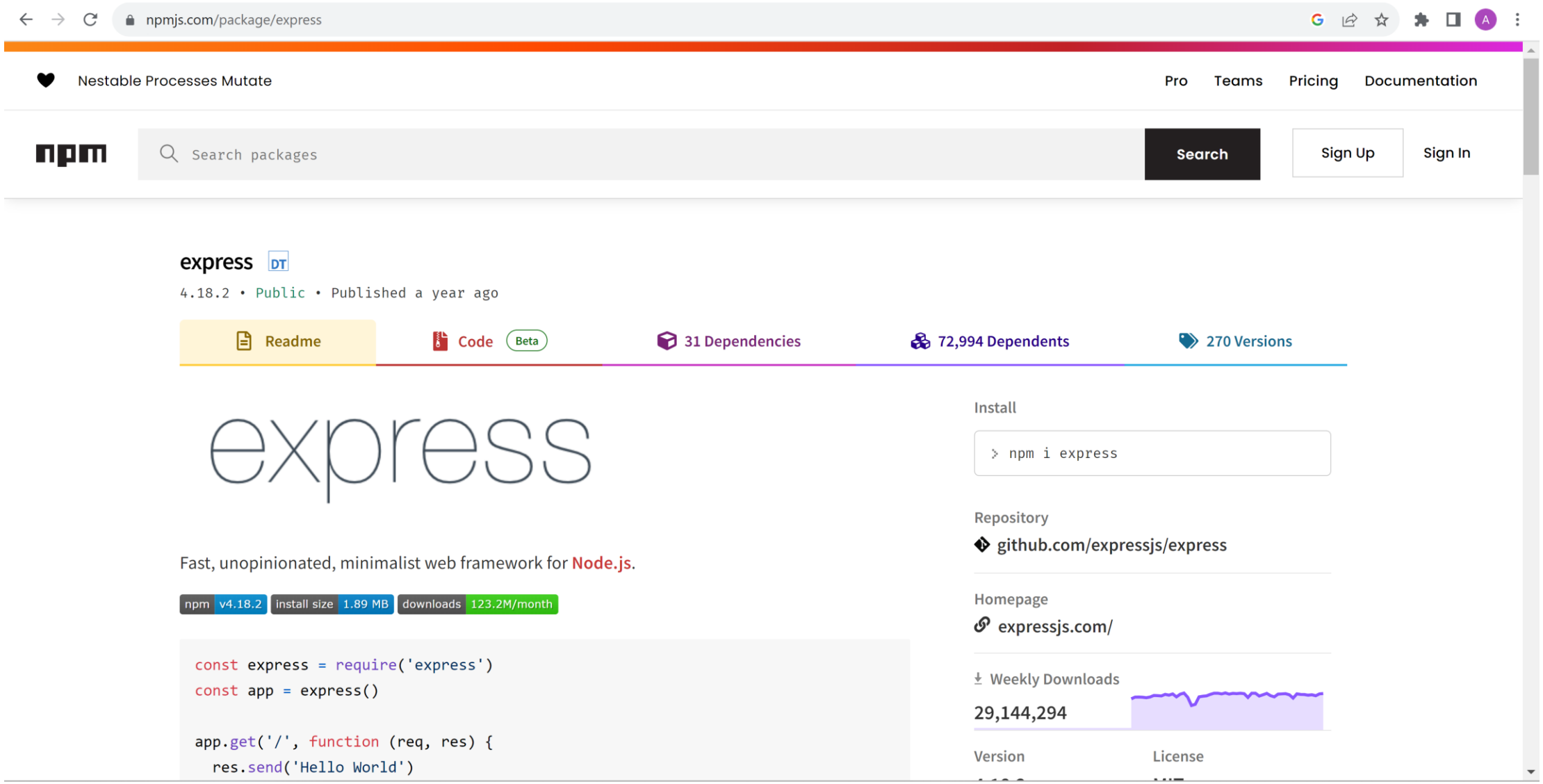
https://npmjs.com



Search « express » package

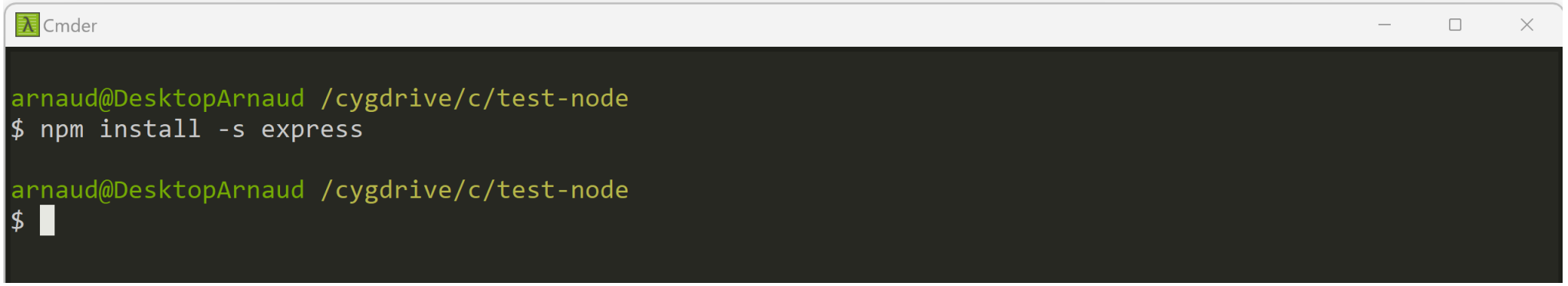


Package express … version / doc / install



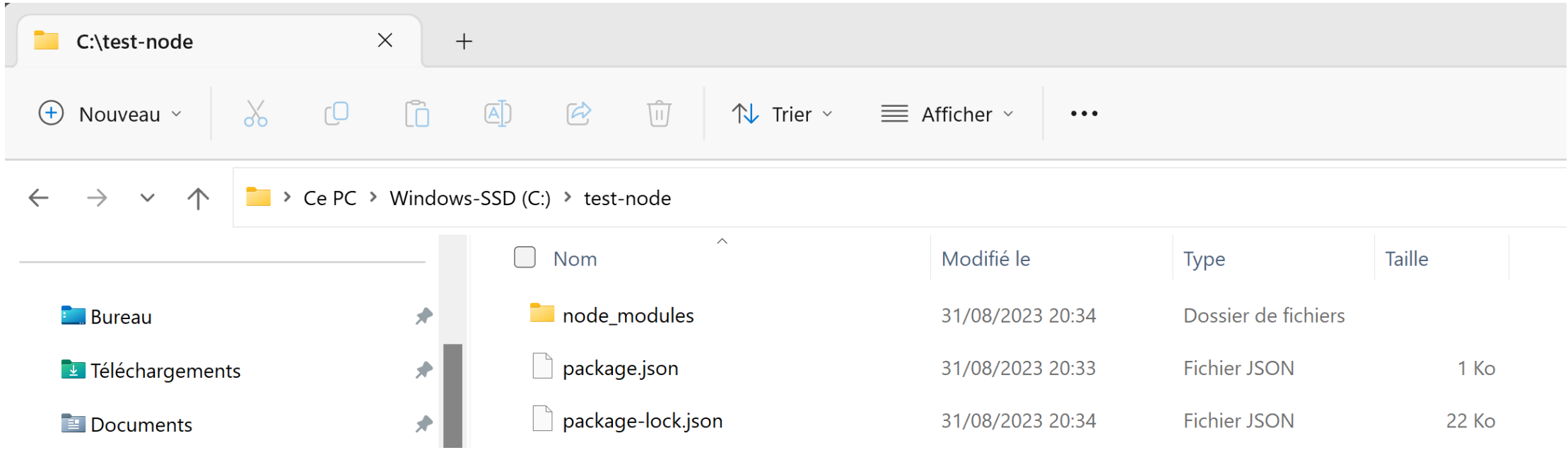
Add npm dependency « express »

C:> npm install –s express
(equivalent: npm install --save express)

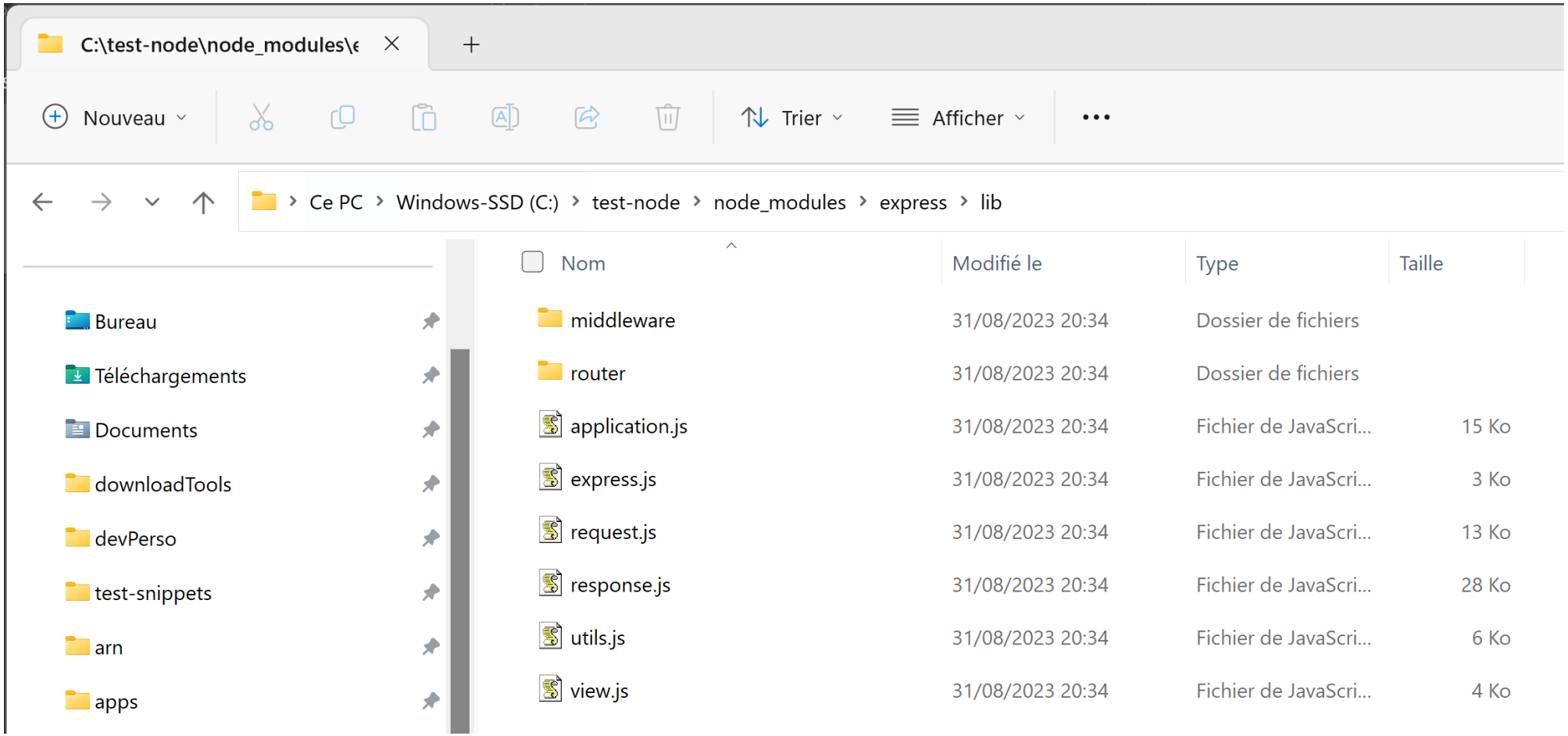


npm --save .... => save to package.json file
+ package-lock.json

+ download **/** to node_modules/

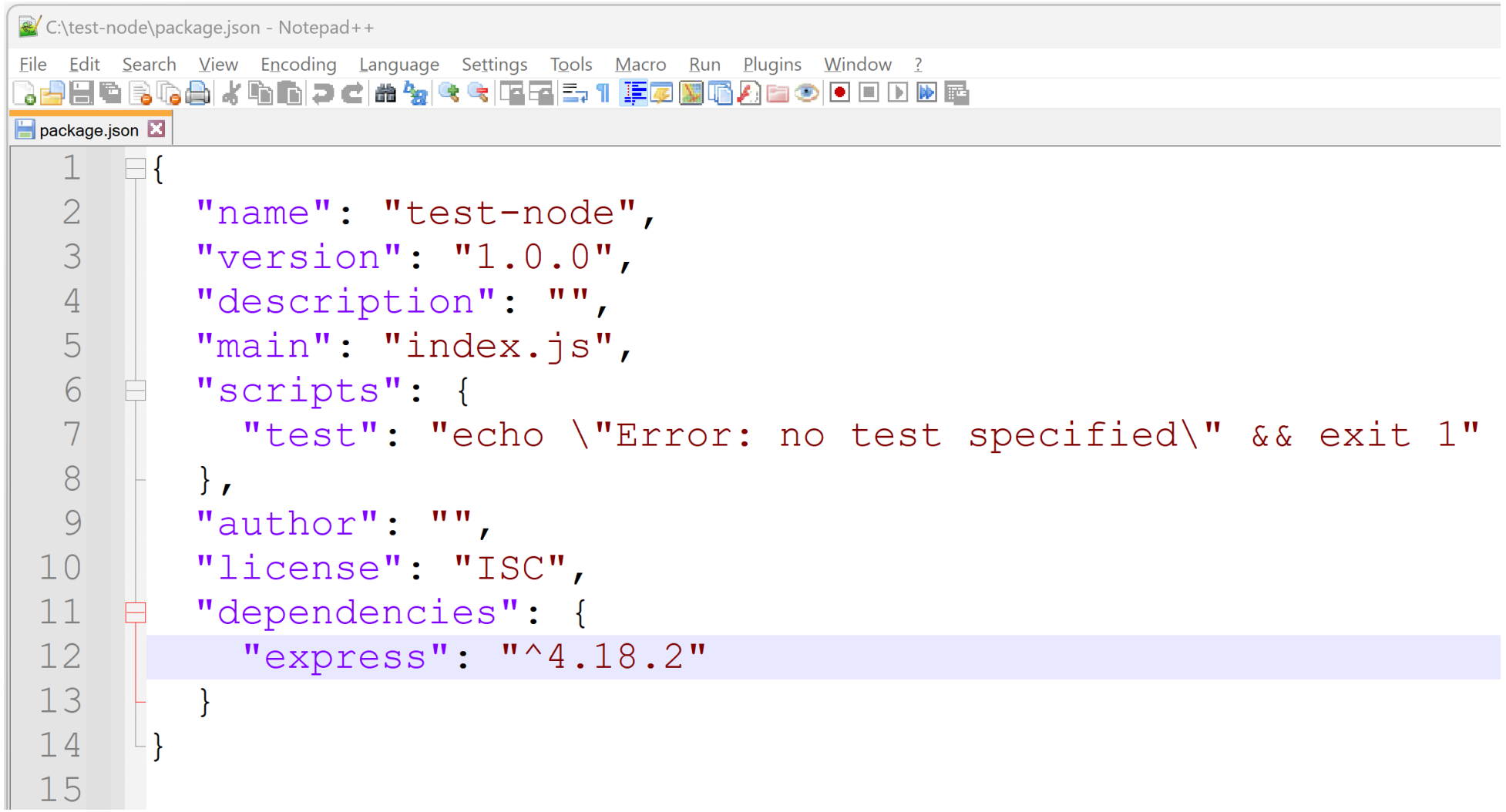


node_modules/express/lib/**/*.js



… 1 Line added in package.json

dependencies: { «pck-name» : « version », … }



https://docs.npmjs.com/about-semantic-versioning

Using semantic versioning to specify update types your package can accept

You can specify which update types your package can accept from dependencies in your package's `package.json` file.

For example, to specify acceptable version ranges up to 1.0.4, use the following syntax:

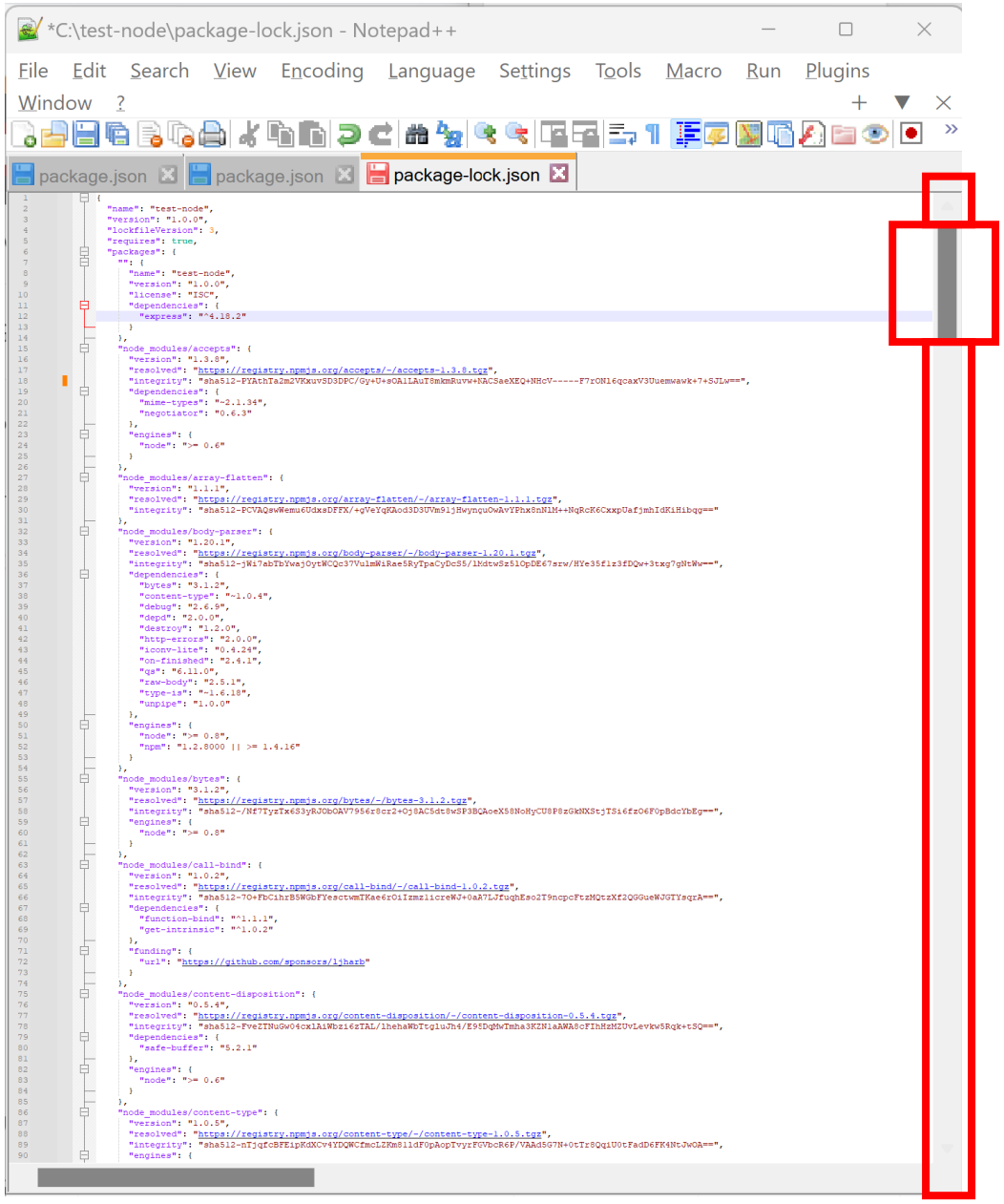
- Patch releases: `1.0` or `1.0.x` or `~1.0.4`
- Minor releases: `1` or `1.x` or `^1.0.4`
- Major releases: `*` or `x`

For more information on semantic versioning syntax, see the npm semver calculator.

Example

```
"dependencies": {
  "my_dep": "^1.0.0",
  "another_dep": "~2.2.0"
},
```

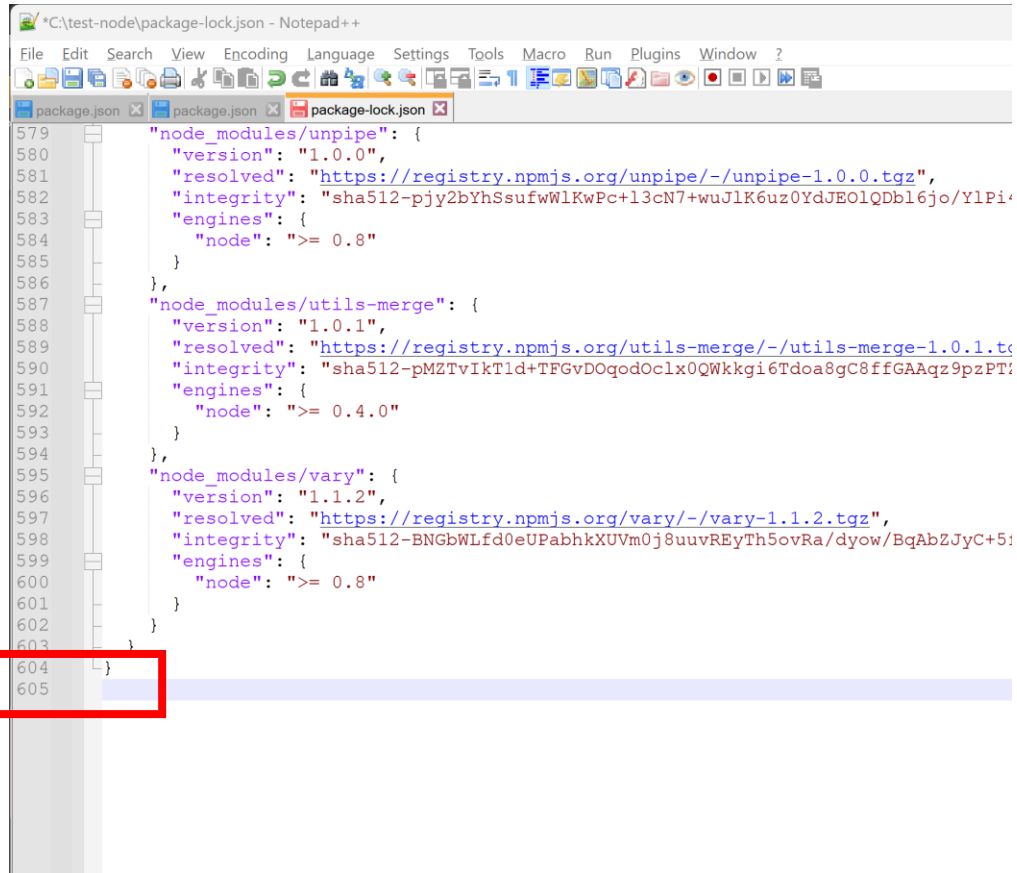
Locking versions number … package-lock.json



File is **HUGE** …

Like expanded package/package/package !!!

ONLY « express » ➔ 605 lines



node_modules/*/*/*/*.js

Directory node_modules

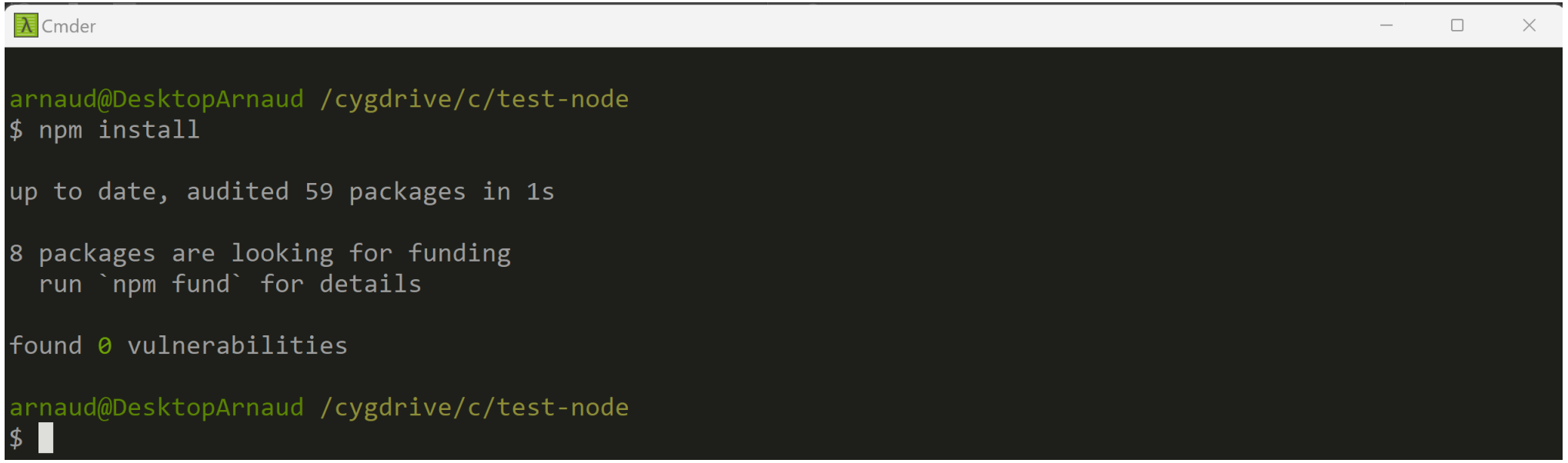
1/ may be BIG
…. ~300 Mega is frequent with few dependencies
$ du -sh node_modules

2/ can be deleted + re-downloaded safely
$ rm –rf node_modules; npm install

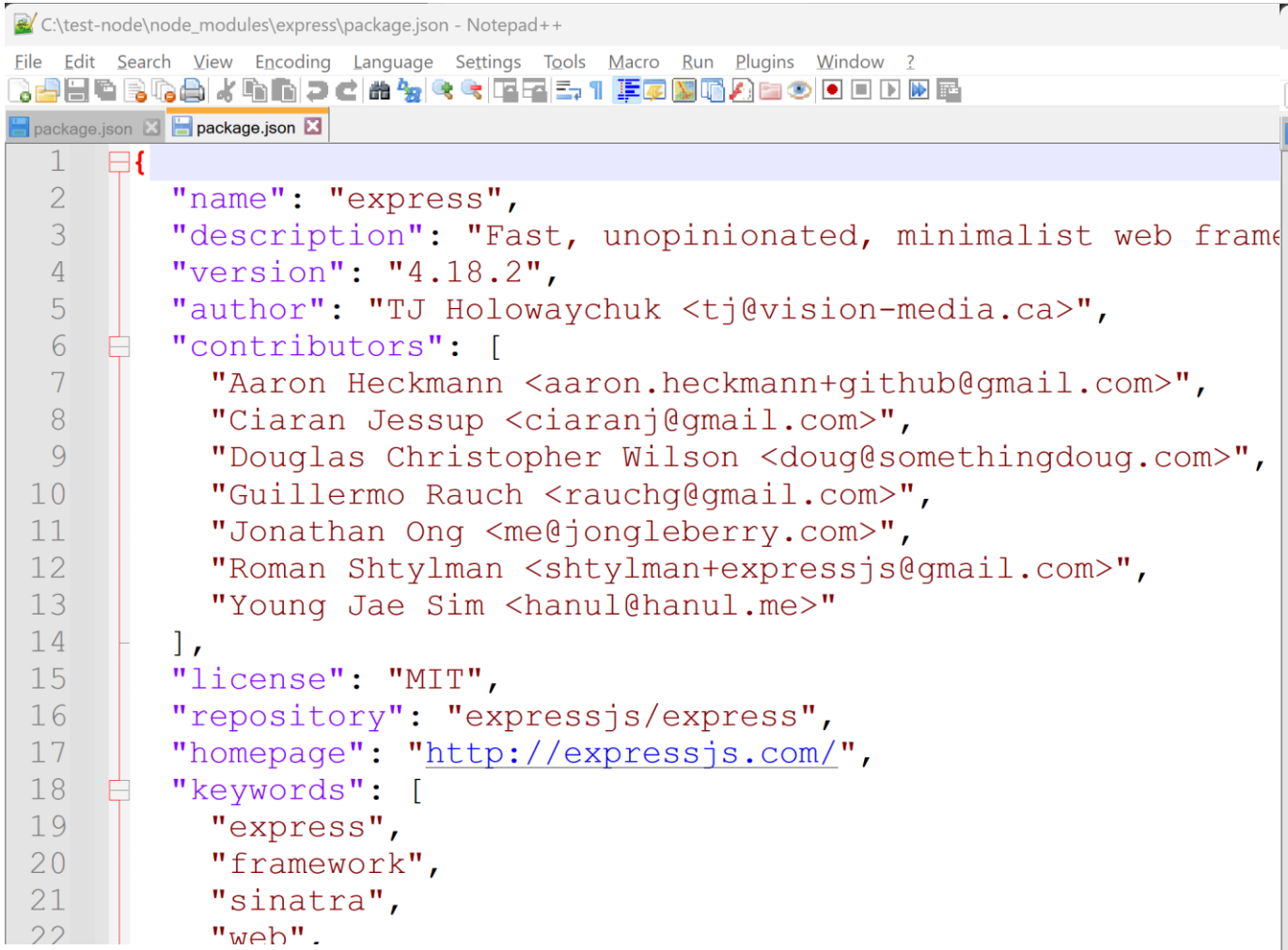
3/ should NEVER be committed to any git repository
$ echo node_modules/ >> .gitignore

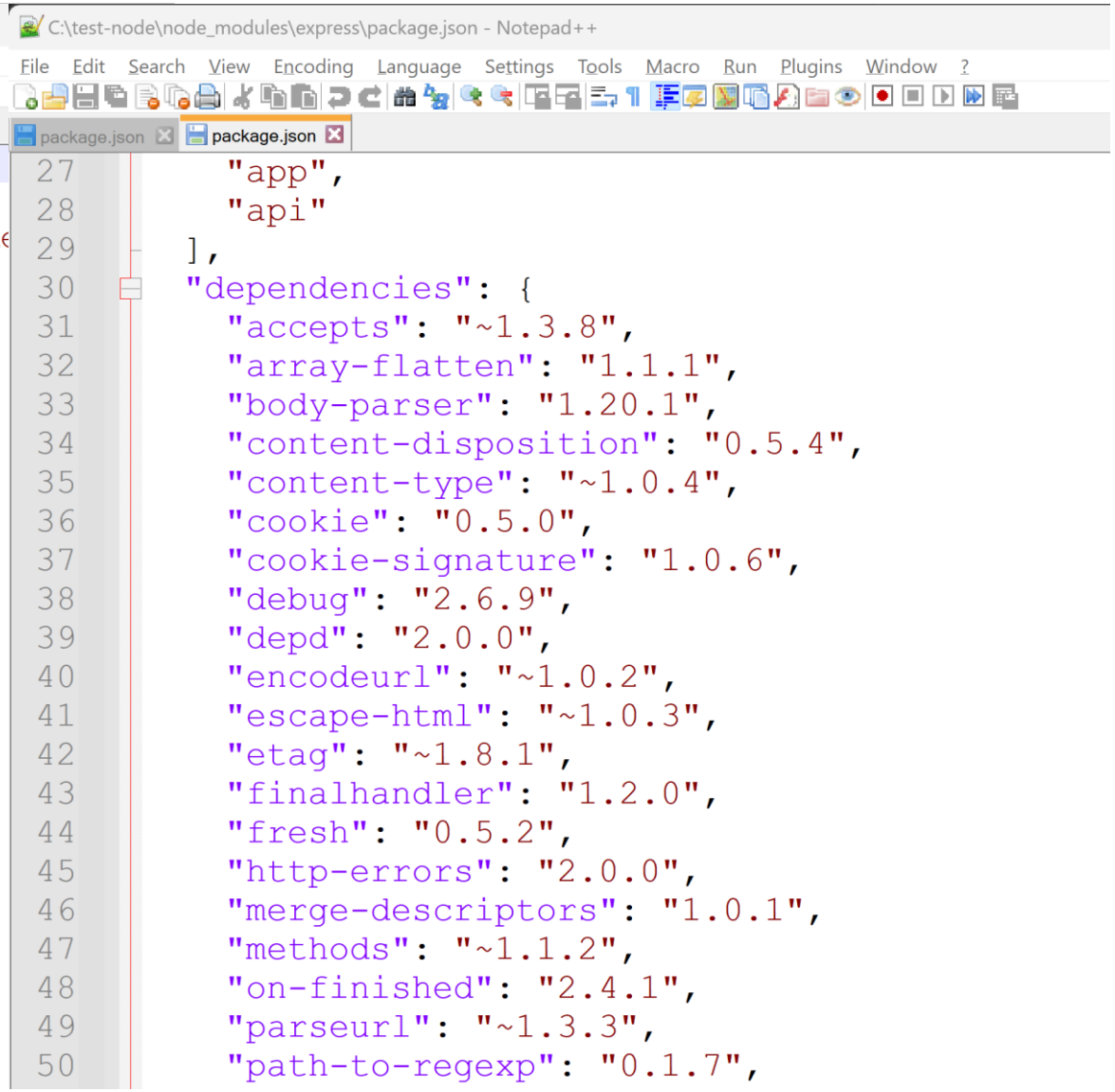
After getting some web project source code…
=> downloads to re-fill node_modules/**
as defined in package.json

$ npm install

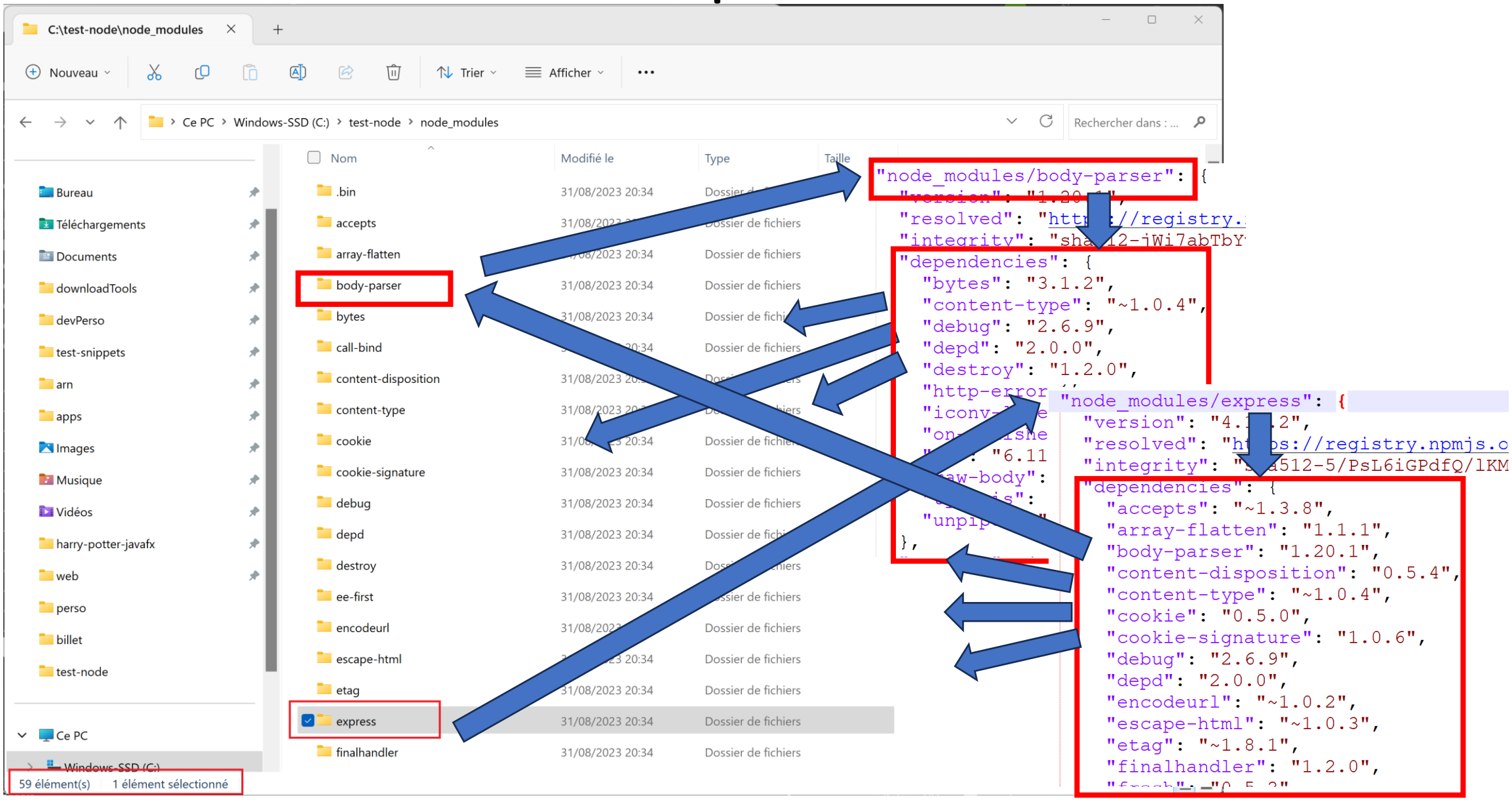


node_modules/express/package.json

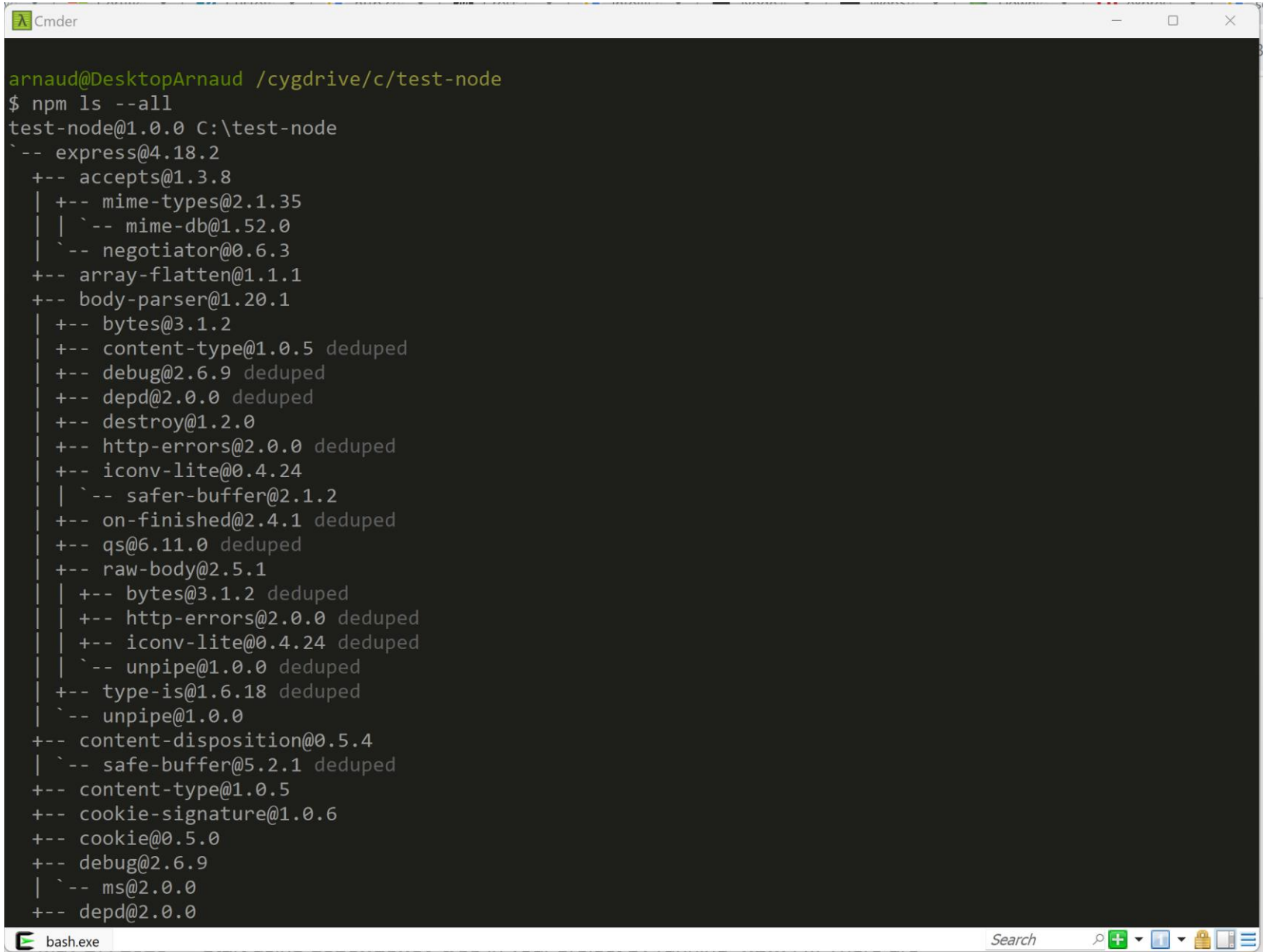




package.json -> package.json -> … package.json
transitive dependencies



npm ls --all


Cmder
arnaud@DesktopArnaud /cygdrive/c/test-node
$ npm ls --all
test-node@1.0.0 C:\test-node
`-- express@4.18.2
+-- accepts@1.3.8
| +-- mime-types@2.1.35
| | `-- mime-db@1.52.0
| `-- negotiator@0.6.3
+-- array-flatten@1.1.1
+-- body-parser@1.20.1
| +-- bytes@3.1.2
| +-- content-type@1.0.5 deduped
| +-- debug@2.6.9 deduped
| +-- depd@2.0.0 deduped
| +-- destroy@1.2.0
| +-- http-errors@2.0.0 deduped
| +-- iconv-lite@0.4.24
| | `-- safer-buffer@2.1.2
| +-- on-finished@2.4.1 deduped
| +-- qs@6.11.0 deduped
| +-- raw-body@2.5.1
| | +-- bytes@3.1.2 deduped
| | +-- http-errors@2.0.0 deduped
| | +-- iconv-lite@0.4.24 deduped
| | `-- unpipe@1.0.0 deduped
| +-- type-is@1.6.18 deduped
| `-- unpipe@1.0.0
+-- content-disposition@0.5.4
| `-- safe-buffer@5.2.1 deduped
+-- content-type@1.0.5
+-- cookie-signature@1.0.6
+-- cookie@0.5.0
+-- debug@2.6.9
| `-- ms@2.0.0
+-- depd@2.0.0
bash.exe
Search

Coffee break (?)

Choose a Text Editor

IntelliJ Ultimate

Notepad++

WebStorm

SublimeText

VisualStudio Code

Vi

Notepad

Wordpad

Eclipse

NetBeans

Choose ~~(NOT a Text Editor)~~ a modern IDE

IntelliJ Ultimate ⟸ **€€€$$$ (or 30 days)**

WebStorm ⟸ **€€$$ (or 30 days)**

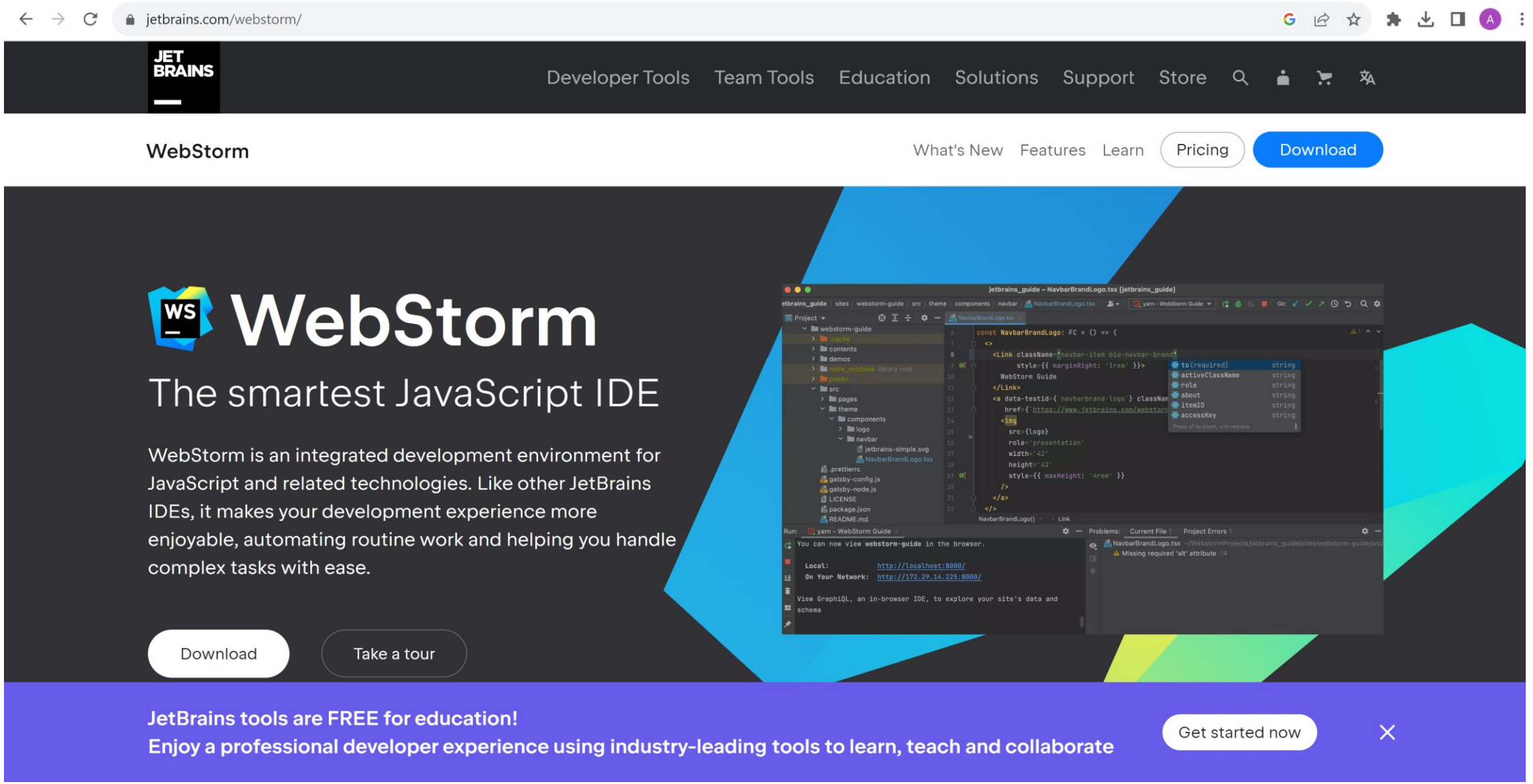
VisualStudio Code

Eclipse + plugins

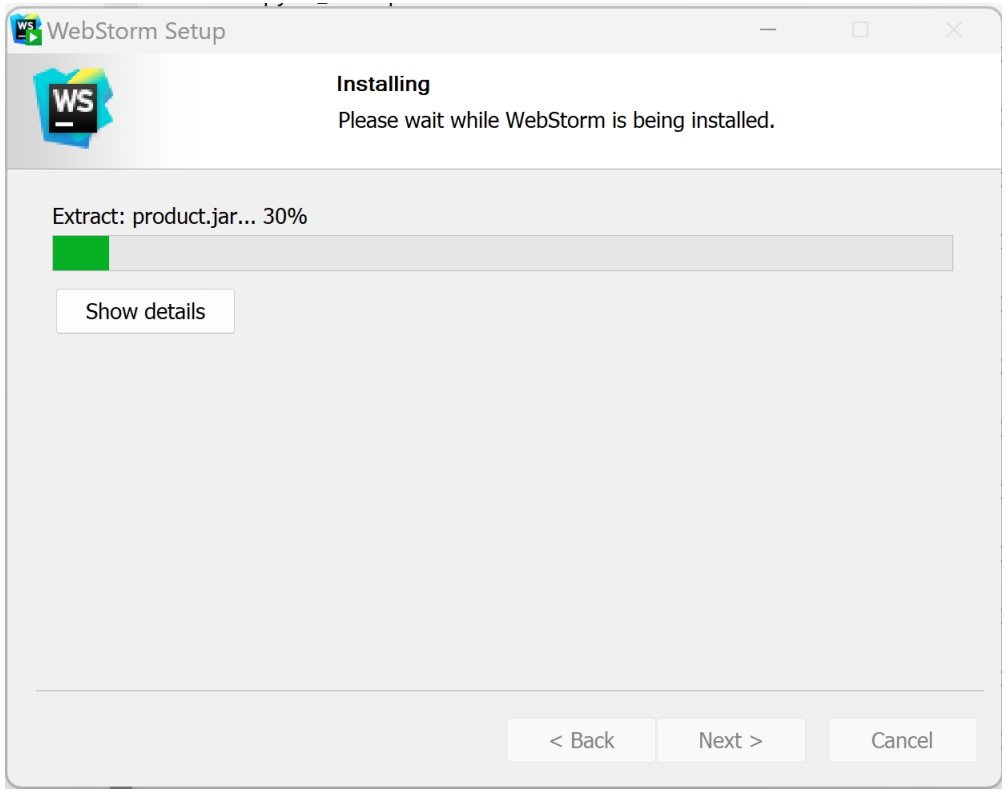
Eclipse(plugins not pre-installed)

IntelliJ (No web js Support)

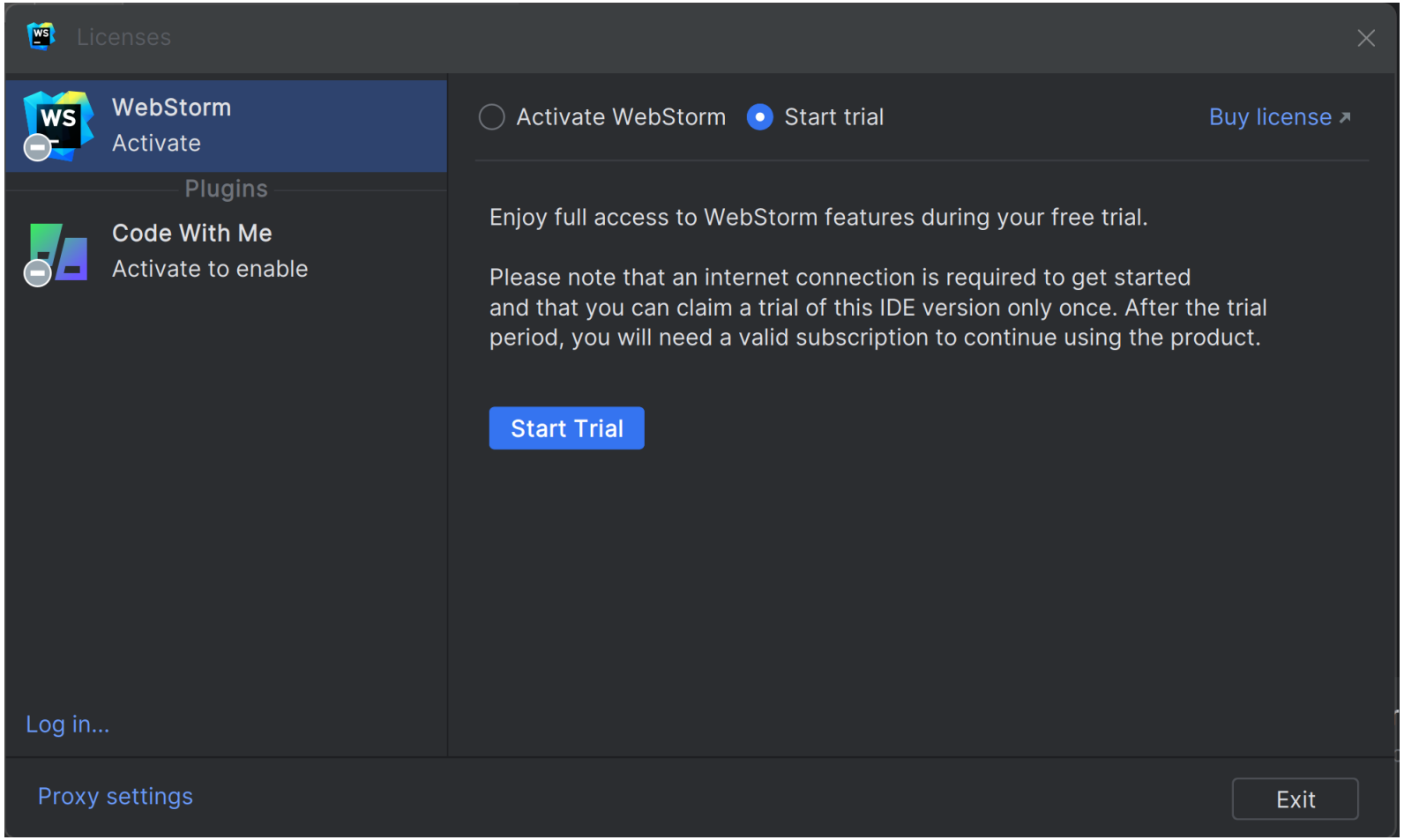
https://www.jetbrains.com/webstorm/



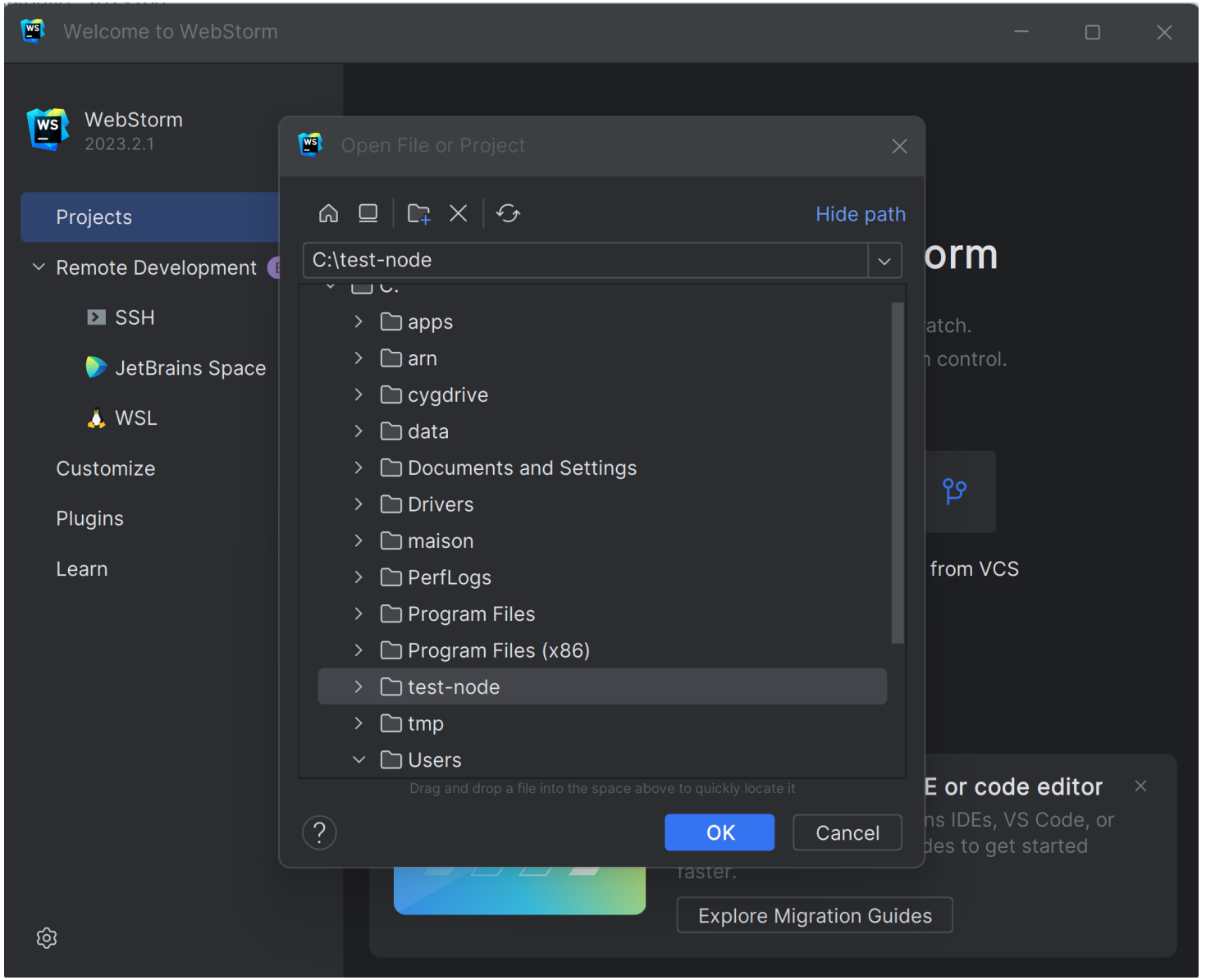
Download 30 days free evaluation + Install …



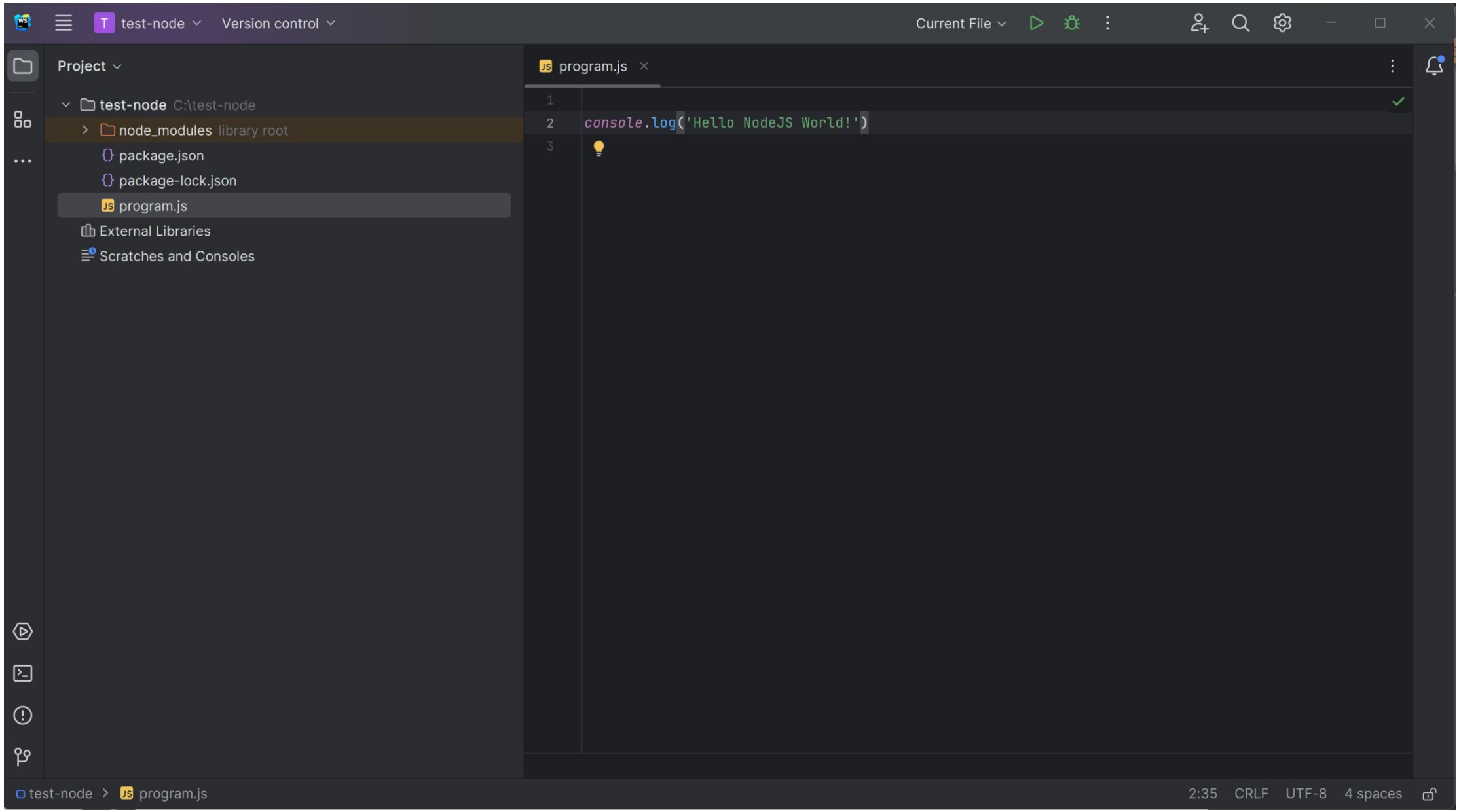
Notice Licenses:
IntelliJ Ultimate = WebStorm + IntelliJ + ….
IntelliJ Community = .. NO web !



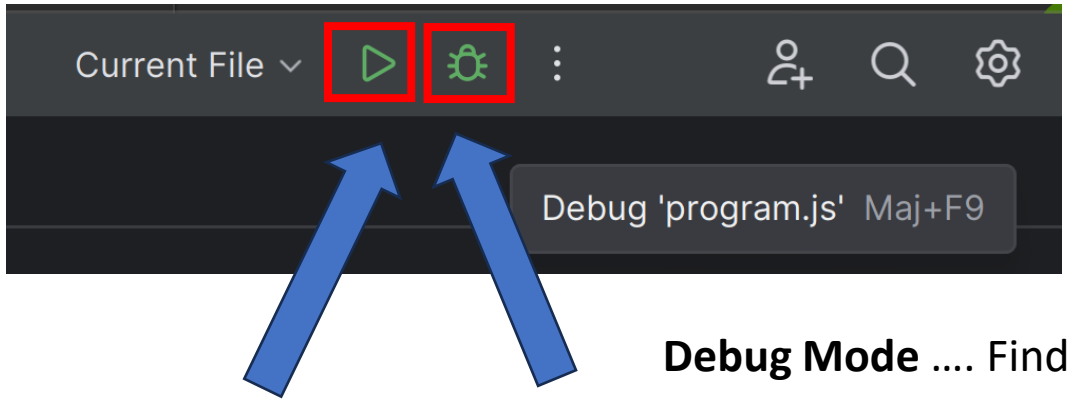
Open Project



Edit program.js File



Click on « Run » or « Debug » ?



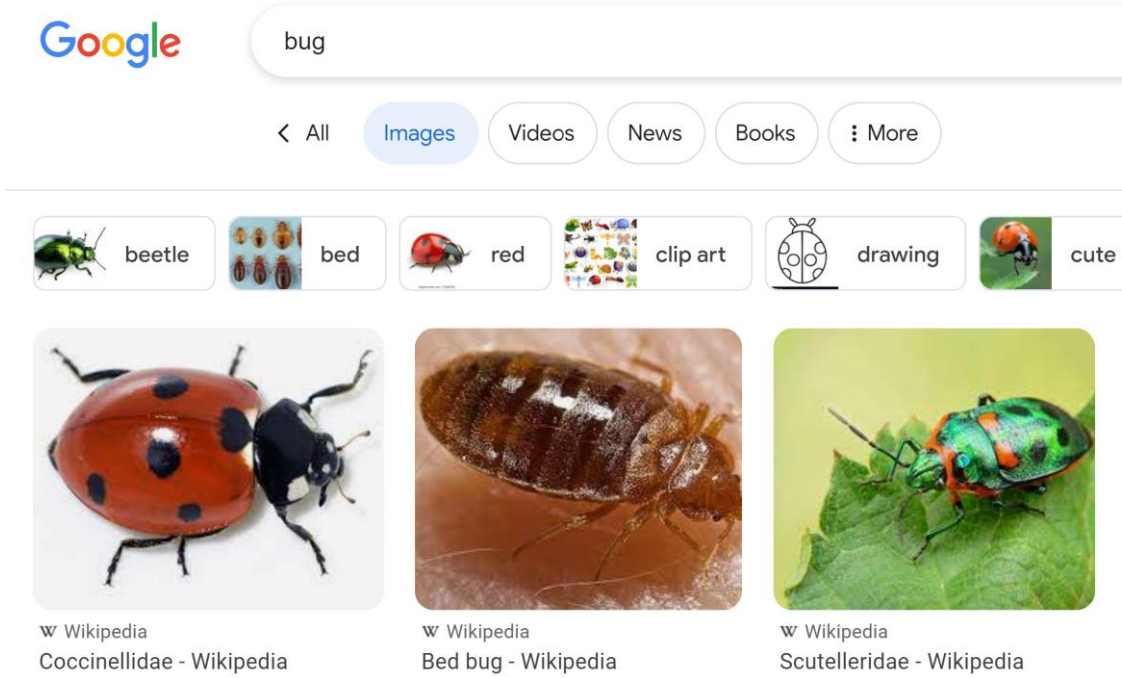
Run Mode ….
Start and forget
Can NOT activate any breakpoint
… can only see logs, and Kill

Why do you use an IDE Debugger

Use directly « C:> node » ??

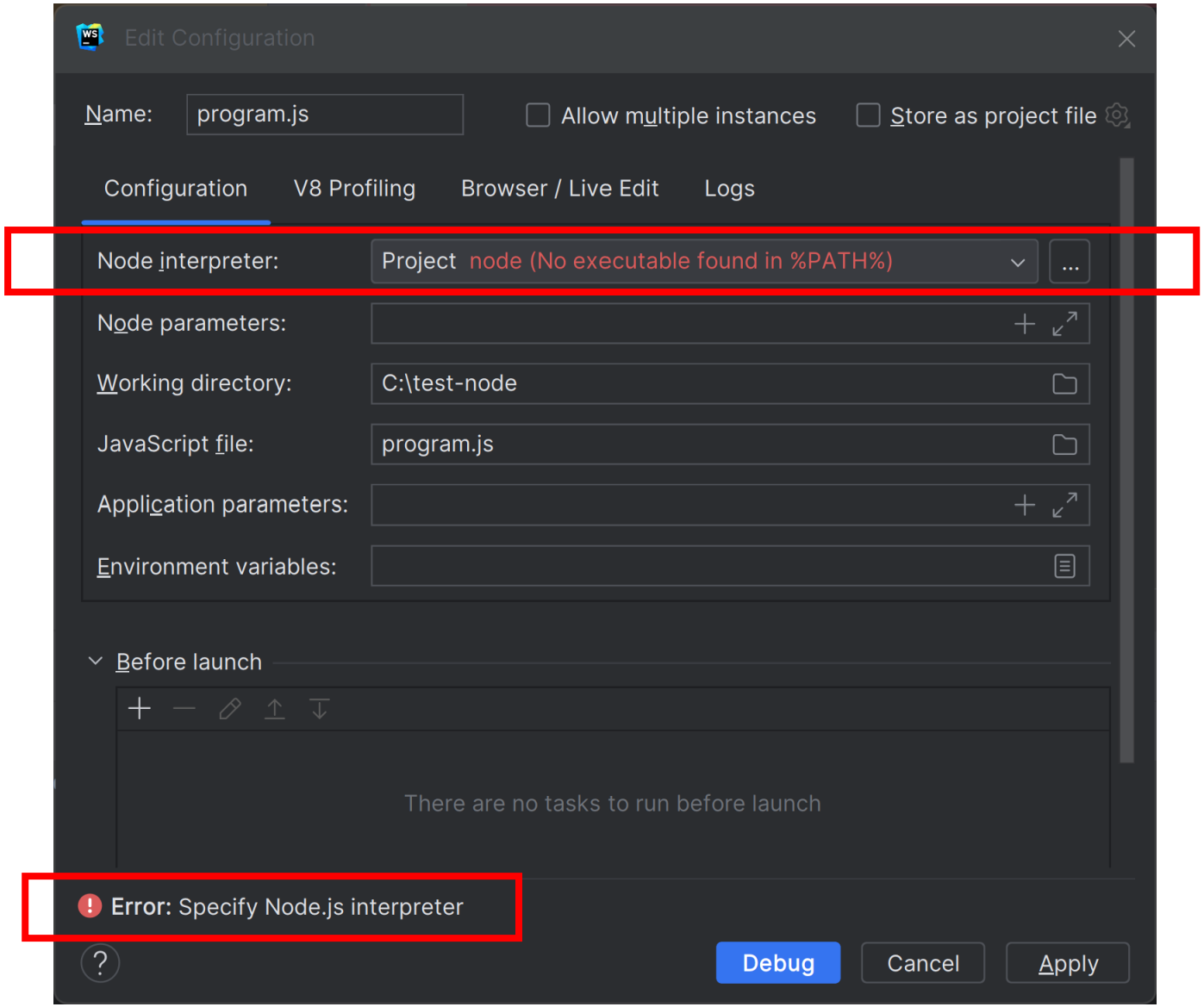
Debug Mode …. Find the « bug »

Can put breakpoints !



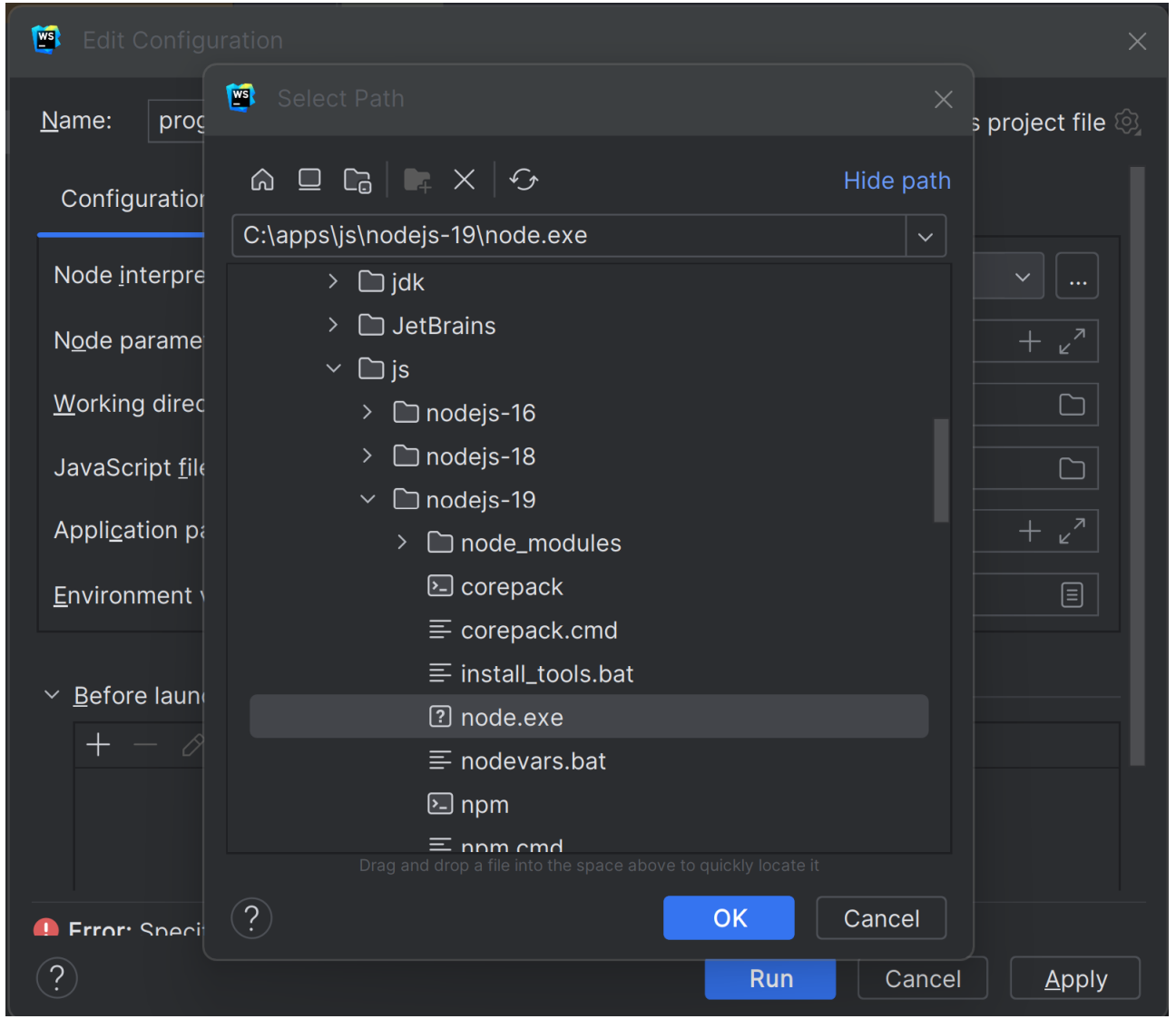
System Diagnostic

… Case undetected « node » not in %PATH%



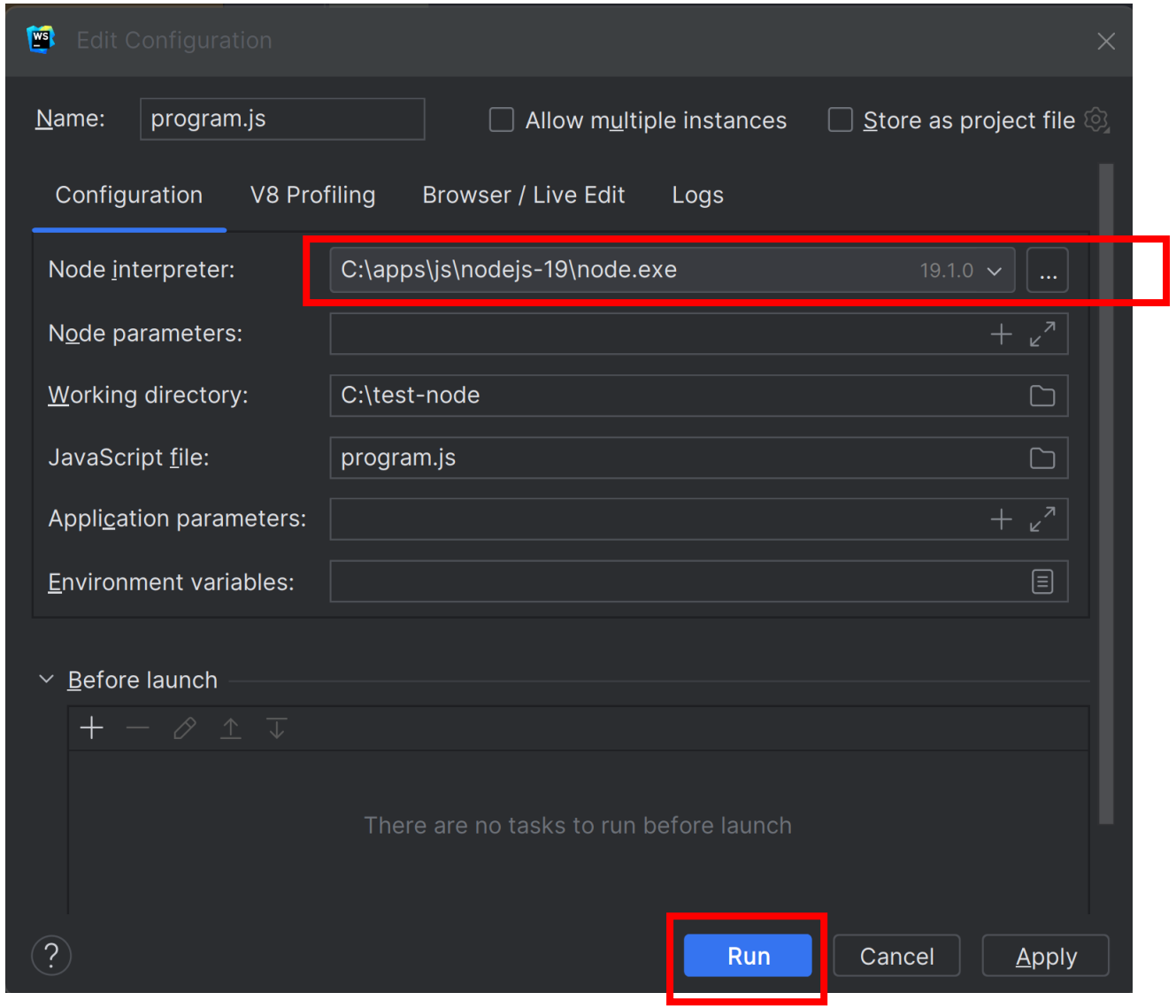
System Diagnostic

Configure … Add node

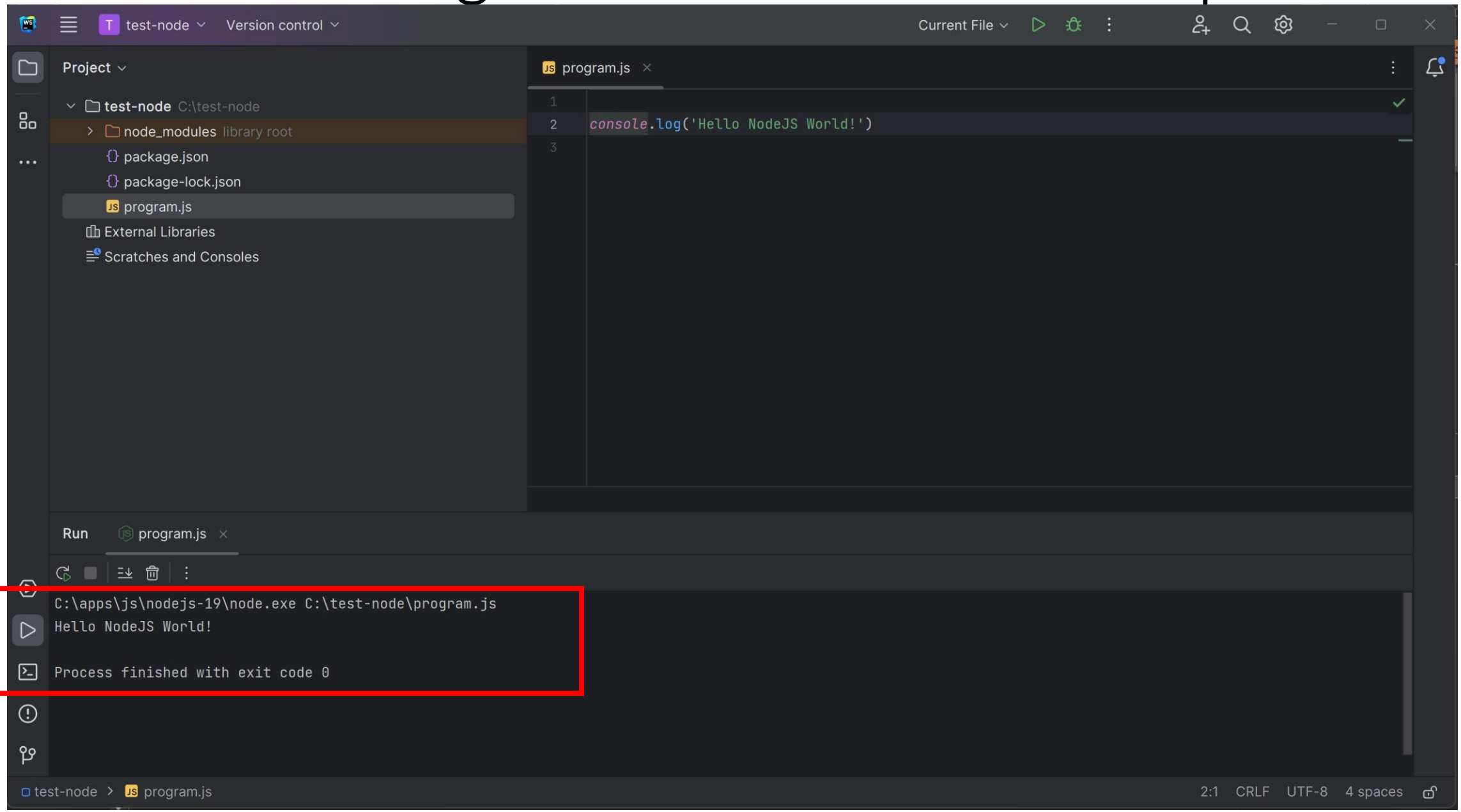


System Diagnostic

Edit Run/Debug configuration…



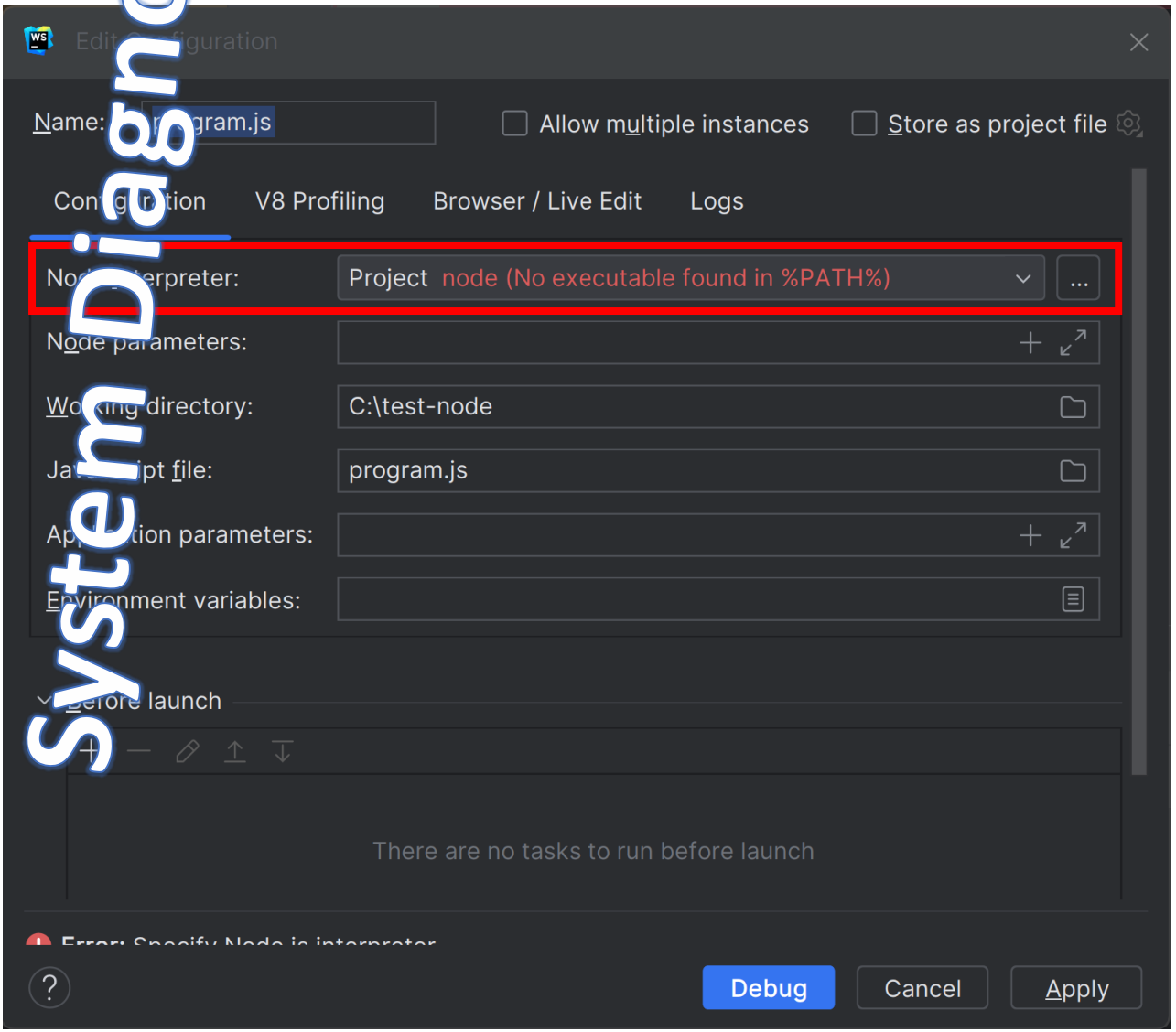
« Run » … Program Finished + console print



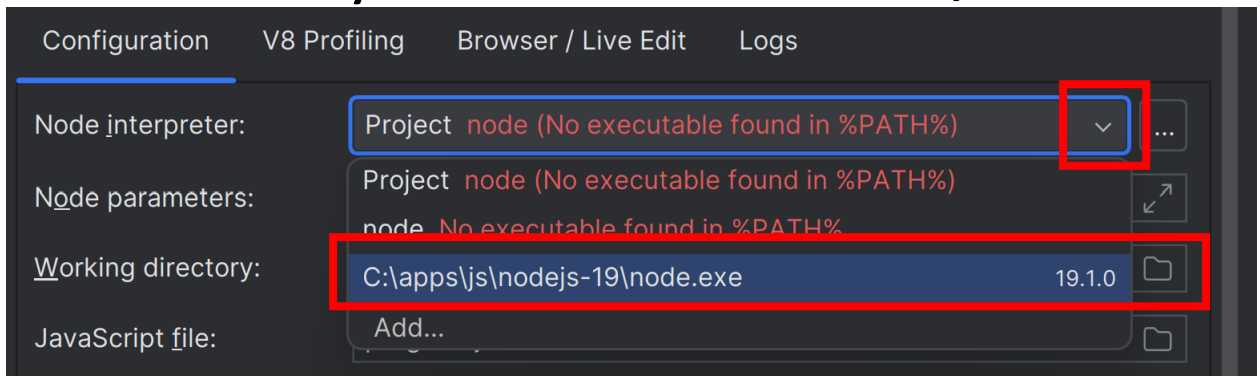
System Diagnostic

Redo « Run » or « Debug »

Need to re-edit node in %PATH% ???



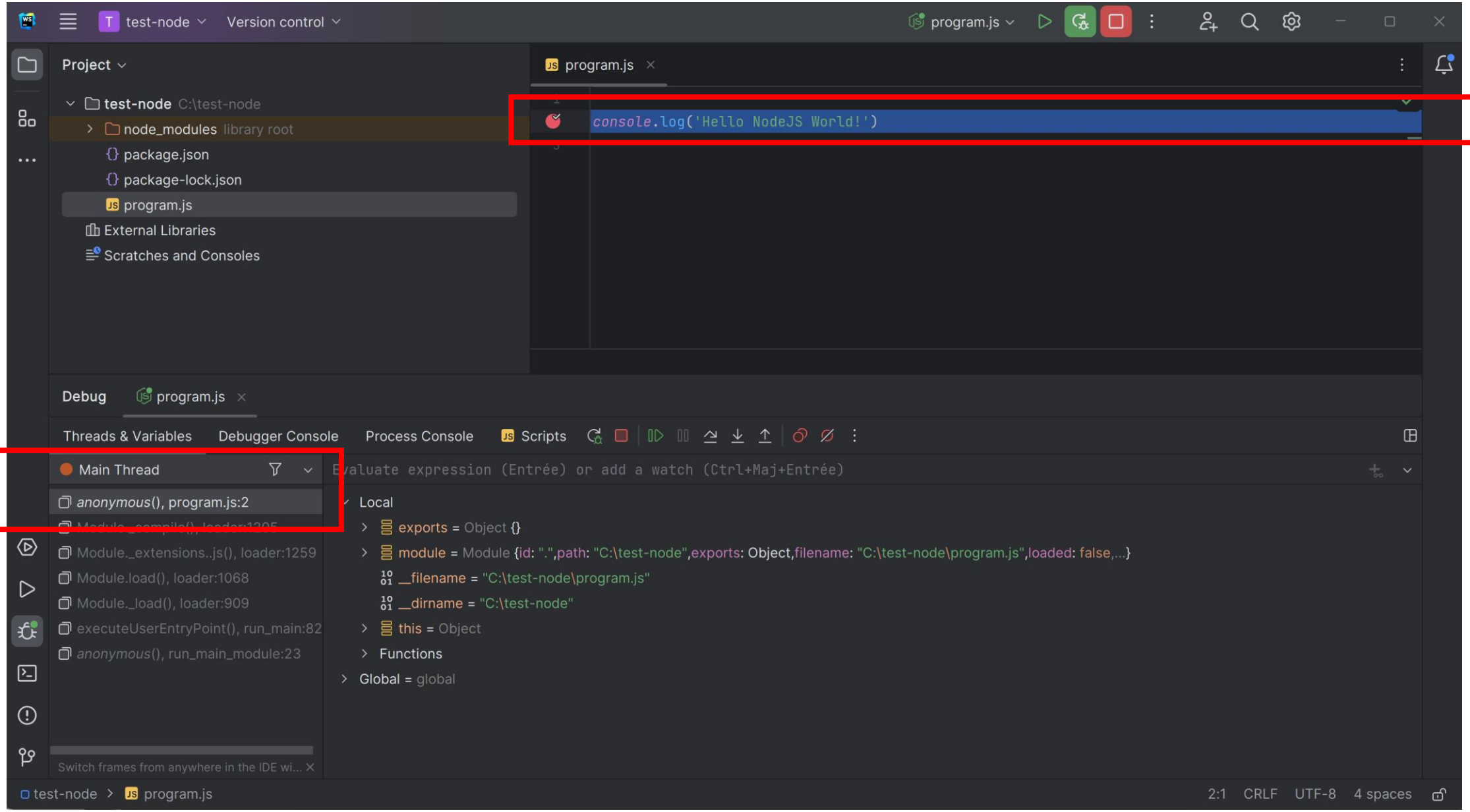
At least, can re-select
already added choice « C:\... » !



System Diagnostic

Change configuration « your-config » instead of « Current File »

Debug « Mode » + Breakpoint => Program Paused

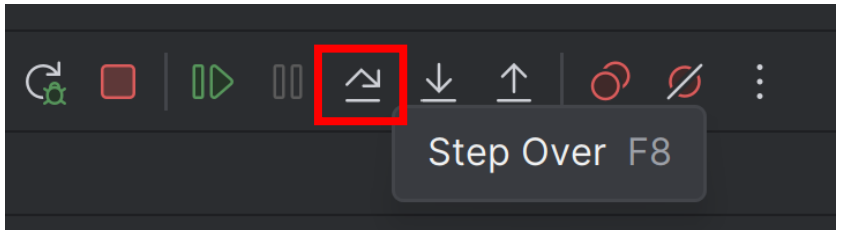


Step Into(F7)

< Step Over(F8)

< Step Out (Maj F8)

< Resume (F9)



Coffee Break

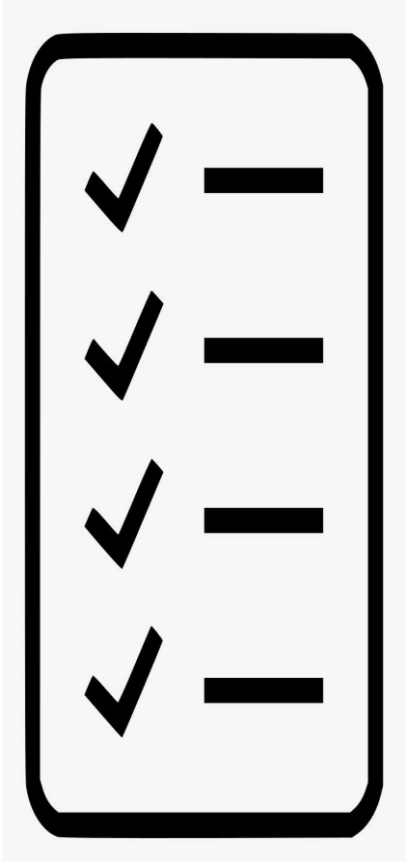
Complains on JavaScript ?

for Christmas Wish List, ask

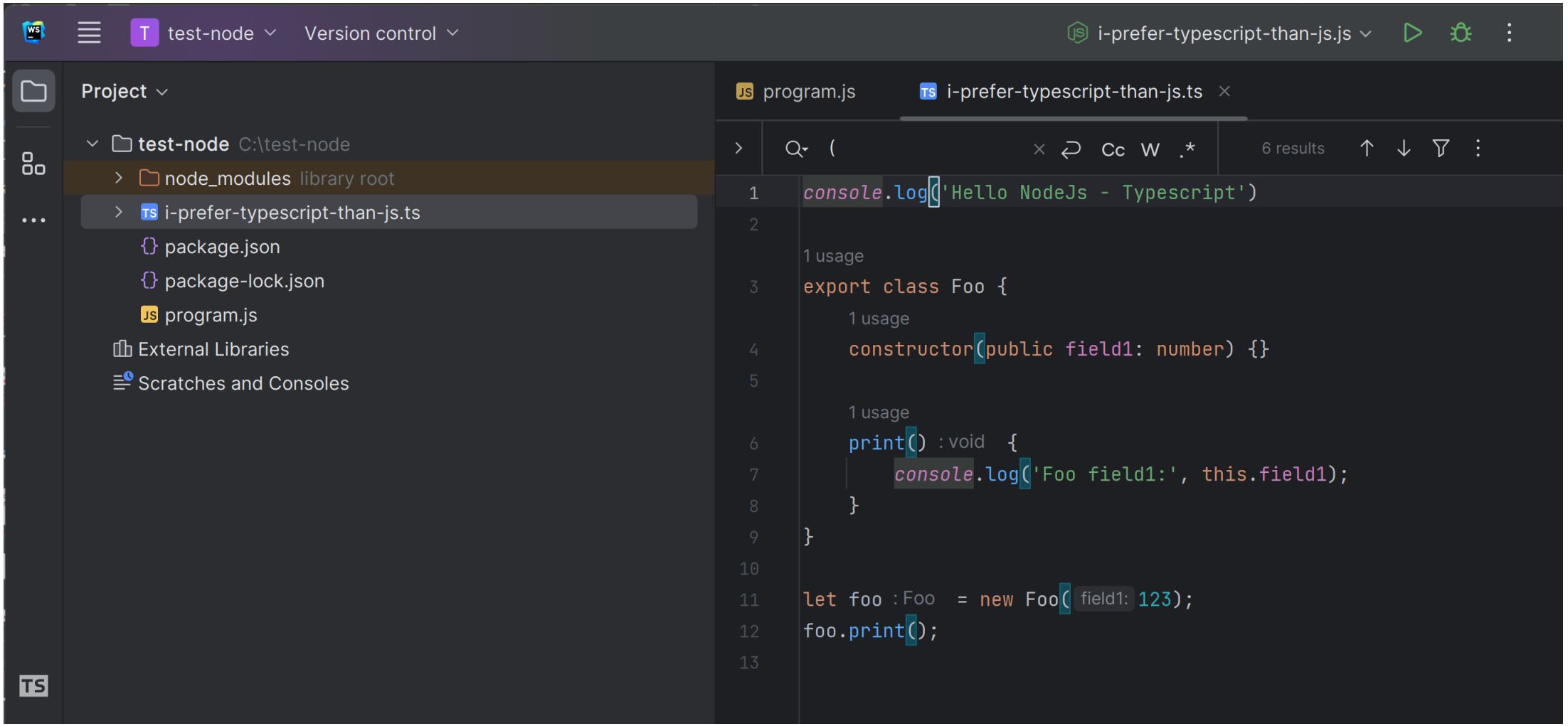
- Types declarations
  (standard types, structural types, union types)
- **Compiled-time Type checking**
- With Implicit types inference
- **Object-Oriented Classes**
- Templates
- Spread operators
- Syntacic sugars

..

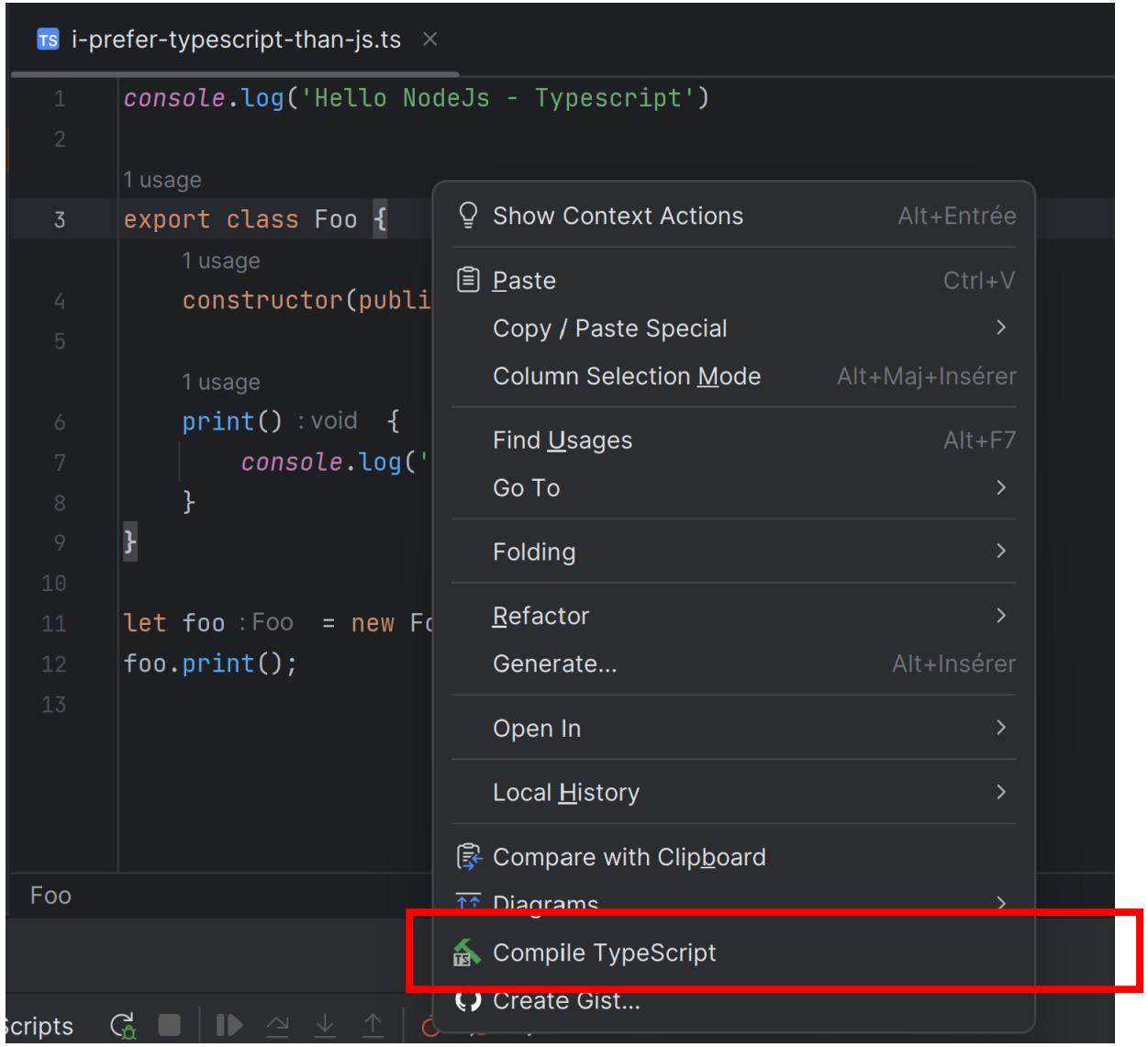
TypeScript



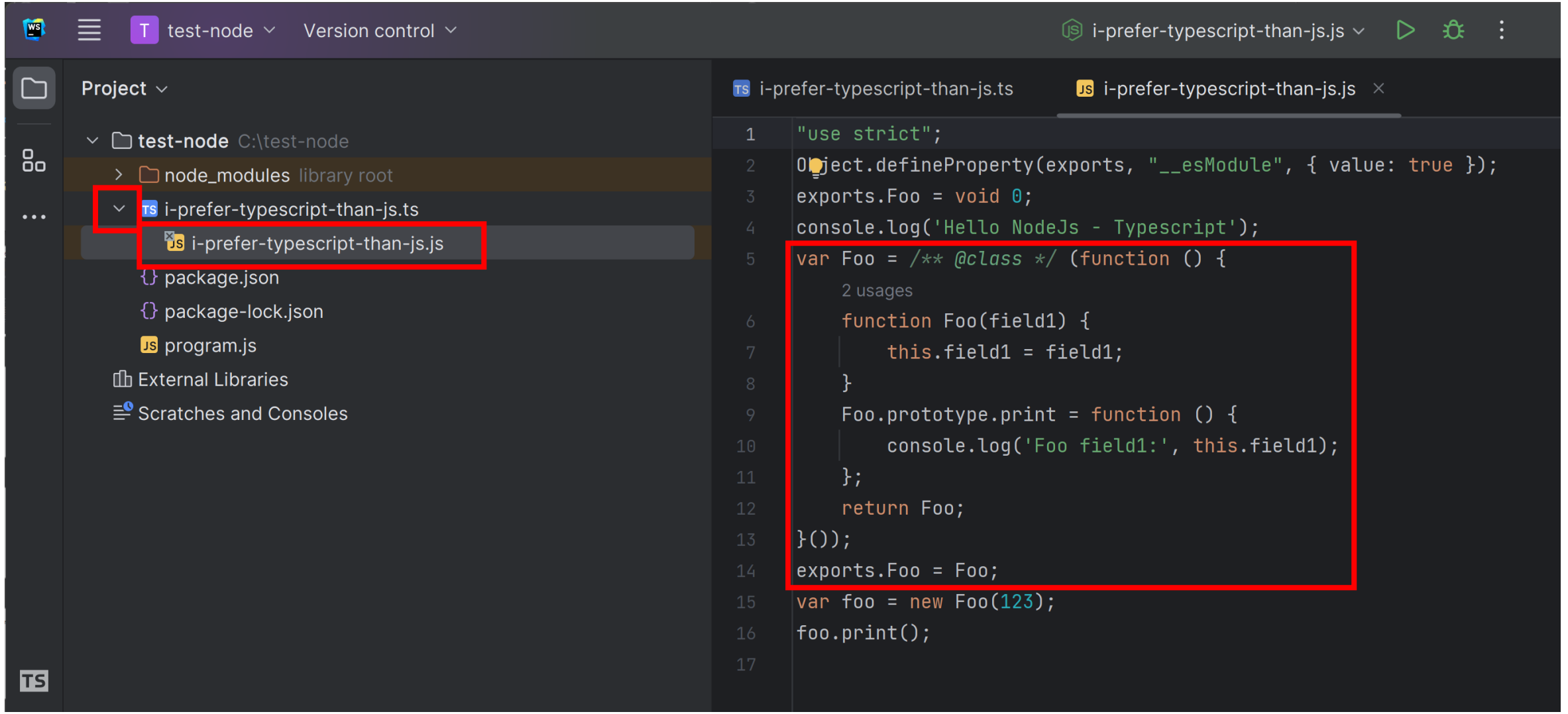
Edit a TypeScript *.ts File



Right Click « Compile TypeScript »

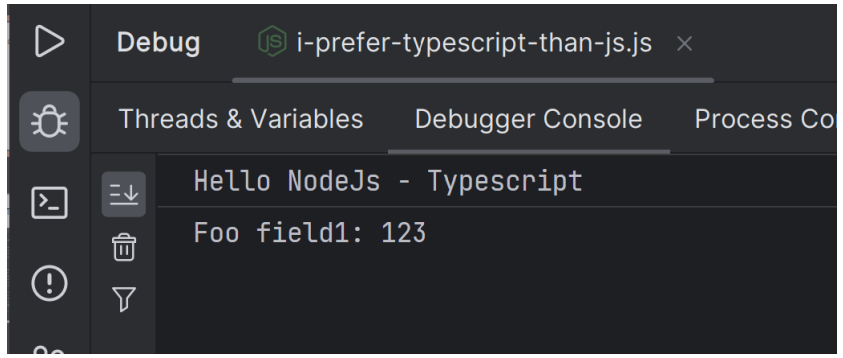


Open Item « .ts » > Sub Item « .js » see generated Js !!

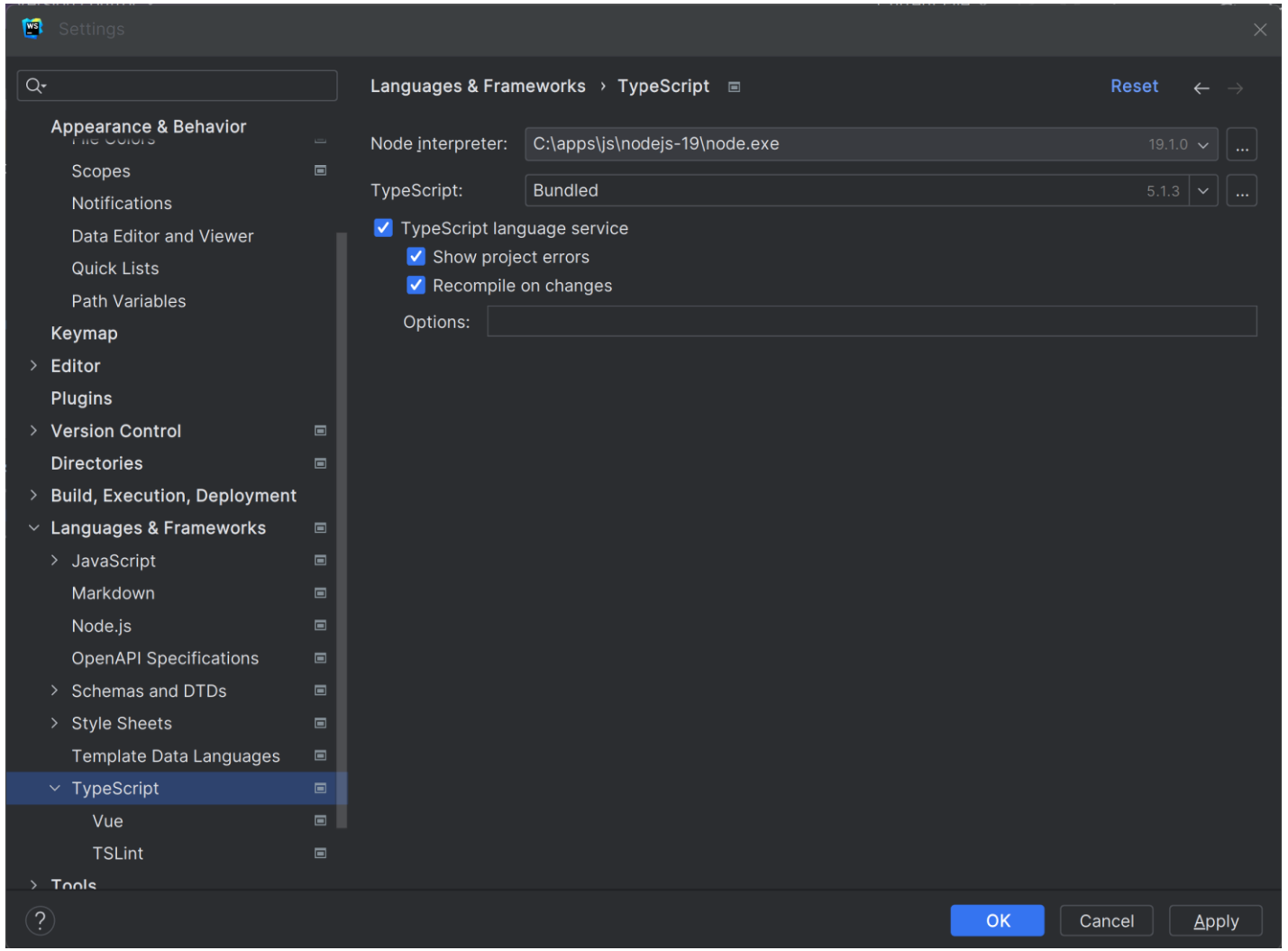


Debug Ts program
… (No Run/Debug/breakpoint in *.ts, but ok in *.js)

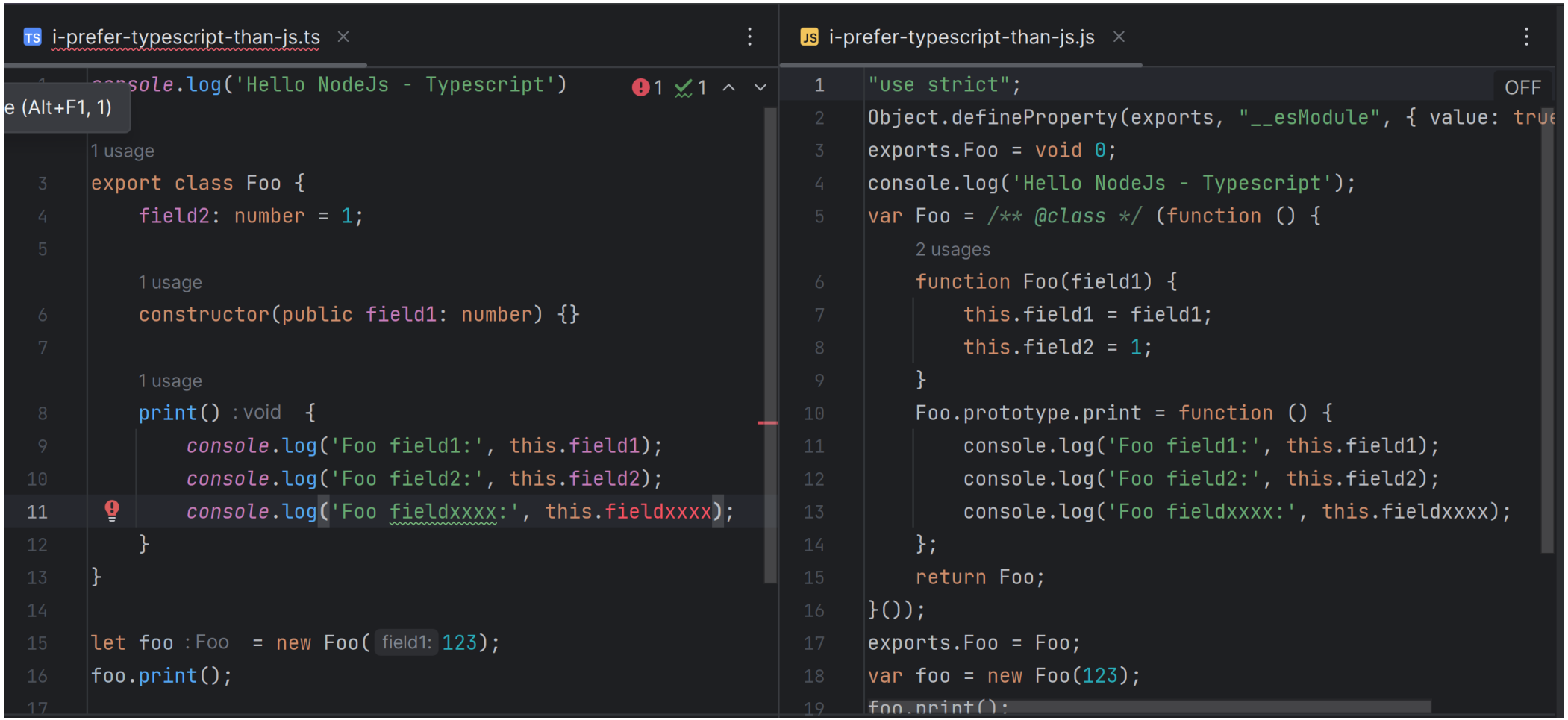
need « .map » file for mapping ts line X <-> js line Y



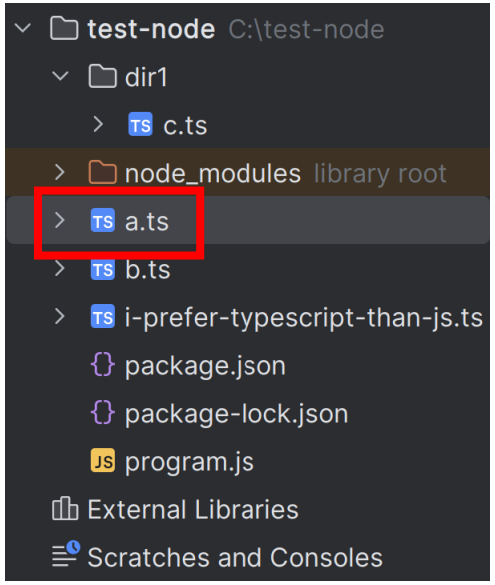
Edit Settings > TypeScript > Recompile on changes



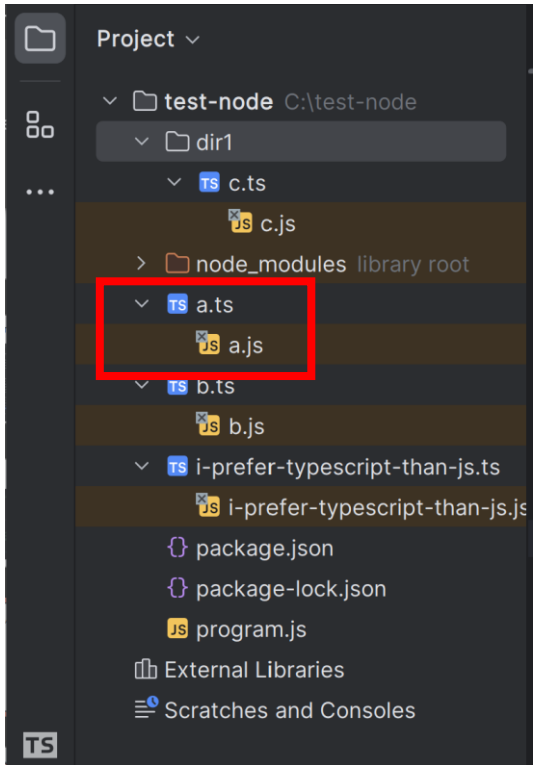
Edit on Left *.ts … see real time change in right *.js



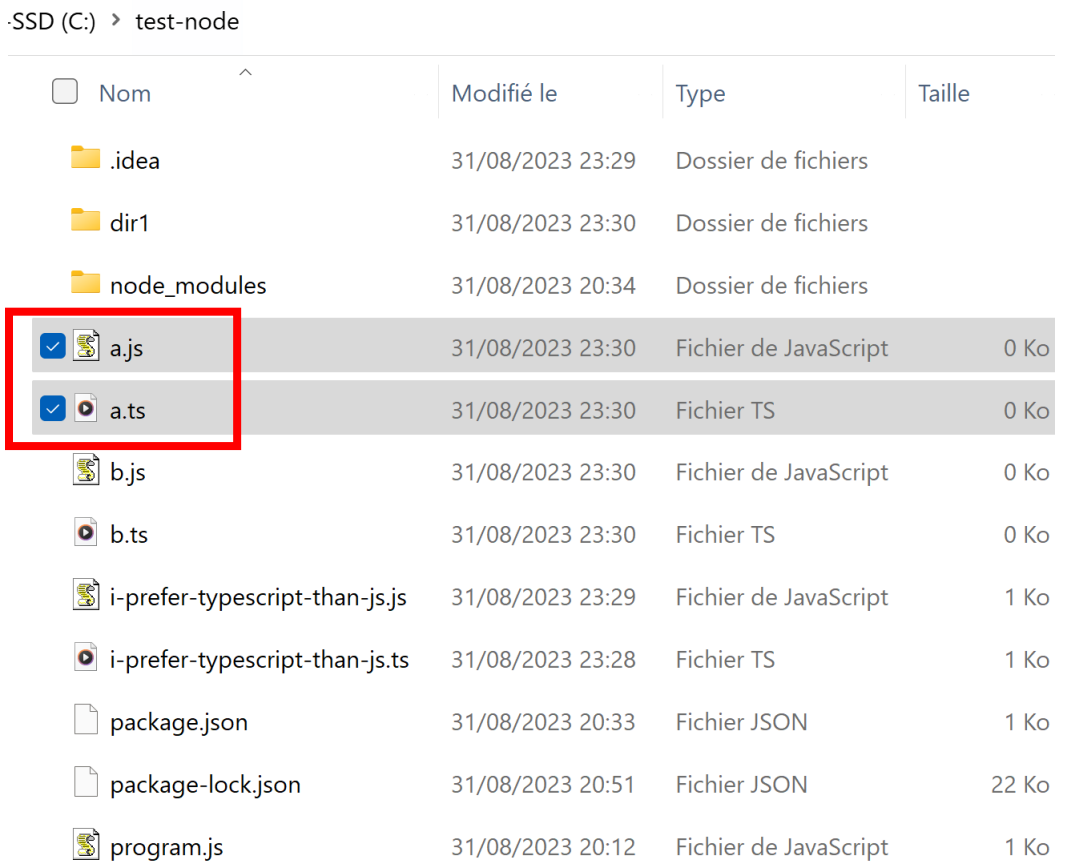
Edit many **/*.ts … see corresponding **/*.js



Expand
views



Check
on FileSystem



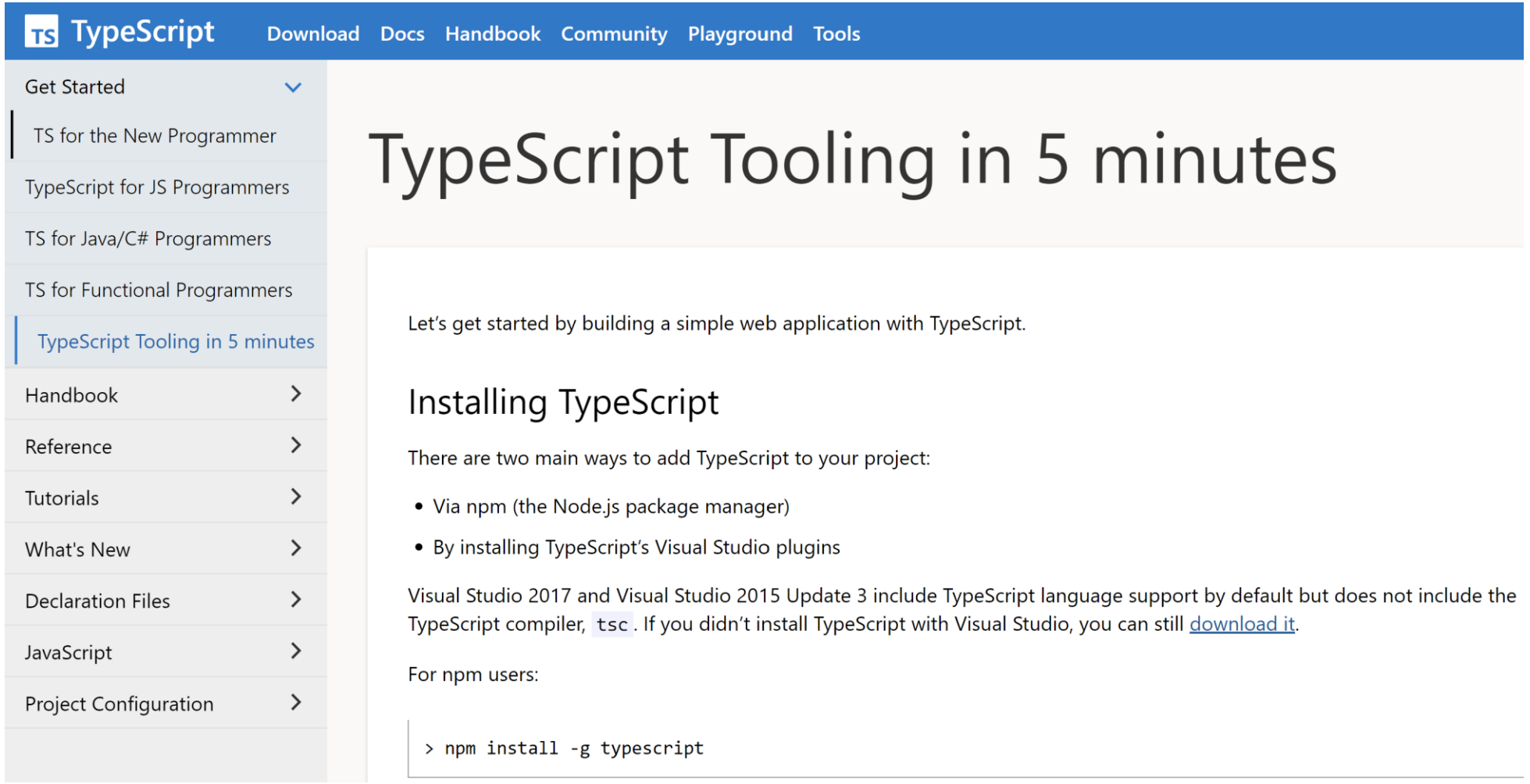
tsc = TypeScript Compiler

… same as in IDE, but in batch mode, for CI-CD

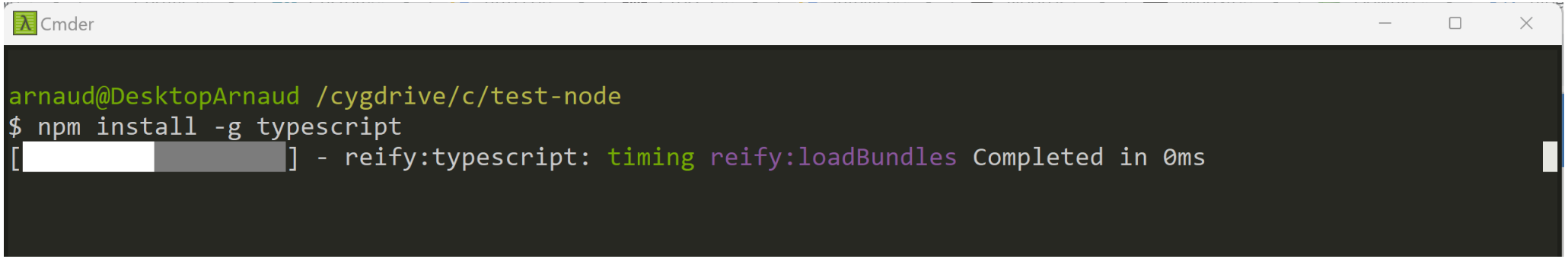
tsc is internally used by WebStorm
… / IntelliJ / VisualStudio Code / Eclipse !!

tsc even generalised it as « langage server »
=> You have exact same compilation in all IDEs !!

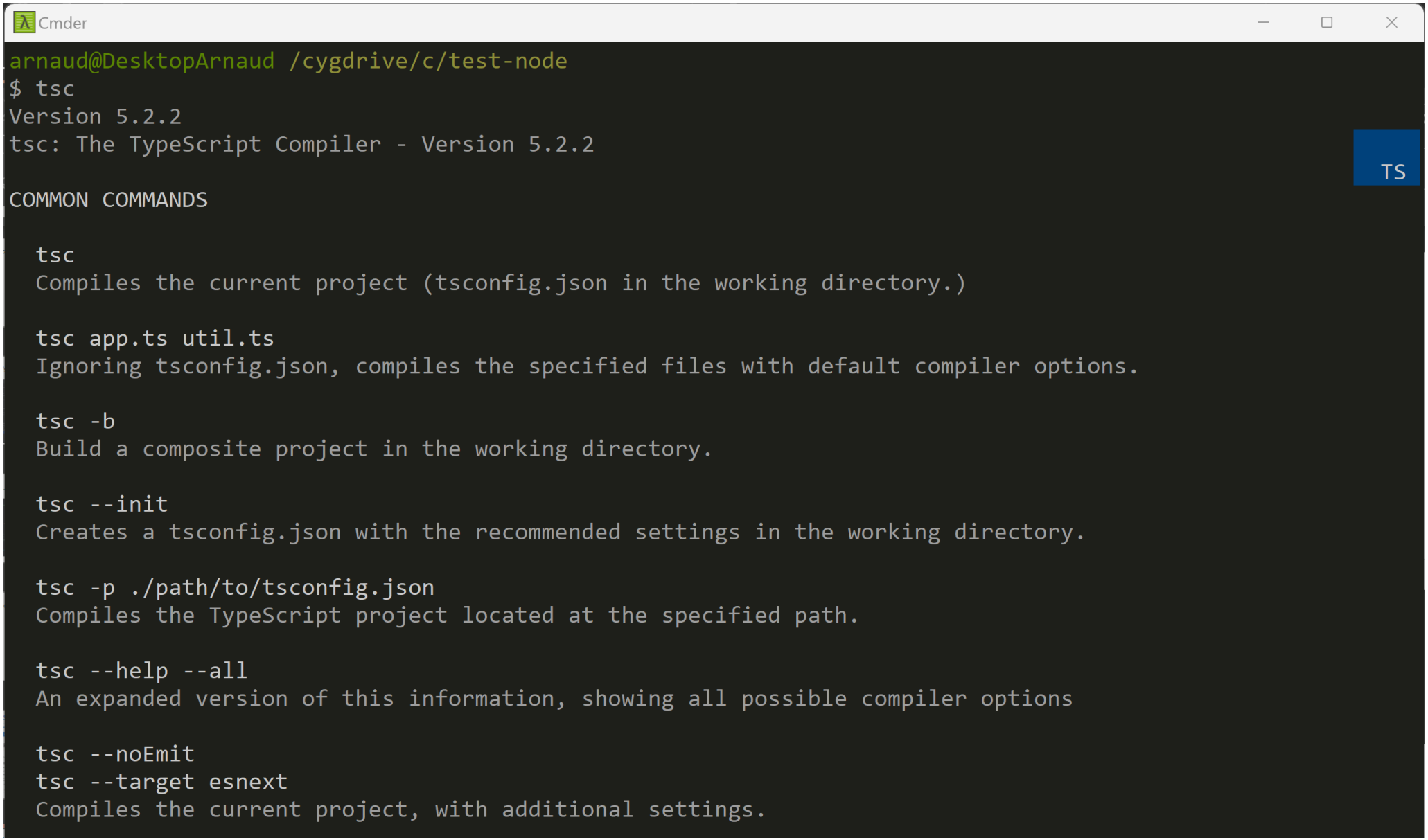
https://www.typescriptlang.org/docs/handbook/typescript-tooling-in-5-minutes.html



$ npm install –g typescript
(« -g » for « --global » … add tsc in node %PATH%)



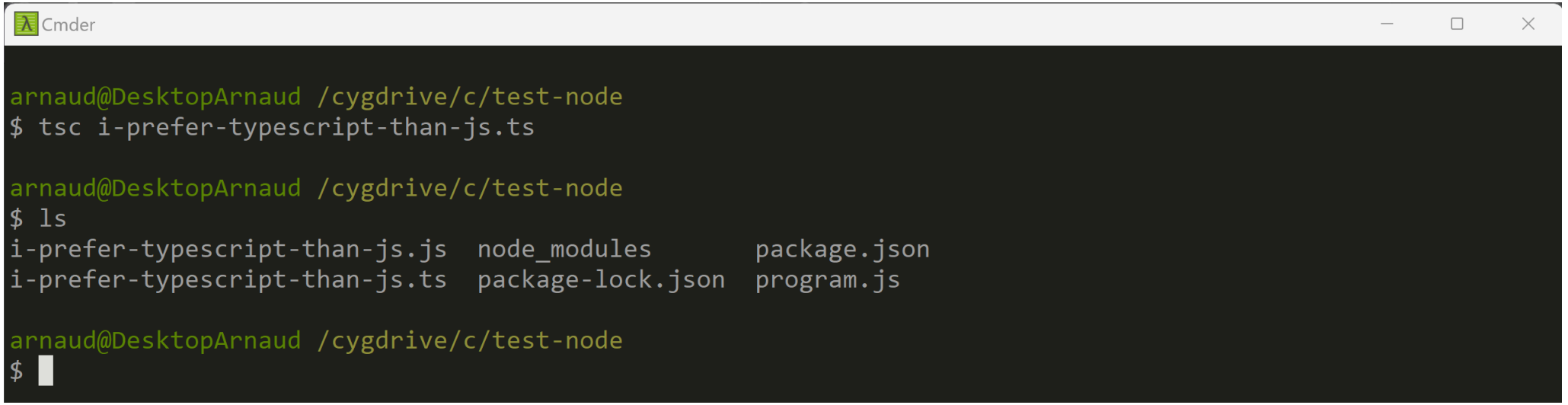
$ tsc


Cmder
arnaud@DesktopArnaud /cygdrive/c/test-node
$ tsc
Version 5.2.2
tsc: The TypeScript Compiler - Version 5.2.2
TS
COMMON COMMANDS
tsc
Compiles the current project (tsconfig.json in the working directory.)
tsc app.ts util.ts
Ignoring tsconfig.json, compiles the specified files with default compiler options.
tsc -b
Build a composite project in the working directory.
tsc --init
Creates a tsconfig.json with the recommended settings in the working directory.
tsc -p ./path/to/tsconfig.json
Compiles the TypeScript project located at the specified path.
tsc --help --all
An expanded version of this information, showing all possible compiler options
tsc --noEmit
tsc --target esnext
Compiles the current project, with additional settings.

3 uses cases …

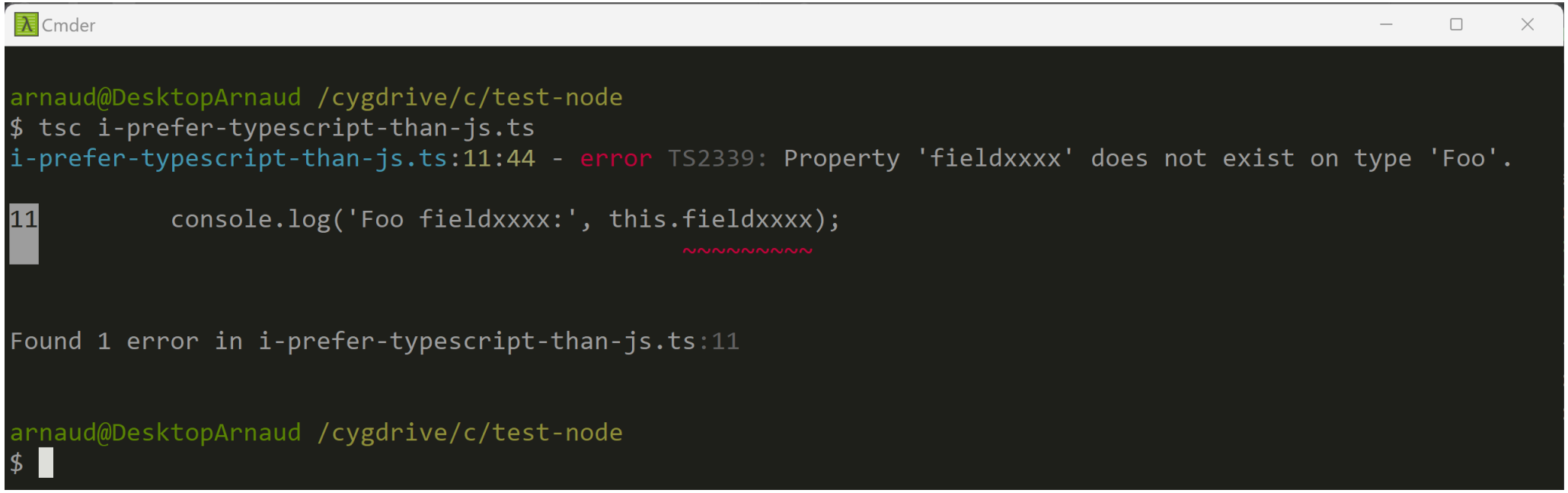
1/ compile 1 single .ts file
2/ compile **/*.ts in project
3/ « -w » real-time watch to recompile on change

$ tsc single-file.ts

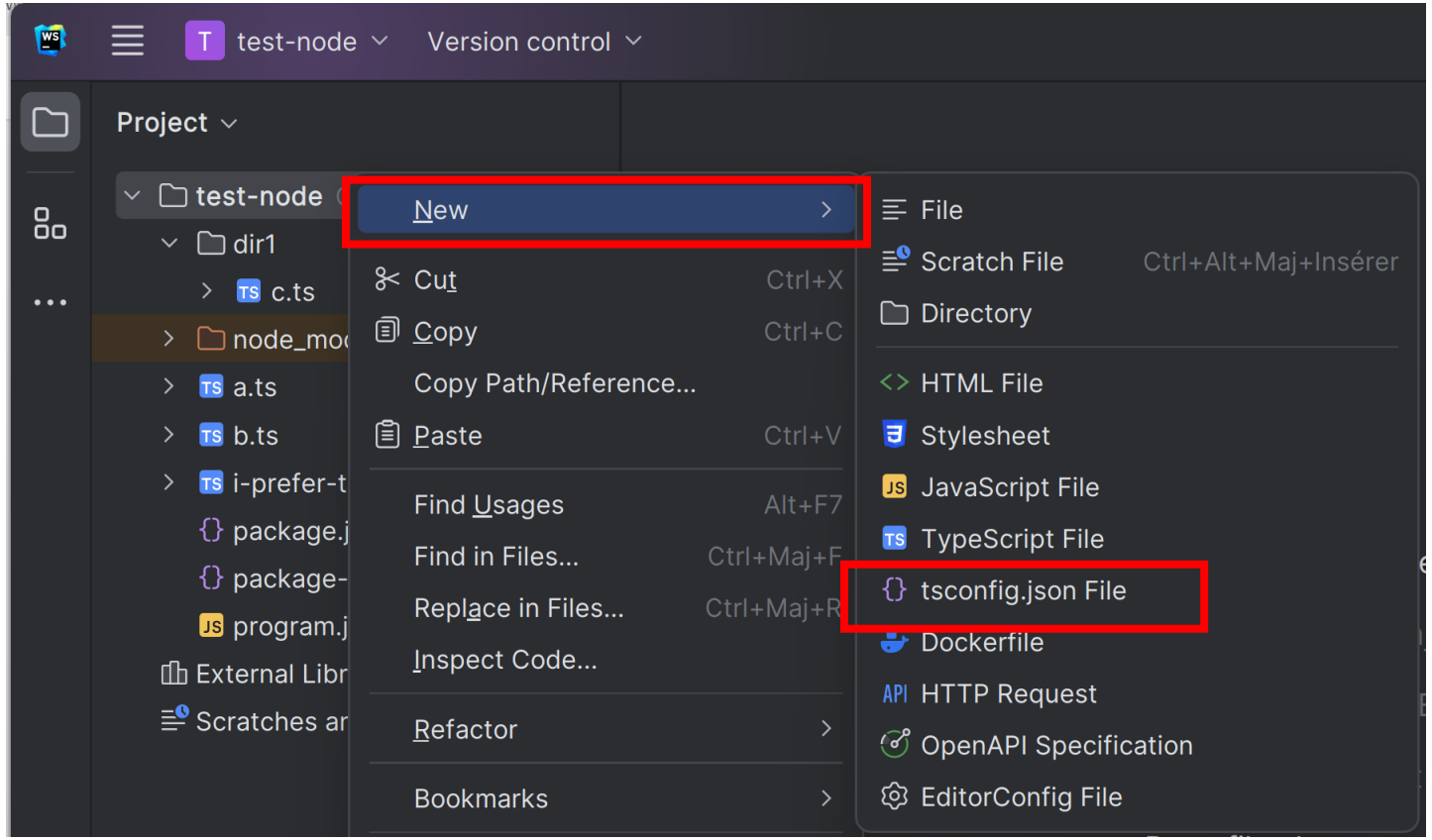


Tsc Compiles … find ERRORs messages at compile-time

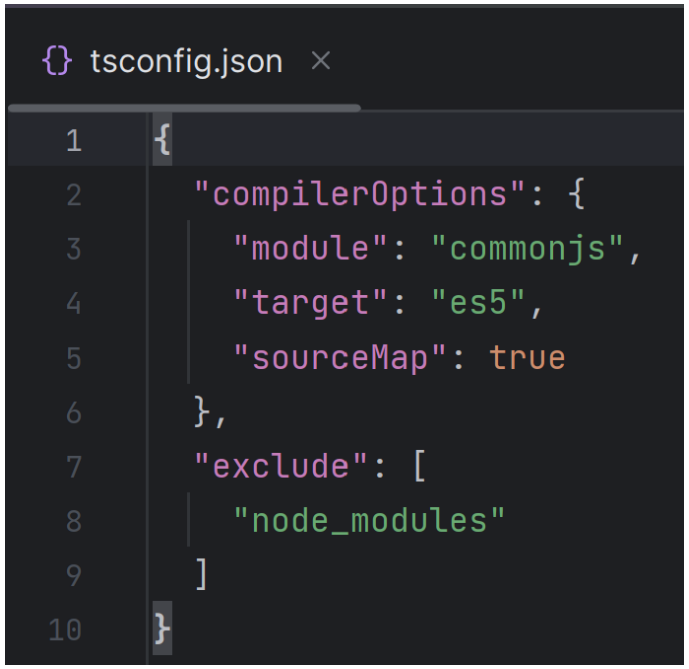
Compare… remember typo found at runtime ?
https://my-broken-web-site-deployed-in-PROD.html



Create tsc configuration file: « tsconfig.json »



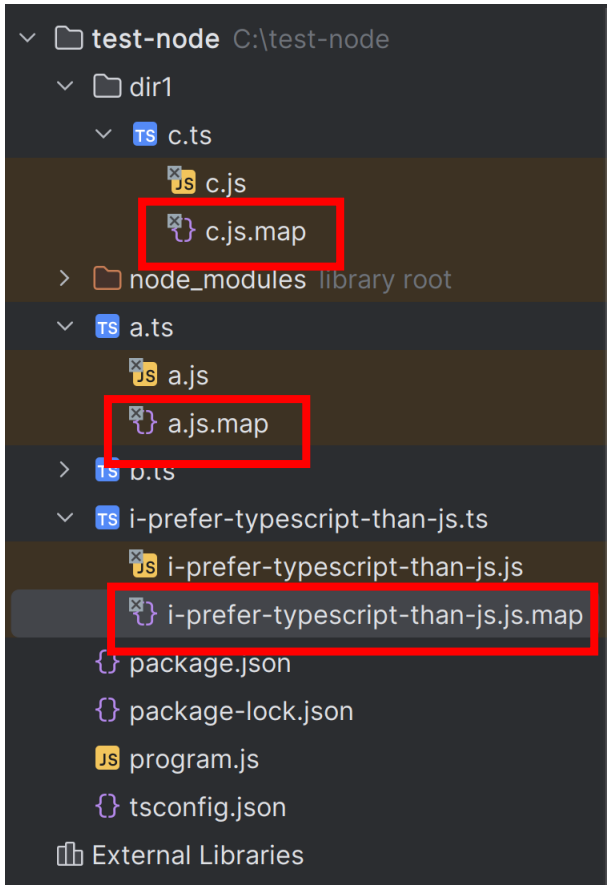
Default tsconfig.json



Notice sourceMap:true in tsconfig.json … and **/*.js.map files generated

tsconfig.json

```
{
  "compilerOptions": {
    "module": "commonjs",
    "target": "es5",
    "sourceMap": true
  },
  "exclude": [
    "node_modules"
  ]
}
```



.map file
… = « lineNumber » debugging info
between source « *.ts » and generated « *.js »

Analogy with java:
between source « *.java » and generated « *.class »

Remark and Analogy

JavaScript is the « low-level » assembler langage (~x64) on the Web
Supported by all « web browsers » AND « node » (~intel/amd processors)

No developper code anymore in assembly langage directly
(they prefer high-level C, C++, Java, C#, Python, …)

Web developper prefers TypeScript rather than Js directly ..

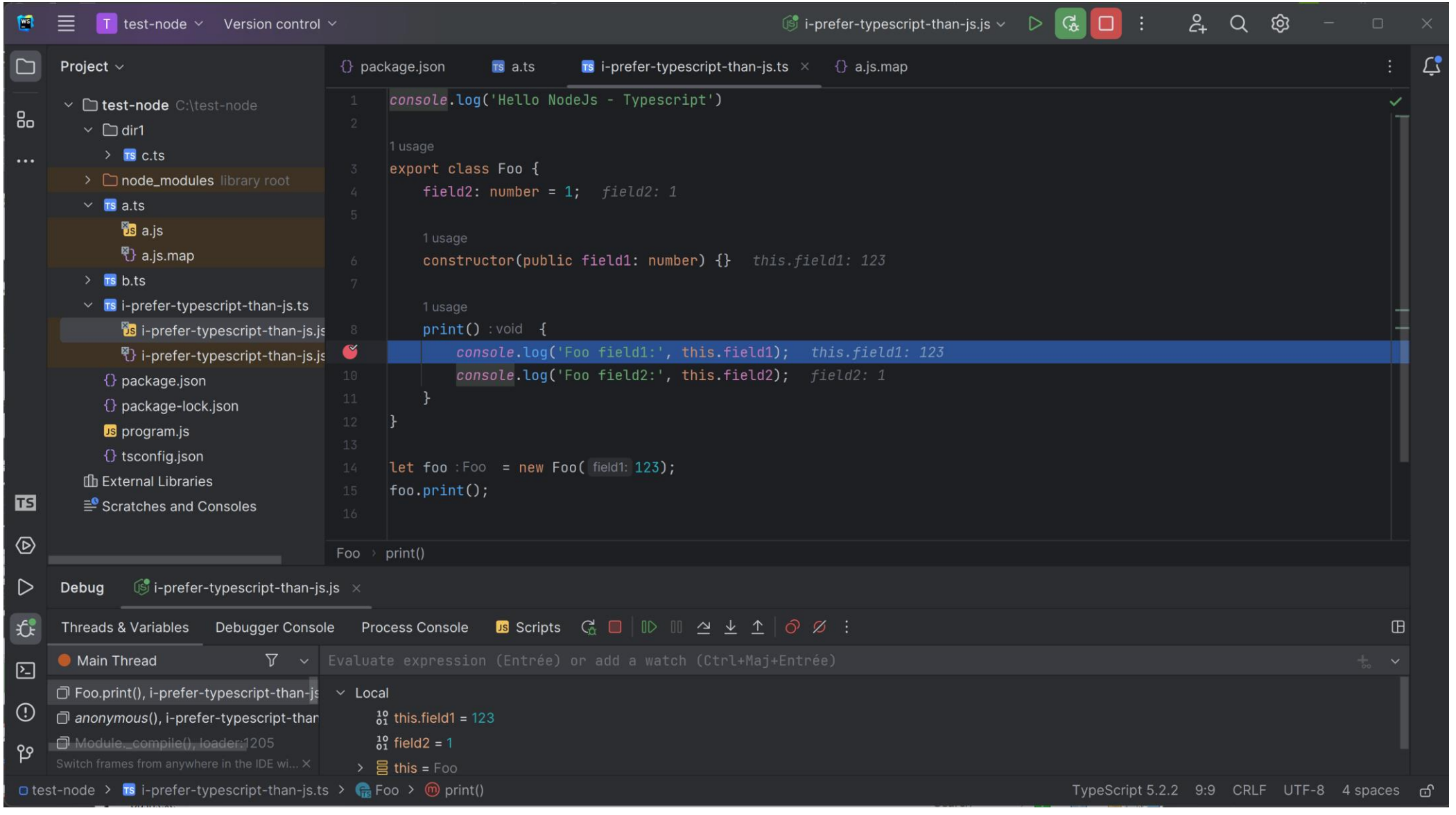
Remark of The Remark … WebAssembly

JavaScript is such an **horrible monster langage**
(with **eval** and strange permissive things)

W3c standard « WebAssembly » use only a SUBSET of JS
… that can be really statically typed / analysed / understood
then **transformed to native x64 code by « compiler »**

« WebAssembly » is trully a fast assembly langage
almost as fast as optimised « normal » code

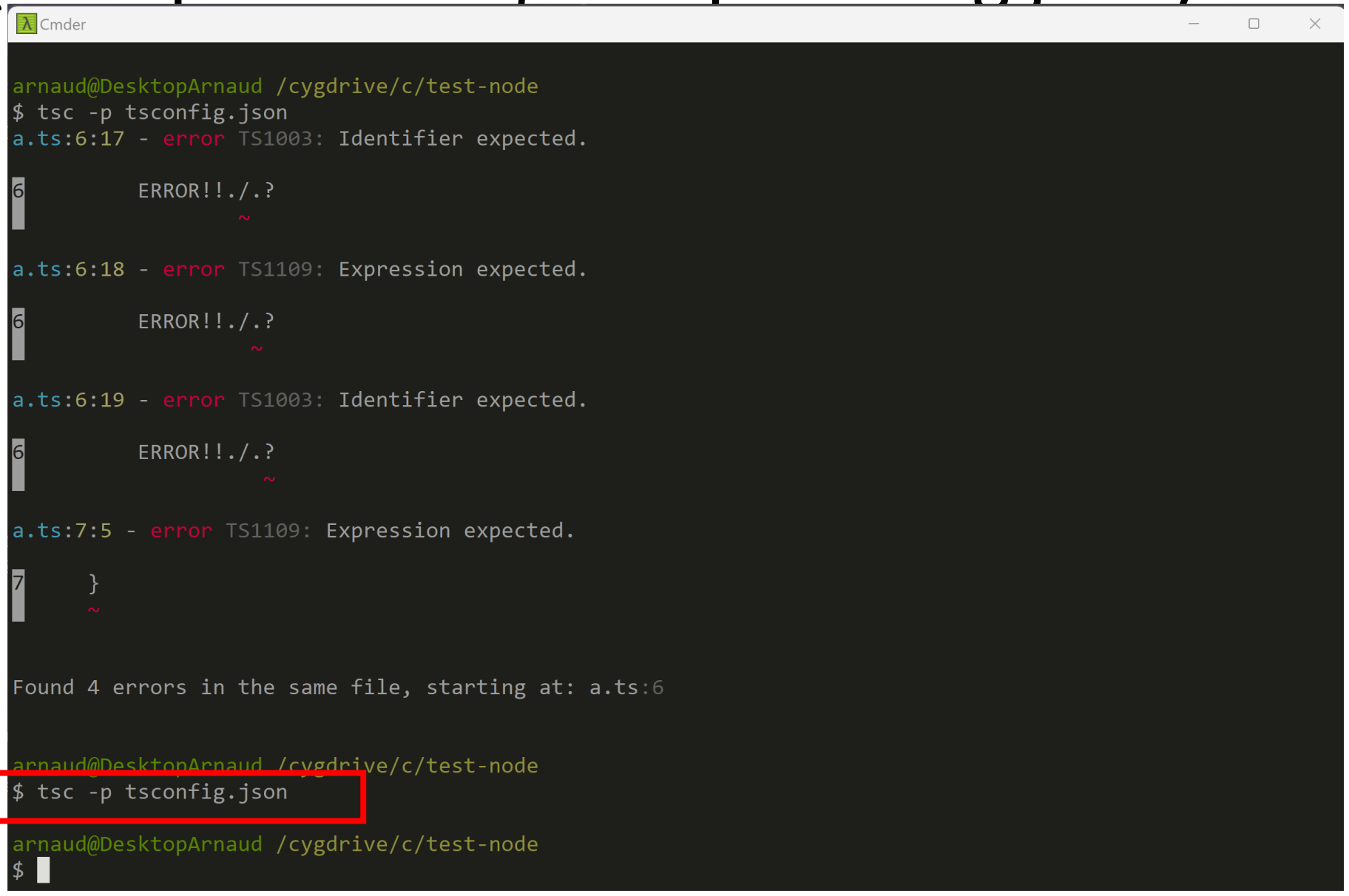
.map Goodies … can put breakpoint on *.ts (still debugging *.js for launch)



$ tsc
(or equivalent : $ tsc –p tsconfig.json)

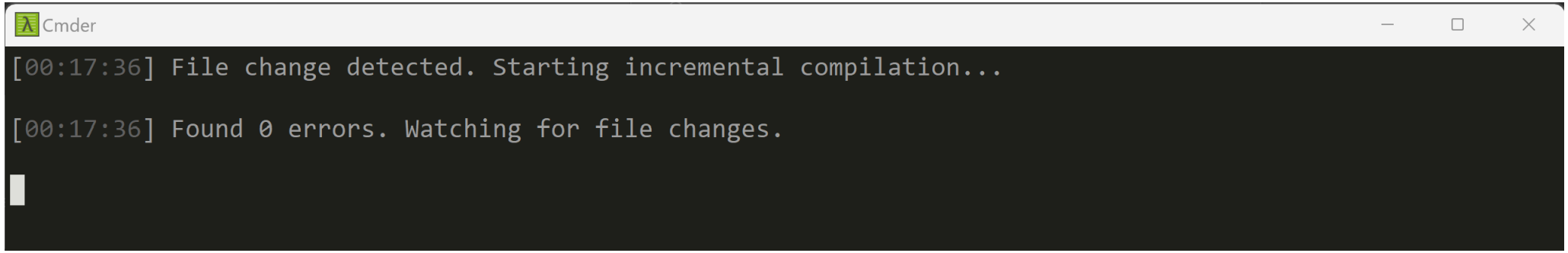
Case when error…

Case when OK…
NO log !!



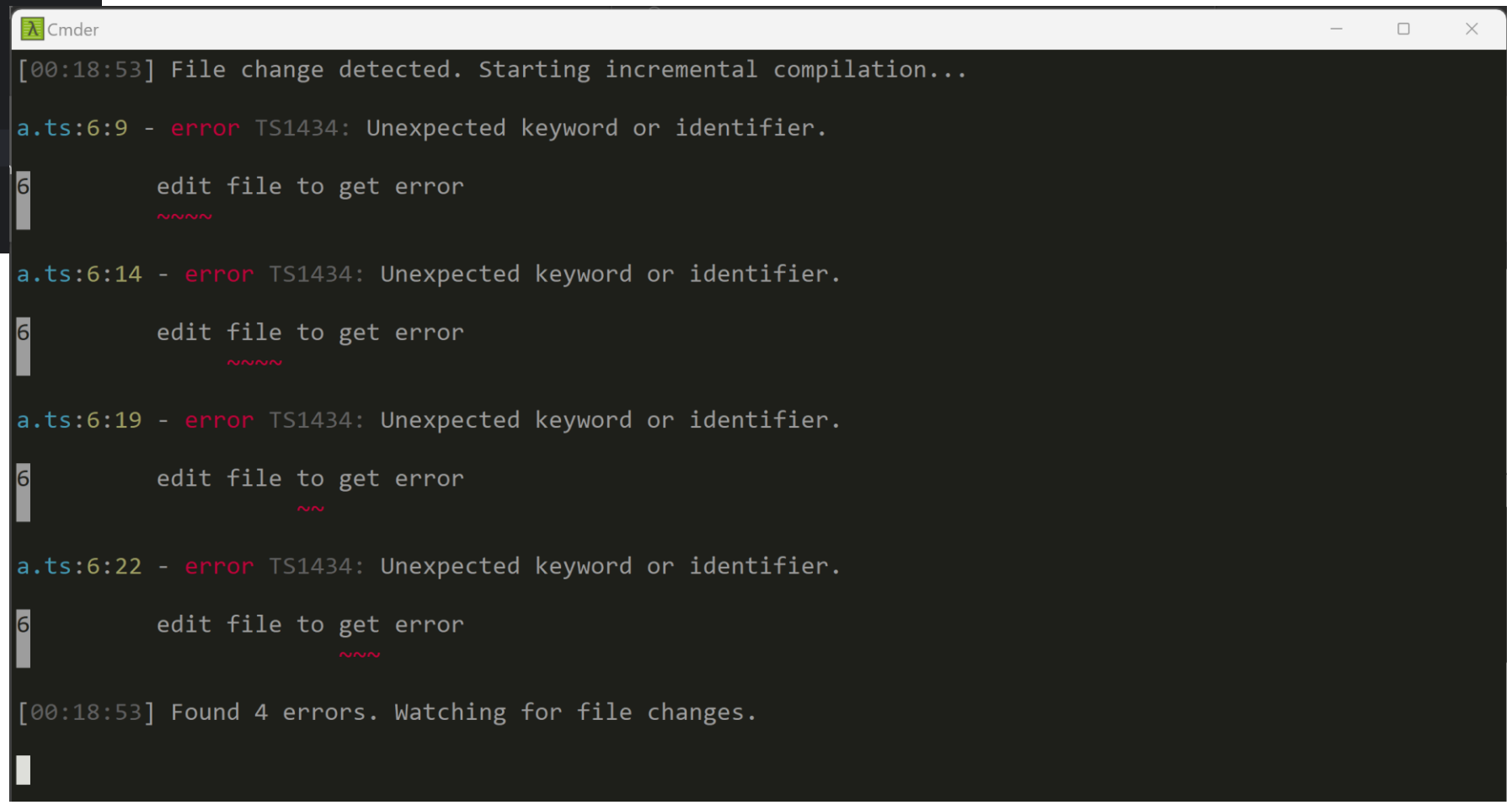
$ tsc -w

Case when OK…
NO log



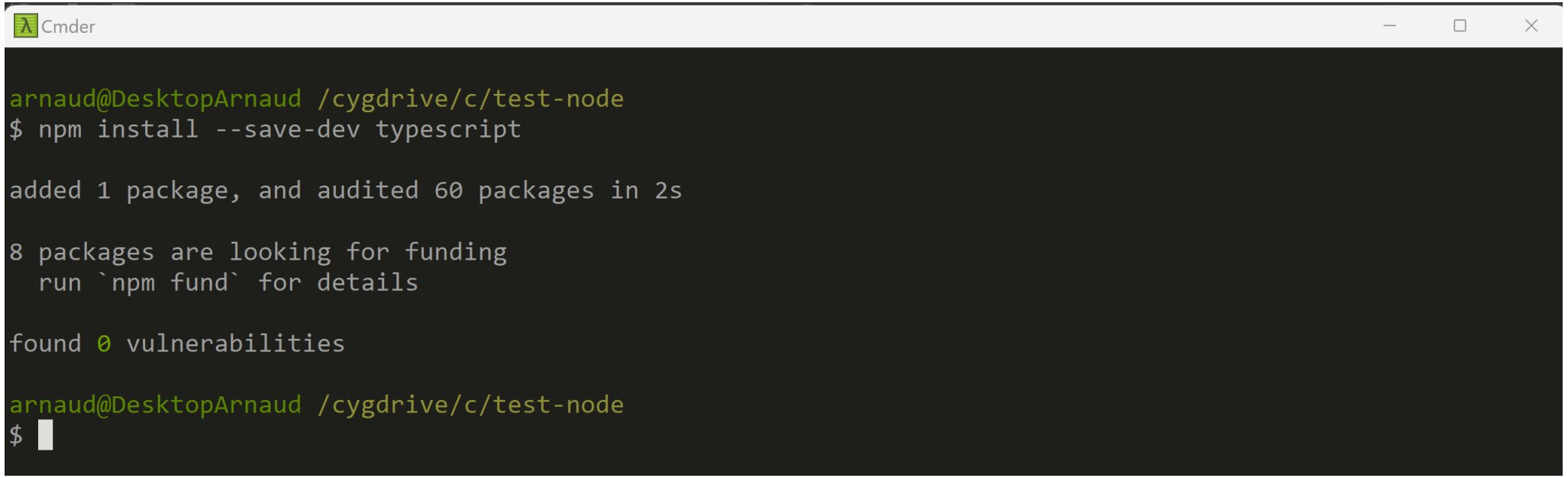
Case when error… cf next slide

Edit file to get Compile Error with –w (watch)

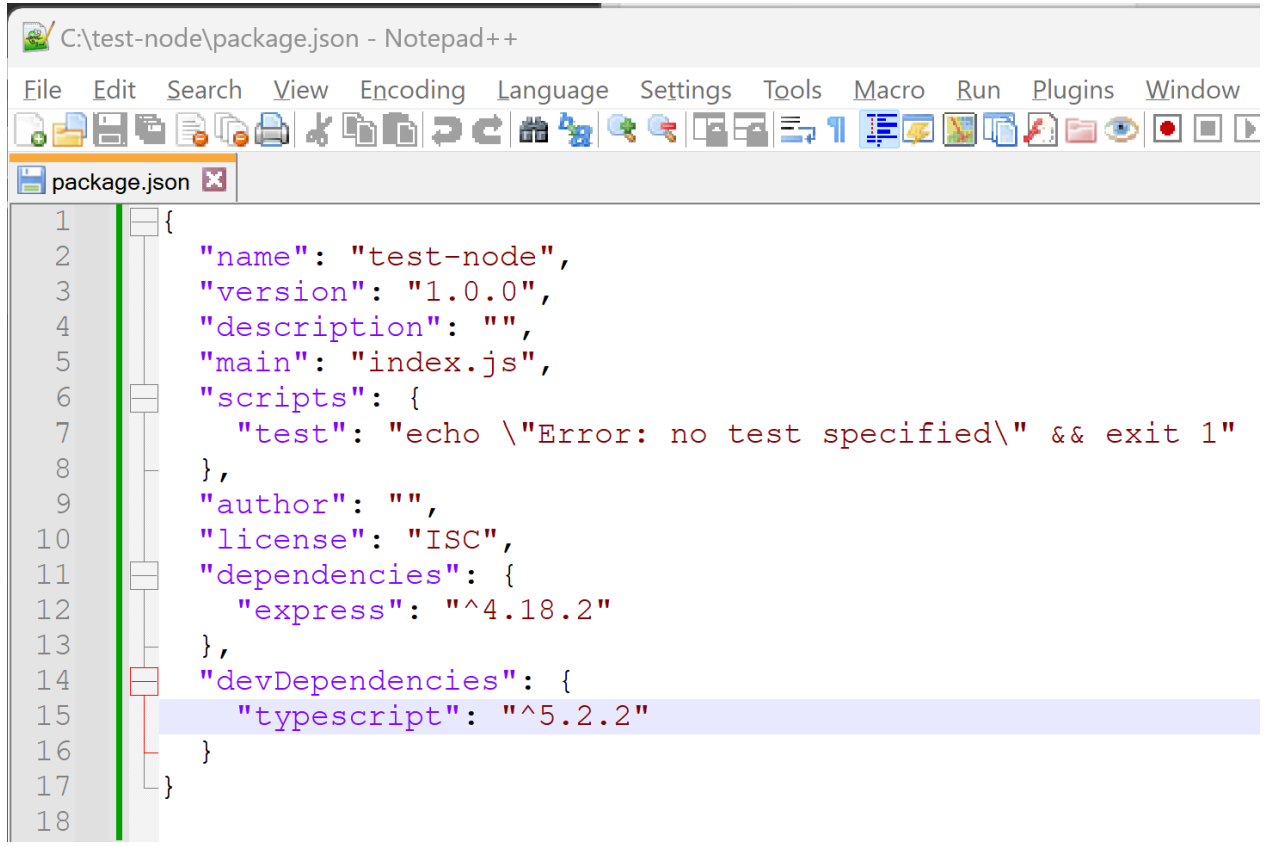


Install « typescript » in package.json devDependencies
(devDependencies != dependencies, != global)

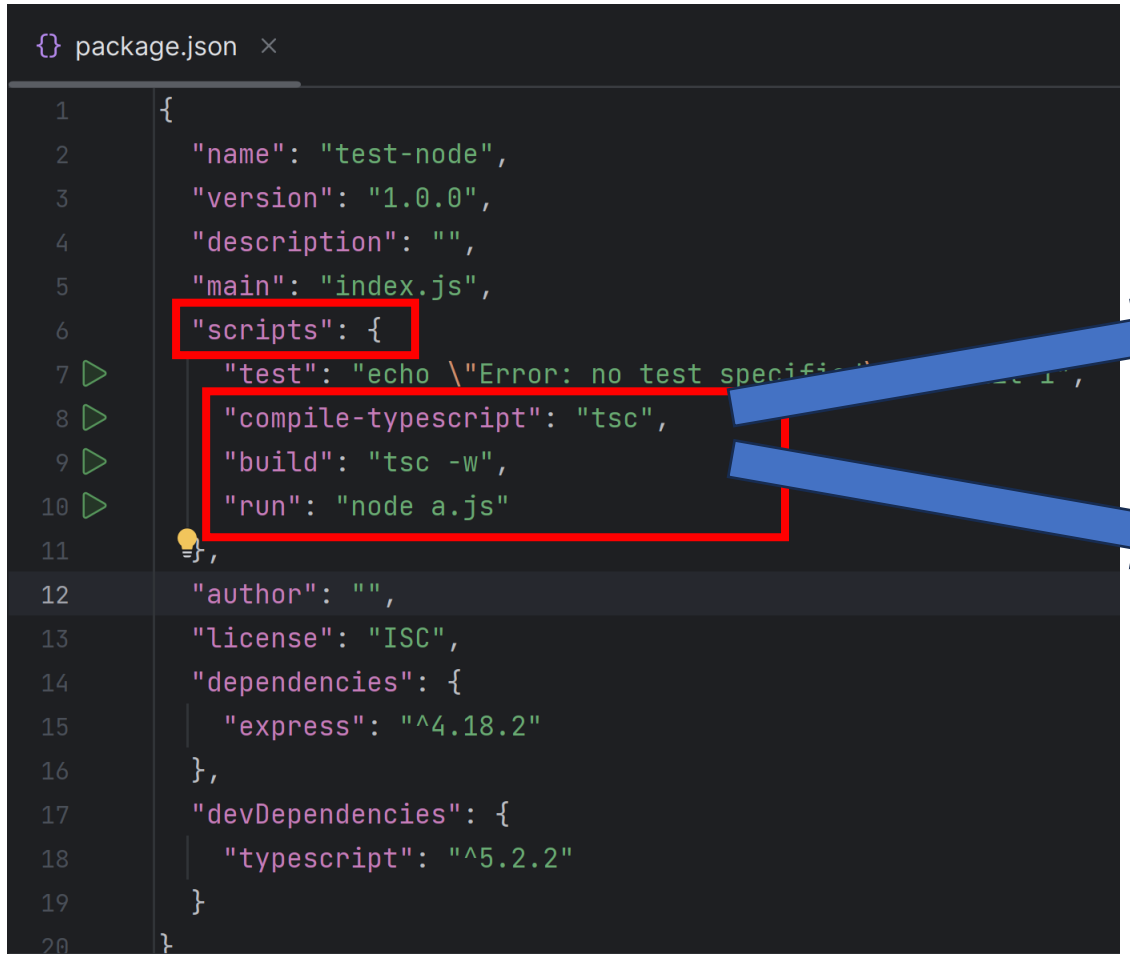
$ npm install –save-dev typescript



devDependencies...

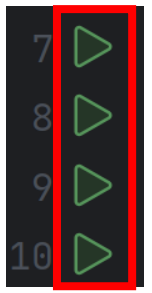


Edit package.json « scripts »



« $ **npm run compile-typescript** »
… is equivalent to **« $ tsc »**

« $ **npm run build** »
… is equivalent to **« $ tsc -w »**

Notice:
There are Run icons  **for each script line**

Understanding Angular « ng » client internals

« **npm run start**»

Internally Call

« **ng serve** »

Internally Call

« **tsc -w**»

Compile all *.ts to *.js
(maybe webpack all into a single app.js)

Internall call « **node** » with **express server**
… http server running on port « http://localhost:4200 »

All commands « npm », « ng », « tsc »

Run internally « **node** » !!!

Also Use « **live-reload.js** » …
so Web browser automatically refresh page on change

Coffe Break (?)

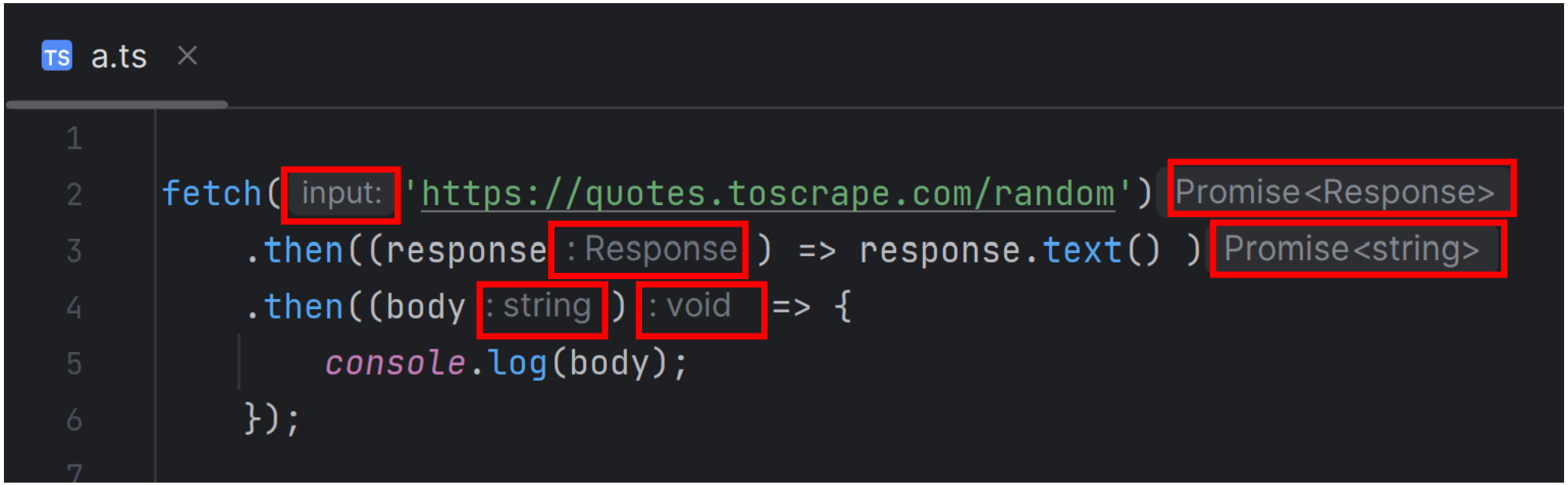
Introducing difficulties with Async Code: Callbacks, Promises, async/await, … Threads in a mono-threaded-user NodeJs model

A simple http GET fetch()
.. The Hello world on http !

a.ts

```
1
2  fetch( input: 'https://quotes.toscrape.com/random') Promise<Response>
3      .then((response : Response ) => response.text() ) Promise<string>
4      .then((body : string ) : void  => {
5          console.log(body);
6      });
7
```

Noticed WebStorm automatically show « **:type** » and parameter « **name:** » in grey



Adding console.log(...)

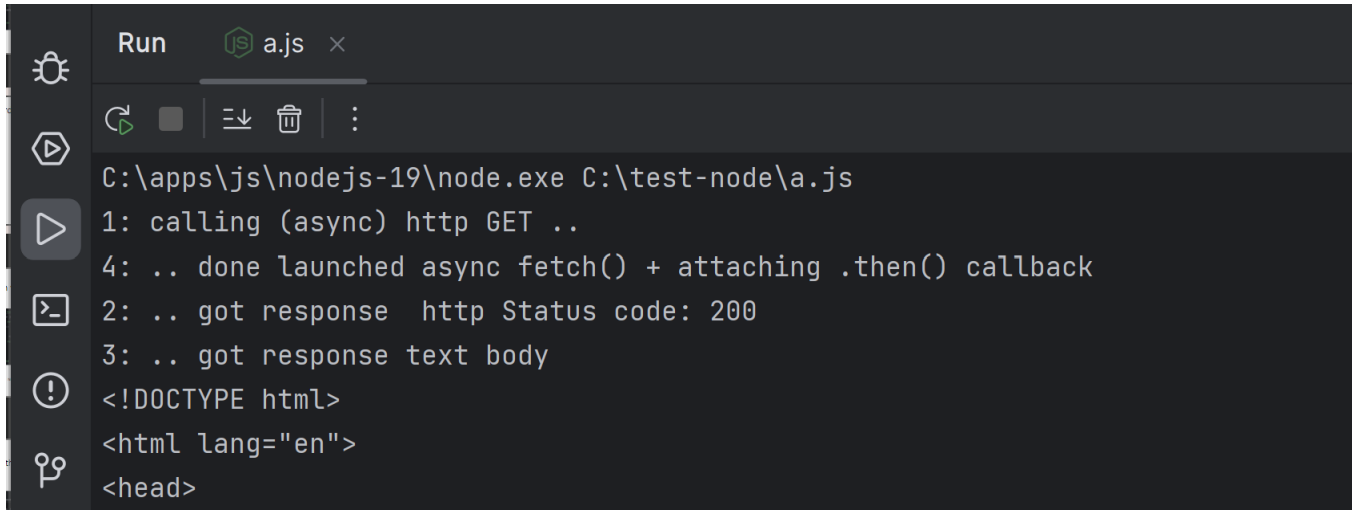
a.ts

```
8
9   console.log("1: calling (async) http GET ..");
10
11  fetch( input: 'https://quotes.toscrape.com/random') Promise<Response>
12      .then((response : Response ) => {
13          console.log('2: .. got response  http Status code: ' + response.status);
14          return response.text() // => cascade (+return) another async reading for consuming http response body ...
15      }) Promise<string>
16      .then((body : string ) : void  => {
17          console.log('3: .. got response text body');
18          console.log(body);
19      });
20
21  console.log('4: .. done launched async fetch() + attaching .then() callback ');
22
```

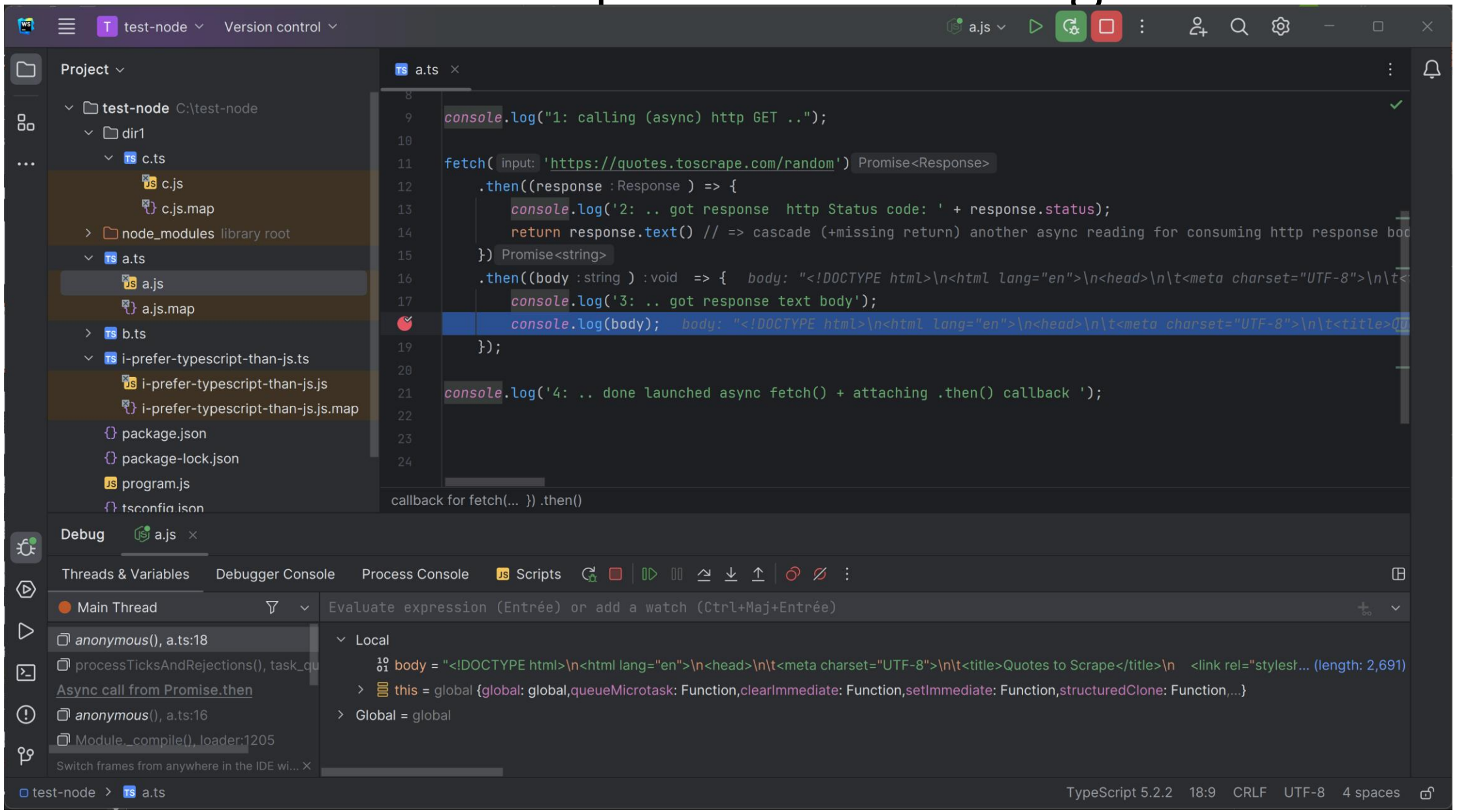
In which order are the console.log()s printed?

1, 2, 3, 4 ? … other? which order?

Response: 1, 4, 2, 3 !!



Put Breakpoint and Debug



Can you spot the bug here ?

```
11  fetch( input: 'https://quotes.toscrape.com/random') Promise<Response>
12      .then((response : Response ) : void  => {
13          console.log('.. got response  http Status code: ' + response.status);
14          response.text() // => cascade (missing return) another async reading for consuming http response body ...
15      }) Promise<void>
16      .then((body : void ) : void  => {
17          console.log('.. got response text body');
18          console.log(body);
19      });
20
21  console.log('.. done launched async fetch() + attaching .then() callback ');
```

Spot the Bug … 4 Clues

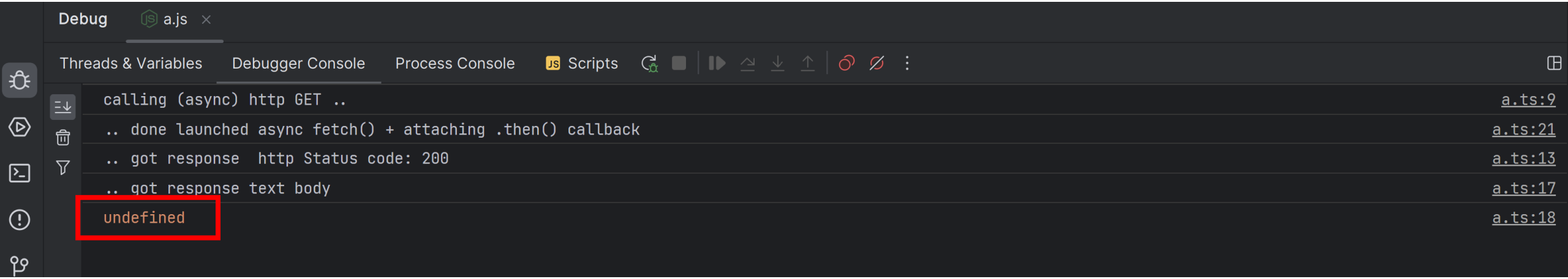
1/ No Compile ERROR … but Running => see in console logs

2/ Executing in Debug mode => check variables values

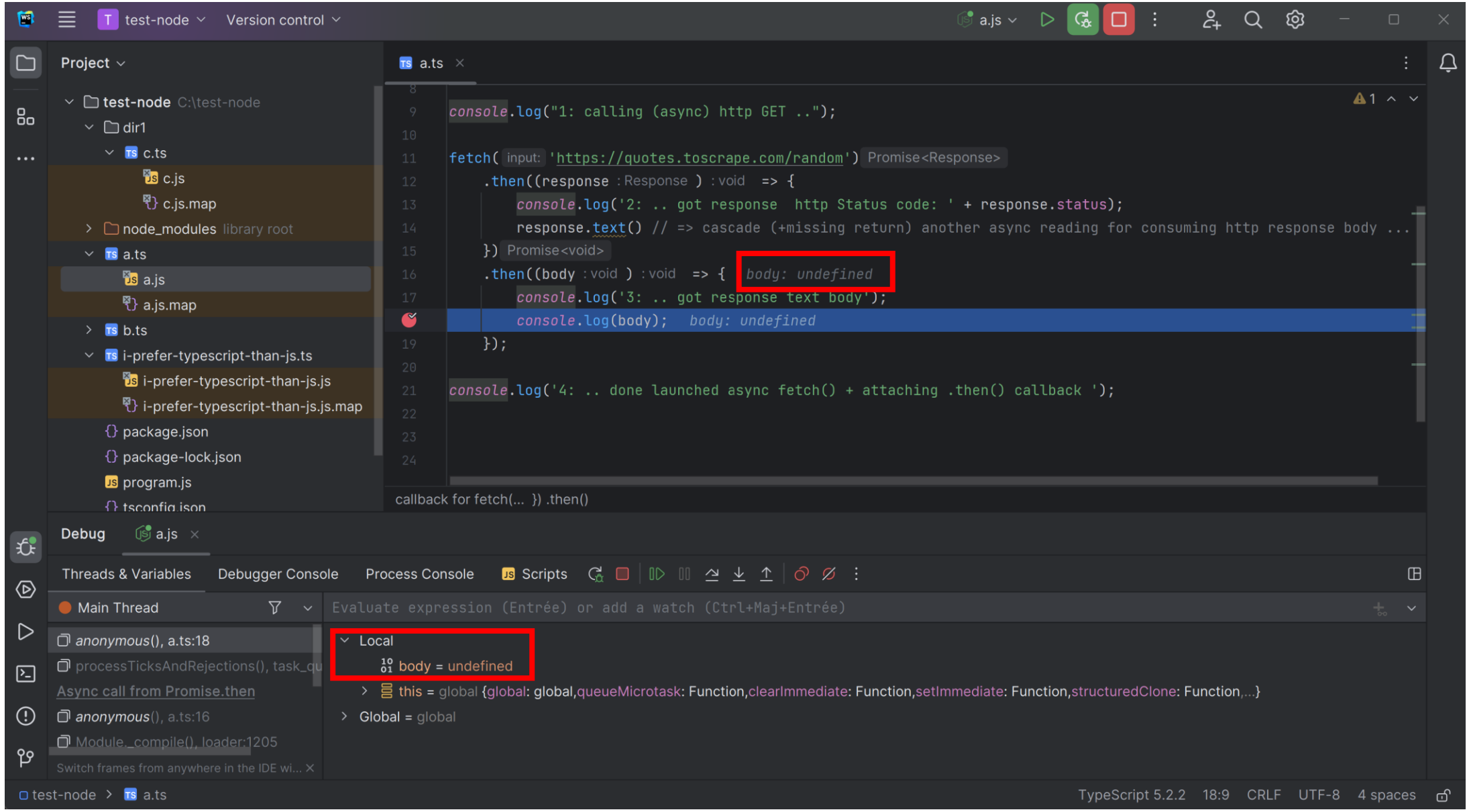
3/ WebStorm underline « .text() » as a possible warning

4/ Can you Spot the inconsistent Type infered by WebStorm
(not a bug in WebStorm)

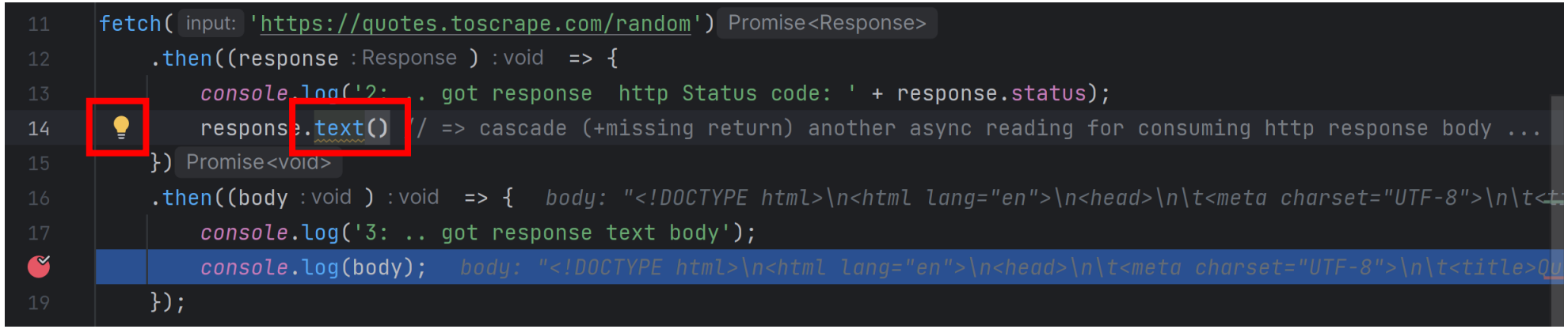
Clue 1/ Run, see in console log
Understand the « undefined » instead of <html>… ?

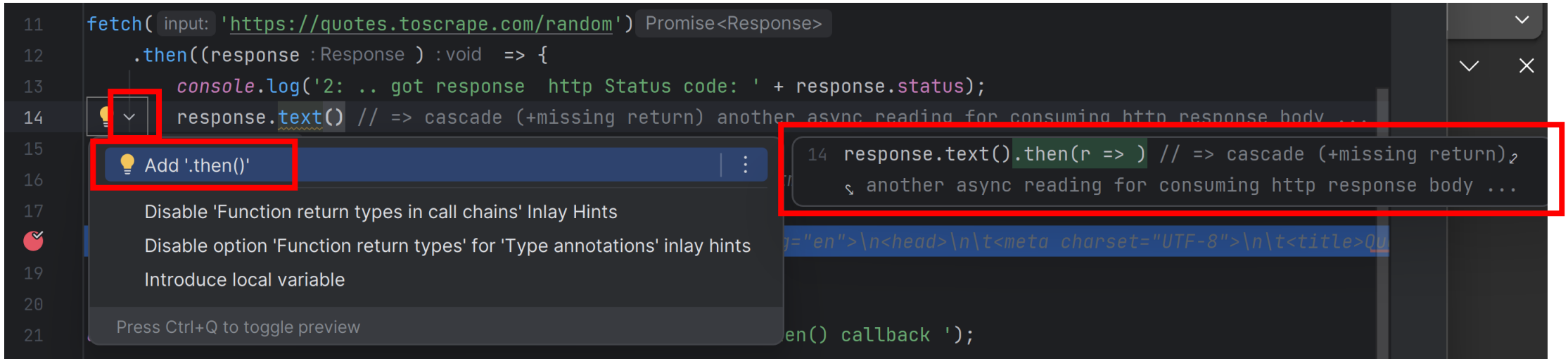


Clue 2 / Debug with Breakpoint
… indeed « body » is undefined



Clue 3/ .text() in underlined in WebStorm ?





Check « .text() » : Browsing code -> definition !

Clue 4/ Inconsistent types
… was expecting « string », got « void »

a.ts

```
console.log("1: calling (async) http GET ..");

fetch( input: 'https://quotes.toscrape.com/random') Promise<Response>
    .then((response : Response ) : void => {
        console.log('2: .. got response  http Status code: ' + response.status);
        response.text() // => cascade (+missing return) another async reading for consuming http response body ...
    }) Promise<void>
    .then((body : void ) : void => {
        console.log('3: .. got response text body');
        console.log(body);
    });

console.log('4: .. done launched async fetch() + attaching .then() callback ');
```

Promise and nested Callbacks => Cascading Hell

async/await Solution

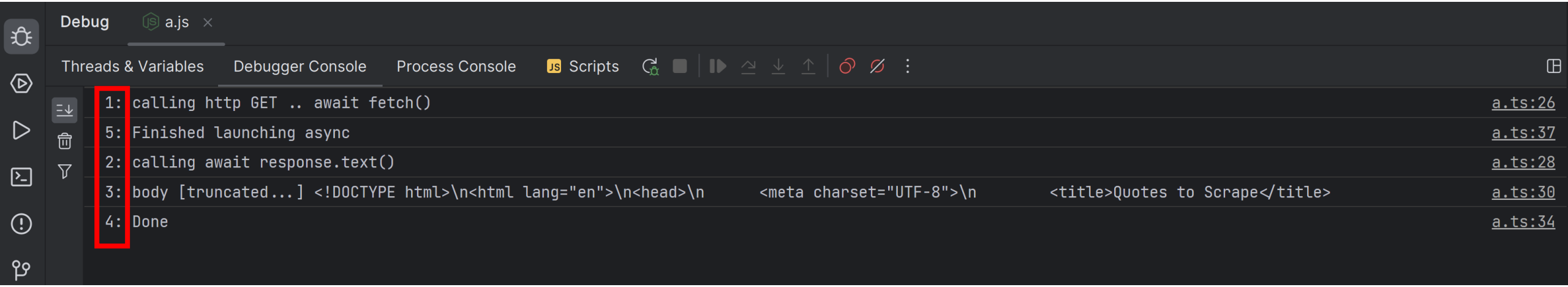
Cf also RxJS (out of scope here… But used in Angular)

Transforming fetch().then() … to await fetch()

a.ts

```
// @ts-ignore
1 usage
async function asyncFetchAndLog() : Promise<void>  {
    console.log("1: calling http GET .. await fetch()");
    let response : Response  = await fetch( input: 'https://quotes.toscrape.com/random');
    console.log("2: calling await response.text()");
    let body : string  = await response.text();
    console.log("3: body [truncated...] " + body.replace( searchValue: /\n/g,  replaceValue: '\\n').substring(0, 100));
}

asyncFetchAndLog().then(resp : void  => {
    console.log('4: Done');
});

console.log('5: Finished launching async');
```

Run async/await ...



Coffe Break / Lunch Break (?)

Next Steps…