

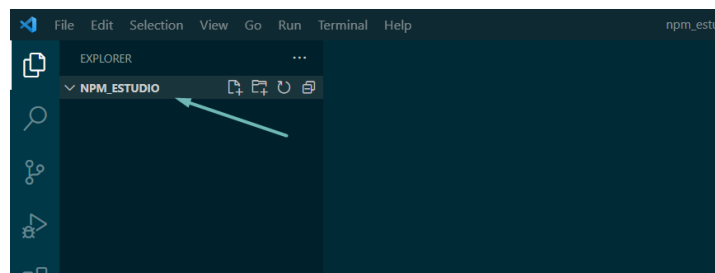
EXERCISES QUE TRABAJAREMOS EN EL CUE

- EXERCISE 1: COMANDOS BÁSICOS MÁS UTILIZADOS EN NPM.
- EXERCISE 2: REQUIRIENDO PAQUETES Y ARCHIVOS.

EXERCISE 1: COMANDOS BÁSICOS MÁS UTILIZADOS EN NPM.

Para profundizar más sobre el mundo de **npm**, haremos un repaso práctico de todos los comandos ya mencionados.

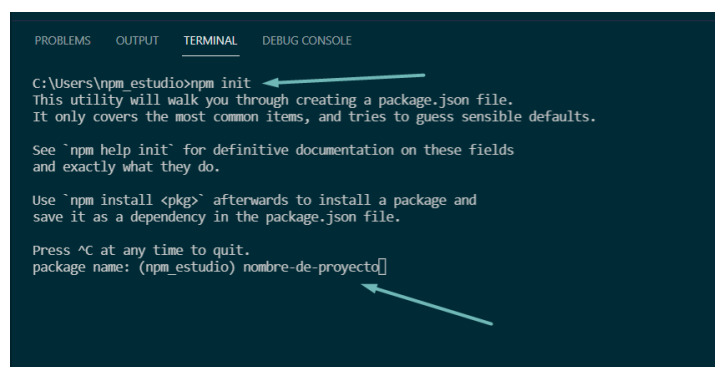
Creamos una carpeta llamada `npm_estudio`, y usamos Visual Studio Code para trabajar en ella.



Inicializaremos un nuevo proyecto con el comando **npm init**, y al ingresarlo, deberás incluir los datos requeridos.

Primero ingresamos el nombre del proyecto, y presionamos la tecla enter.

Si bien no existe una convención para nombrar los proyectos **npm**, se suelen utilizar palabras en letras minúsculas, separadas por un guion medio: "nombre-de-proyecto".




Luego, **npm** consultará por la versión de tu proyecto. Puedes presionar la tecla enter sin ingresar un valor, y éste usará el valor por defecto 1.0.0.

```
PROBLEMS  OUTPUT  TERMINAL  DEBUG CONSOLE

C:\Users\npm_estudio>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: (npm_estudio) nombre-de-proyecto
version: (1.0.0) 
```


Ahora, debes ingresar una descripción resumida de tu proyecto; si lo deseas, puedes dejar este campo en blanco, y presionar la tecla enter.

```
PROBLEMS  OUTPUT  TERMINAL  DEBUG CONSOLE

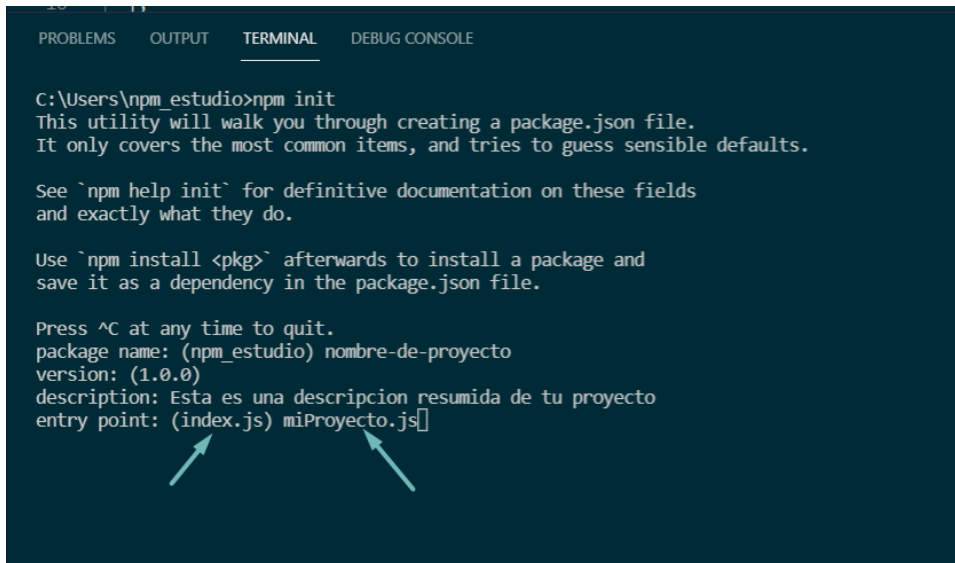
C:\Users\npm_estudio>npm init
This utility will walk you through creating a package.json file.
It only covers the most common items, and tries to guess sensible defaults.

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: (npm_estudio) nombre-de-proyecto
version: (1.0.0)
description: Esta es una descripcion resumida de tu proyecto 
```

npm preguntará por el nombre del archivo que inicia tu programa (el **entry point** o punto de entrada). Por defecto, es `index.js`, pero si tu archivo se llama `miProyecto.js`, puedes escribirlo en la consola y presionar la tecla enter. Este nombre solo es usado para identificar el archivo principal de tu proyecto, en caso de que sea requerido como paquete dentro de otro proyecto **node**.



```
PROBLEMS OUTPUT TERMINAL DEBUG CONSOLE

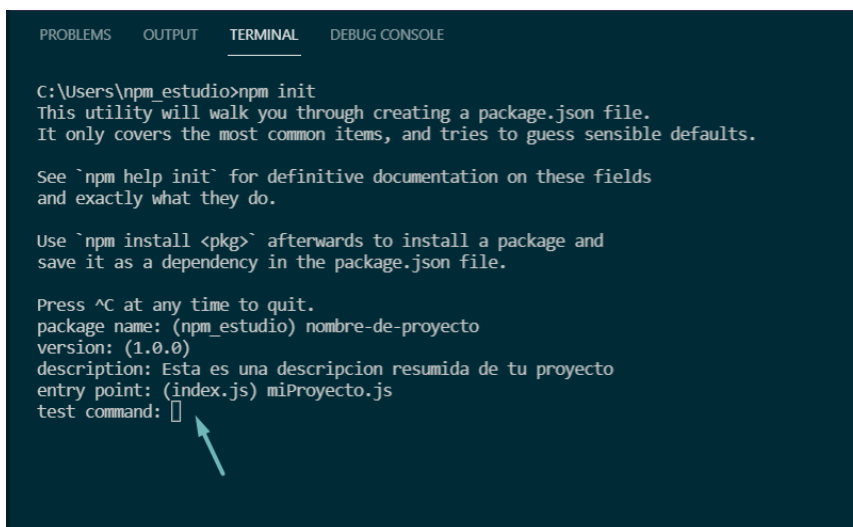
C:\Users\npm_estudio>npm init
This utility will walk you through creating a package.json file.
It only covers the most common items, and tries to guess sensible defaults.

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: (npm_estudio) nombre-de-proyecto
version: (1.0.0)
description: Esta es una descripción resumida de tu proyecto
entry point: (index.js) miProyecto.js
```

También preguntará el comando para comenzar a realizar el **testing** de tu aplicación, si es que tienes alguno definido; por ahora, dejaremos el campo en blanco, y presionaremos la tecla enter.



```
PROBLEMS OUTPUT TERMINAL DEBUG CONSOLE

C:\Users\npm_estudio>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: (npm_estudio) nombre-de-proyecto
version: (1.0.0)
description: Esta es una descripción resumida de tu proyecto
entry point: (index.js) miProyecto.js
test command:
```


Puedes especificar un repositorio de **git** para tu proyecto, o también dejar el campo vacío, y presionar la tecla enter.

```
PROBLEMS  OUTPUT  TERMINAL  DEBUG CONSOLE

C:\Users\npm_estudio>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: (npm_estudio) nombre-de-proyecto
version: (1.0.0)
description: Esta es una descripcion resumida de tu proyecto
entry point: (index.js) miProyecto.js
test command:
git repository:  
```

Luego, **npm** te pedirá que ingreses palabras claves que pueden ayudar a que tu proyecto sea encontrado, esto en el caso de que quieras subirlo para ser utilizado por otros desarrolladores. Al igual que las veces anteriores, también puedes dejar el campo vacío, y presionar la tecla enter.

```
PROBLEMS  OUTPUT  TERMINAL  DEBUG CONSOLE

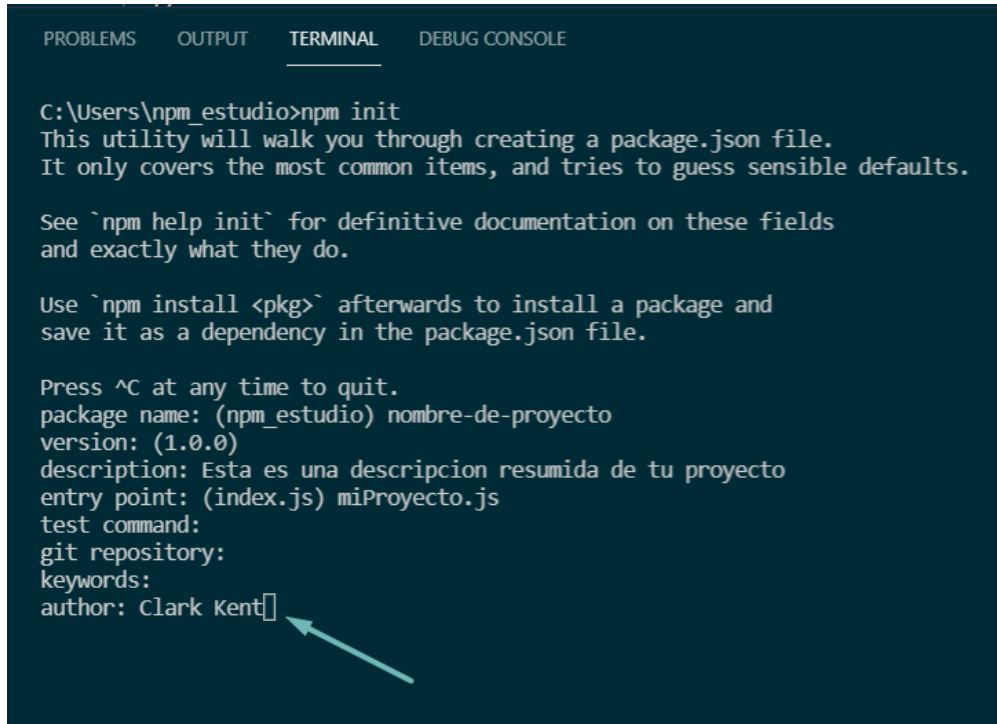
C:\Users\npm_estudio>npm init
This utility will walk you through creating a package.json file.
It only covers the most common items, and tries to guess sensible defaults.

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: (npm_estudio) nombre-de-proyecto
version: (1.0.0)
description: Esta es una descripcion resumida de tu proyecto
entry point: (index.js) miProyecto.js
test command:
git repository:
keywords:  
```

Puedes ingresar el nombre del autor.



```
PROBLEMS  OUTPUT  TERMINAL  DEBUG CONSOLE

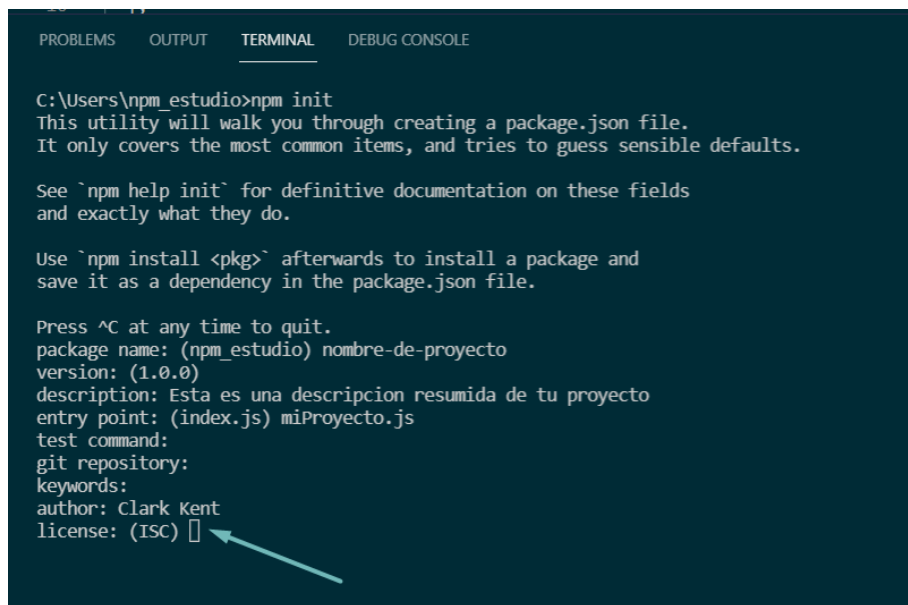
C:\Users\npm_estudio>npm init
This utility will walk you through creating a package.json file.
It only covers the most common items, and tries to guess sensible defaults.

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: (npm_estudio) nombre-de-proyecto
version: (1.0.0)
description: Esta es una descripcion resumida de tu proyecto
entry point: (index.js) miProyecto.js
test command:
git repository:
keywords:
author: Clark Kent
```

Elegir el tipo de licencia que define el uso que se le puede dar a tu proyecto, o dejar la opción por defecto, y presionar la tecla enter.



```
PROBLEMS  OUTPUT  TERMINAL  DEBUG CONSOLE

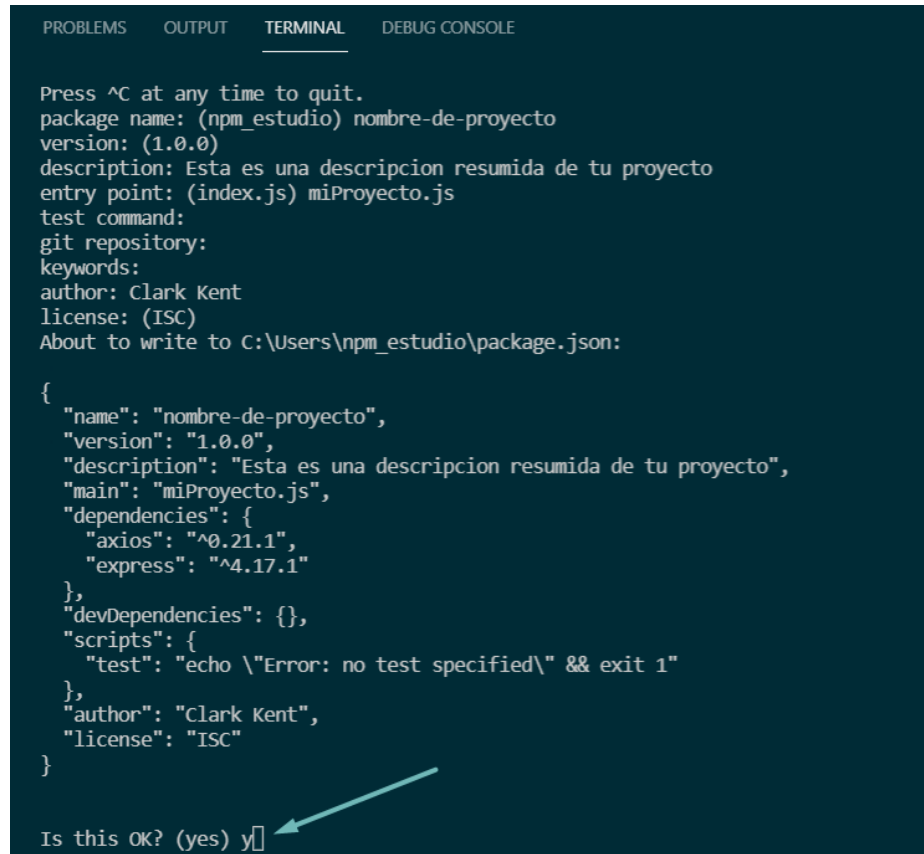
C:\Users\npm_estudio>npm init
This utility will walk you through creating a package.json file.
It only covers the most common items, and tries to guess sensible defaults.

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: (npm_estudio) nombre-de-proyecto
version: (1.0.0)
description: Esta es una descripcion resumida de tu proyecto
entry point: (index.js) miProyecto.js
test command:
git repository:
keywords:
author: Clark Kent
license: (ISC)
```

Finalmente, **npm** te solicitará que confirmes la información ingresada. Presiona la tecla "y", y luego la tecla enter.



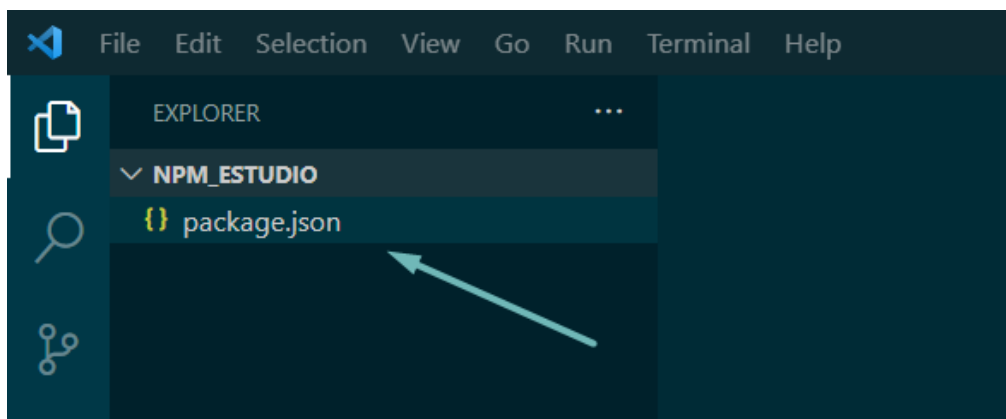
```
PROBLEMS OUTPUT TERMINAL DEBUG CONSOLE

Press ^C at any time to quit.
package name: (npm_estudio) nombre-de-proyecto
version: (1.0.0)
description: Esta es una descripcion resumida de tu proyecto
entry point: (index.js) miProyecto.js
test command:
git repository:
keywords:
author: Clark Kent
license: (ISC)
About to write to C:\Users\npm_estudio\package.json:

{
  "name": "nombre-de-proyecto",
  "version": "1.0.0",
  "description": "Esta es una descripcion resumida de tu proyecto",
  "main": "miProyecto.js",
  "dependencies": {
    "axios": "^0.21.1",
    "express": "^4.17.1"
  },
  "devDependencies": {},
  "scripts": {
    "test": "echo \"Error: no test specified\" && exit 1"
  },
  "author": "Clark Kent",
  "license": "ISC"
}

Is this OK? (yes) y
```

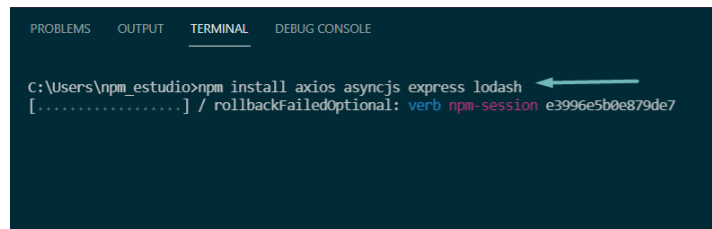
Luego de este proceso, deberías ver un nuevo archivo creado en tu proyecto, del cual pronto se conocerá su función.



Ahora, comenzaremos con la instalación de los siguientes paquetes:

- Express.
- Axios.
- Lodash.
- Asyncjs.

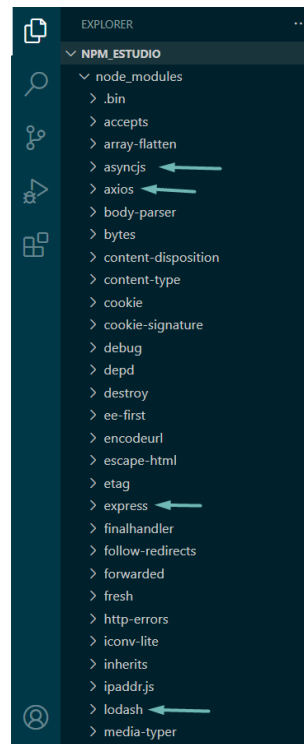
Puedes instalar más de uno en una sola línea.



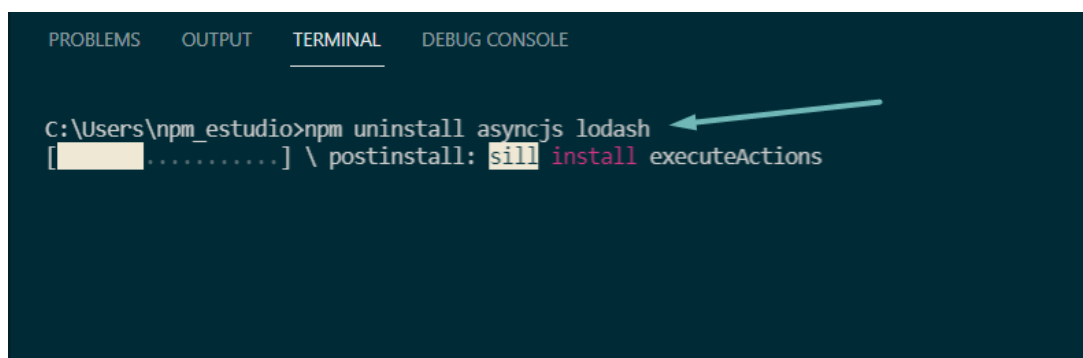
```
PROBLEMS  OUTPUT  TERMINAL  DEBUG CONSOLE

C:\Users\npm_estudio>npm install axios asyncjs express lodash
[.....] / rollbackFailedOptional: verb npm-session e3996e5b0e879de7
```

Revisaremos la carpeta `node_modules`, y ver los archivos que conforman un paquete. Podemos notar que, con unos cuantos paquetes instalados, esta carpeta ya contiene una gran cantidad de archivos y subdirectorios.



De pronto hemos decidido que no haremos uso de la librería **asyncjs** y **lodash**. Desinstala estos paquetes utilizando el comando **npm uninstall**. Al igual que el comando **npm install**, puedes desinstalar más de un paquete al mismo tiempo.



Ahora revisaremos el estado en que se encuentran nuestros paquetes, y si debemos tener un cuidado especial con alguna vulnerabilidad conocida.


```
PROBLEMS  OUTPUT  TERMINAL  DEBUG CONSOLE

C:\Users\npm_estudio>npm audit ←
                               === npm audit security report ===
found 0 vulnerabilities
in 52 scanned packages

C:\Users\npm_estudio>
```

Por último, veamos la lista de los paquetes que tenemos instalados.

```
PROBLEMS  OUTPUT  TERMINAL  DEBUG CONSOLE

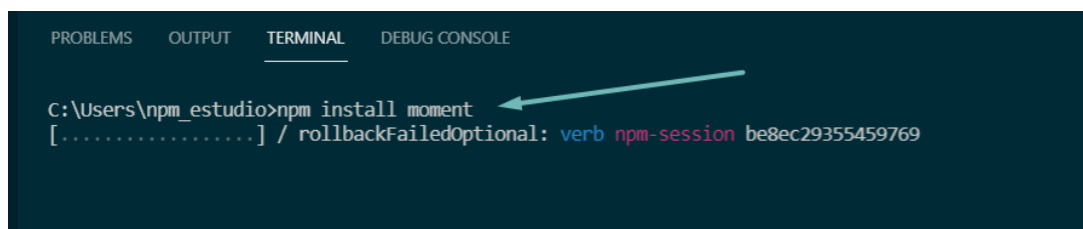
C:\Users\npm_estudio>npm list
npm_estudio@1.0.0 C:\Users\npm_estudio
+-- axios@0.21.1
|  `-- follow-redirects@1.14.1
-- express@4.17.1
   +-- accepts@1.3.7
   |   +-- mime-types@2.1.31
   |   |   `-- mime-db@1.48.0
   |   `-- negotiator@0.6.2
   +-- array-flatten@1.1.1
   +-- body-parser@1.19.0
   |   +-- bytes@3.1.0
   |   +-- content-type@1.0.4 deduped
   |   +-- debug@2.6.9 deduped
   |   +-- depd@1.1.2 deduped
   |   +-- http-errors@1.7.2
   |   |   +-- depd@1.1.2 deduped
   |   |   +-- inherits@2.0.3
   |   |   +-- setprototypeof@1.1.1 deduped
   |   |   +-- statuses@1.5.0 deduped
   |   |   `-- toidentifier@1.0.0
   |   +-- iconv-lite@0.4.24
   |   |   `-- safer-buffer@2.1.2
   |   +-- on-finished@2.3.0 deduped
   |   +-- qs@6.7.0 deduped
   |   +-- raw-body@2.4.0
   |   |   +-- bytes@3.1.0 deduped
   |   |   +-- http-errors@1.7.2 deduped
   |   |   +-- iconv-lite@0.4.24 deduped
   |   |   `-- unpipe@1.0.0 deduped
   |   `-- type-is@1.6.18 deduped
   +-- content-disposition@0.5.3
   |   `-- safe-buffer@5.1.2 deduped
   +-- content-type@1.0.4
   +-- cookie@0.4.0
```

Con esto concluye nuestro repaso por los comandos básicos más utilizados en el mundo de **npm**. Existen muchos otros que el **CLI de npm** puede soportar, pero con los que hemos revisado en este documento tienes todo lo necesario para comenzar a utilizar los paquetes de **node**, y beneficiarte del gran ecosistema de **JavaScript** y **Node.js**.

EXERCISE 2: REQUIRIENDO PAQUETES Y ARCHIVOS.

Para ejercitar nuestro uso de requerimiento de paquetes en **node**, instalaremos **Moment.js**. Este es un paquete muy popular, y aunque su uso está disminuyendo, e incluso en la documentación se señala que ya no existen razones de peso para preferirlo (esto debido a las actualizaciones en el uso de fechas nativo en **JavaScript**), nos servirá de ejemplo para seguir practicando cómo usar paquetes en nuestro código.

En el mismo proyecto que hemos estado usando en el último Text Class, ejecutaremos el comando **npm install moment**.



```
PROBLEMS  OUTPUT  TERMINAL  DEBUG CONSOLE

C:\Users\npm_estudio>npm install moment
[.....] / rollbackFailedOptional: verb npm-session be8ec29355459769
```

Dentro del archivo **utils**, importaremos el paquete **momentjs**, y crearemos una función que nos devuelva la fecha actual. Recuerda que siempre puedes leer la documentación para conocer el uso de un paquete en particular (<https://momentjs.com/docs/#/manipulating/subtract/> - <https://momentjs.com/docs/#/displaying/calendar-time/>).

Al hacer uso de **moment()**, le estamos solicitando a nuestro paquete que nos indique cuál es la fecha actual. Luego de eso, emplearemos **subtract()**, una función que permite conocer qué fecha fue hace x cantidad de tiempo; esta recibe dos argumentos: el primero la cantidad de tiempo que queremos restar, y en el segundo le decimos qué unidad de tiempo queremos restar. Por último, utilizamos el método **calendar()**, que nos devolverá la fecha en un formato distinto.

Por su parte, nuestra función **queDiaFue()**, toma un argumento, que será la cantidad de días que le pasaremos a la función **subtract** para saber qué fecha fue hace x cantidad de días.

```
JS utils.js  X  JS index.js

JS utils.js > queDiaFue
1  var moment = require("moment");
2
3
4  function queDiaFue(dias){
5
6      return moment().subtract(dias, 'days').calendar()
7
8  }
9
10
```

Exportamos la función.

```
JS utils.js  X  JS index.js

JS utils.js > multiplicaDosEnteros
1  var moment = require("moment");
2
3
4  function queDiaFue(dias){
5
6      return moment().subtract(dias, 'days').calendar()
7
8  }
9
10
11  function sumaDosEnteros(enteroUno, enteroDos) {
12      return enteroUno + enteroDos
13  }
14
15  function multiplicaDosEnteros(enteroUno, enteroDos) {
16      return enteroUno * enteroDos
17  }
18
19  module.exports = { sumaDosEnteros: sumaDosEnteros, multiplicaDosEnteros: multiplicaDosEnteros, queDiaFue: queDiaFue }
```

Y ahora, en nuestro archivo **index.js**, haremos uso de esta función y corremos nuestro programa.

```

JS utils.js JS index.js X
JS index.js > ...
1 var axios = require('axios');
2 var faker = require('faker');
3 var utils = require('./utils');
4
5 faker.locale = "es";
6
7 console.log(faker.address.streetAddress());
8
9 console.log(utils.sumaDosEnteros(3, 4));
10
11 console.log(utils.multiplicaDosEnteros(3, 4));
12
13 console.log(utils.queDiaFue(3));

PROBLEMS OUTPUT TERMINAL DEBUG CONSOLE

C:\Users\npm_estudio>nodemon
[nodemon] 2.0.7
[nodemon] to restart at any time, enter `rs`
[nodemon] watching path(s): *.*
[nodemon] watching extensions: js,mjs,json
[nodemon] starting `node index.js`
07145 Lovato Glorieta
7
12
Last Wednesday at 10:58 PM
[nodemon] clean exit - waiting for changes before restart
  
```

¿Qué pasa si quisiéramos cambiar el idioma de **momentjs**? vamos a la documentación, en la sección “Changing locales locally” (<https://momentjs.com/docs/#/i18n/instance-locale/>).

Changing locales locally 1.7.0+

edit

```

// From version 2.8.1 onward
moment().locale(String|String[]|Boolean);

// Deprecated version 2.8.1
moment().lang(String|String[]|Boolean);
  
```

A global locale configuration can be problematic when passing around moments that may need to be formatted into different locale.

```
moment.locale('en'); // default the locale to English
```

Escribimos en nuestro código tal como lo pide la documentación, en la función del archivo **utils.js**.

```

JS utils.js  X  JS index.js
JS utils.js > queDiaFue
1  var moment = require("moment");
2
3
4  function queDiaFue(dias){
5
6      moment.locale('es');
7
8      return moment().subtract(dias, 'days').calendar()
9
10 }
11
12
PROBLEMS  OUTPUT  TERMINAL  DEBUG CONSOLE

C:\Users\npm_estudio>nodemon
[nodemon] 2.0.7
[nodemon] to restart at any time, enter `rs`
[nodemon] watching path(s): *.*
[nodemon] watching extensions: js,mjs,json
[nodemon] starting `node index.js`
6919 Delagarza Parque
7
12
el miércoles pasado a las 23:04
[nodemon] clean exit - waiting for changes before restart

```

Ahora podemos ver cómo cambia el idioma de nuestra librería; es importante mencionar que los mensajes de tipo string solo funcionarán hasta 7 días atrás, luego de eso, la función entregará la fecha en formato DD/MM/YYYY.

Crearemos un nuevo archivo llamado **config**. Éste contendrá un objeto de configuración, el cual tendrá como pares de llave - valor el nombre de un idioma, y su respectiva abreviación para usar en la selección de lenguaje en **moment.js**.

```

VS Code  File  Edit  Selection  View  Go  Run  Terminal  Help
config.js - npm_estudio - V
EXPLORER
NPM_ESTUDIO
> node_modules
JS config.js
JS index.js
package-lock.json
package.json
utils.js

JS config.js > idiomas
1  const idiomas = {
2      ingles: 'en',
3      español: 'es',
4      ruso: 'ru',
5      esperanto: 'eo'
6  }

```

Exportaremos el objeto.

```
JS config.js > ...
1  const idiomas = {
2    ingles: 'en',
3    español: 'es',
4    ruso: 'ru',
5    esperanto: 'eo'
6  }
7
8
9  module.exports = { idiomas: idiomas };
```

Y lo importamos en nuestro archivo **index.js**.

```
JS index.js > ...
1  var moment = require("moment");
2  var config = require('./config');
3
4
```

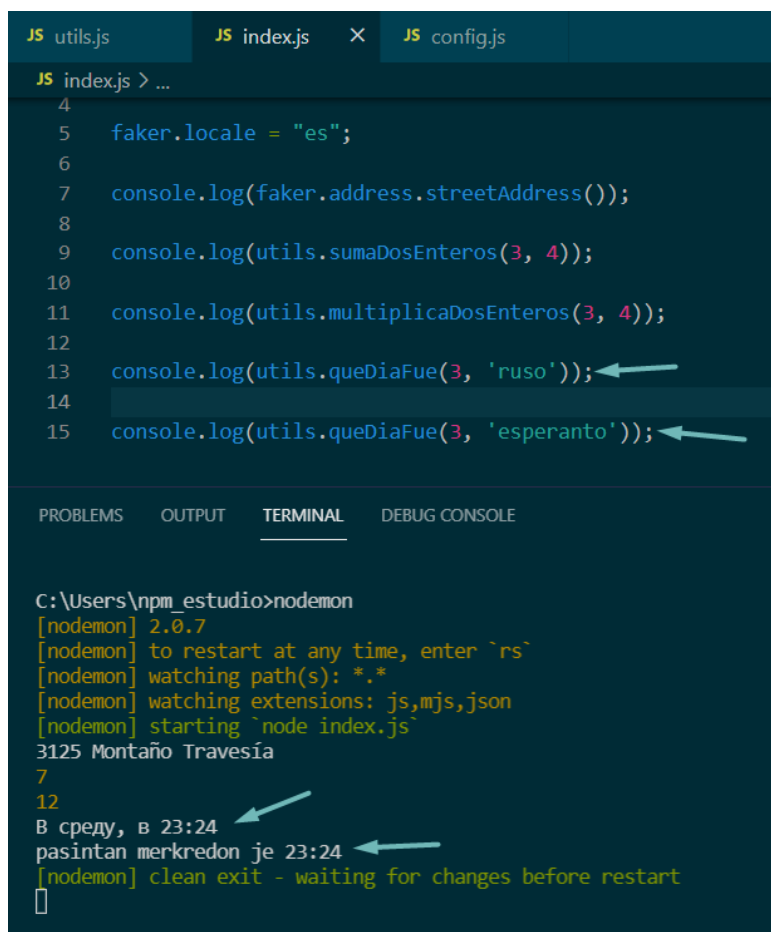
Ahora modificaremos nuestra función en el archivo **utils**, para que ésta pueda recibir como argumento un valor que determinará el idioma en que queremos que nuestra aplicación muestre los resultados.

```
JS utils.js > queDiaFue
1  var moment = require("moment");
2  var config = require('./config');
3
4
5  function queDiaFue(dias, idioma){
6    moment.locale(config.idiomas[idioma]);
7    return moment().subtract(dias, 'days').calendar()
8  }
```

Aquí es importante comentar la línea 7 de nuestro código, para acceder a los valores del objeto, al cual generalmente accederíamos con: **config.idiomas.ruso** - **config.idiomas.ingles**,

entre otros. En este caso, lo estamos haciendo de forma “dinámica”, y para que dicho objeto entienda que el argumento “idioma” no es una llave que pertenece a él, usamos la sintaxis con corchetes: `config.idiomas[idioma]`.

Ahora invocamos a nuestra función en el archivo `index.js`, y le pasamos el idioma que queremos utilizar.



The screenshot shows the VS Code editor with three tabs: `utils.js`, `index.js`, and `config.js`. The `index.js` file is active and contains the following code:

```
4
5  faker.locale = "es";
6
7  console.log(faker.address.streetAddress());
8
9  console.log(utils.sumaDosEnteros(3, 4));
10
11 console.log(utils.multiplicaDosEnteros(3, 4));
12
13 console.log(utils.queDiaFue(3, 'ruso'));
14
15 console.log(utils.queDiaFue(3, 'esperanto'));
```

Two arrows point to the last two lines of code, indicating the dynamic language selection. Below the editor, the terminal shows the output of running `nodemon`:

```
C:\Users\npm_estudio>nodemon
[nodemon] 2.0.7
[nodemon] to restart at any time, enter `rs`
[nodemon] watching path(s): *.*
[nodemon] watching extensions: js,mjs,json
[nodemon] starting `node index.js`
3125 Montaña Travesía
7
12
В среду, в 23:24
pasintan merkredon je 23:24
[nodemon] clean exit - waiting for changes before restart
█
```

Two arrows point to the output lines `В среду, в 23:24` and `pasintan merkredon je 23:24`, which correspond to the `'ruso'` and `'esperanto'` calls in the code above.

Puedes elegir el idioma que quieras desde `index.js`, sin necesidad de modificar directamente el archivo `utils.js`, logrando así un código modular.