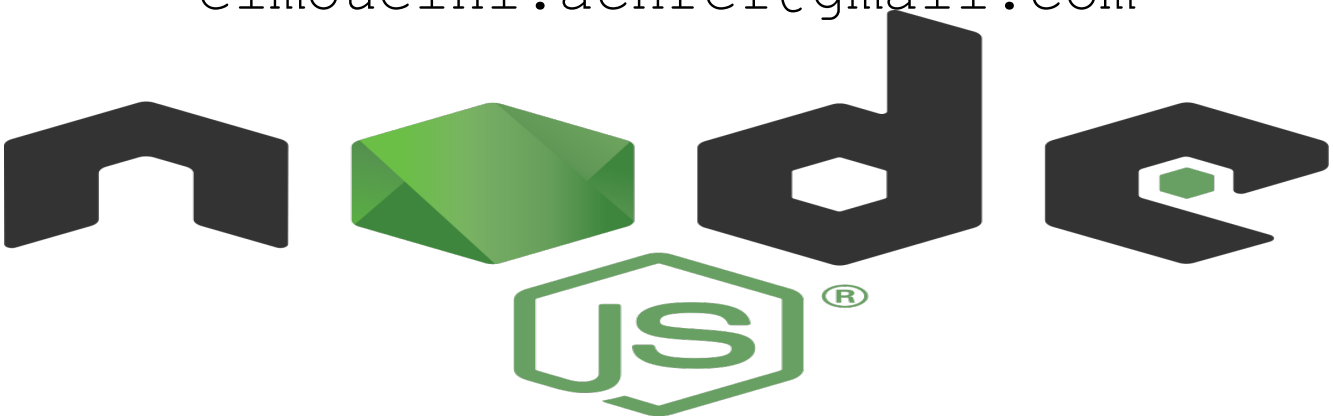
Node.js : modules natifs

Achref El Mouelhi

Docteur de l'université d'Aix-Marseille
Chercheur en programmation par contrainte (IA)
Ingénieur en génie logiciel

elmouelhi.achref@gmail.com



Plan

Plan

Node.js

Trois types de modules

- **Core modules** : définis dans le noyau du **Node.js**. Pour les utiliser, il faut les importer.
- **Community modules** : proposés par la communauté **Node.js**. Pour les utiliser, il faut les installer et ensuite les importer.
- **Modules personnalisés** : pour les utiliser, il faut les exporter puis les importer.

Node.js

Pour importer un `Core module`

```
const module = require('nom-module');
```

Node.js

Pour importer un `Core module`

```
const module = require('nom-module');
```

Pour les distinguer des autres modules, on peut aussi ajouter le préfixe `node`

```
const module = require('node:nom-module');
```

Node.js

Quelques **Core modules**

- **File System**
- **Http**
- **OS**
- **Path**
- ...

Node.js

Module **F**ile **S**ystem (**fs**)

- Module **Node.js** offrant des fonctions, synchrones et asynchrones (`callback` et `promise`) pour gérer les fichiers
  - Création
  - Lecture
  - Écriture
  - Renommage
  - Suppression
  - ...
- Documentation officielle : `https://nodejs.org/api/documentation.html`

Node.js

Pour l'utiliser, il faut l'importer

```
const fs = require('node:fs');
```

Node.js

Pour l'utiliser, il faut l'importer

```
const fs = require('node:fs');
```

Et pour les promesses

```
const fs = require('node:fs/promises');
```

Node.js

Pour la suite, créer les trois fichiers suivants

- `sync.js`
- `async.js`
- `promesse.js`
- `salutation.txt`

Node.js

Pour la suite, créer les trois fichiers suivants

- `sync.js`
- `async.js`
- `promesse.js`
- `salutation.txt`

Dans salutation.txt

```
bonjour
bonsoir
salut
hello
```

Node.js

Pour lire le contenu d'un fichier (à ajouter dans `sync.js`)

```
const fs = require('node:fs');
const content = fs.readFileSync('./salutation.txt');
console.log(content.toString());
console.log('end of file');
```

Node.js

Pour lire le contenu d'un fichier (à ajouter dans `sync.js`)

```
const fs = require('node:fs');
const content = fs.readFileSync('./salutation.txt');
console.log(content.toString());
console.log('end of file');
```

Ou en mode asynchrone (à ajouter dans `async.js`)

```
const fs = require('node:fs');
fs.readFile('./salutation.txt', (err, result) => {
    if (err)
        return console.error(err);
    return console.log(result.toString());
});
console.log('end of file');
```

Node.js

Exécuter les deux fichiers séparément

Avez-vous remarqué une différence ?

Node.js

Une écriture utilisant les promesses

```
const fs = require('node:fs/promises');

fs.readFile("salutation.txt")
    .then(content => console.log(content.toString()))
    .catch(err => console.log("Problème de lecture"))

console.log('end of file');
```

Node.js

Une écriture utilisant les promesses

```
const fs = require('node:fs/promises');

fs.readFile("salutation.txt")
    .then(content => console.log(content.toString()))
    .catch(err => console.log("Problème de lecture"))

console.log('end of file');
```

Ou

```
const fs = require('node:fs/promises');

try {
    let content = await readFile("salutation.txt")
    console.log(content.toString())
} catch (error) {
    console.error("Problème de lecture", error)
}

console.log('end of file');
```

Node.js

Pour écrire dans un fichier (à ajouter dans `sync.js`)

```
const fs = require('node:fs');
fs.writeFileSync('./salutation.txt', 'hello');
```

Node.js

Pour écrire dans un fichier (à ajouter dans `sync.js`)

```
const fs = require('node:fs');
fs.writeFileSync('./salutation.txt', 'hello');
```

Ou en mode asynchrone (à ajouter dans `async.js`)

```
const fs = require('node:fs');

fs.writeFile('salutation.txt', 'bonjour', (err) => {
  if (err)
        throw err;
  console.log('écriture effectuée avec succès');
});
```

Node.js

Remarques

- `writeFileSync` et `writeFile` créent le fichier s'il n'existe pas.
- Si le fichier existe son contenu précédent sera remplacé par le nouveau.

Node.js

Pour écrire à la suite dans un fichier (à ajouter dans `sync.js`)

```
const fs = require('node:fs');
fs.appendFileSync('./salutation.txt', 'other');
```

Node.js

Pour écrire à la suite dans un fichier (à ajouter dans `sync.js`)

```
const fs = require('node:fs');
fs.appendFileSync('./salutation.txt', 'other');
```

Ou en mode asynchrone (à ajouter dans `async.js`)

```
const fs = require('node:fs');

fs.appendFile('salutation.txt', 'autre', (err) => {
  if (err)
  throw err;
  console.log('écriture effectuée avec succès');
});
```

Node.js

Remarques

- `appendFileSync` et `appendFile` créent le fichier s'il n'existe pas.
- Si le fichier existe, le nouveau contenu sera ajouté à la suite du précédent.

Node.js

Exercice

- Écrire un programme **Node.js** qui permet de copier le contenu du fichier `salutation.txt` dans un nouveau fichier `salutation-copy.txt`
- Proposer deux versions : une première en utilisant les fonctions synchrones et une deuxième en utilisant les fonctions asynchrones (`callback` ou `promise`)

Node.js

Correction

```
var fs = require("node:fs");

fs.readFile("salutation.txt", "utf8", (err, result) => {
  if (err) return console.error(err);

  fs.writeFile("salutation-copy.txt", result, (err) => {
    if (err) return console.error(err);
    console.log("C'est fait");
  });
});
```

Node.js

Pour renommer un fichier (à ajouter dans `sync.js`)

```
const fs = require('node:fs');
fs.renameSync('./salutation.txt', 'salutation-fr.txt');
```

Node.js

Pour renommer un fichier (à ajouter dans `sync.js`)

```
const fs = require('node:fs');
fs.renameSync('./salutation.txt', 'salutation-fr.txt');
```

Ou en mode asynchrone (à ajouter dans `async.js`)

```
const fs = require('node:fs');

fs.rename('salutation-fr.txt', 'salutation.txt', (err) => {
    if (err) throw err;
    console.log('Fichier renommé avec succès');
});
```

Node.js

Remarques

- Si le fichier à renommer n'existe pas, une exception sera levée.
- `renameSync` et `rename` permettent de renommer les fichiers et les répertoires.
- `renameSync` et `rename` permettent également de déplacer les fichiers et les répertoires.

Node.js

Pour copier un fichier (à ajouter dans `sync.js`)

```
const fs = require('node:fs');
fs.copyFileSync('salutation.txt', 'salutation-fr.txt');
```

Node.js

Pour copier un fichier (à ajouter dans `sync.js`)

```
const fs = require('node:fs');
fs.copyFileSync('salutation.txt', 'salutation-fr.txt');
```

Ou en mode asynchrone (à ajouter dans `async.js`)

```
const fs = require('node:fs');

fs.copyFile('salutation-fr.txt', 'salutation.txt', (err) => {
    if (err) throw err;
    console.log('Fichier copié avec succès');
});
```

Node.js

Remarques

- Si le fichier à copier n'existe pas, une exception sera levée.
- `copyFileSync` et `copyFile` copient le fichier sans le supprimer (elles ne le déplacent pas).

Node.js

Pour supprimer un fichier (à ajouter dans `sync.js`)

```
const fs = require('node:fs');
fs.unlinkSync('salutation.txt');
```

Node.js

Pour supprimer un fichier (à ajouter dans `sync.js`)

```
const fs = require('node:fs');
fs.unlinkSync('salutation.txt');
```

Ou en mode asynchrone (à ajouter dans `async.js`)

```
const fs = require('node:fs');

fs.unlink('salutation.txt', (err) => {
  if (err) throw err;
  console.log('Fichier supprimé');
});
```

Node.js

Remarques

- Si le fichier à supprimer n'existe pas, une exception sera levée.
- `unlinkSync` et `unlink` permettent de supprimer les fichiers mais pas les répertoires.

Node.js

Pour créer un répertoire (à ajouter dans `sync.js`)

```
const fs = require('node:fs');
fs.mkdirSync('a');
```

Node.js

Pour créer un répertoire (à ajouter dans `sync.js`)

```
const fs = require('node:fs');
fs.mkdirSync('a');
```

Ou en mode asynchrone (à ajouter dans `async.js`)

```
const fs = require('node:fs');

fs.mkdir('b', (err) => {
    if (err) throw err;
    console.log('Répertoire créé avec succès');
});
```

Node.js

Pour créer récursivement une arborescence de répertoires (à ajouter dans `sync.js`)

```
const fs = require('fs');
fs.mkdirSync('a/b/c', { recursive: true });
```

Node.js

Pour créer récursivement une arborescence de répertoires (à ajouter dans `sync.js`)

```
const fs = require('fs');
fs.mkdirSync('a/b/c', { recursive: true });
```

Ou en mode asynchrone (à ajouter dans `async.js`)

```
const fs = require('node:fs');

fs.mkdir('x/y/z', { recursive: true }, (err) => {
    if (err) throw err;
    console.log('Répertoires créés avec succès');
});
```

Node.js

Pour supprimer le répertoire vide `c` (à ajouter dans `sync.js`)

```
const fs = require('node:fs');
fs.rmdirSync('a/b/c');
```

Node.js

Pour supprimer le répertoire vide `c` (à ajouter dans `sync.js`)

```
const fs = require('node:fs');
fs.rmdirSync('a/b/c');
```

Ou en mode asynchrone (à ajouter dans `async.js`)

```
const fs = require('node:fs');

fs.rmdir('x/y/z', (err) => {
    if (err) throw err;
    console.log('Répertoire supprimé avec succès');
});
```

Node.js

`rmdirSync` **et** `rmdir` **ne permettent pas de supprimer un dossier non vide**

```
const fs = require('node:fs');
fs.rmdirSync('a');
```

Node.js

rmdirSync et rmdir ne permettent pas de supprimer un dossier non vide

```
const fs = require('node:fs');
fs.rmdirSync('a');
```


Remarque

L'instruction précédente génère l'erreur suivante :

```
Error: ENOTEMPTY: directory not empty, rmdir 'a'
```

Node.js

Pour supprimer un fichier ou un dossier vide ou non-vide (à ajouter dans `sync.js`)

```
const fs = require('node:fs');
fs.rmSync('a', { recursive: true, force: true });
```

Node.js

Pour supprimer un fichier ou un dossier vide ou non-vide (à ajouter dans `sync.js`)

```
const fs = require('node:fs');
fs.rmSync('a', { recursive: true, force: true });
```

Ou en mode asynchrone (à ajouter dans `async.js`)

```
const fs = require('node:fs');

fs.rm('x', { recursive: true, force: true }, (err, res) => {
    if (err) console.log("Problème de suppression");
    if (res) console.log('Suppression effectué avec succès');
});
```

Node.js

Pour vérifier l'existence d'un répertoire ou d'un fichier (à ajouter dans `sync.js`)

```
const fs = require('node:fs');

if (fs.existsSync('x/y')) {
    console.log("existe");
} else {
    console.log("n'existe pas");
}
```

Node.js

Pour vérifier l'existence d'un répertoire ou d'un fichier (à ajouter dans `sync.js`)

```
const fs = require('node:fs');

if (fs.existsSync('x/y')) {
    console.log("existe");
} else {
    console.log("n'existe pas");
}
```


Ou (à ajouter dans `sync.js`)

```
try {
    fs.statSync('x/y')
    console.log("existe");
} catch (e) {
    console.log("n'existe pas");
}
```

Node.js

Ou en mode asynchrone (à ajouter dans `async.js`)

```
const fs = require('node:fs');

fs.stat('x/y', (err, res) => {
    if (err) console.log("chemin inexistant");
    if (res) console.log('Chemin existant');
});
```

Node.js

Ou en mode asynchrone (à ajouter dans `async.js`)

```
const fs = require('node:fs');

fs.stat('x/y', (err, res) => {
    if (err) console.log("chemin inexistant");
    if (res) console.log('Chemin existant');
});
```

Remarques

- `existsSync`, `statSync` et `stat` vérifie les répertoires et les fichiers.
- `existsSync` retourne un booléen tandis que `statSync` et `stat` retournent les stats du paramètre.
- `exists` est dépréciée.

Pour tester s'il s'agit d'un fichier ou un répertoire (à ajouter dans `sync.js`)

```
const fs = require('node:fs');

if (!fs.statSync('x/y')) {
    console.log("n'existe pas");
} else {
    console.log(fs.statSync('x/y').isFile());
    console.log(fs.statSync('x/y').isDirectory());
}
```

Pour tester s'il s'agit d'un fichier ou un répertoire (à ajouter dans `sync.js`)

```
const fs = require('node:fs');

if (!fs.statSync('x/y')) {
    console.log("n'existe pas");
} else {
    console.log(fs.statSync('x/y').isFile());
    console.log(fs.statSync('x/y').isDirectory());
}
```

Ou en mode asynchrone (à ajouter dans `async.js`)

```
const fs = require('node:fs');

fs.stat('x/y', (err, res) => {
    if (err) {
        console.log("Le chemin n'existe pas");
    } else {
        console.log("Le chemin existe")
        console.log(res);
        console.log(res.isFile());
        console.log(res.isDirectory());
    }
});
```

Node.js

`fstat` **est similaire à** `stat` **mais elle n'accepte comme paramètre qu'un file descriptor** `sync.js`**)**

```
const fs = require('node:fs');

const elt = fs.openSync('main.js')
if (fs.fstatSync(elt)) {
    console.log("existe");
} else {
    console.log("n'existe pas");
}
```

Node.js

Pour lire le contenu d'un répertoire (à ajouter dans `sync.js`)

```
const fs = require('node:fs');
const contents = fs.readdirSync("./dossier");
for (const content of contents) {
    console.log(content);
}
```

Node.js

Pour lire le contenu d'un répertoire (à ajouter dans `sync.js`)

```
const fs = require('node:fs');
const contents = fs.readdirSync("./dossier");
for (const content of contents) {
    console.log(content);
}
```

Ou en mode asynchrone (à ajouter dans `async.js`)

```
const fs = require('node:fs');

fs.readdir('./dossier', (err, contents) => {
    if (err) console.log("Erreur de lecture");
    for (const content of contents) {
        console.log(content);
    }
});
```

Node.js

Considérons le tableau suivant

```
const files = ["file1.txt", "file2.txt", "file3.txt"];
```

Node.js

Considérons le tableau suivant

```
const files = ["file1.txt", "file2.txt", "file3.txt"];
```


Exercice

- Écrire un programme **Node.js** qui permet de créer un répertoire `monDossier` et trois fichiers `file1.txt`, `file2.txt` et `file3.txt` (dans `monDossier`)
- Proposer deux versions : une utilisant les fonctions synchrones et une utilisant les fonctions asynchrones

Node.js

Solution synchrone

```
const fs = require("fs");
const files = ["file1.txt", "file2.txt", "file3.txt"];

if (fs.existsSync('monDossier')) {
    console.error('dossier existe déjà');
} else {
    fs.mkdirSync('monDossier');
}
for (const file of files) {
    fs.writeFileSync(`monDossier/${file}`, "contenu");
}
```

Node.js

Solution asynchrone

```
const fs = require("fs");
const files = ["file1.txt", "file2.txt", "file3.txt"];

fs.stat('monDossier', (err) => {
    if (!err) console.error('dossier existe déjà', err)
    else {
        fs.mkdir('monDossier', (err) => {
            if (err) console.error("problème de création de dossier")
            else {
                console.log(`dossier créé`)
                for (const file of files) {
                    fs.writeFile(`monDossier/${file}`, "contenu", (err) => {
                        if (err) console.error(`problème de création de fichier ${file}`)
                        else console.log(`fichier ${file} créé`)
                    });
                }
            }
        });
    }
});
```

Node.js

Exercice

- Écrire un programme **Node.js** qui permet de générer un rapport illustrant le contenu d'un répertoire appelé `monDossier` : nom du fichier/dossier + type (fichier ou dossier)
- Proposer deux versions : une utilisant les fonctions synchrones et une utilisant les fonctions asynchrones

Node.js

Solution

```
let fs = require('fs');
let rapport = "rapport.txt";
let dossier = 'monDossier';
let arbo = fs.readdirSync(dossier);

for (const elt of arbo) {
    const chemin = `./${dossier}/${elt}`
    const fileOrDir = `${fs.statSync(chemin).isDirectory() ? 'dossier' : 'fichier'}`
    fs.appendFileSync(rapport, `${elt} : ${fileOrDir} \n`);
}
```

Node.js

Pour récupérer le nom du fichier courant

```
console.log(__filename)
```

Node.js

Pour récupérer le nom du fichier courant

```
console.log(__filename)
```

Pour récupérer le nom du répertoire courant

```
console.log(__dirname)
```

Node.js

Comparatif : Sync vs Callback vs Promise vs Async/Await

Critère	**Sync**	**Callback**	**Promise**	**Async/Await**
Type	Bloquant	Non bloquant	Non bloquant	Non bloquant
Syntaxe	Synchrone	Callbacks imbriqués	`.then()` et `.catch()`	`async/await`
Gestion d'erreurs	`try/catch`	`(err, result)`	`.catch()`	`try/catch`
Lisibilité du code	Excellente	Mauvaise si imbriqué	Moyenne	Excellente
Recommandé pour	Scripts simples	Ancien style, à éviter	Code moderne	Code moderne

Node.js

Module **Path**

Module **Node.js** offrant des fonctions synchrones pour gérer les chemins

Node.js

Module **Path**

Module **Node.js** offrant des fonctions synchrones pour gérer les chemins

Pour l'utiliser, il faut l'importer

```
const path = require('node:path');
```

Node.js

Pour récupérer le nom du dossier parent

```
console.log(path.dirname('monDossier/file1.txt'));
// affiche monDossier
```

Node.js

Pour récupérer le nom du dossier parent

```
console.log(path.dirname('monDossier/file1.txt'));
// affiche monDossier
```

Pour récupérer la dernière partie d'un chemin

```
console.log(path.basename('monDossier/file1.txt'));
// affiche file1.txt
```

Node.js

Pour récupérer le nom du dossier parent

```
console.log(path.dirname('monDossier/file1.txt'));
// affiche monDossier
```

Pour récupérer la dernière partie d'un chemin

```
console.log(path.basename('monDossier/file1.txt'));
// affiche file1.txt
```

Pour récupérer l'extension d'un fichier

```
console.log(path.extname('monDossier/file1.txt'));
// affiche .txt
```

Node.js

Pour déterminer si un chemin est absolu

```
console.log(path.isAbsolute('monDossier/file1.txt'));
// affiche false
```

Node.js

Pour déterminer si un chemin est absolu

```
console.log(path.isAbsolute('monDossier/file1.txt'));
// affiche false
```

Pour construire le chemin absolu vers un paramètre

```
console.log(path.resolve('monDossier/file1.txt'));
// affiche chemin absolu vers file1.txt
```

Node.js

Pour construire un chemin relatif à partir de plusieurs paramètres

```
console.log(path.join( 'public', 'index.html'));
// affiche public\index.html
```

Node.js

Pour construire un chemin relatif à partir de plusieurs paramètres

```
console.log(path.join( 'public', 'index.html'));
// affiche public\index.html
```

Pour construire un chemin absolu à partir de plusieurs paramètres

```
console.log(path.resolve( 'public', 'index.html'));
// affiche C:\Users\Admin\Desktop\cours-node\public\index.html
```

Node.js

Pour construire un chemin relatif à partir de plusieurs paramètres

```
console.log(path.join( 'public', 'index.html'));
// affiche public\index.html
```

Pour construire un chemin absolu à partir de plusieurs paramètres

```
console.log(path.resolve( 'public', 'index.html'));
// affiche C:\Users\Admin\Desktop\cours-node\public\index.html
```


Ces méthodes ne permettent pas la construction physique d'un chemin.

Node.js

Objet **process**

Objet global **Node.js** offrant des propriétés et des fonctions sur le processus

Node.js

Objet **process**

Objet global **Node.js** offrant des propriétés et des fonctions sur le processus

Pour récupérer le PID du processus

```
console.log(process.pid)
```

Node.js

Objet **process**

Objet global **Node.js** offrant des propriétés et des fonctions sur le processus

Pour récupérer le PID du processus

```
console.log(process.pid)
```

Pour récupérer la version utilisée de Node.js

```
console.log(process.version)
```

Node.js

process.argv : tableau contenant

- process.argv[0] : la commande (node)
- process.argv[1] : chemin absolu vers le fichier exécuté
- process.argv[2] : le premier argument s'il y en a
- ...

Node.js

process.argv : tableau contenant

- process.argv[0] : la commande (node)
- process.argv[1] : chemin absolu vers le fichier exécuté
- process.argv[2] : le premier argument s'il y en a
- ...

Pour tester

```
console.log(process.argv[0])
console.log(process.argv[1])
```

Node.js

Qu'affiche le programme suivant ?

```
process.on("beforeExit", () => {
    console.log("beforeExit");
});

console.log("Begin");

setTimeout(() => {
    console.log("the asynchronous timeout");
    console.log("End");
}, 2000);
```

Node.js

Qu'affiche le programme suivant ?

```
process.on("beforeExit", () => {
    console.log("beforeExit");
});

console.log("Begin");

setTimeout(() => {
    console.log("the asynchronous timeout");
    console.log("End");
}, 2000);
```


Résultat

```
Begin
the asynchronous timeout
End
beforeExit
```

Node.js

Qu'affiche le programme suivant ?

```
process.on("beforeExit", () => {
    console.log("Last call before end");
});

process.on("exit", (code) => {
    console.log("The end of the program :", code);
});

console.log("Begin");
setTimeout(() => {
    console.log("the asynchronous timeout");
    console.log("End");
}, 2000);
```

Node.js

Qu'affiche le programme suivant ? pourquoi ?

```
process.on('uncaughtException', (err) => {
    console.log('Exception captée:  ' + err);
});


const obj = { a: 5, b: 3 };
console.log(obj.c.d);
```

Node.js

Module **Readline**

Le module **readline** de **Node.js** fournit une interface permettant de lire des données depuis un flux (stream), typiquement le clavier.

Node.js

Module **Readline**

Le module **readline** de **Node.js** fournit une interface permettant de lire des données depuis un flux (stream), typiquement le clavier.

Étape 1 : importation du module

```
const readline = require("node:readline");
```

Node.js

Module **Readline**

Le module **readline** de **Node.js** fournit une interface permettant de lire des données depuis un flux (stream), typiquement le clavier.

Étape 1 : importation du module

```
const readline = require("node:readline");
```

Étape 2 : création de l'interface de lecture

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

Node.js

Lire une donnée saisie par l'utilisateur, puis fermer l'interface :

```
rl.question("Votre nom : ", (nom) => {
    console.log(`Bonjour ${nom}`);
    rl.close();
});
```

Node.js

Lire une donnée saisie par l'utilisateur, puis fermer l'interface :

```
rl.question("Votre nom : ", (nom) => {
    console.log(`Bonjour ${nom}`);
    rl.close();
});
```

Remarques

- Sans appel à `rl.close()`, le programme reste en attente dans le terminal.
- Une fois fermée, l'interface `rl` ne peut plus être utilisée.

Node.js

Pour exécuter une fonction à la fermeture de la console

```
r1.on("close", () => {
    console.log("Fin!");
});
```

Node.js

Pour exécuter une fonction à la fermeture de la console

```
r1.on("close", () => {
    console.log("Fin!");
});
```


Remarque

Il existe une version synchrone et une version avec `promise` de `question`.

Node.js

Exercice

En utilisant la version promesse de `readline`, écrire un script **Node.js** qui permet demande à l'utilisateur de saisir son nom puis son prénom et qui les affiche dans un message de bienvenue.

Node.js

Module **O**perating **S**ystem : **os**

Module **Node.js** offrant des fonctions synchrones permettant de fournir de données sur le système d'exploitation

- nombre de CPU
- hostname
- ...

Node.js

Module **O**perating **S**ystem : **os**

Module **Node.js** offrant des fonctions synchrones permettant de fournir de données sur le système d'exploitation

- nombre de CPU
- hostname
- ...

Pour l'utiliser, il faut l'importer

```
const os = require('node:os');
```

Node.js

Exemple

```
console.log("Mémoire", os.totalmem());
console.log("Nombre CPU", os.cpus().length);
console.log("Hostname", os.hostname());
console.log("Utilisateur", os.userInfo().username);
```

Node.js

Pour utiliser le module `colors` , il faut l'installer

```
npm install colors
```

Node.js

Pour utiliser le module `colors` , il faut l'installer

```
npm install colors
```

Et ensuite l'importation

```
require('colors');
```

Node.js

Pour modifier la couleur d'affichage

```
console.log('Attention!'.red);
console.log("c'est correct!".green.underline);
console.log('celui-ci aussi'.bgGreen.white.bold);
console.log('lettres en couleurs'.rainbow);
console.log('drapeau américain et français'.america);
console.log("c'est quoi ce truc".trap);
```

Node.js

Pour modifier le style

```
console.log('texte souligné'.underline);
console.log('texte italic'.italic);
console.log('texte gras'.bold);
console.log('texte à couleurs inversées'.inverse);
```

Node.js

Module Events

- Module natif de Node.js pour gérer des événements asynchrones.
- Implémente le **patron d'observateur** (EventEmitter).
- Permet de définir, émettre et écouter des événements personnalisés.

Node.js

Module Events

- Module natif de Node.js pour gérer des événements asynchrones.
- Implémente le **patron d'observateur** (EventEmitter).
- Permet de définir, émettre et écouter des événements personnalisés.

Pour importer le module

```
const EventEmitter = require('events');
```

Node.js

Pour créer une instance d'EventEmitter

```
const myEmitter = new EventEmitter();
```

Node.js

Pour créer une instance d'EventEmitter

```
const myEmitter = new EventEmitter();
```

Pour écouter un événement avec `.on()`

```
myEmitter.on('saluer', (nom) => {
  console.log(`Bonjour, ${nom} !`);
});
```

Node.js

Pour créer une instance d'EventEmitter

```
const myEmitter = new EventEmitter();
```

Pour écouter un événement avec `.on()`

```
myEmitter.on('saluer', (nom) => {
  console.log(`Bonjour, ${nom} !`);
});
```


Pour émettre un événement avec `.emit()`

```
myEmitter.emit('saluer', 'Wick');
```

Node.js

Autres méthodes utiles

- `.on(event, listener)` : associe un écouteur.
- `.emit(event, ...args)` : déclenche un événement.
- `.once(event, listener)` : écoute une seule fois.
- `.removeListener(event, listener)` : supprime un écouteur.
- ...

Node.js

Module **http**

Module **Node.js** permettant la création d'un serveur **HTTP** et donc recevoir des requêtes et retourner des réponses

Node.js

Module **http**

Module **Node.js** permettant la création d'un serveur **HTTP** et donc recevoir des requêtes et retourner des réponses

Première étape : importer le package `http`

```
const http  = require('node:http');
```

Node.js

Deuxième étape : utiliser le module `http` pour créer un serveur : la fonction `createServer` prend en paramètre une fonction qui sera exécutée à chaque fois qu'on appelle le serveur

```
const server = http.createServer((request, response) => {
    response.write('Hello World!');
    response.end();
});
```

Node.js

Deuxième étape : utiliser le module `http` pour créer un serveur : la fonction `createServer` prend en paramètre une fonction qui sera exécutée à chaque fois qu'on appelle le serveur

```
const server = http.createServer((request, response) => {
    response.write('Hello World!');
    response.end();
});
```


On peut aussi fusionner `write` et `end`

```
const server = http.createServer((request, response) => {
    response.end('Hello World!');
});
```

Node.js

On peut aussi se contenter d'importer la fonction `createServer`

```
const { createServer } = require('http');

const server = createServer((request, response) => {
    response.end('Hello World!');
});
```

Node.js

On peut aussi se contenter d'importer la fonction `createServer`

```
const { createServer } = require('http');

const server = createServer((request, response) => {
    response.end('Hello World!');
});
```

On peut également indiquer le type de la réponse et le code serveur

```
const { createServer } = require('http');

const server = createServer((request, response) => {
    response.writeHead(200, { 'Content-Type': 'text/plain' });
    response.end('Hello World!');
});
```

Node.js

Troisième étape : lancer le serveur pour écouter les requêtes sur le port 3000

```
server.listen(3000);
```

Node.js

Troisième étape : lancer le serveur pour écouter les requêtes sur le port 3000

```
server.listen(3000);
```

On peut aussi ajouter une fonction à exécuter pendant que le serveur est à l'écoute

```
server.listen(3000, () =>
    console.log('Adresse du serveur : http://localhost:3000')
);
```

Node.js

Code final

```
const { createServer } = require('http');

const server = createServer((request, response) => {
    response.writeHead(200, { 'Content-Type': 'text/plain' });
    response.end('Hello World!');
});

server.listen(3000, () =>
    console.log('Adresse du serveur : http://localhost:3000')
);
```

Node.js

Code final

```
const { createServer } = require('http');

const server = createServer((request, response) => {
    response.writeHead(200, { 'Content-Type': 'text/plain' });
    response.end('Hello World!');
});

server.listen(3000, () =>
    console.log('Adresse du serveur : http://localhost:3000')
);
```

Pour tester depuis un navigateur, aller à l'URL `http://localhost:3000`

Node.js

On peut aussi tester en utilisant la méthode `request` de `http` (code à placer dans `client.js`)

```
const http = require('http');

const options = {
    host: 'localhost',
    port: 3000,
    path: '',
    method: 'GET'
};

http
    .request(options, (response) => {
        console.log(`STATUS: ${response.statusCode}`);
    })
    .on("error", (err) => {
        console.log("Erreur : ", err)
    })
    .end();
```

Node.js

Pour tester

- Ouvrez un terminal et lancez le premier programme
- Ouvrez un deuxième terminal et lancez le deuxième programme et vérifiez l'affichage suivant : `STATUS: 200`

Node.js

Module **Dotenv**

- Module sans dépendance
- Permettant de charger les variables d'environnement d'un fichier `.env` dans `process.env`

Node.js

Module **Dotenv**

- Module sans dépendance
- Permettant de charger les variables d'environnement d'un fichier `.env` dans `process.env`

Pour utiliser un fichier de configuration (`.env`), il faut installer `dotenv`

```
npm install dotenv
```

Node.js

Et ensuite l'importer

```
require('dotenv').config();
```

Node.js

Et ensuite l'importer

```
require('dotenv').config();
```

Ou

```
import 'dotenv/config'
```

Node.js

À la racine du projet, créons un fichier `.env` avec le contenu suivant (fichier souvent placé dans `.gitignore`)

```
PORT=5555
NODE_ENV=development
```

Node.js

À la racine du projet, créons un fichier `.env` avec le contenu suivant (fichier souvent placé dans `.gitignore`)

```
PORT=5555
NODE_ENV=development
```

Chargeons dynamiquement le port

```
const port = process.env.PORT || 3000;
server.listen(port, () =>
    console.log(`Adresse du serveur : http://localhost:${port}`)
);
```

Node.js

À la racine du projet, créons un fichier `.env` avec le contenu suivant (fichier souvent placé dans `.gitignore`)

```
PORT=5555
NODE_ENV=development
```

Chargeons dynamiquement le port

```
const port = process.env.PORT || 3000;
server.listen(port, () =>
    console.log(`Adresse du serveur : http://localhost:${port}`)
);
```


Pour tester

Relance l'application et vérifiez que `Adresse du serveur : http://localhost:5555` s'affiche dans la console

Node.js

Module **url**

Module **Node.js** permettant de récupérer les différents fragments d'une **URL**

Node.js

Module **url**

Module **Node.js** permettant de récupérer les différents fragments d'une **URL**

Première étape : importer le package `http`

```
const url  = require('node:url');
```

Node.js

En supposant que l'URL demandée est : `http://localhost:3000/home?nom=doe&prenom=john`

```
const { createServer } = require('node:http');
const url  = require('node:url');

const server = createServer((request, response) => {
    response.writeHead(200, { 'Content-Type': 'text/plain' });
    const requestUrl = url.parse(request.url, true);

    console.log(requestUrl.href);
    // retourne /home?nom=doe&prenom=john

    console.log(requestUrl.pathname);
    // retourne /home

    console.log(requestUrl.search);
    // retourne ?nom=doe&prenom=john

    console.log(requestUrl.query);
    // retourne un objet : { nom: 'doe', prenom: 'john' }

    response.end('Hello World!');
});

server.listen(3000, () => {
    console.log('Adresse du serveur : http://localhost:3000')
});
```

Node.js

Pour récupérer les paramètres un par un

```
const { createServer } = require('node:http');
const url = require('node:url');

const server = createServer((request, response) => {
    response.writeHead(200, { 'Content-Type': 'text/plain' });
    const requestUrl = url.parse(request.url, true);
    const { nom, prenom } = requestUrl.query
    console.log(`Bonjour ${nom} ${prenom}`);
    response.end('Hello World!');
});

server.listen(3000, () => {
    console.log('Adresse du serveur : http://localhost:3000')

});
```

Node.js

Exercice

- Écrire un programme qui affiche le résultat d'une opération arithmétique des nombres passés en paramètre
- Si l'adresse saisie dans la barre d'adresse contient `/calcul?op=plus&value1=2&value2=5`, la réponse attendue est `2 + 5 = 7`
- Les valeurs possibles de `op` sont `plus`, `moins`, `fois` et `div`

Node.js

Exercice

Écrire un programme qui génère dans un fichier `calcul.txt` la table de multiplication, addition, soustraction ou division d'un nombre positif passé en paramètre

Node.js

nodemon

- Package **Node.js**
- Il surveille les modifications de fichiers sources
- Et il redémarre automatiquement le serveur de l'application
- Repository **GitHub** : `https://github.com/remy/nodemon`
- Documentation : `https://www.npmjs.com/package/nodemon`

Node.js

Installation

```
npm install -g nodemon
```

Node.js

Installation

```
npm install -g nodemon
```

Utilisation

```
nodemon [répertoire ou fichier]
```