

Introduction to Node.js

By Matthew Honour



What exactly is Node.js?

- Node.js is a runtime environment for JavaScript.
- Node.js is not a language.

What is a Runtime Environment?

- A runtime environment is simply an environment that a programming language can be executed in.
- Until 2009 the only runtime environment available for JavaScript was in the browser.



How do browsers run JavaScript?

- Browsers run JS using their respective JS Engine.
- Some example include:
 - Chrome: V8 Engine
 - Firefox: SpiderMonkey
 - Microsoft Edge: Chakra



How does Node.js run JavaScript?

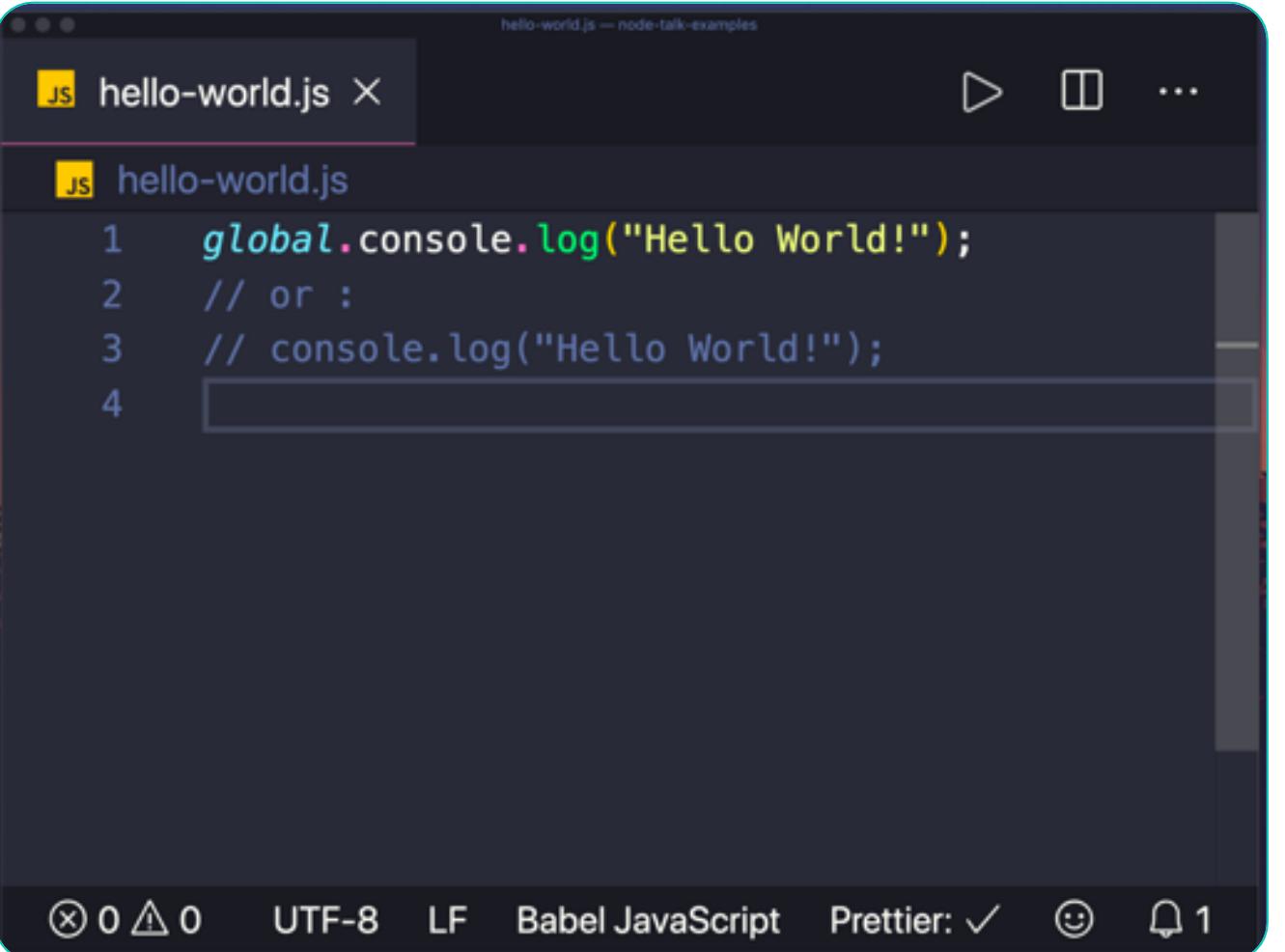
Node uses Chrome's 'V8 JavaScript Engine' to execute JavaScript Code.



What else does Node.js do?

- Node.js adds several useful libraries that developers can use to write server-side applications. Some of these libraries include:
 - Path – working with file paths
 - Os – working with the Operating System
 - Http – used to make web servers and transfer data over http.
 - Fs – working with the file system of the machine.

Let's start with a classic Hello World!



A screenshot of a code editor window titled "hello-world.js" in a dark-themed interface. The code editor displays two ways to log "Hello World!" to the console:

```
1 global.console.log("Hello World!");
2 // or :
3 // console.log("Hello World!");
4
```

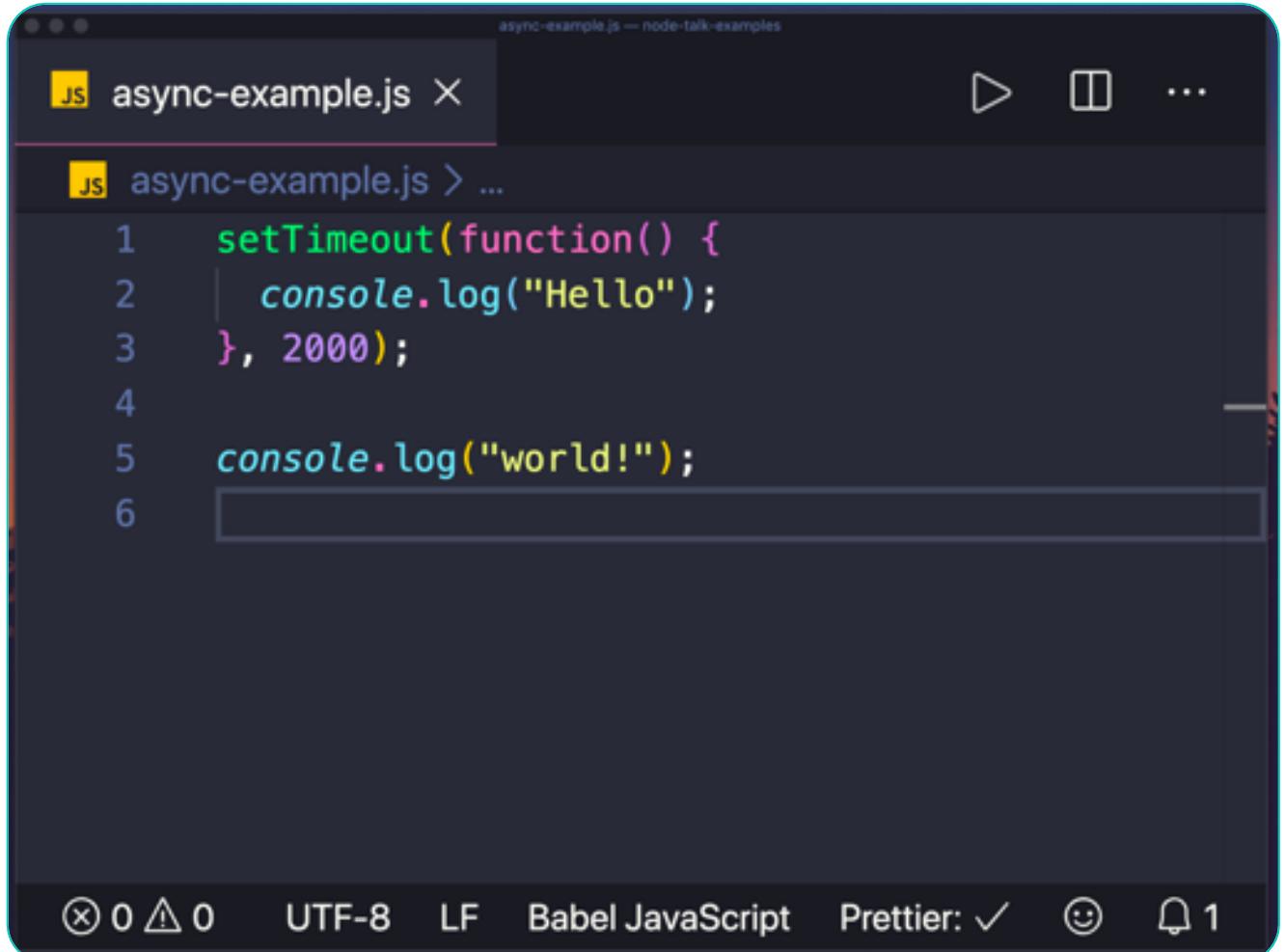
The status bar at the bottom of the editor shows the following information: 0 errors, 0 warnings, UTF-8 encoding, LF line endings, Babel JavaScript, Prettier: ✓, a smiley face icon, and a notification bell with the number 1.

Asynchronous by Default

- All JavaScript code executed using Node.js is asynchronous by default.
- This means that Node.js never waits for a line of code to be executed...
- ... Instead, it proceeds on to the next line of code while the previous line is being executed in the background.
- This makes Node.js perfect for programs that contain a lot of I/O operations (e.g. reading and writing to a database).



Async Example



A screenshot of a code editor window titled "async-example.js". The code editor has a dark theme. The file content is as follows:

```
async-example.js — node-talk-examples
JS async-example.js ×
JS async-example.js > ...
1 setTimeout(function() {
2   console.log("Hello");
3 }, 2000);
4
5 console.log("world!");
6
```

The status bar at the bottom of the editor shows the following information from left to right: 0 errors, 0 warnings, UTF-8 encoding, LF line endings, Babel JavaScript, Prettier checked, a smiley face icon, and a notification bell with the number 1.

Modules in Node.js

- In Node.js code is organised into modules.
- Each JS file within a node project is considered a module and it is the way that Node.js deals with organising code.
- There are three types of module in Node.js...

app.js

```
creating-your-own-modules > app.js > ...
1 const g = require("./greetingsGenerator");
2
3 // console.log(g.getHappyGreeting());
4 console.log(g.getMeanGreeting());
5
```

⑧ 0 ▲ 0 UTF-8 LF Babel JavaScript Prettier: ✓ 😊 ⏱ 1

greetingsGenerator.js

```
creating-your-own-modules > greetingsGenerator.js > ...
1 const happyGreeting = "Hello, you are a nice person :)";
2 const meanGreeting = "Hello, you are a meany >:)";
3
4 module.exports.getHappyGreeting = function() {
5   return happyGreeting;
6 };
7
8 module.exports.getMeanGreeting = function() {
9   return meanGreeting;
10};
11
```

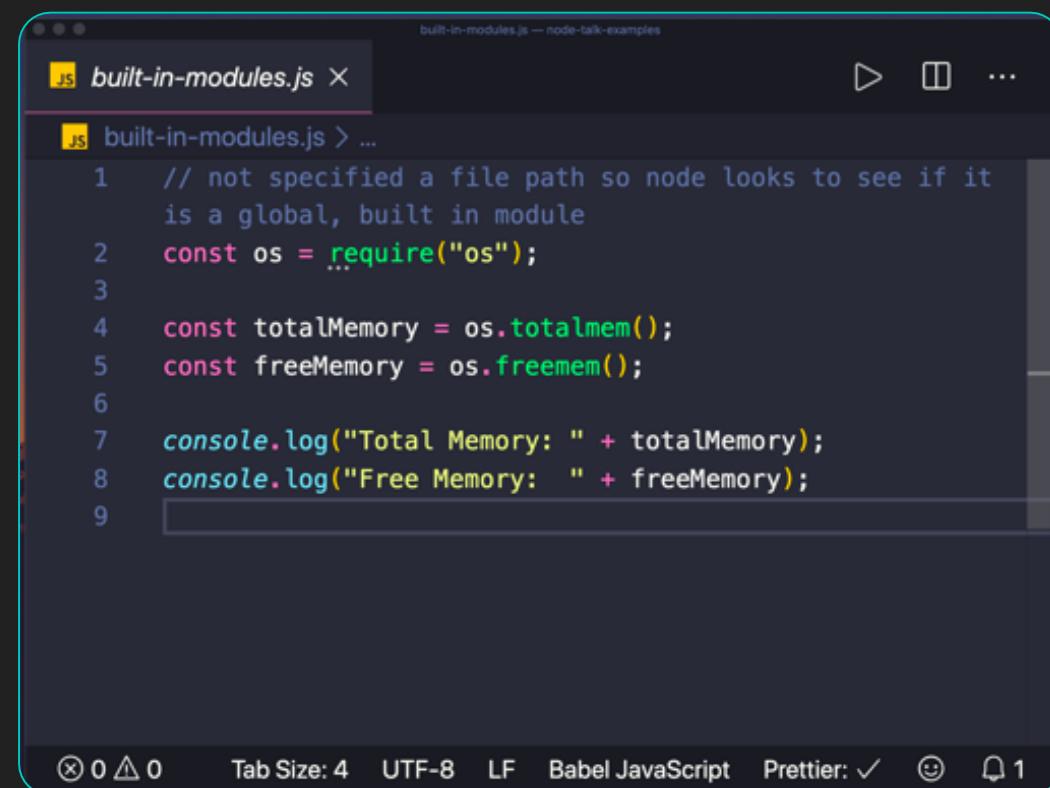
⑧ 0 ▲ 0 Tab Size: 4 UTF-8 LF Babel JavaScript Prettier: ✓ 😊 ⏱ 1

...Your own modules

Each File that you make is a module and can be used within other files.

...Node.js Core Modules

- These are the modules that are globally available in a Node.js application.
- ...We mentioned some earlier like: path, os, fs and http.



The screenshot shows a code editor window with a dark theme. The file is titled 'built-in-modules.js'. The code uses the 'os' module to log total and free memory to the console:

```
// not specified a file path so node looks to see if it
// is a global, built in module
const os = require("os");

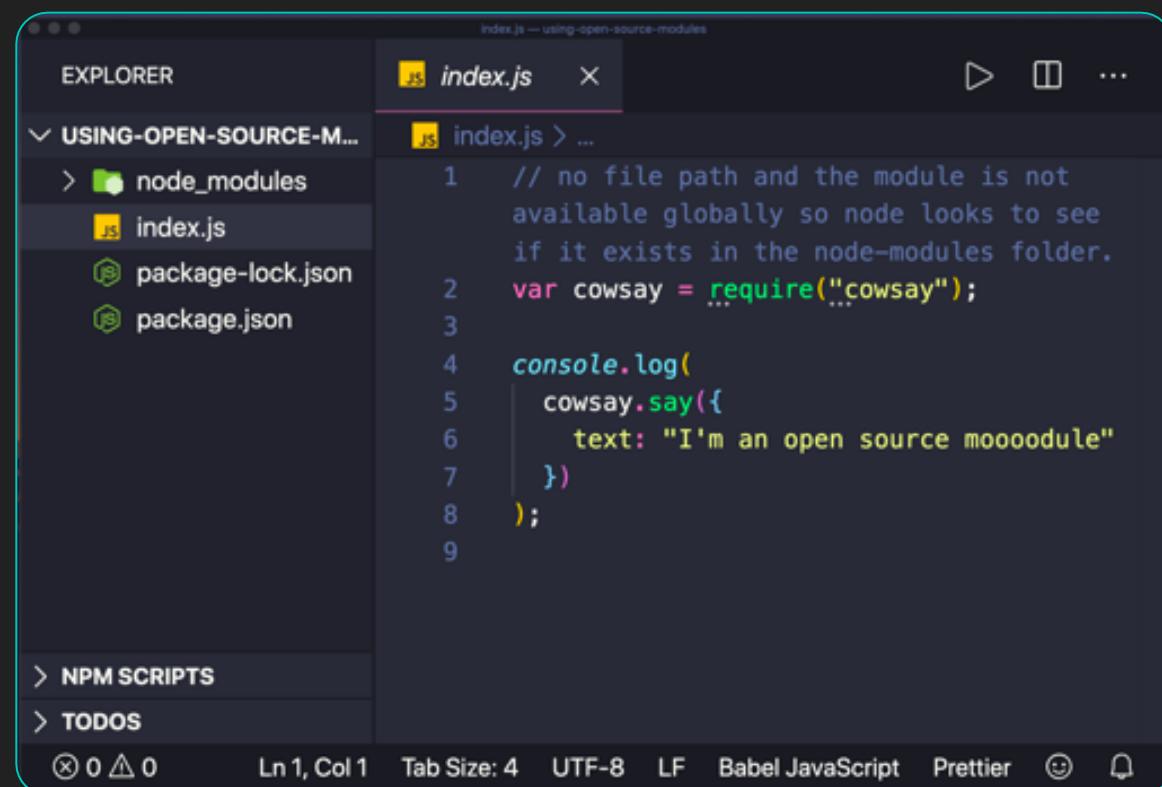
const totalMemory = os.totalmem();
const freeMemory = os.freemem();

console.log("Total Memory: " + totalMemory);
console.log("Free Memory: " + freeMemory);
```

At the bottom of the editor, there are status icons and text: ⊗ 0 ▲ 0, Tab Size: 4, UTF-8, LF, Babel JavaScript, Prettier: ✓, and a notification bell icon with a '1'.

... Open Source Modules

- Lucky for us developers there are thousands of modules already created for Node.js.
- These are stored in a repository and made available via the 'Node Package Manager' (npm).
- Npm is a command line tool that you can use to add open source packages to your Node.js project.
- Here it is:
<https://www.npmjs.com/>



The screenshot shows a code editor window with the following details:

- EXPLORER:** Shows a project structure with a folder named "USING-OPEN-SOURCE-M..." containing "node_modules", "index.js", "package-lock.json", and "package.json".
- EDITOR:** The "index.js" file is open, showing the following code:

```
// no file path and the module is not
available globally so node looks to see
if it exists in the node-modules folder.
var cowsay = require("cowsay");
console.log(
  cowsay.say({
    text: "I'm an open source mooodule"
});
)
```
- STATUS BAR:** Shows "Ln 1, Col 1", "Tab Size: 4", "UTF-8", "LF", "Babel JavaScript", "Prettier", and some icons for file operations.

When to use Node.js?



- As aforementioned, Node is good for I/O operations like accessing databases.
- Due to its asynchronous nature, it can deal with many client requests without making each user wait until the previous request is resolved.
- This makes Node.js perfect for creating RESTful APIs.

When not to use Node.js?

- Node is single threaded.
- This means that for any tasks that require multi-threading to make full use of the CPU:
 - e.g. Anything that we ever did with Marius last year...

Use another language :D! I hear C++ is a good one !



To Conclude...

- Node.js is the best :D
- Ps. I accept tips!



Any Questions?