**📦 Node.js Modules – require & module.exports Explained**

### ❓ Why Do We Need Modules in Node.js?

✅ **Real-world projects get big** — putting everything in one file becomes messy and hard to maintain.  
✅ **Modules** let us **split code into smaller, reusable files** — each handling a specific task.

🛠️ In Node.js, we use the **require() function** to import these files (modules) and keep our code **clean, organized, and maintainable**.

“We use modules to break large applications into smaller, manageable parts. Node.js supports this using require() to import code from other files.”

**🔄 What is require() in Node.js?**

✅ require() is a **built-in function** in Node.js used to **import code from other files or modules**.

**🧠 Key Points:**

* It allows you to **reuse functions, objects, or variables** defined in other files.

✅ require() can import:

* **Built-in modules** → like fs, http, path
* **Custom modules** → your own .js files in the project

🛠️ When called, require() **loads and executes** the module immediately.

**📦 Node.js Module System – All Import & Export Syntaxes with Examples**

**🔹 PART 1: CommonJS (Default in Node.js)**

**✅ Case 1: Export a single function**

**math.js**

js

CopyEdit

function add(a, b) {

return a + b;

}

module.exports = add;

**app.js**

js

CopyEdit

const add = require('./math');

console.log(add(2, 3)); // ➝ 5

**✅ Case 2: Export a single variable**

**config.js**

js

CopyEdit

const PORT = 3000;

module.exports = PORT;

**server.js**

js

CopyEdit

const PORT = require('./config');

console.log(PORT); // ➝ 3000

**✅ Case 3: Export multiple functions/values**

**math.js**

js

CopyEdit

function add(a, b) { return a + b; }

function sub(a, b) { return a - b; }

module.exports = { add, sub };

**app.js**

js

CopyEdit

const { add, sub } = require('./math');

console.log(add(10, 5)); // ➝ 15

**✅ Case 4: Export with alias (custom names)**

**math.js**

function sum(a, b) { return a + b; }

function difference(a, b) { return a - b; }

module.exports = {

total: sum,

minus: difference

};

**app.js**

const { total, minus } = require('./math');

console.log(total(5, 2)); // ➝ 7

**🔸 PART 2: ES Modules (Modern JS)**

🔧 **Enable in Node.js**:

* Add "type": "module" to package.json, or
* Use .mjs file extension

**✅ Case 1: Default export (single value/function)**

**math.js**

export default function add(a, b) {

return a + b;

}

**app.js**

import add from './math.js';

console.log(add(3, 4)); // ➝ 7

**✅ Case 2: Named exports (multiple)**

**math.js**

export function add(a, b) {

return a + b;

}

export function multiply(a, b) {

return a \* b;

}

**app.js**

import { add, multiply } from './math.js';

console.log(add(2, 3)); // ➝ 5

console.log(multiply(2, 3)); // ➝ 6

**✅ Case 3: Import with alias**

**app.js**

import { add as sum } from './math.js';

console.log(sum(4, 6)); // ➝ 10

**✅ Case 4: Export with alias**

**math.js**

function add(a, b) { return a + b; }

function sub(a, b) { return a - b; }

export { add as sum, sub as difference };

**app.js**

import { sum, difference } from './math.js';

console.log(sum(5, 2)); // ➝ 7

**✅ Case 5: Import everything as object**

**app.js**

import \* as math from './math.js';

console.log(math.add(1, 2)); // ➝ 3

**✅ Case 6: Mixed export**

**math.js**

js

CopyEdit

export const PI = 3.14;

export default function circleArea(r) {

return PI \* r \* r;

}

**app.js**

js

CopyEdit

import circleArea, { PI } from './math.js';

console.log(circleArea(2)); // ➝ 12.56

**🔒 Want to Protect Variables/Functions in Node.js?**

Yes! You can **hide internal logic** by **not exporting** everything from a module.

This provides:

* **Encapsulation** (hiding implementation details)
* **Clean API** (expose only what's necessary)

**✅ Example 1: Hiding private helper function**

**math.js**

js

CopyEdit

// 🔒 Private function (not exported)

function logToConsole(msg) {

console.log("LOG:", msg);

}

// ✅ Public function (exported)

function add(a, b) {

logToConsole("Add function called");

return a + b;

}

module.exports = {

add,

};

**app.js**

js

CopyEdit

const { add } = require('./math');

console.log(add(5, 3)); // Add function called ,8

**logToConsole is a private helper function, but it still runs internally when called by an exported function like add().**

**✅ What “private” means here:**

* It means the function is **not exported** using module.exports.
* So it's **not accessible from outside** the file (e.g., app.js cannot call it directly).
* But inside the same file (math.js), it **can be freely used**.

**✅ Example 2: Protecting internal constants or configs**

**config.js**

js

CopyEdit

// 🔒 Internal API key — not exposed

const API\_KEY = "secret-key-123";

// ✅ Only expose base URL

const BASE\_URL = "https://api.example.com";

module.exports = {

BASE\_URL,

};

**app.js**

js

CopyEdit

const { BASE\_URL } = require('./config');

console.log(BASE\_URL); // works

// ❌ Cannot access API\_KEY — it's private inside config.js

**💬 One-Liner (for Interviews):**

"In Node.js, you can hide implementation details by not exporting them — this promotes encapsulation and helps create clean, modular code."

## 🔒 Why Make Things Private or Internal in Node.js?

To **hide internal details**, protect logic, and expose only what other modules actually need. This makes your code **cleaner, safer, and easier to maintain**.

**🧱 Types of Module Systems**

| **Feature** | **CommonJS (CJS)** | **ES Modules (ESM)** |
| --- | --- | --- |
| Used by default in... | Node.js | Browsers (modern), Node (opt-in) |
| Import syntax | require() | import |
| Export syntax | module.exports | export |
| Execution | Sync | Async |
| File extension | .js | .mjs or use "type": "module" |

**🔁 Easy Difference Between CommonJS and ES Modules (Sync vs Async)**

**🧱 CommonJS → uses require()**

* **Loads one file at a time** (step by step)
* Code **waits** until the file is fully loaded
* Good for **Node.js (server-side)**

📌 Think:

"Load now, run now."

**🌐 ES Modules → uses import**

* **Loads files in the background** (non-blocking)
* Code can continue while loading
* Great for **browsers** and **modern apps**

## 🧱 CommonJS (require) — 🔁 Code ****Waits****

js

CopyEdit

// sync-example.js (CommonJS)

console.log("1️⃣ Start");

const fs = require('fs'); // 🔒 Blocking — waits for this to complete

const data = fs.readFileSync('demo.txt', 'utf8');

console.log("2️⃣ File content:", data);

console.log("3️⃣ End");

### 🧠 Output (in order):

sql

CopyEdit

1️⃣ Start

2️⃣ File content: Hello from demo.txt

3️⃣ End

✅ Code **waited** for require() and readFileSync() to finish before moving on.

## 🌐 ES Modules (import) — 🔄 Code ****Does NOT Wait****

js

CopyEdit

// async-example.mjs (ESM)

// Run this with `"type": "module"` in package.json

import fs from 'fs/promises';

console.log("1️⃣ Start");

fs.readFile('demo.txt', 'utf8')

.then(data => {

console.log("2️⃣ File content:", data);

});

console.log("3️⃣ End"); // Runs before the file is read

### 🧠 Output (order is different!):

sql

CopyEdit

1️⃣ Start

3️⃣ End

2️⃣ File content: Hello from demo.txt

✅ fs.readFile() runs in the background (non-blocking), so **code doesn't wait** — it moves on immediately.

**⚠️ Strict Mode in ES Modules**

* ES Modules always run in **strict mode** (no silent errors)
* CommonJS runs in **non-strict mode** by default
* ES modules give better safety, parsing, and error handling

**📁 Enabling ES Modules in Node.js**

You must:

1. Create a package.json file
2. Add this line inside it:

{

"type": "module"

}

This tells Node.js to treat your files as **ES Modules**.

**🔍 module.exports in Node.js – With & Without Export**

**❌ Case 1: NOT Exporting Anything**

**📁 math.js (No export)**

js

CopyEdit

function add(a, b) {

return a + b;

}

// Nothing is exported

**📁 app.js**

js

CopyEdit

const { add } = require('./math');

console.log(add(2, 3)); // ❌ Error: add is undefined

**🧠 Why?**

* Since add is not exported from math.js, it remains **private**.
* require('./math') returns an **empty object {}**.
* So, add is undefined.

**✅ Case 2: Exporting a Function Properly**

**📁 math.js (With export)**

js

CopyEdit

function add(a, b) {

return a + b;

}

// Exporting 'add' explicitly

module.exports = {

add,

};

**📁 app.js**

js

CopyEdit

const { add } = require('./math');

console.log(add(2, 3)); // ✅ Output: 5

**🧠 Why?**

* add is part of the module.exports object.
* require('./math') returns: { add: [Function] }
* So add is accessible and works as expected.

Bottom of Form

**require() Internally Returns module.exports**

In Node.js (CommonJS), when you do:

js

CopyEdit

const something = require('./math');

What you actually get back is:

js

CopyEdit

something === module.exports // from math.js

**🔁 Default Behavior**

By default, **every Node.js file** has a hidden empty object:

js

CopyEdit

module.exports = {};

If **you don’t modify it**, then require('./file') will return that **empty object** {}.

**❌ Example: Nothing Exported**

**math.js**

js

CopyEdit

function add(a, b) {

return a + b;

}

// Not exporting anything

// So module.exports remains {} by default

**app.js**

js

CopyEdit

const { add } = require('./math'); // 💥 Error: add is undefined

🧠 Here’s why:

* Internally: require('./math') → returns {} (empty object)
* So: const { add } = {} → add is undefined

**✅ Example: You Modify module.exports**

**math.js**

js

CopyEdit

function add(a, b) {

return a + b;

}

module.exports = { add }; // You override the empty object

Now:

* require('./math') returns { add: [Function: add] }
* So add(2, 3) works!