Nodejs Modules crash course

🔥 **NODE.JS MODULES — CRASH COURSE (BEGINNER → INDUSTRY LEVEL)**

(Clean • Practical • Interview + Real-Project Ready)

🚀 WHAT ARE MODULES IN NODE.JS?

👉 In Node.js, **every file is a module.**

A **module** is a reusable block of code whose variables & functions are **private by default**.

Why modules exist:

- Code reusability
- Maintainability
- Encapsulation
- Scalability (large projects)

🧠 MODULE SYSTEM IN NODE.JS

Node.js uses **CommonJS Module System** (default).

Key keywords:

- `require()` → import
- `module.exports` → export
- `exports` → shorthand export

1️⃣ TYPES OF MODULES IN NODE.JS

Node.js has **3 types of modules**:

✅ 1. CORE MODULES (Built-in)

Provided by Node.js itself.

Examples:

- `fs` → File system
- `http` → Server
- `path` → Path handling
- `os` → System info
- `events` → Event handling

Example:

js Copy code

```js
const fs = require("fs");

fs.writeFileSync("test.txt", "Hello Node");
```

📌 No installation required.

✅ 2. LOCAL (CUSTOM) MODULES

Modules created by **you**.

📁 File structure

lua Copy code

```lua
math.js
app.js
```

math.js

js Copy code

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

module.exports = add;
```

app.js

js Copy code

```js
const add = require("./math");

console.log(add(5, 3));
```

📌 `./` is mandatory for local modules.

✅ 3. THIRD-PARTY MODULES (NPM)

Installed via **npm**.

Example:

bash Copy code

```bash
npm install express
```

js Copy code

```js
const express = require("express");
```

Examples:

- express
- mongoose
- dotenv
- axios
- lodash

2️⃣ HOW `require()` WORKS INTERNALLY

When Node sees:

js Copy code

```js
require("./math");
```

Node follows this order:

1. Check core module
2. Check file (`.js` , `.json` , `.node`)
3. Check folder → `index.js`

4. Check `node_modules`

📌 This process is called **Module Resolution**.

3️⃣ module.exports vs exports (VERY IMPORTANT)

✅ module.exports (REAL EXPORT OBJECT)

js Copy code

```js
module.exports = function () {
  console.log("Hello");
};
```

⚠️ exports (REFERENCE ONLY)

js Copy code

```js
exports.sayHello = () => {
  console.log("Hello");
};
```

🚫 Wrong:

js Copy code

```js
exports = function () {}; // breaks reference
```

📌 **Rule:**

👉 Always prefer `module.exports`.

4️⃣ EXPORTING MULTIPLE VALUES

js Copy code

```js
module.exports = {
  add,
  sub,
  mul
};
```

Import:

js Copy code

```js
const math = require("./math");
math.add(2, 3);
```

5️⃣ MODULE WRAPPER FUNCTION (CORE CONCEPT)

Every Node file is wrapped internally like this:

js Copy code

```js
(function (exports, require, module, __filename, __dirname) {
  // your code
})();
```

This gives access to:

- `exports`
- `require`
- `module`
- `__filename`
- `__dirname`

📌 This is why variables are **not global**.

6️⃣ __filename & __dirname

js Copy code

```js
console.log(__filename); // full file path
console.log(__dirname);  // folder path
```

Used in:

- file uploads
- path resolution
- config loading

7️⃣ CACHING IN NODE MODULES

Modules are loaded **only once**.

```js
require("./math");
require("./math");
```

✔ Executed only first time

✔ Cached in memory

Used for:

- DB connections
- Config files
- Singleton services

8️⃣ JSON MODULES

You can import JSON directly.

```js
const data = require("./config.json");
```

📌 Auto-parsed into JS object.

9️⃣ INDEX.JS PATTERN (INDUSTRY STANDARD)

Folder modules:

```pgsql
services/
 ├─ user.js
 ├─ order.js
 └─ index.js
```

index.js

js

```js
module.exports = {
  userService: require("./user"),
  orderService: require("./order")
};
```

Usage:

js Copy code

```js
const services = require("./services");
```

🔟 COMMONJS vs ES MODULES (MODERN NODE)

CommonJS (Default)

js Copy code

```js
const fs = require("fs");
module.exports = fs;
```

ES Modules (Modern)

js Copy code

```js
import fs from "fs";
export default fs;
```

Enable ES modules:

json Copy code

```json
{
  "type": "module"
}
```

📌 Many production apps still use CommonJS.

1️⃣1️⃣ WHEN MODULES CRASH (COMMON ERRORS)

❌ `Cannot find module`

- Wrong path
- Missing `./`

❌ `exports is not defined`

- ES module + CommonJS mix

❌ Circular dependency

- Module A → B → A

1️⃣2️⃣ REAL-WORLD PROJECT STRUCTURE

arduino

```
src/
 ├─ controllers/
 ├─ services/
 ├─ models/
 ├─ routes/
 ├─ utils/
 ├─ config/
 └─ index.js
```

Each folder = module

Each file = module

1️⃣3️⃣ BEST PRACTICES (INTERVIEW + INDUSTRY)

✔ One module = one responsibility

✔ Use `module.exports` clearly

✔ Avoid circular dependencies

✔ Group related logic

✔ Use index.js for clean imports

✔ Don't pollute global scope

🎯 WHAT NEXT?