Modules (CommonJS, AMD, ES6 modules)

📦 JavaScript Modules (CommonJS, AMD, ES6)

Modules allow you to break your JavaScript code into separate, reusable files. This helps maintain **clean, scalable**, and **manageable** codebases, especially in large applications.

🔷 1. Why Modules?

- Avoid global scope pollution
- Encapsulation (private vs public code)
- Code reuse
- Maintainability
- Dependency management

🔷 2. CommonJS (CJS)

Used in: Node.js (still the default for many Node environments)

👉 Syntax:

```
// math.js
const add = (a, b) => a + b;
module.exports = { add };

// app.js
const math = require('./math');
console.log(math.add(2, 3)); // 5
```

✅ Characteristics:

- Synchronous (loads modules at runtime)
- `require()` for importing
- `module.exports` or `exports` for exporting
- Best suited for server-side code (Node.js)
- Not natively supported in browsers without bundlers like Webpack

---

🔷 3. AMD (Asynchronous Module Definition)

Used in: Browsers (historically), older apps with RequireJS

👉 Syntax (with RequireJS):

```
// math.js
define([], function () {
  return {
    add: function (a, b) {
      return a + b;
    }
  };
});

// app.js
require(['math'], function (math) {
  console.log(math.add(2, 3)); // 5
});
```

✅ Characteristics:

- Asynchronous loading (good for browser)
- Uses `define()` and `require()`
- Designed for client-side JavaScript
- Mostly replaced by ES modules now

🔷 4. ES6 Modules (ESM)

Used in: Modern browsers and Node.js (with `.mjs` or `"type": "module"` in package.json)

👉 Exporting:

```
// math.js
export const add = (a, b) => a + b;
export default function multiply(a, b) {
  return a * b;
}
```

👉 Importing:

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

console.log(add(2, 3));       // 5
console.log(multiply(2, 3));  // 6
```

✅ Characteristics:

- Static analysis (parsed before execution)
- `import` / `export` syntax
- Supports **named** and **default** exports
- Works both in browsers and Node.js
- Supports **tree-shaking** for optimization

🔷 5. Key Differences

Feature	CommonJS	AMD	ES6 Modules
Load Type	Synchronous	Asynchronous	Static (asynchronous in browsers)
Syntax	`require()`	`define()`	`import` / `export`

Platform	Node.js	Browser (older)	Browser & Node.js
Default Export	`module.exports`	Return from define	`export default`
Tree Shaking	❌	❌	✅

🔷 6. How to Use ES Modules in Node.js

Option 1: Use `.mjs` extension

```
node app.mjs
```

Option 2: Set `"type": "module"` in `package.json`

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

Then use `.js` extensions with `import` / `export`.

🔷 7. Bundlers and Transpilers

- **Webpack**, **Rollup**, **Parcel** are used to bundle modules for production.
- **Babel** can transpile ES6 module code to CommonJS for backward compatibility.

🔷 8. Use Case Summary

Use Case	Recommended Module Format
Node.js (legacy)	CommonJS (`require`)
Modern browser apps	ES6 Modules (`import`)
RequireJS-based apps	AMD
Tree-shakable bundling	ES6 Modules

✅ Conclusion

- Use **ES6 modules** for modern development (browser + Node).
- Understand **CommonJS** for working in traditional Node.js environments.
- Know **AMD** for legacy browser code.

Would you like a working example of converting CommonJS to ES6 Modules?