

<WA/>

2025

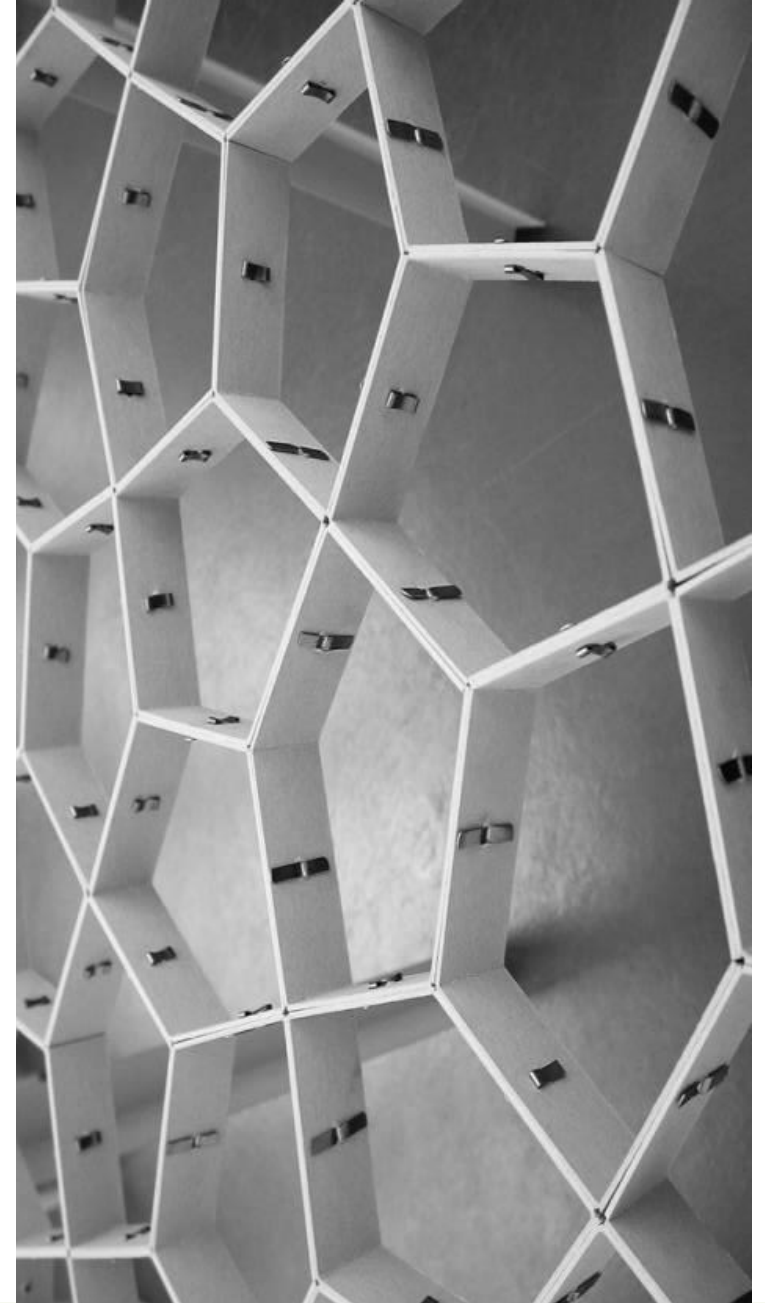
JavaScript Modules

“The” language of the Web

Fulvio Corno

Luigi De Russis

Enrico Masala





JavaScript: The Definitive Guide, 7th Edition
Chapter 10. Modules

Mozilla Developer Network

- [Web technology for developers » JavaScript » JavaScript Guide » JavaScript Modules](#)

Modular JS programming

MODULES

Modules

- Mechanisms for splitting JavaScript programs into separate files that can be imported when needed
- Encapsulate or hide private implementation details and keep the global namespace tidy so that modules can not accidentally modify the variables, functions and classes defined by other modules
- 3 kinds of modules \Rightarrow

1. *Do-It-Yourself* (with classes, objects, IIFE and closures)
2. **ES6 modules** (using `export` and `import`)
 1. ECMA Standard
 2. Supported by recent browsers
 3. Supported by Node (v13+)
3. Node.js modules (using `require()`) – called **CommonJS**
 1. Based on closures
 2. Never standardized by ECMA, but the normal practice with Node

ES6

CJS

ES6 Modules

- A module is a JavaScript file that **exports** one or more values (objects, functions or variables), using the **export** keyword
 - each module is a piece of code that is executed once it is loaded
- Any other JavaScript module can **import** the functionality offered by another module by importing it, with the **import** keyword
- Imports and exports must be at the *top level*
- Two main kinds of exports:
 - **named** exports (several per module)
 - **default** export (one per module)

Default Export

- Modules that only export **single values**
 - Only one per module
 - You are exporting a values, but not the name of the resource
- Syntax
 - `export default <value>`

```
export default str =>  
str.toUpperCase();
```

```
// OTHER examples  
export default {x: 5, y: 6};
```

```
export default "name";
```

```
function grades(student) {...};  
export default grades;
```

Named Exports

- Modules that export **one or more values**
 - several per module
 - Exports **also the names**
- Syntax
 - `export <value>`
 - `export {<value>, <...>}`

```
export const name = 'Luigi';
```

```
function grades(student) {...};  
export grades;
```

```
const name = 'Luigi';  
const anotherName = 'Fulvio';  
export { name, anotherName }  
// we can also rename them...  
// export {name, anotherName as teacher}
```

Imports

- To import something exported by another module
- Syntax
 - `import package from 'module-name'`
- Imports are:
 - hoisted
 - read-only views on exports

Import From a Default Export

```
//--- module1.js ---  
export default str =>  
str.toUpperCase();
```

```
//--- module2.js ---  
import toUpperCase from './module1.js';  
// you choose the name!  
  
// another example  
import uppercase from '/home/app/module1.js';  
  
// usage of the imported function  
uppercase('test');
```


Import From a Named Export

```
//--- module1.js ---  
const name = 'Luigi';  
const anotherName = 'Fulvio';  
  
export { name, anotherName };
```

```
//--- module2.js ---  
import { name, anotherName } from './module1.js';  
  
// you can rename imported values, if you want  
import { name as first, anotherName as second }  
      from './module1.js';  
  
// usage  
console.log(first);
```

Other Imports Options

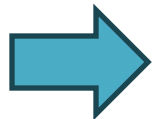
- You can import everything a module exports
 - `import * from 'module'`
- You can import a few of the exports (e.g., if `export {a, b, c}`):
 - `import {a} from 'module'`
- You can import the default export alongside with any named exports:
 - `import default, { name } from 'module'`

ES6 Modules In The Browser

- File extension
 - Preferred: `.mjs` (ensure the server sets Content-Type: text/javascript)
 - Also accepted: `.js`
- Load in HTML
 - `<script type="module" src="main.js"></script>`
 - Only load the “main” modules, others will be loaded by import statements
 - **Only files loaded with `type="module"` may use `import` and `export`**
 - Modules are **automatically** loaded in **defer** mode
 - Note: locally loading modules (`file:///`) **does not work** due to CORS

ES6 Modules In Node.js

- Node.js started to support ES6 modules only recently
- From Node.js v14 (LTS)
 - Enabled by default
 - Must use a file extension of `.mjs` *or* specify `"type": "module"` in `package.json`
 - <https://nodejs.org/docs/latest-v20.x/api/esm.html#enabling>
- **Beware:** not all Node.js modules are provided as ES6 modules



In Node.js, use CommonJS modules with `require()`

CommonJS Modules

- The standard module format in Node.js
- Uses the `.js` or `.cjs` extension
- Not natively supported by browsers
 - Unless you use libraries such as RequireJS (<https://requirejs.org/>)
- It is basically a wrapper around your module code

```
(function(exports, require, module, __filename, __dirname) {  
  // Module code actually lives in here  
});
```

<https://nodejs.org/docs/latest-v20.x/api/modules.html>

CommonJS Imports

- To import something exported by another module
- `const package = require('module-name')`
 - Looked up in `node_modules`
- `const myLocalModule = require('./path/myLocalModule');`
 - Looked up in a relative path from `__dirname` or `$cwd`

CommonJS Exports

- Assign your exported variables by creating **new properties in the object** `module.exports` (shortcut: `exports`)
- Examples:
 - `exports.area = (r) => Math.PI*(r**2);`
 - `module.exports = class Square {
 constructor(width) {
 this.width = width;
 }
 area() {
 return this.width ** 2;
 }
};`

```
'use strict';

/* Data Access Object (DAO) */
const db = require('./db');
const dayjs = require("dayjs");

// Retrieves the whole list of films
exports.listFilms = (filter) => {
  return new Promise((res, rej) => {
    const sql = 'SELECT * FROM films';
    db.all(sql, (err, rows) => { ...
    ...
  }}}}
```

License

- These slides are distributed under a Creative Commons license “**Attribution-NonCommercial-ShareAlike 4.0 International (CC BY-NC-SA 4.0)**”
- **You are free to:**
 - **Share** — copy and redistribute the material in any medium or format
 - **Adapt** — remix, transform, and build upon the material
 - The licensor cannot revoke these freedoms as long as you follow the license terms.
- **Under the following terms:**
 - **Attribution** — You must give [appropriate credit](#), provide a link to the license, and [indicate if changes were made](#). You may do so in any reasonable manner, but not in any way that suggests the licensor endorses you or your use.
 - **NonCommercial** — You may not use the material for [commercial purposes](#).
 - **ShareAlike** — If you remix, transform, or build upon the material, you must distribute your contributions under the [same license](#) as the original.
 - **No additional restrictions** — You may not apply legal terms or [technological measures](#) that legally restrict others from doing anything the license permits.
- <https://creativecommons.org/licenses/by-nc-sa/4.0/>

