ScopeTreasure Courses

JAVASCRIPT COURSES · LESSON 3.17 (SHORT) · CODE SCOPE SERIES

ES6: Modules (import/export)

Split code into files that share: export what others need, import what you use, named vs default exports — one concept, from the Short.

Introduction

One giant `script.js` works for a small page — and then the page grows, and finding anything in that file becomes the real job. **Modules** fix this: split the code into files, **one job per file**, and let the files share with each other in a controlled way. A file marks what it shares with `export`, and another file takes what it needs with `import`. *(Roman Urdu: Har file ka aik kaam — export share karta hai, import zaroorat ki cheez le leta hai.)*

Key points

- **`export` = this file shares it:** put `export` in front of a value, a function or a class, and other files are allowed to use it. Anything **without** `export` stays private inside its file.
- **`import` = this file takes it:** `import { menu } from "./data.js";` — the path says which file, the braces say what you are taking.
- **Named exports:** a file can share **several** named things. You import them with `{ }`, using their **exact names** — the name in the import must match the name in the export.

  (Roman Urdu: Named export mein naam bilkul wahi likho jo export kiya gaya tha.)
- **Default export:** a file can have **one** main thing as its `export default`. You import it **without** braces, and the name is **your choice** — `import makeCard from "./card.js";`.
- **The page opts in:** a module only loads when the script tag says so — the page needs the module type on its script tag (shown in Example 3). A normal script tag will reject `import` and `export`.
- **Imports go at the top:** all `import` lines live at the top of the file, before the other code — that is the rule, not just a style.
- **Files have walls:** a variable inside a module is private unless it is exported — the same idea as a function's scope from the earlier lessons, now at file level.

  (Roman Urdu: Jo export nahi kiya, woh file ke andar hi rehta hai — bahar koi nahi dekh sakta.)

Code examples

Example 1 — Named exports: share the menu

File: data.js

```
export const menu = ["Chai", "Coffee", "Lassi"];
export const shopName = "Code Scope Cafe";
```

File: main.js

```
import { menu, shopName } from "./data.js";

console.log(shopName);
console.log(menu.join(", "));
```

What it does: The data file shares two named things, `menu` and `shopName`, each with `export` in front of it. The main file takes both with one `import`, using curly braces and the exact same names, and uses them like its own variables.

What you will see: when the page loads `main.js` as a module, the console prints `Code Scope Cafe`, then `Chai, Coffee, Lassi`.

Example 2 — One default export: the card builder

File: card.js

```
export default function makeCard(name, price) {
  return name + " - Rs " + price;
}
```

File: main.js

```
import buildCard from "./card.js";

console.log(buildCard("Masala Chai", 250));
```

What it does: The card file has one main job, so it uses `export default` — and a file may only ever have **one** default. The main file imports it **without** braces, and because it is the default, the importing side may choose the name: here `makeCard` is imported as `buildCard`, and it works exactly the same.

What you will see: when the page loads `main.js` as a module, the console prints `Masala Chai - Rs 250`.

Example 3 — A tiny live page shape: data, ui, main

File: index.html

```
<!DOCTYPE html>
<html>
<head>
  <title>Module Cafe</title>
</head>
<body>
  <h1 id="title"></h1>
  <div id="list"></div>

  <script type="module" src="main.js"></script>
</body>
</html>
```

File: data.js

```
export const menu = ["Chai", "Coffee", "Lassi"];
```

File: ui.js

```
export function cardText(item) {
  return "Fresh " + item;
}
```

File: main.js

```
import { menu } from "./data.js";
import { cardText } from "./ui.js";

document.getElementById("title").textContent = "Today's Menu";
document.getElementById("list").textContent = menu.map(cardText).join(" | ");
```

What it does: Three files, three jobs: `data.js` holds the menu, `ui.js` holds a card-text builder, and `main.js` imports both and runs the page. The key line is in the HTML: the script tag carries `type="module"` — without it, the browser refuses the `import` lines. Notice `menu.map(cardText)` — the imported function is used as a callback, just like your own functions.

What you will see: the page shows the heading `Today's Menu` and below it the line `Fresh Chai | Fresh Coffee | Fresh Lassi`.

Quick recap

- `export` shares, `import` takes — named exports use `{ }` and exact names; one default export per file imports without braces, under any name you choose.

- The page must opt in with a module script tag, and all imports sit at the top of the file.
- What is not exported stays private: each module file has walls, like a function's scope.

Common mistakes

1. **Braces in the wrong place** — importing a named export without `{ }` (or a default export with them) gives you `undefined` or an error, because the two kinds are taken differently. The fix: named = braces and exact name; default = no braces, your choice of name.
2. **Forgetting the module type on the script tag** — with a plain script tag, the browser throws an error on the very first `import` line and nothing runs. The fix: the script that starts your modules must be marked as a module, as in Example 3.
3. **Trying to use a variable that was never exported** — it does not matter that the other file "has" it; without `export`, the wall holds and the import fails. The fix: export exactly what other files need — no more, no less.

Practice

1. Create a file `prices.js` that exports a named array `prices` with the values `250`, `200` and `300`, and a named function `total` that adds up an array with `reduce`. In `main.js`, import both, and print the total of the prices.
2. Create a file `greet.js` whose default export is a function that takes a name and returns the text `Welcome,` plus that name. Import it in `main.js` under a name of your own choice, call it with your name, and print the result.

Practice quiz

1. How do named exports and a default export differ in import?

- Both use braces and exact names
- Named exports use { } and exact names, one default export per file imports without braces under any name you choose
- Named exports have no braces, default exports have braces
- There is only one kind of export

Answer: Named exports use { } and exact names, one default export per file imports without braces under any name you choose

2. What happens to a value in a module file that has no export?

- It is shared with every file automatically
- It stays private inside its file, like a function's scope at file level
- It becomes a global variable
- It is deleted

Answer: It stays private inside its file, like a function's scope at file level

3. What must the page's script tag say for modules to load, and where do imports live?

- Nothing special, and imports can go anywhere
- The script tag must be a module type, and all import lines live at the top of the file
- The script tag must be in the head only
- Imports live at the bottom of the file

Answer: The script tag must be a module type, and all import lines live at the top of the file

---

Watch the Code Scope channel: https://www.youtube.com/@CodeScopePK — this lesson's own video joins the channel when it is recorded.

Read online: https://scopetreasure.com/notes/js-modules-import-export · More free tools and notes at https://scopetreasure.com