Modern JavaScript: Modules & Tooling

> 🎓 **CS Perspective:** ES6 modules implement a **static module system** with compile-time dependency resolution (imports are hoisted and analyzed before execution). This enables **tree-shaking**—dead code elimination during bundling. Unlike CommonJS (dynamic, runtime resolution), ES6 imports are **live bindings** (references, not copies), which has implications for circular dependencies. The module pattern using IIFEs demonstrates **closures for encapsulation**—creating private scope through function invocation. Bundlers (Webpack, Parcel) perform **dependency graph analysis** and code transformation. Transpilers like Babel implement **source-to-source compilation**, converting modern JS to older versions. This toolchain represents the modern **build pipeline** concept in software engineering.

📌 ES6 Modules vs Scripts

Feature	ES6 Modules	Scripts
Top-level variables	Scoped to module	Global
Default mode	Strict mode	"Sloppy" mode
Top-level `this`	`undefined`	`window`
Imports/Exports	✅	❌
HTML linking	`<script type="module">`	`<script>`
File downloading	Async	Sync (blocking)

📌 Exporting

Named Exports

```
// shoppingCart.js
const shippingCost = 10;
const cart = [];

export const addToCart = function(product, quantity) {
  cart.push({ product, quantity });
  console.log(`${quantity} ${product} added`);
};

export { shippingCost, cart };

// Can rename on export
export { totalPrice as price, totalQuantity as tq };
```

Default Export

One per module, usually for main functionality:

```
// shoppingCart.js
export default function(product, quantity) {
  cart.push({ product, quantity });
  console.log(`${quantity} ${product} added`);
}
```

📌 Importing

Named Imports

```
import { addToCart, totalPrice as price } from './shoppingCart.js';

addToCart('bread', 5);
console.log(price);
```

Import All (Namespace)

```
import * as ShoppingCart from './shoppingCart.js';

ShoppingCart.addToCart('bread', 5);
console.log(ShoppingCart.shippingCost);
```

Default Import

```
// Name it whatever you want
import add from './shoppingCart.js';

add('pizza', 2);
```

Mixed Imports

```
// Default + Named (not recommended)
import add, { cart, totalPrice } from './shoppingCart.js';
```

📌 Important Module Characteristics

Live Connection

Imports are **live bindings**, not copies:

```
// shoppingCart.js
export const cart = [];

// script.js
import { cart } from './shoppingCart.js';
ShoppingCart.addToCart('pizza', 2);
console.log(cart); // [{ product: 'pizza', quantity: 2 }]
```

Modules Execute Once

Even if imported multiple times:

```
import './shoppingCart.js';
import './shoppingCart.js';
import './shoppingCart.js';
// Code runs only ONCE
```

📌 Top-Level Await (Modules Only)

```
// Blocks module execution
const res = await fetch('https://api.example.com/data');
const data = await res.json();
console.log(data);

export { data };
```

⚠ **Warning**: Blocks importing module too!

📌 The Module Pattern (Old Way)

Before ES6 modules, used IIFEs:

```
const ShoppingCart = (function() {
  const cart = [];
  const shippingCost = 10;

  const addToCart = function(product, quantity) {
    cart.push({ product, quantity });
  };

  const orderStock = function(product, quantity) {
```

```
    console.log(`${quantity} ${product} ordered`);
  };

  return {
    addToCart,
    cart
  };
})();

ShoppingCart.addToCart('apple', 4);
console.log(ShoppingCart.cart); // Works
console.log(ShoppingCart.shippingCost); // undefined (private)
```

📌 CommonJS Modules (Node.js)

```
// Export (Node.js)
exports.addToCart = function(product, quantity) {
  cart.push({ product, quantity });
};

// Import (Node.js)
const { addToCart } = require('./shoppingCart.js');
```

📌 NPM (Node Package Manager)

Initialize Project

```
npm init    # Interactive
npm init -y # Default settings
```

Install Packages

```
npm install lodash-es      # Production dependency
npm install parcel --save-dev  # Dev dependency
npm install                    # Install all from package.json
```

package.json

```
{
  "name": "project",
  "version": "1.0.0",
  "scripts": {
```

```
    "start": "parcel index.html",
    "build": "parcel build index.html"
  },
  "dependencies": {
    "lodash-es": "^4.17.21"
  },
  "devDependencies": {
    "parcel": "^2.0.0"
  }
}
```

Run Scripts

```
npm run start
npm run build
```

📌 Bundling with Parcel

Install

```
npm install parcel --save-dev
```

Run Dev Server

```
npx parcel index.html
# or add to package.json scripts
```

Hot Module Replacement

```
if (module.hot) {
  module.hot.accept();
}
```

Build for Production

```
npx parcel build index.html
```

📌 Importing Non-JS Assets

```
// Images
import imgPath from './img/logo.png';
imgEl.src = imgPath;

// JSON (no await needed)
import data from './data.json';
```

📌 Transpiling & Polyfilling

Transpiling (Babel)

Converts modern syntax to ES5:

- Arrow functions → Regular functions
- Classes → Constructor functions
- `const/let` → `var`

Parcel includes Babel automatically.

Polyfilling

Adds missing features to old browsers:

```
import 'core-js/stable';  // Polyfill features
import 'regenerator-runtime/runtime';  // Async/await polyfill
```

Only polyfills what you use (tree-shaking).

📌 Clean Code Principles

Readable Code

```
// ❌ Bad
const x = [1, 2, 3];
const y = x.map(n => n * 2);

// ✅ Good
const numbers = [1, 2, 3];
const doubledNumbers = numbers.map(num => num * 2);
```

Functional Principles

```

```

```
// ❌ Mutating (impure)
const addToCart = function(product) {
  cart.push(product);
};

// ✅ Non-mutating (pure)
const addToCart = function(cart, product) {
  return [...cart, product];
};
```

Declarative vs Imperative

```
// ❌ Imperative (how)
const doubled = [];
for (let i = 0; i < arr.length; i++) {
  doubled.push(arr[i] * 2);
}

// ✅ Declarative (what)
const doubled = arr.map(n => n * 2);
```

📌 Modern JavaScript Features Summary

Feature	ES Version
Modules	ES6 (2015)
Arrow functions	ES6
Template literals	ES6
Destructuring	ES6
Spread/Rest	ES6
Promises	ES6
Classes	ES6
`async/await`	ES2017
Object spread	ES2018
Optional chaining `?.`	ES2020
Nullish coalescing `??`	ES2020
`Promise.allSettled`	ES2020
Private fields `#`	ES2022

Feature	ES Version
Top-level await	ES2022

🧪 Quick Reference Cheatsheet

```
// Named export
export const name = 'value';
export { name1, name2 };

// Default export
export default function() {}

// Named import
import { name } from './module.js';
import { name as alias } from './module.js';

// Namespace import
import * as Module from './module.js';

// Default import
import anyName from './module.js';

// HTML
<script type="module" src="script.js"></script>

// NPM
npm init -y
npm install package
npm install package --save-dev

// Parcel
npx parcel index.html        // Dev
npx parcel build index.html  // Production

// Polyfills
import 'core-js/stable';
import 'regenerator-runtime/runtime';
```

✅ Exam Tips

1. ES6 modules are **strict mode** by default
2. Module top-level variables are **scoped**, not global
3. Imports are **hoisted** (moved to top)
4. Imports are **live bindings**, not copies
5. Each module executes **only once**
6. Use `type="module"` in HTML for ES6 modules
7. **Named exports** can have multiple per file

8. **Default export** is one per file
9. Top-level `await` **blocks** module execution
10. `node_modules` should be in `.gitignore`
11. Parcel does **bundling + transpiling + polyfilling**
12. `dependencies` = production, `devDependencies` = build tools