

THE COMPLETE JAVASCRIPT COURSE

FROM ZERO TO EXPERT!

SECTION

MODERN JAVASCRIPT
DEVELOPMENT: MODULES AND
TOOLING

LECTURE

AN OVERVIEW OF MODULES
IN JAVASCRIPT

JS

AN OVERVIEW OF MODULES

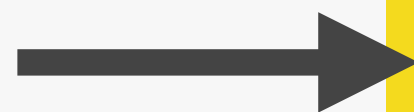
MODULE

- ➔ Reusable piece of code that **encapsulates** implementation details;
- ➔ Usually a **standalone file**, but it doesn't have to be.

WHY
MODULES?

- ➔ **Compose software:** Modules are small building blocks that we put together to build complex applications;
- ➔ **Isolate components:** Modules can be developed in isolation without thinking about the entire codebase;
- ➔ **Abstract code:** Implement low-level code in modules and import these abstractions into other modules;
- ➔ **Organized code:** Modules naturally lead to a more organized codebase;
- ➔ **Reuse code:** Modules allow us to easily reuse the same code, even across multiple projects.

IMPORT
(DEPENDENCY)



MODULE

```
import { rand } from './math.js';  
const diceP1 = rand(1, 6, 2);  
const diceP2 = rand(1, 6, 2);  
const scores = { diceP1, diceP2 };  
export { scores };
```

Module code



EXPORT
(PUBLIC API)

NATIVE JAVASCRIPT (ES6) MODULES

ES6 MODULES

Modules stored in files, **exactly one module per file.**

```
import { rand } from './math.js';  
const diceP1 = rand(1, 6, 2);  
const diceP2 = rand(1, 6, 2);  
const scores = { diceP1, diceP2 };  
export { scores };
```

import and export
syntax

Need to happen at top-level
Imports are hoisted!

ES6 MODULE

SCRIPT

👉 Top-level variables

Scoped to module

Global

👉 Default mode

Strict mode

"Sloppy" mode

👉 Top-level this

undefined

window

👉 Imports and exports

✅ YES

❌ NO

👉 HTML linking

`<script type="module">`

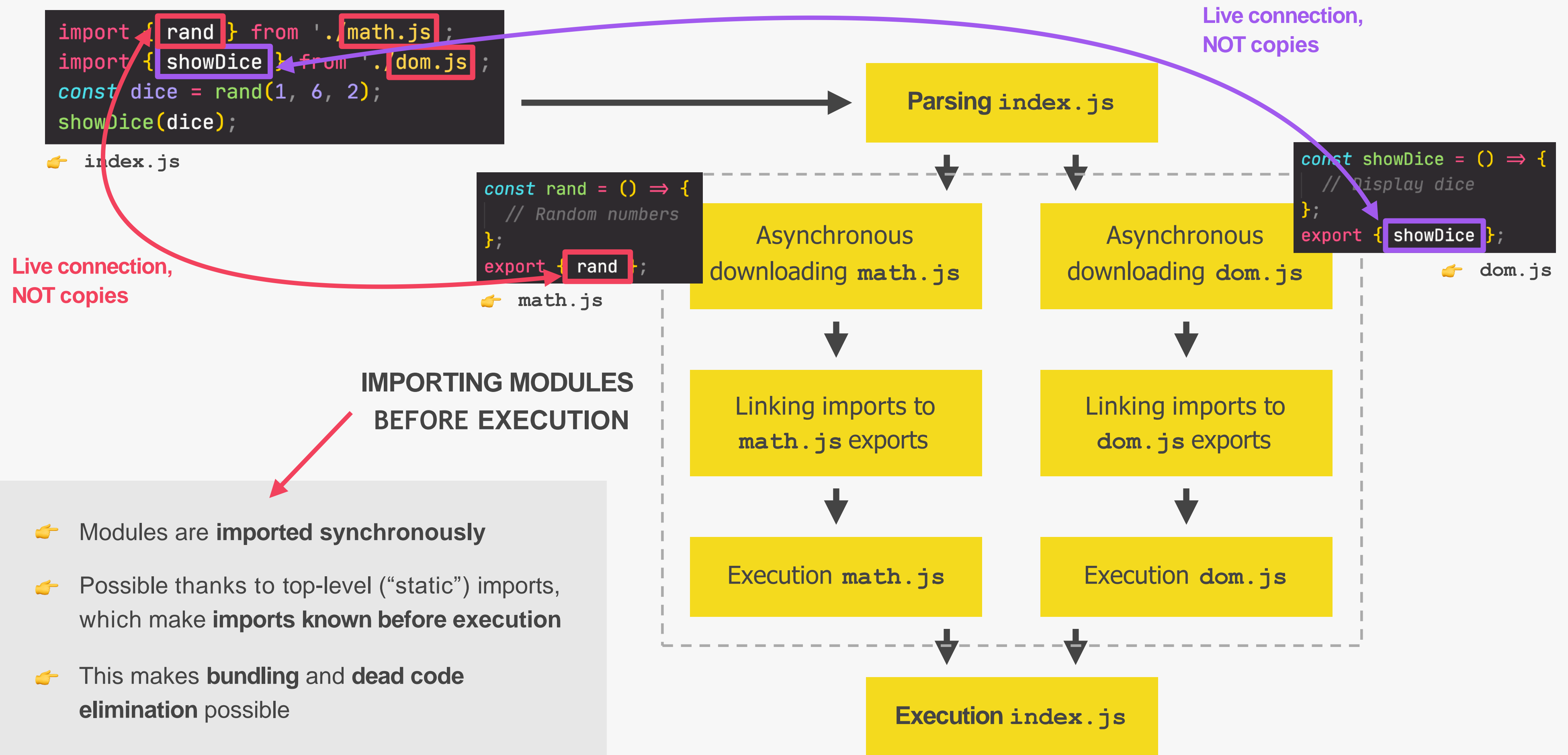
`<script>`

👉 File downloading

Asynchronous

Synchronous

HOW ES6 MODULES ARE IMPORTED



THE COMPLETE JAVASCRIPT COURSE

FROM ZERO TO EXPERT!

SECTION

MODERN JAVASCRIPT
DEVELOPMENT: MODULES AND
TOOLING

LECTURE

REVIEW: WRITING CLEAN AND
MODERN JAVASCRIPT

JS

REVIEW: MODERN AND CLEAN CODE

READABLE CODE

- 👉 Write code so that **others** can understand it
- 👉 Write code so that **you** can understand it in 1 year
- 👉 Avoid too “clever” and overcomplicated solutions
- 👉 Use descriptive variable names: **what they contain**
- 👉 Use descriptive function names: **what they do**

GENERAL

- 👉 Use DRY principle (refactor your code)
- 👉 Don't pollute global namespace, encapsulate instead
- 👉 Don't use **var**
- 👉 Use strong type checks (**===** and **!==**)

FUNCTIONS

- 👉 Generally, functions should do **only one thing**
- 👉 Don't use more than 3 function parameters
- 👉 Use default parameters whenever possible
- 👉 Generally, return same data type as received
- 👉 Use arrow functions when they make code more readable

OOP

- 👉 Use ES6 classes
- 👉 Encapsulate data and **don't mutate** it from outside the class
- 👉 Implement method chaining
- 👉 Do **not** use arrow functions as methods (in regular objects)

REVIEW: MODERN AND CLEAN CODE

AVOID NESTED CODE

- 👉 Use early **return** (guard clauses)
- 👉 Use ternary (conditional) or logical operators instead of **if**
- 👉 Use multiple **if** instead of **if/else-if**
- 👉 Avoid **for** loops, use array methods instead
- 👉 Avoid callback-based asynchronous APIs

ASYNCHRONOUS CODE

- 👉 Consume promises with **async/await** for best readability
- 👉 Whenever possible, run promises in **parallel** (**Promise.all**)
- 👉 Handle errors and promise rejections