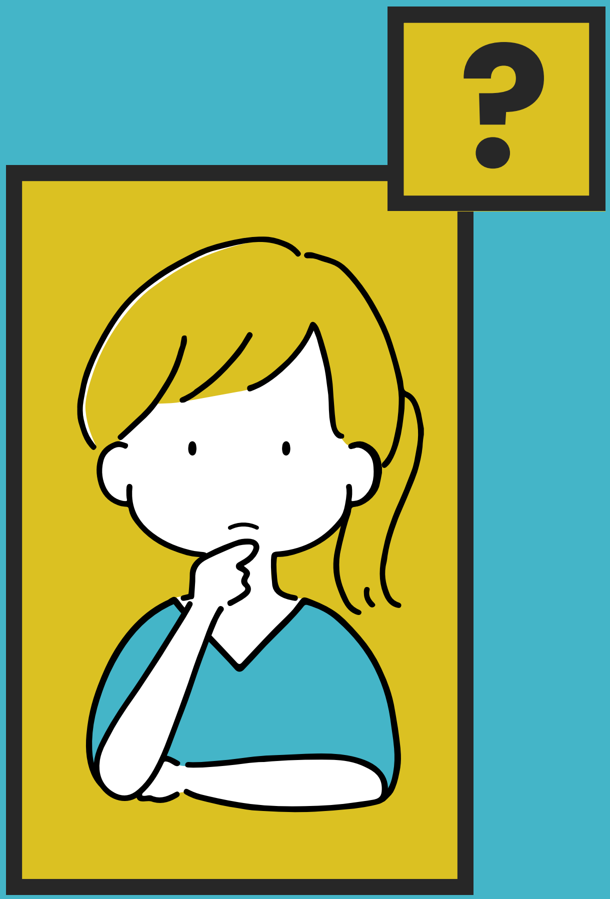
Slim

JavaScript tips

What Are JavaScript Modules?



What is a **Module** in JavaScript?

A module in JavaScript is a way to split your code into smaller, reusable pieces. By using modules, you can:

- **Organize your code**: Break down large applications into smaller, more manageable pieces.
- **Reuse code**: Import modules where needed, reducing repetition.
- **Maintainability**: Keep related functionality together, making it easier to debug and modify.

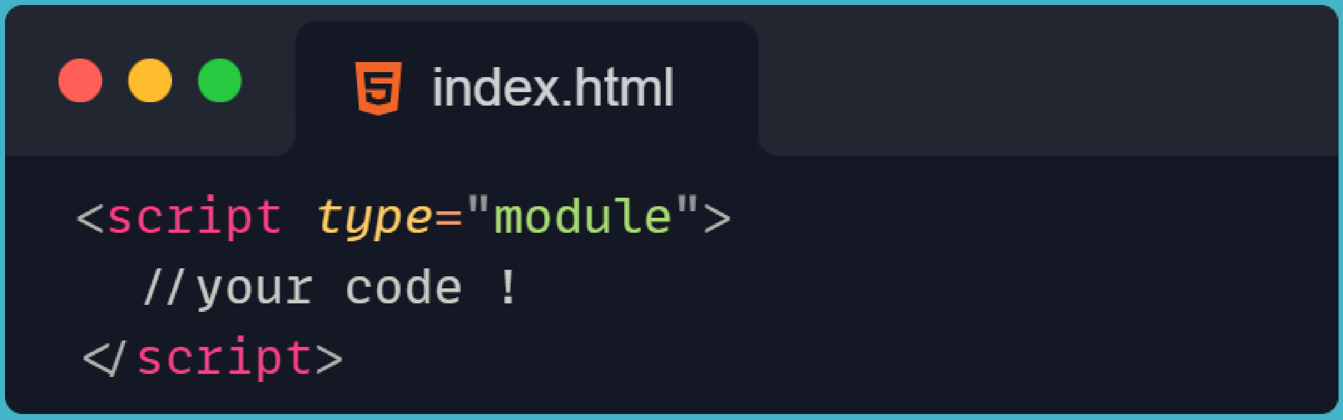
JavaScript modules allow you to use import and export statements to make code in one file available to others.

How to Enable Modules in JavaScript ?

To use JavaScript modules, you need to enable them in your environment. Here's how:

1- In the Browser (Using <script type="module">)

To use JavaScript modules in a web browser, you need to include the type="module" attribute in your <script> tag.


index.html

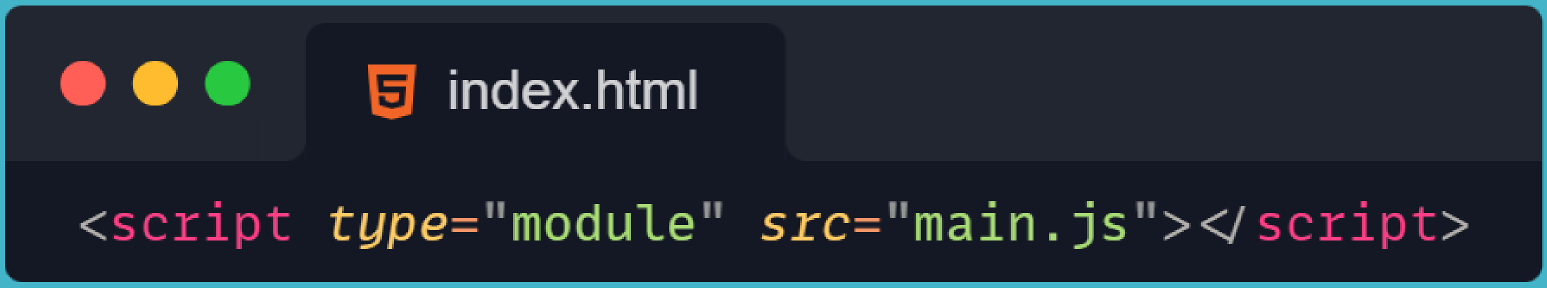
```
<script type="module">
  //your code !
</script>
```

Explanation:

- The **type="module"** tells the browser that this script contains modules, and you can use import and export in your code.
- Modules are always loaded in **strict mode**, which means some behaviors are different (like variable scoping).

2- Using Modules with External Files

If you’re working with external JavaScript files, you’ll also use the type="module" attribute for them.


index.html

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

Explanation:

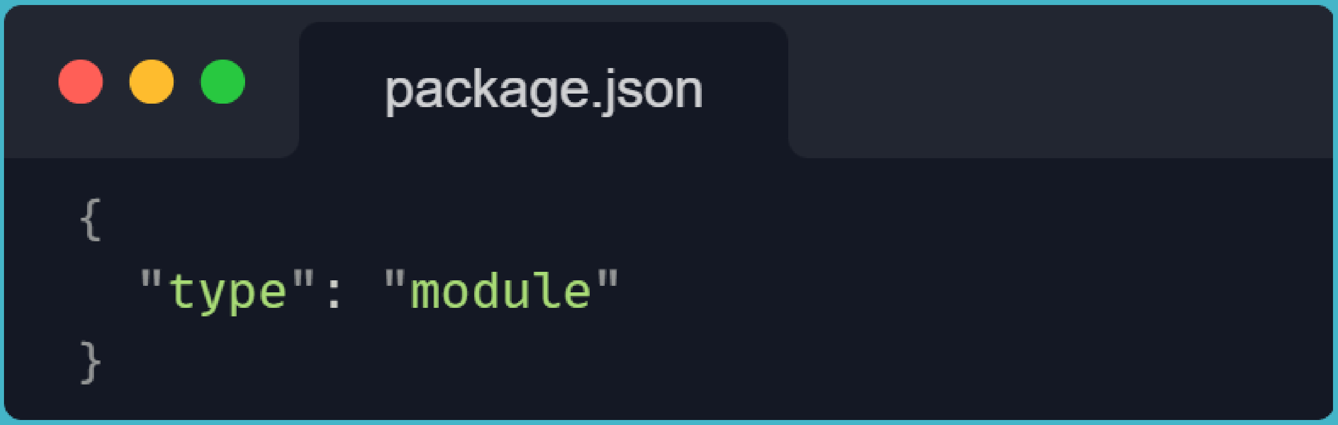
- The main.js file is now treated as a module, and you can use import and export within it.

3- In Node.js (With ES Module Support)

For Node js, ES modules are supported, but you must enable them explicitly:

- Rename Files to **.mjs**

  or
- Set **"type": "module"** in package.json.


package.json

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

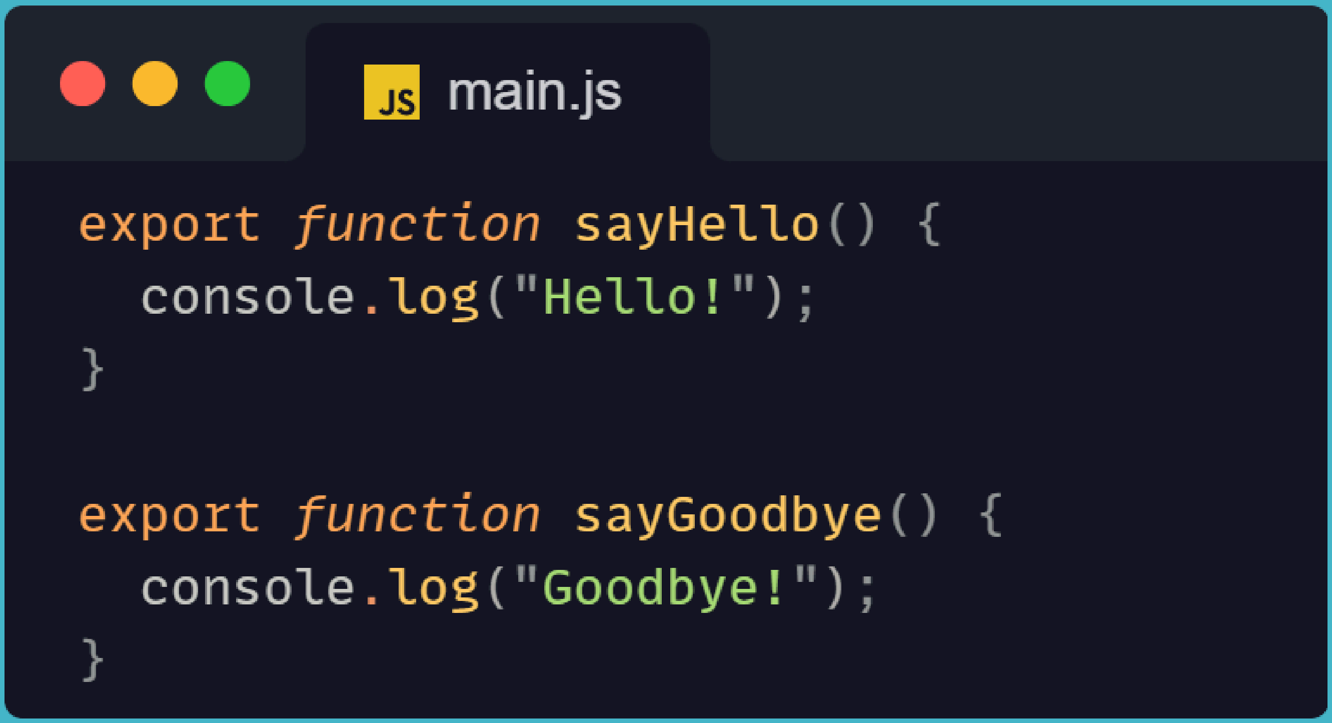
How to Use JavaScript Modules ?

Once you've enabled modules, you can use the **import** and **export** statements to share functionality between different files.

Exporting Code

You can export variables, functions, objects, or even entire classes from one file so that other files can access them.

1- Named Exports: Export multiple values by name.



main.js

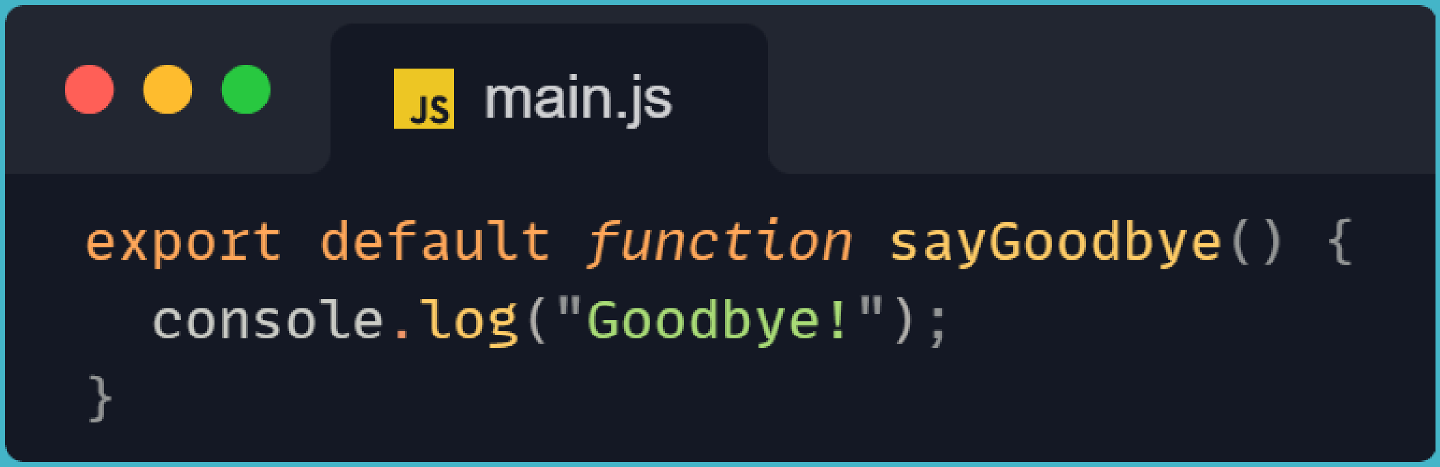
```javascript
export function sayHello() {
  console.log("Hello!");
}

export function sayGoodbye() {
  console.log("Goodbye!");
}
```

Explanation:

- Each function (**sayHello**, **sayGoodbye**) is exported by name, so other files can import them individually.

2 -Default Exports: Export a single item as the default export.


main.js

```javascript
export default function sayGoodbye() {
  console.log("Goodbye!");
}
```

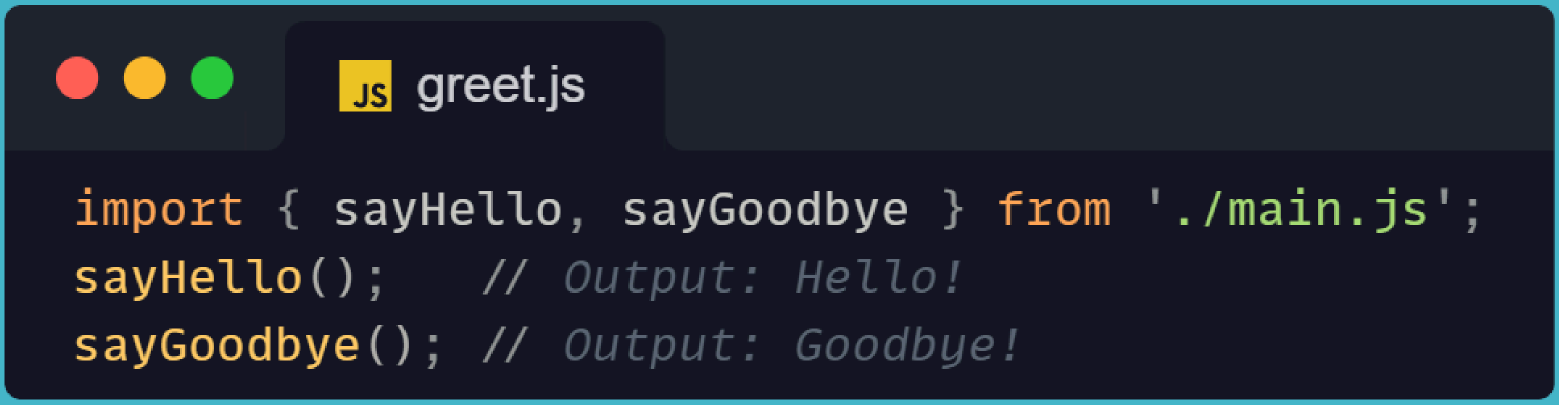
Explanation:

- export default allows you to export a **single** function, object, or class. There can only be one default export per file.

Importing Code

Once you've exported code, you can import it into other files.

1- Importing Named Exports:

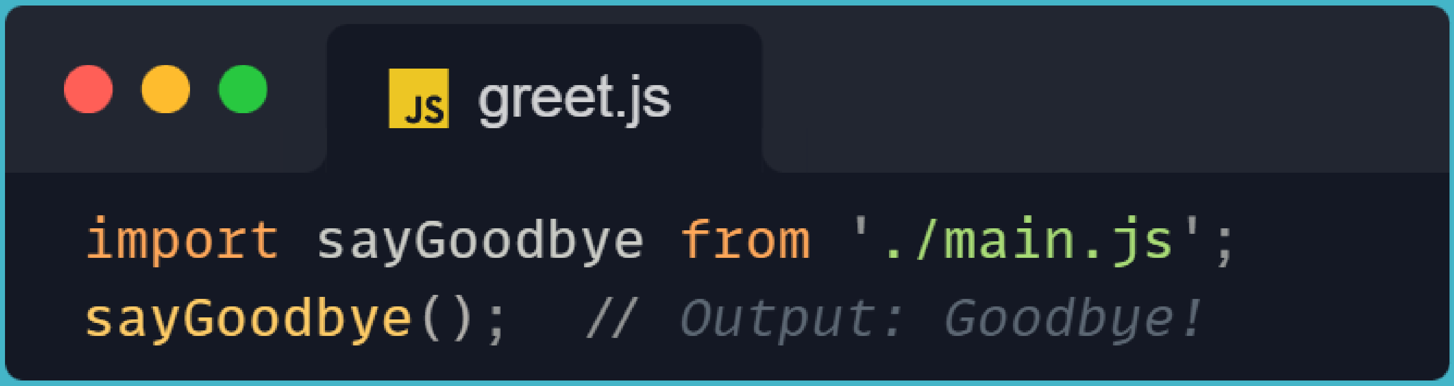


```javascript
import { sayHello, sayGoodbye } from './main.js';
sayHello();    // Output: Hello!
sayGoodbye(); // Output: Goodbye!
```

Explanation:

- The **{ }** syntax is used to import **named** exports by their exact names.

2- Importing Default Exports:

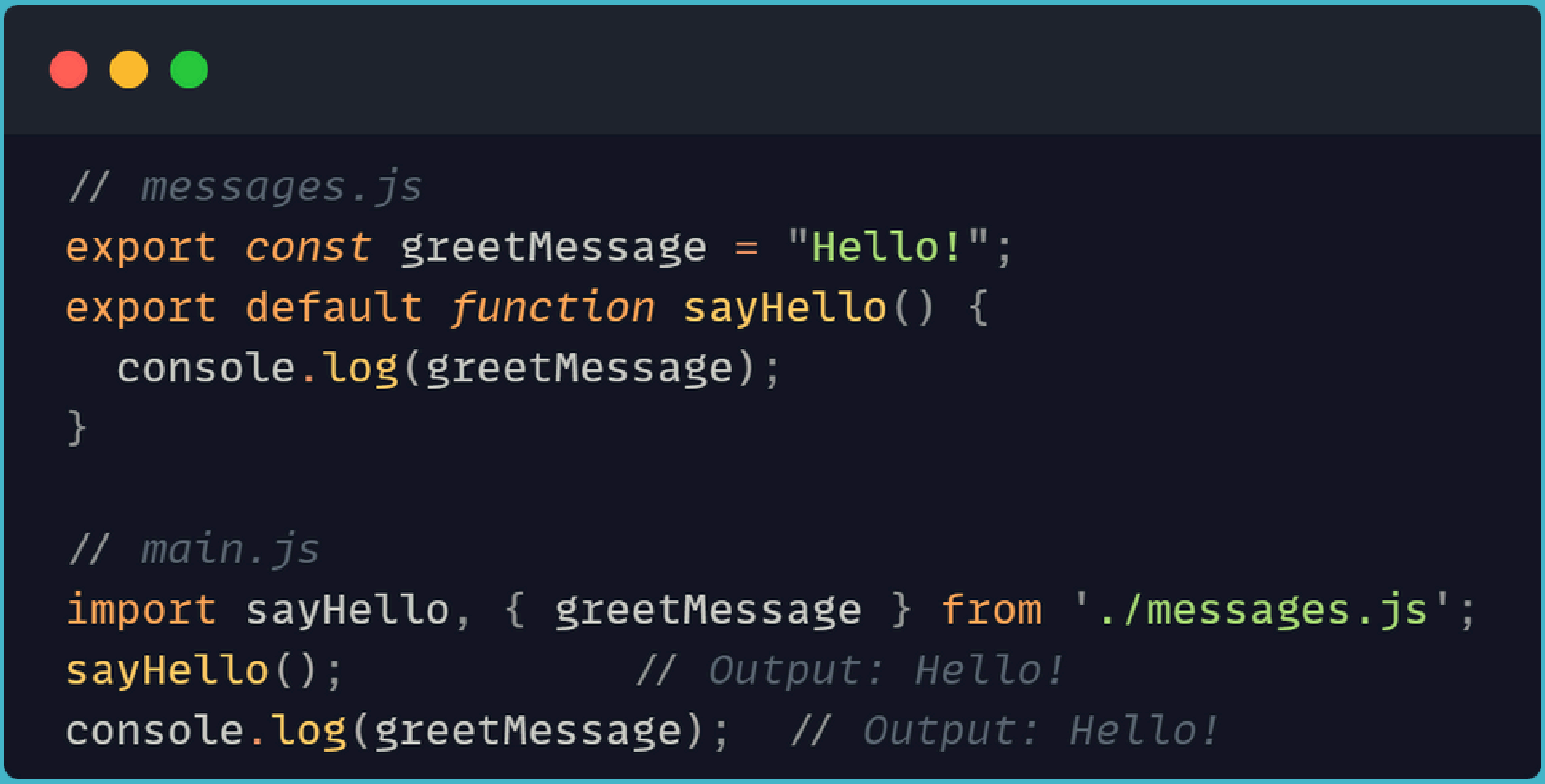


```javascript
import sayGoodbye from './main.js';
sayGoodbye();   // Output: Goodbye!
```

Explanation:

- For default exports, you **don't need** to use { }, and you can give the import **any name** you like.

3-Importing Both Named and Default Exports :



```
// messages.js
export const greetMessage = "Hello!";
export default function sayHello() {
  console.log(greetMessage);
}

// main.js
import sayHello, { greetMessage } from './messages.js';
sayHello();             // Output: Hello!
console.log(greetMessage);  // Output: Hello!
```

Explanation:

- You can import both **named exports** (greetMessage) and **default exports** (sayHello) from the same file.

Slim

Hopefully You Found It Usefull!

“Be sure to save this post so you can come back to it later”

like

Comment

Share