Modular JavaScript

- module → is a self-contained piece of code that provides functions & methods that can then be used in other files & by other modules
- Keeps code:
  - organized in separate, reusable files
  - improves code maintainability
  - loosely coupled
  - interchangeable

Coupling

- refers to how dependent certain elements or modules of code are on each other

"Two pieces of code are said to be tightly coupled if one relies on the other to run"
"two pieces of code are said to loosely coupled if one piece of code can be easily substituted by another without affecting the final outcome"

ES6 Modules

"There are a few important points about modules that are worth keeping in mind:

All code in modules is always in strict mode without the need for 'use strict' and there is no way to opt out of this.
A module has its own global scope, so any variables created in the top-level of a module can only be accessed within that module.
The value of this in the top level of a module is undefined, rather than the global object.
You can't use HTML-style comments in modules (although this isn't very common in any JavaScript program these days)"

Browser Support

Default Exports

"Default exports refer to a single variable, function or class in a module that can be imported without having to be explicitly named. The syntax for default exports is purposely easier to read because this is how modules were designed to be used"

Don't Use More Than One Default Exports

"Having more than one default export will result in a syntax error.

To import these default values, you would use the following code:

```
import PI from './pi.js';
import square from './square.js';
import stats from './stats.js';
```

The big difference with default exports is that you don't need to use curly braces or make any mention of the value that is being imported, making the statement read more elegantly"

Aliases

- the alias assigned to the imported module does not have to match its name in the actual module