# Theory: Modules



11 minutes 0 / 5 problems solved



Imagine you have a task to develop a new social network. To create one, you need to write more than a 100 000 lines of code. Of course, you can put all this code in a single file, but you will likely deal with a lot of problems supporting it. In this topic, you will learn how to better organize your code by splitting it into multiple files using modules.



## §1. What are modules?

A **module** is a single file with independent code that can be downloaded from another file. The file that uses the module only works with its result and does not know about its internal implementation. This helps maintain the program. When you want to rewrite a module, you only change its own code without affecting other parts of the program.

If you're trying to code a big website without using modules, you can write more than a thousand lines of code in a single file. When doing so, it is very easy to write hundreds of global variables and accidentally change some of them, which leads to errors. To solve this problem, you can split the logic into several functions with local variables and divide them into several files.

Some programmers like to copy and paste code that has already been written to perform the same task. This is a really bad idea because it violates the main rule of developing — "don't repeat yourself" — which means that you don't need to write the same code again with small changes. To avoid repetitions, it is much better to write the code once in a module and then re-use it when needed.

As you can see, modules have a lot of benefits. Let's see how we can create them!

## §2. Import and export

If you want to use a function from another file, in ES6 you first need to write the **export** keyword before its name.

```
// library.js
export function myBestFunction() {
  //...
}
```

Second, you have to import it to the file where you want to call it using the **import** keyword. After that, you can launch the imported function wherever you want.

```
// main.js
import { myBestFunction } from './library.js';

myBestFunction();
```

You can also rename an imported function using the **as** keyword.

```
// main.js
import { myBestFunction as myFunction } from './library.js';

myFunction();
```

Sometimes you need to use multiple functions from a single module. Instead of listing them all, you can import the entire module using the `*` symbol.

```
// main.js
import * as myLib from './library.js';

myLib.myBestFunction();
```

You can also avoid writing the function name when importing it. To do this, add the **default** keyword after `export` and import the function using any name without brackets.

```
// library.js
export default function myBestFunction() {
  //...
}

// main.js
import anyNameToFunction from './library.js';

anyNameToFunction();
```

You can only have one default export in a file, but you can combine default and non-default exports from a single file.

```
// library.js
export function myFunction() {
  //...
}

export default function myBestFunction() {
  //...
}


// main.js

import anyNameToFunction, {myFunction} from './library.js';



anyNameToFunction();



myFunction();
```

There is a big debate among the developers whether they should use exports by default. The problem is that each programmer creates their own name when importing the default function, which results in inconsistent code. To avoid this, you should name the imported function according to its file name.

```
// main.js
import library, {myFunction} from './library.js';

library();

myFunction();
```

This is how you can work with functions via modules. In real life, programmers use modules not only for functions, but also for classes, constants, and so on.

## §3. Conclusion

Modules help you encapsulate code and avoid potential errors. This is an excellent reason to use them, especially since it is quite effortless. You only need to write the `export` keyword before the function name when it is created and write the `import` keyword when you import it to another file.

Report a typo

**48** users liked this theory. **0** didn't like it. **What about you?**

Start practicing

Comments (2) Hints (0) Useful links (0) Show discussion