COMP1531

🛠️ Development - Multi-File & Importing

Lecture 2.2

Author(s): Hayden Smith

(Download as PDF)

In This Lecture

- **Why? 🤔**
  - Most software projects involve working across multiple files that need to interact
- **What? 📰**
  - Importing libraries
  - Importing our own files

📕 Importing Libraries

Similar to C, NodeJS (Javascript) has a number of built-in libraries. These libraries come with the interpret and don't need to be installed. These libraries were written by other programmers.

Examples of importing are below.

C

```c
#include <stdio.h>
```

Javascript

```javascript
import fs from 'fs';
```

📕 Importing Libraries

How we use them is different too.

C

```c
#include <stdio.h>

int main() {
  printf("Hello\n");
}
```

Javascript

```javascript
import path from 'path';
console.log(path.resolve('./'));
```

Importing Our Own Files

We're also able to import our own files into another file, allowing us to separate out logic!

```
1 function manyString(repeat, str) {
2   let outString = '';
3   for (let i = 0; i < repeat; i++) {
4     outString += str;
5   }
6   return outString;
7 }
8
9 console.log(manyString(5, 'hello '));
```

2.2_split_before.js

🦸 Importing Our Own Files

We've now split the code up into two files.

```
1 function manyString(repeat, str) {
2   let outString = '';
3   for (let i = 0; i < repeat; i++) {
4     outString += str;
5   }
6   return outString;
7 }
8
9 export default manyString;
```

2.2_split_after_lib.js

```
1 import manyString from './2.2_split_after_lib.js';
2
3 console.log(manyString(5, 'hello '));
```

2.2_split_after_main.js

We use `export default X` when we want to export a single thing.

Importing Our Own Files

We've now split the code up into two files.

```
1 function manyString(repeat, str) {
2   let outString = '';
3   for (let i = 0; i < repeat; i++) {
4     outString += str;
5   }
6   return outString;
7 }
8
9 export default manyString;
```

2.2_split_after_lib.js

```
1 import manyString from './2.2_split_after_lib.js';
2
3 console.log(manyString(5, 'hello '));
```

2.2_split_after_main.js

We use `export default X` when we want to export a single thing.

Importing Our Own Files

We've now split the code up into two files.

```
1 function manyString(repeat, str) {
2   let outString = '';
3   for (let i = 0; i < repeat; i++) {
4     outString += str;
5   }
6   return outString;
7 }
8
9 export default manyString;
```

2.2_split_after_lib.js

```
1 import manyString from './2.2_split_after_lib.js';
2
3 console.log(manyString(5, 'hello '));
```

2.2_split_after_main.js

We use `export default X` when we want to export a single thing.

But now what if we want to export multiple things?

Importing Our Own Files

For example, how do we export both functions from this file?

```
1  function manyString(repeat, str) {
2    let outString = '';
3    for (let i = 0; i < repeat; i++) {
4      outString += str;
5    }
6    return outString;
7  }
8
9  function addBrackets(string) {
10   return `(${string})`;
11 }
12
13 export default manyString; // How do we add more??
```

2.2_multi_export_lib_p_1.js

🦸 Importing Our Own Files

Instead of exporting one thing, we can export many things! **Please note, this is not exporting an object.**

```
1  function manyString(repeat, str) {
2    let outString = '';
3    for (let i = 0; i < repeat; i++) {
4      outString += str;
5    }
6    return outString;
7  }
8
9  function addBrackets(string) {
10   return `(${string})`;
11 }
12
13 export {
14   manyString,
15   addBrackets,
16 };
```

2.2_multi_export_lib_p_2.js

Importing Our Own Files

We can use **destructuring** for importing as well.

```
1  function manyString(repeat, str) {
2    let outString = '';
3    for (let i = 0; i < repeat; i++) {
4      outString += str;
5    }
6    return outString;
7  }
8
9  function addBrackets(string) {
10   return `(${string})`;
11 }
12
13 export {
14   manyString,
15   addBrackets,
16 };
```

2.2_multi_export_lib_p_2.js

```
1 import { brackets, manyString } from './2.2_multi_export_lib_p_2.j
2
3 const b = brackets('hello ');
4 const many = manyString(5, b);
5 console.log(many);
```

2.2_multi_export_main_p_2.js

Importing Our Own Files

In summary:

Default Export

```
1 function a() {
2   return 0;
3 }
4
5 export default a;
```

```
1 import a from './above.js'
```

Exporting one thing

Named Export

```
 1 function a() {
 2   return 0;
 3 }
 4
 5 function b() {
 6   return 0;
 7 }
 8
 9 export {
10   a,
11   b,
12 };
```

```
1 import { a, b } from './above.js';
```

Exporting one or more things

🤔 Other Notes About Importing

In general we try and always use named exports. This:

1. Adds `flexibility` to our code design as it allows us to add more exports later.
2. Means that users of a library by default have to reuse the same name of your function. This is not true when you return a function directly. This avoids confusion.

Fns.Js

```
1 function sum(a, b) {
2   return a + b;
3 }
4
5 export default sum;
```

Main.Js

```
1 import anyNameIWant from './fns.js';
2
3 console.log(anyNameIWant(1,2));
```

🤔 Other Notes About Importing

That being said, with named exports (i.e. an export within an object) we can still alias the name if we need to. This is especially useful if you already have functions in your file that use that name.

Fns.Js

```
1 function sum(a, b) {
2   return a + b;
3 }
4
5 export {
6   sum,
7 };
```

Main.Js

```
1 import { sum as simpleSum } from './fns.js';
2
3 function sum(a, b, c) {
4   return a + b + c;
5 }
6
7 console.log(simpleSum(1,2));
```

🤔 Other Notes About Importing

And if we're importing too many things, we can just break it up over multiple lines.

```
import {
  function1,
  function2,
  function3,
  function4,
} from './bigfile.js';
```

👂 Feedback

Or go to the form here.