JavaScript modules, imports - exports

Before you start creating the next great app, let’s explore a little more about modules.

Modules can help you to save and access your code in a more structured way, and in this reading, you'll learn about some foundational concepts of working with JavaScript modules.

This knowledge is crucial in order to understand the syntax and the logic behind how the example React apps in this course are put together.

This reading will cover the three main concepts:

1. JavaScript modules
1. Module exports
1. Module imports

JavaScript Modules

In JavaScript, a module is simply a file.

The purpose of a module is to have more modular code, where you can work with smaller files, and import and export them so that the apps you build are more customizable and have more composable parts.

A module can be as simple as a single function in a separate file.

Consider the following function declaration:

```
function addTwo(a, b) {
    console.log(a + b);
}
```

Say that you have a file named **addTwo.js** that contains only the above code.

How would you make this file a JavaScript module?

All that you would need to do to make it a JavaScript module is use the export syntax.

Module Exports

There is more than one way to export a module in JavaScript.

While all the various syntactical differences are not listed, here are a few examples that will cover all the ways that the importing and exporting of JavaScript modules will be done in this course.

In general, there are two ways to export modules in JavaScript:

1. Using default exports
1. Using named exports

Default Exports

You can have **one default export** per JavaScript module.

Using the above **addTwo.js** file as an example, here are two ways to perform a default export:

```
export default function addTwo(a, b) {
    console.log(a + b);
}
```

So, in the above example, you're adding the **`export default`** keywords in front of the **`addTwo`** function declaration.

Here's an alternative syntax:

```
export default addTwo;
```

Named Exports

Named exports are a way to export only certain parts of a given JavaScript file.

In contrast with default exports, you can export as many items from any JavaScript file as you want.

In other words, there can be only one default export, but as many named exports as you want.

For example:

```
function addTwo(a, b) {
    console.log(a + b);
}

function addThree(a + b + c) {
    console.log(a + b + c);
}
```

If you want to export both the **`addTwo`** and the **`addThree`** functions as named exports, one way to do it would be the following:

```
export function addTwo(a, b) {
    console.log(a + b);
}

export function addThree(a + b + c) {
    console.log(a + b + c);
}
```

Here's another way you could do it:

```
function addTwo(a, b) {
```

```
function addTwo(a, b) {
    console.log(a + b);
}

function addThree(a + b + c) {
    console.log(a + b + c);
}

export { addTwo, addThree };
```

Importing Modules

Just like when exporting modules in JavaScript, there are several ways to import them.

The exact syntax depends on how the module was exported.

Say that you have two modules in a folder.

The first module is **addTwo.js** and the second module is **mathOperations.js**.

You want to import the **addTwo.js** module into the **mathOperations.js** module.

Importing a Module that was Exported as Default

Consider the previous example of exporting the **`addTwo`** function as a default module:

```
// addTwo.js module:
function addTwo(a, b) {
    console.log(a + b);
}

export default addTwo;
```

To import it into the **mathOperations.js** module, you could use the following syntax:

```
import addTwo from "./addTwo";

// the rest of the mathOperations.js code goes here
```

So, you could start this import with the **`import`** keyword, then the name under which you'll use this imported code inside the **mathOperations.js** file. You would then type the keyword **`from`**, and finally the location of the file, *without the .js extension.*

Contrast the above import of the default **`addTwo export`** with the different import syntax if the **`addTwo`** function was instead a named export:

```
import { addTwo } from "./addTwo";

// the rest of the mathOperations.js code goes here
```

Conclusion

In this reading, you've learned about the very basics of what modules are in JavaScript, why they are used and how they get exported and imported.

The examples you've seen here are the core of how you'll deal with imports and exports of various modules in the example React apps on this course.

However, please note that there are many more caveats, rules, and implementations of working with modules in JavaScript. The examples given in this reading are there just to make it easier to comprehend what is happening in React apps that you'll be building in this course. The intent of this reading was just to get you familiar with the most common syntax used - not as a comprehensive overview of modules in JavaScript.