ES6 Modules

Summary: in this tutorial, you will learn about **ES6 modules** and how to export variables, functions, and classes from a module, and reuse them in other modules.

Introduction to ES6 modules

In the early days, JavaScript initially served small scripting tasks that provided interactivity to web pages. Nowadays, JavaScript has evolved to power complete applications in browsers and servers (Node.js).

To handle this growth, it is necessary to modularize JavaScript code into modules and make them reusable across applications.

ES6 introduced the concept of modules. A module is a JavaScript file that executes in *strict mode*. It means that any variables or functions declared in the module won't be added automatically to the global scope.

The good news is that modern web browsers and Node.js support native ES6 modules. This native support streamlines module loading and optimizes performance.

> ES6 modules are supported in Node. js versions 13 and above.

ES6 modules example

We'll create a new project with the following directory and file structure:

```
├── index.html
└── js
    ├── index.js
    └── lib.js
```

First, define a function called `display()` in the `lib.js` module:

```
function display(message) {
  const el = document.createElement('div');
  el.innerText = message;
  document.body.appendChild(el);
}
```

The `display()` function displays a message on the web page by creating a `div` element and appending it to the `body` element.

Second, load the `index.js` file in the `index.html` file:

```
<!DOCTYPE html>
<html lang="en">

    <head>
        <meta charset="UTF-8">
        <meta name="viewport" content="width=device-width, initial-scale=1.0">
        <title>ES6 Modules</title>
    </head>

    <body>

        <script src="js/index.js"></script>
    </body>

</html>
```

To use the `display()` function of the `lib.js` file in the `index.js` file, you use ES6 modules. The following steps illustrate how to accomplish it:

Step 1. Export the `display()` function in the `lib.js` file using the `export` statement:

```
function display(message) {
  const el = document.createElement('div');
  el.innerText = message;
```

```
  document.body.appendChild(el);
}

export { display };
```

In this example, we place the function name `display` inside curly braces after the `export` keyword. This allows the `display` function to be used in other modules.

Step 2. Import the `display` function from the `lib.js` module using the `import` statement and call the `display()` function to show the `Hi` message on the webpage:

```
import { display } from './lib.js';

display('Hi');
```

In this example, we place the `display` function name that we want to import inside the curly braces and specify the module name from which we want to import (`lib.js`).

Step 3. Add the `type="module"` to the `script` tag in the `index.html` to instruct the web browser to load the `index.js` file as a module:

```
<!DOCTYPE html>
<html lang="en">
    <head>
        <meta charset="UTF-8">
        <meta name="viewport" content="width=device-width, initial-scale=1.0">
        <title>ES6 Modules</title>
    </head>
    <body>

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

If you open the `index.html` in the web browser, you'll see the `Hi` message on the webpage.

Summary

- ES6 modules allow you to organize JavaScript files into modules.
- ES modules are JavaScript files that execute in strict mode.
- Use the `export` statement to export variables, functions, and classes.
- Use the `import` statement to import variables, functions, and classes from other modules.
- Use `type="module"` in the script tag to instruct the web browser to load a JavaScript file as a module.