✓ 100 XP

Exercise - Export and import module components

5 minutes

Let's see how to organize variable, class, interface, and function declarations into multiple modules and then use these components in code.

Note

You must use an IDE, such as Visual Studio Code, to implement modules. It is not possible to do this in the TypeScript Playground.

Export a module component

To export a module component, use the `export` keyword.

In this part of the exercise, you'll organize related functions into separate modules and then export the function declarations.

1. Open a new Visual Studio Code workspace.

2. Create a new file called **greetings_module.ts** and then add the the following function called `returnGreeting` to it. Add the `export` keyword before the function name so it is available to other modules.

   TypeScript

   ```
   export function returnGreeting (greeting: string) {
       console.log(`The message from Greetings_module is ${greeting}.`);
   }
   ```

3. Create a second file called **greetings-utilities_module.ts** and then add the following two functions, `returnGreeting` and `getLength`, to the new file. Add `export` before the `returnGreeting` function so it is available to other modules. It is not necessary to export

the `getLength` function because it is only used within the scope of the module.

TypeScript

```typescript
export function returnGreeting (greeting: string) {
    let greetingLength = getLength(greeting);
    console.log(`The message from GreetingsLength_module is ${greeting}. It
is ${greetingLength} characters long.`);
}
function getLength(message: string): number {
    return message.length
}
```

Import a module component

To use the exported components from a module, use the `import` statement. The `import` statement can take several forms depending on your objectives.

To import a single export from a module:

TypeScript

```typescript
import { <component name> } from '<module name>'
```

To rename an import, use the as keyword:

TypeScript

```typescript
import { <component name> as <new name> } from '<module name>'
```

To import the entire module into a single variable, and use it to access the module exports:

TypeScript

```typescript
import * as <variable name> from '<module name>'
```

In next part of the exercise, you'll import components from each of the two modules into a new module.

1. Create a new file called **main.ts**. This file will contain the main code of the application,

including the `import` statements.

2. Import the `returnGreeting` function from **greetings_module.ts** using the `import` keyword.

TypeScript

```
import { returnGreeting } from './greetings_module.js';         // imports a
single function in the module
```

3. If **greetings_module.ts** had contained multiple components, you could import the entire module into a single variable (for example, `allGreetingFunctions`), as shown in the following statement. You can then use the variable to access all the module exports.

TypeScript

```
import * as allGreetingFunctions from './greetings_module.js';  // imports
all exported components in the module
```

4. Try importing the `returnGreeting` function from **greetings-utilities_module.ts** using the statement `import { returnGreeting } from './greetings-utilities_module.js'`. You'll notice an error because both files contain a `returnGreeting` function and you now have a naming conflict in the global scope of **main.ts**.

5. Correct the naming conflict by assigning the second instance of `returnGreeting` a new name. Replace `{ returnGreeting }` with `{ returnGreeting as returnGreetingLength }`. You can now use `returnGreetingLength` in place of the function name in your code.

TypeScript

```
import { returnGreeting as returnGreetingLength } from './greetings-
utilities_module.js';
```

Important

If you want to run the resulting JavaScript in a web browser, you must append the **.js** file extension to the file name in the `import` statement. To learn more, see **Compiled JavaScript import is missing file extension** .

6. Now, you can use the `returnGreetings` functions in your code.

TypeScript

```
returnGreeting('Hola!')  // Displays 'The message from Greetings_module is
Hola!'
allGreetingFunctions.returnGreeting('Bonjour');  // Displays 'The message
from Greetings_module is Bonjour!'
returnGreetingLength('Ciao!');  // Displays 'The message from
GreetingsWithLength_module is Ciao! It is 5 characters long.'
```

Next unit: Exercise - Compile modules

Continue >

How are we doing? ☆ ☆ ☆ ☆ ☆