Default Module

In this lesson, you will look at what a default module is.

WE'LL COVER THE FOLLOWING

Module with Default Export

Another concept around module is that each module can have a default export. This concept is unique for each module. The syntax is the keyword `default` between `export` and the actual element to expose.

A default export opens an import without specifying an element or to loading everything with the star syntax. In other words, it allows access by importing without using the curly brackets and exporting *without* even naming the element. The main use case is exporting a single point of a module.

```
//One file named "module4.ts"
interface module4Interface { m1: string }
export default module4Interface;

// Another file
import Def from "./module4";
const defModule4: Def = { m1: "A string" }
```

The export doesn't need to export a function or a variable that has a name. It doesn't matter because it is the import that defines under which identifier the default element is referenced.

```
export default (input: number) => input * 10;
```

Avoiding Default Module

The default module exists but should be used sporadically. The default might reduce the need to point to a particular module of a package but also make it harder to refactor. The reason is that all use of a module with default may have different names.

In the following example, **line 5** and **line 8** import the same function but assign two different names. Refactoring the `nameABC` will not refactor `nameXYZ`. Not using default helps increasing consistency in the code base as well.

```
// Module1.ts
export default function () {console.log("function");}

// Import in file1.ts
import nameABC from "./Module1"

// Import in file1.ts
import nameXYZ from "./Module1"
```

If you are using the default module with the web, you may lose potential optimization in size with *tree shaking* because the default module may load more than needed.

Be cautious with default export!