Module

This lesson introduces the concept of a module.

WE'LL COVER THE FOLLOWING

How is module different from namespace?

TypeScript uses the same concept of the module as specified by ECMAScript 2015. A module is different from namespace in many ways. The first difference is that modules do not use global scope, but rather their own scope. A module can be created using the keyword `export` and it can load another module by importing using the keyword `import`.

Module requirements

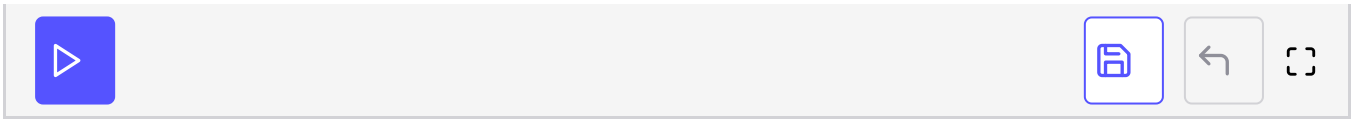
A module requires a module loader, of which there are many flavors. The most common ones are CommonJs, AMD, UMD, System, and ECMAScript. The choice of a loader depends on which system the executable (JavaScript) is at runtime. As technology improves, ECMAScript is taking the lead to handle modules.

Separation of the module

Modules are separated on a per-file basis. A single file is a single module, which means that code cannot be shared across files. Everything in a file requires the use of `export` before what it wants to expose. Otherwise, it will be private to the module. A module can contain the same content as a

namespace, i.e. interfaces, classes, types, functions, or variables. It's possible to also not specify `export` directly on the element but anywhere in the file by using `export` followed by a curly bracket and the element's name. This might look more verbose, which is true, but it's a way to export something with a different name. That's right; before closing the curly brackets, it's possible to use `as` followed by the desired name to export.

```
export const module1_variable1 = "test";
export interface module1_interface1 { m1: string; }
```



What can a module do?

A module allows exposing the code of other module as well. See this feature as a way to proxy the code from multiple modules. This way, you can import a single module and access all other module codes. To create this proxy around a module, you must use `export * from` followed by the module's name.

```
export * from "./module1";
```

When it's time to consume a module, you need to import. To do this, use the `import` keyword followed by a curly bracket and the name of the exported element. Again, it's possible when importing to rename the module by using `as` after the exported element name and before closing the curly brackets. Another option is to import everything from the module. Importing a whole module uses the syntax `import * as X from Y` where `Y` is the module and `X` holds all exports. To access an exported element, you need to use `X` dot the element.

```
import { module1_variable1, module1_interface1 } from "./module1";
import * as EverythingFromModule1 from "./module1";

console.log(module1_variable1);
console.log(EverythingFromModule1.module1_variable1);
```

Modules are everywhere. If you are using external libraries, you will use their modules. If you are sharing code with a JavaScript codebase, you will use module. Module should be the reflex when dividing piece of code. Most of the

time, a module is a single file.