Modules

- In TypeScript you can use modules to organize your code, avoid polluting the global space, and expose functionalities for others to use.
- Multiple modules can be defined in the same file. However, it makes more sense to keep on module per file
- If you want, you can split a single module across multiple files
- If you decide to split a module across different files, this is how you would do it:
  - Create the module file: `mymodule.ts` and declare your module there: `module MyModule {}`
  - Create another file: `mymodule.ext1.ts` and on top of the file add: `/// <reference path="mymodule.ts" />`. Then in the file, you can use the same name of the module and add more stuff to it: `module MyModule { // other stuff... }`
  - Then in your main file, you need two things on top of the file:
    - `/// <reference path="mymodule.ts" />`
    - `/// <reference path="mymodule.ext1.ts" />`
  - Then, you can use the name of your module to refer to the symbols defined: `MyModule.something`, `MyModule.somethingElse`
- TypeScript has two system: one used internally and the other used externally
- External modules are used if your app uses CommonJS or AMD modules. Otherwise, you can use TypeScript's internal module system
- Using TypeScript's internal module system, you can:
  - use the `module` keyword to define a module: `module MyModule { ... }`
  - split modules into different files that contribute to a single module
  - use the `/// <reference path="File.ts" />` tag to tell the compiler how files are related to each other when modules are split across files
- Using TypeScript's external module system:
  - you cannot use the `module` keyword. The `module` keyword is used only by the internal module system.
  - instead of the `reference` tag, you can use the `import` keyword to define the relationship between modules
  - you can import symbols using the file name: `import mymodule = require('mymodule')`