**TYPESCRIPT CODE SPLITTING**

There are two major approaches to perform code splitting in typescript.

1. Namespaces and File Bundling(Older approach).
2. ES6 import/exports(Modern approach).

* **Namespaces and File Bundling(**outputs single JS file**):**
  + The keyword **namespace**  is used to define a namespace.
  + The namespaces are basically utilized to bundle the entire typescript code to a single JavaScript file.
  + The code/functions inside a namespace can only be used within a namespace.
  + To export the code/functions inside a namespace we utilize the **export** keyword.
  + To utilize the namespace features in destination file, a special syntax is utilized i.e.

*/// <reference path=’file-with-namespace.ts’>*

* + The source file and the destination file content must be wrapped inside the same namespace in order to utilize the functionality.
  + It is a common practice to bundle all the javascript code to a single file while using the namespaces.
    - Set the **outFile**  property of the tsConfig to the output file e.g. ‘./dist/bundle.js’
    - Set the **module** property of the tsConfig to **AMD**.

**Problems with Namespaces:**

* + The imports and exports are harder to maintain.
  + We have to manually check that whether the file contains related imports or not.
  + It shows no error during compilation, even if some imports are missing.
  + We can remain clueless, that what goes wrong.
  + Each file should have imports that it needs.
* **ES6 Import/exports:** 
  + The keyword **import** and **export**  is used to perform ES6 import/export.
  + To Utilize the ES6 imports/exports we have to set the property **type=’module’**  on the script import in the index.html file.
    - *<script type=’module’ src=’./file-path.js’></script>*
  + The imported file must be kept with the **.js** extension instead of **.ts**.
  + We have to set the **module**  property of the tsConfig.json file to **ES6** or upper.
  + We have to assume in imports that the typescript files are already complied.
  + Export syntax:
    - *export default <function/variable/etc> (default export)*
    - *export const <function/variable/etc> (named exports)*
  + Import syntax:
    - *Import <function/variable> from ‘./file-path’ (default import)*
    - *Import {<function/variable>} from ‘./file-path (named import)*

**Problems with ES6 modules:**

* They are not support by all the older browsers.
* The imports statement must have .js file extension, even though we work in .ts files.
* A lot of HTTP requests in the browser for files.