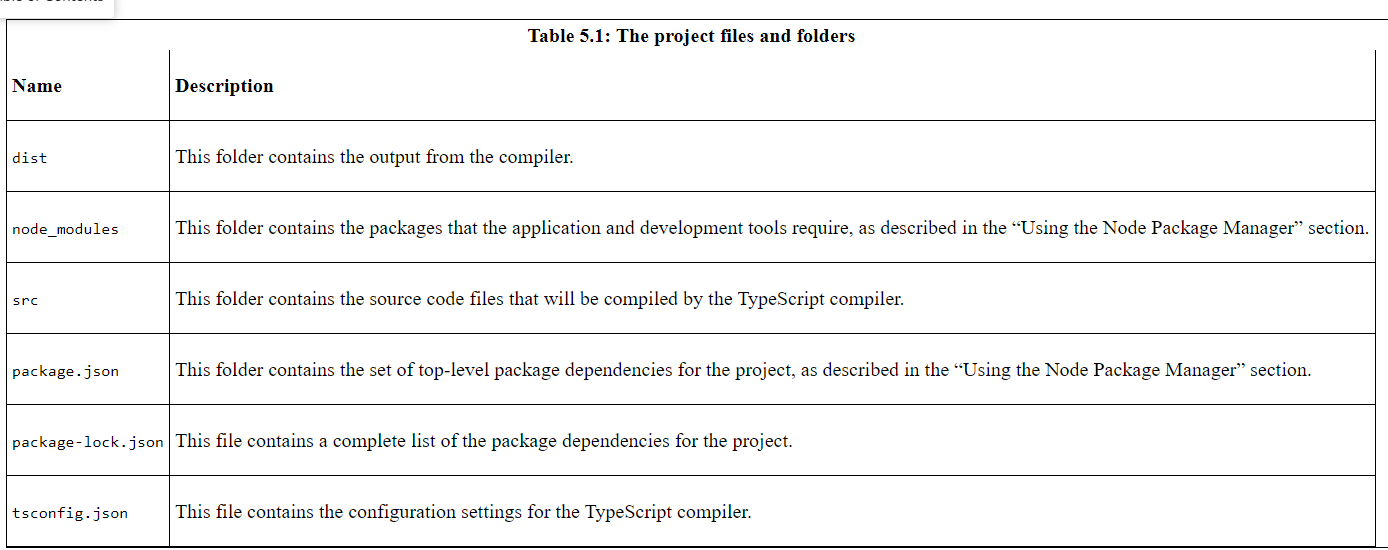
**TypeScript**

A diagram of a typewriter

Description automatically generated

* TypeScript is a superset of JavaScript by adding static type definitions.
* TypeScript helps developers to catch the errors early through type checking at compile time and it facilitates the development of large scale applications with improved code quality and maintainability.
* npm install typescript 🡪 installs typescript OR npm install –global typescript@5.5.2
* npx --package typescript tsc –init
* npx tsc –verison 🡪will give u the typescript version installed
* Create a ts file and npx tsc <filename>.ts 🡪 It will create <filename>.js
* Once js file is created we can execute in two ways:
  + Create a html and call this JS from it and run the html in browser
  + Use command 🡪 *node <filename>.js* This will execute and run the command there itself.
* public id : number 🡪 This is called *type annotation* which tells TypeScript compiler that *id* property can only be assigned *number* type. Typescript assumes all the methods and properties are public by default unless specified otherwise.
* npm install @babel/plugin-transform-modules-commonjs
* *node <filename>.js* command is used to execute javascript code as node itself is a javascript engine
* **TypeScript Compiler**:
  + npm init –yes 🡪 Creates packages.json in the folder
  + npm install –save-dev [typescript@5.0.2](mailto:typescript@5.0.2) 🡪 Here ‘—save-dev’ tells NPM that these are packages that are used for development not for application.
  + npm install –save-dev [tsc-watch@6.0.0](mailto:tsc-watch@6.0.0) 🡪 This installs tsc-watch
  + npx tsc –init 🡪 This creates tsconfig.json
    - A computer code with text

      Description automatically generated Once tsconfig.json is added run the **npx tsc** command so that ‘dist’ folder gets created.



npx tsc –watch 🡪 watches the specified .ts files (from the folder specified in tsconfig) and keeps generating js file.

npx tsc-watch –onsuccess “node dist/index.js” 🡪 watches the specified ts file and also compiles the specified js file.

**Annotations:** A close-up of a computer screen

Description automatically generated

Apart from above the typescript compiler can infer the type of the variable , like when numericValue is assigned 100 it will be inferred as number and when a string value is getting assigned an error will be shown.

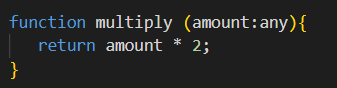
TypeScript compiler can also infer the type of a function by inferring the type the function returns based on the type it accepts.

**declaration:** A computer code with black text

Description automatically generatedThe declaration setting tells compiler to generate another file while converting ts file to js file.

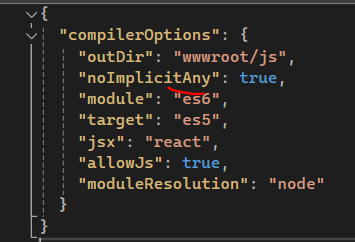
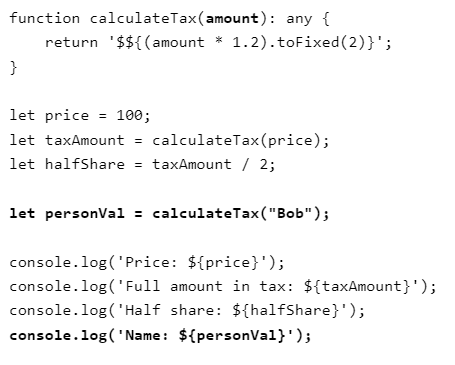
**any 🡪** This type can hold any value and any operation can be performed on it. We use this type when we want to opt-out type check. We use this when we have a value when we don’t want to type check.

**Unknown** 🡪 We can also use ‘unknown’ type as it can also hold any value but if ‘unknown’ type is used we will have to first check for type safety as it ensures type assertion or type guard before any operations are performed. We can use this type when we want to handle external data or input types only when we want to handle type checks.

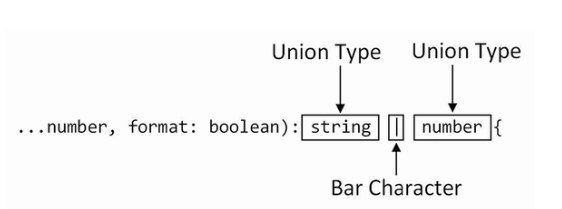
A black background with white text

Description automatically generated

**Implicit Types:** The TypeScript compiler will use *any* when it is assigning types implicitly. If we don’t want typescript to implicitly convert types then set the setting ‘noImplicitAny’ to true.

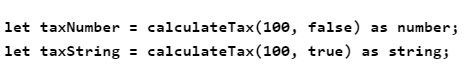


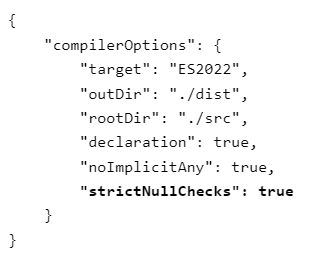
**Unions:** It is important to understand unions are handled as a type of their own i.e.. only common methods between two different types will be only be available to the ‘union’ type object.

A diagram of a stringing system

Description automatically generated

**Type Assertions:** A type assertion tells typescript compiler that to treat a value as a specific type known as *type narrowing*.



**Null Checks:** Strict null setting tells typescript compiler to not assign any null or undefined type to any other types.

**Non-Null Assertive:** A non null value is asserted by applying a ! character after the value which tells the compiler that the result will not null which allows it to assigned to either string | number type.



**Definitive Assertion:** A definitive assertion means we are sure that a particular type value is not null or undefined. This is useful when we are certain that a variable has been initialized or assigned a value before being used

A screen shot of a computer code

Description automatically generated

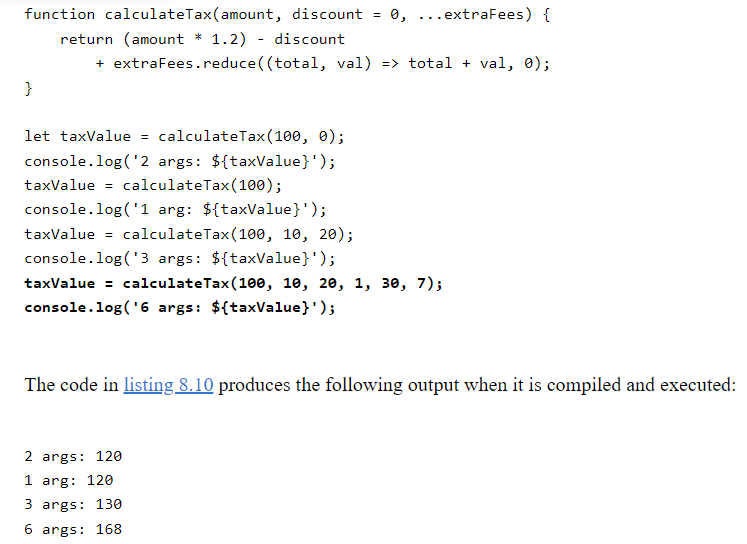
In this code , element! asserts that *element* is definitely not null, allowing us to assign to nonNullElement without typescript complaining about potential null values.

**Functions:** A diagram of a tax calculation

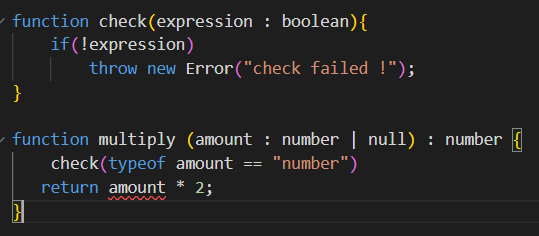
Description automatically generated with medium confidence  This tells that if discount is undefined then it will use 0.

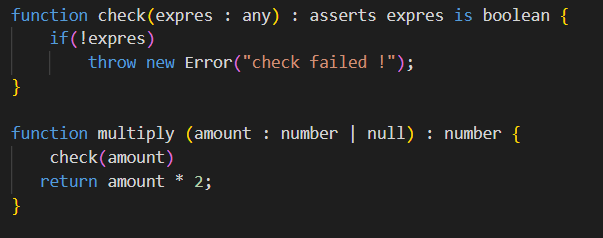
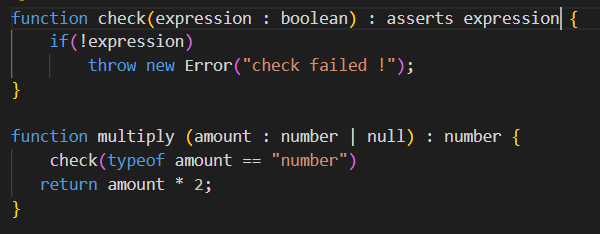
**Rest Parameter:**  It allows functions to accept variable number of arguments which are grouped and presented together.

A diagram of a function

Description automatically generated

By default typescript compiler assigns all the function parameters to be of *any* type.

**Assert Function:** An assert function evaluates an expression and throws an error if the condition is not true. A normal function can’t be used as assert function because typescript compiler will not come to know if function is assert or not and it might throw an error like below :  Hence we will use syntax “asserts expression” for the function declaration like below:



**Array:** A diagram of a number

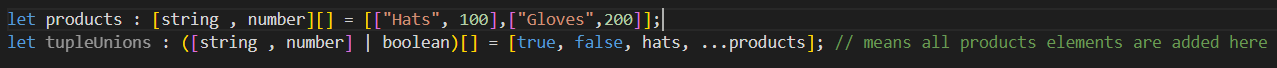
Description automatically generated

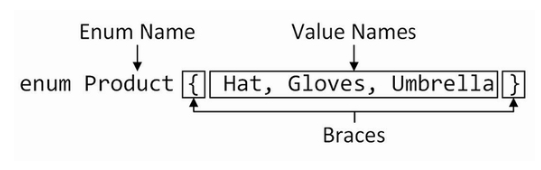
**Tuples:** These are fixed length arrays where each element can have different datatype.

Tuples are a data structure that is provided by the TypeScript compiler and implemented using regular JavaScript arrays in the compiled code.

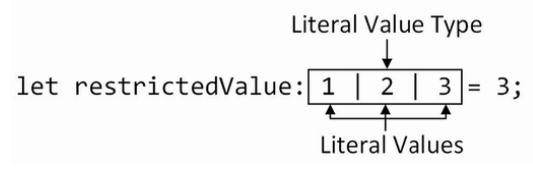
A diagram of a number of objects

Description automatically generated





**LiteralValue Type:** A literal value type specifies the set of values that can be assigned to a variable. If we assign any other value compiler will throw error.



**Type Aliases**: It can define a new type which objects can use it as data type.

type Person = {

id : string,

name : string,

age : string

}

let obj : Person = new Person { “1”,”Rajesh”,39 };

**Generic Collections:**

A screenshot of a computer

Description automatically generated