Enzigma Software Pvt. Ltd.

What is TypeScript?

Typescript is open source, strongly type , object-oriented programming language.It is developed by microsoft.It is used for developing large web based applications which is compiled not interpreted to javascript.Typescript compiler compiles ts code in javascript i.e. why it is superset of javascript.It is used to develop javascript application executed at both client side(angularJs) and server side(Nodejs).

Index

1. Difference between TypeScript and Javascript	3
2. Why to use Typescript?	3
3. What happens when we compile .ts file ?	4
4. What do we mean by Superset of JS?	5
5. Beauty of Typescript	6
6. Steps to write the First Typescript program	6
7. TypeScript- Coding Convention & Styling Guide.	7
8. Why is Typescript said to be Big?	8
9. Why not to use Typescript?	8
10. Variable Declaration in Typescript	8
11. Data Types in Typescript	9
12. Data Types in Typescript	10
13. More on TypeScript	11
14. More on Functions	15
15. Object Oriented Programming	19
16. Polymorphism	20
17. Encapsulation	21
18. Abstraction	22
19. TypeScript: Generic Types	23
20. TypeScript Module	24
21. How about directly running the typescript file	25
22. Footnotes	26
23. Assignment TypeScript	26

1. Difference between TypeScript and Javascript

- Typescript is known as Object oriented programming language whereas JavaScript is a scripting language.
- TypeScript has a feature known as Static typing but JavaScript does not have this feature.
- TypeScript gives support for modules whereas JavaScript does not support modules.
- TypeScript has Interface but JavaScript does not have Interface.
- TypeScript support optional parameter function but JavaScript does not support optional parameter function.
- TypeScript gives you error at the compile time itself which is more safer as compared to Javascript.

2. Why to use Typescript?

TypeScript always point out the compilation errors at the time of development only. Because of this at the run-time the chance of getting errors are very less whereas JavaScript is an interpreted language.

Example :

```
var SSN_Number= 1234567;
SSN_number  = '123456';
```

Javascript doesn't throw error can not assign string to number at compile time.

TypeScript has a feature which is strongly-typed or supports static typing. That means Static typing allows for checking type correctness at compile time. This is not available in JavaScript.

Example :

```
var my_Mobilenumber = 1234567;
my_Mobilenumber  = '123456' / ‘xyz’;
```

typescript throw error can not assign string to number at compile time.

TypeScript is nothing but JavaScript and some additional features i.e. ES6 features. It may not be supported in your target browser but TypeScript compiler can compile the **.ts** files into ES3,ES4 , ES5 and ES6 also.

Features of ES6(ECMAScript):

ES6 refers to version 6 of the ECMAScript programming language. ECMAScript is the standardized name for JavaScript. Following are some major changes that ES6 brings to JavaScript.

Constants:
Example : const price = 4.0;

price = 3.5;

TS throws error as :

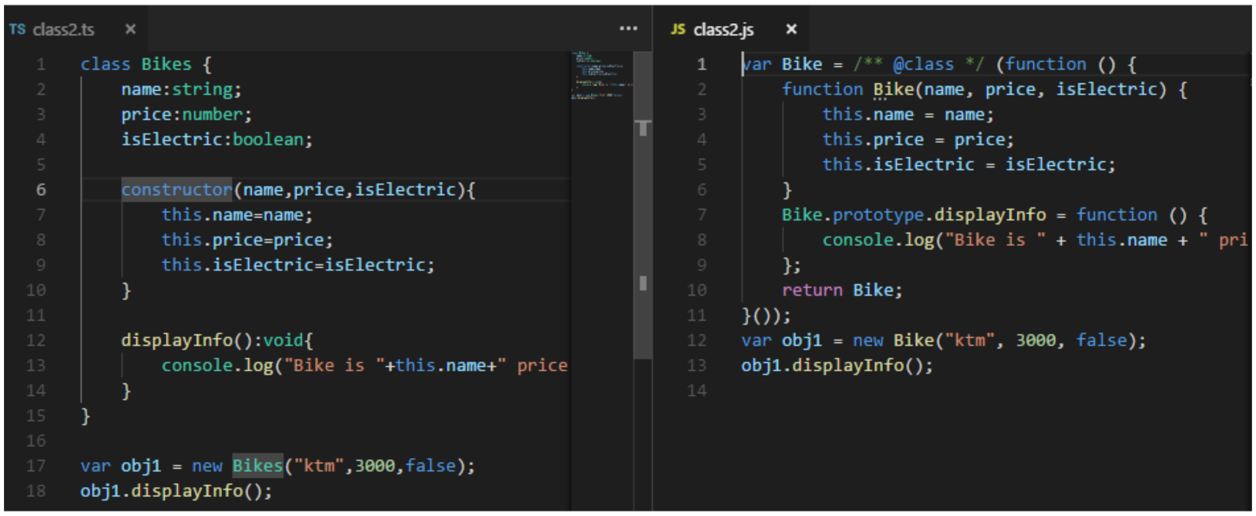
Cannot assign to 'price' because it is a constant or a read-only property.

Block-Scoped variables and functions.

```
let me = "Enzigma";

{     const number1 = 2.9;

try {

let number1 = 2.9;

 let me = "Enzigma Software Pvt Ltd"; console.log("inner block me: " + me);

        }

 catch (error) { }

 console.log("outer block me = " + me);
```

3. What happens when we compile .ts file ?

Below attached is screenshot of Typescript code which is converted into Javascript, as we can see that the converted Javascript is fully ready for usage.



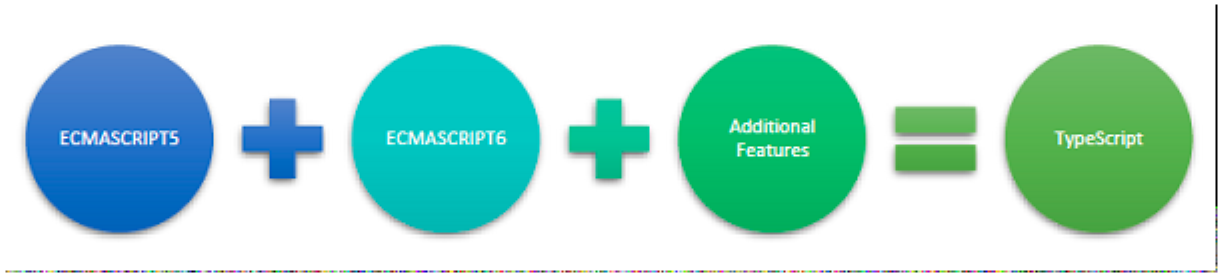
TS class2.ts ×

```
class Bikes {
    name:string;
    price:number;
    isElectric:boolean;

    constructor(name,price,isElectric){
        this.name=name;
        this.price=price;
        this.isElectric=isElectric;
    }

    displayInfo():void{
        console.log("Bike is "+this.name+" price
    }
}

var obj1 = new Bikes("ktm",3000,false);
obj1.displayInfo();
```

JS class2.js ×

```
var Bike = /** @class */ (function () {
    function Bike(name, price, isElectric) {
        this.name = name;
        this.price = price;
        this.isElectric = isElectric;
    }
    Bike.prototype.displayInfo = function () {
        console.log("Bike is " + this.name + " pri
    };
    return Bike;
}());
var obj1 = new Bike("ktm", 3000, false);
obj1.displayInfo();
```

4. What do we mean by Superset of JS?



In this case, it contains the Features of (ECMASCRIPT 5 + ECMASCRIPT 6), along with that it also contains the Additional Features which make Typescript.

5. Beauty of Typescript

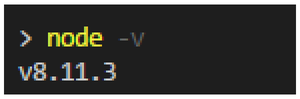
- Portable, typescript can be run on any browser
- Runs on both (Client + Server Side)
    - Typescript can be written at both the client and server side as well.
- Similar Syntax like other Languages
    - Typescript shares a similar syntax with Java, Scala.
- DOM Manipulation
    - TypeScript can be used to perform DOM Manipulation just like Javascript.
- TypeScript gives the Object Oriented Goodness
    - Typescript provides the four pillars of Object Oriented Programming.
        - Encapsulation
        - Polymorphism
        - Inheritance
        - Abstraction
- Code written in Typescript is more readable
- Code written in Typescript is less error prone.
- TypeScript is more easy to maintain
- Static Type
    - Code Written in Static Type is more robust and better
- Best of Best of Both World (Static & Dynamic Type)
    - In this case, Typescript allows us to have the best of both the worlds in which we can use typescript to be static typed or dynamically typed.

6. Steps to write the First Typescript program

Before writing the program let's ensure that the prerequisite are installed and configure on the machine. Make sure Node is installed on the machine, along with it we also need tsc which is typescript compiler.

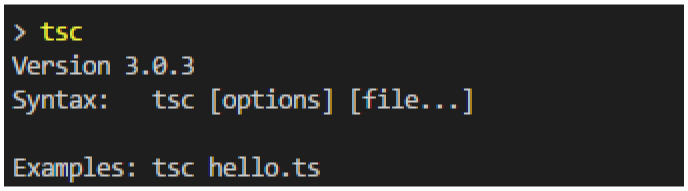
Verify that the node & tsc compiler are present, by running the below commands:

node -v



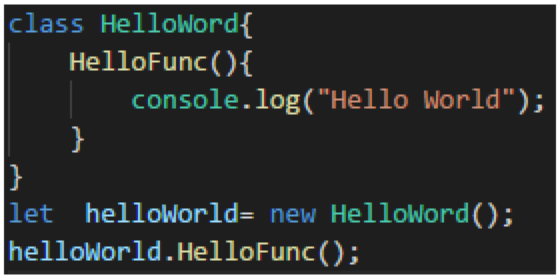
```
> node -v
v8.11.3
```

tsc



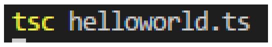
```
> tsc
Version 3.0.3
Syntax:   tsc [options] [file...]

Examples: tsc hello.ts
```

Below is a screenshot of Hello World Program.



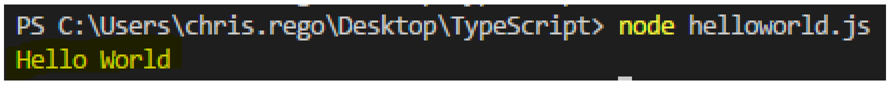
```
class HelloWord{
    HelloFunc(){
        console.log("Hello World");
    }
}
let  helloWorld= new HelloWord();
helloWorld.HelloFunc();
```

To Compile the program, we to enter tsc followed by the name of typescript file.



```
tsc helloworld.ts
```

To run the .js file we need to type node followed by the filename.js



```
PS C:\Users\chris.rego\Desktop\TypeScript> node helloworld.js
Hello World
```

7. TypeScript- Coding Convention & Styling Guide.

TypeScript has certain coding convention which needs to be followed.

lower camelCase is when the each word in the middle of the phrase begins with a capital letter. camelCase is used when we have multiple words combined as one.
Example:

PascalCase is subset of camelcase in this case the first letter of each word that is combined is Uppercase. Example-

Typescript Convention & Styling-

- Variable- camelCase
- Function- camelCase
- Class- PascalCase
- Interface- PascalCase (Don't prefix with I)
- Namespace- PascalCase
- Enum- PascalCase
- Null VS Undefined- Encourage the use of undefined rather than null
- Filename- camelCase

8. Why is Typescript said to be Big?

TypeScript is used by Major Tech Companies, because of the goodness that it gives with it, Typescript has the likes of Microsoft, as well as also adopted by Google.TypeScript also is used to develop Angular. (Google Venture).
Biggest Breakthrough of Typescript was with the help of Angular 2 itself.

9. Why not to use Typescript?

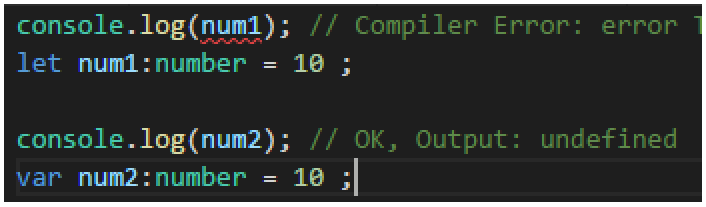
- Generally TypeScript takes time to compile the code
- TypeScript cannot be directly used in your web browser as how Javascript works.

10. Variable Declaration in Typescript

Variable Declaration in Typescript is possible, The two ways in which variables can be declared are.

- Var declaration
- Let declaration

Let Declaration is where the scope is limited to the block level cannot be redeclared.



```
console.log(num1); // Compiler Error: error T
let num1:number = 10 ;

console.log(num2); // OK, Output: undefined
var num2:number = 10 ;
```

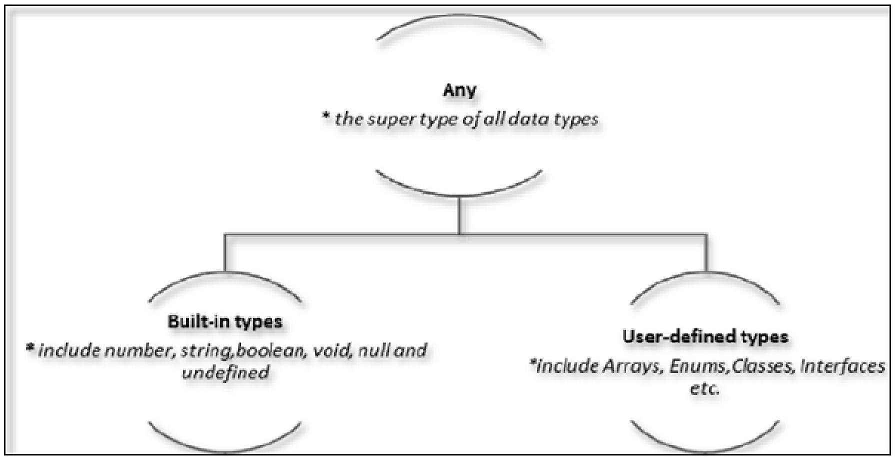
They cannot be accessed until they are declared yet.

11. Data Types in Typescript

Essence of typescript is the heavy usage of type in typescript. Typescript generally contains couple of Data types which are present.

We can categories the types into two types majorly.

- Built-in Type.
- User-Defined Type.


Any
* the super type of all data types
Built-in types
* include number, string,boolean, void, null and undefined
User-defined types
*include Arrays, Enums,Classes, Interfaces etc.

Built-in Types	There are 6 total built-in Data types in Typescript. ● any ● Number ● String ● Boolean ● Void

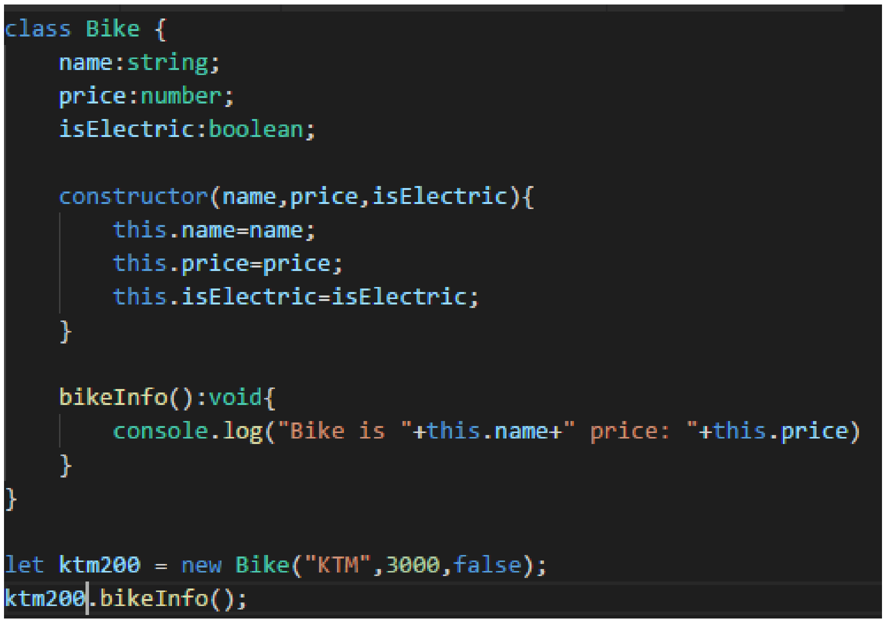
	● Null ● Undefined

User Defined	User-Defined Includes the following: 1. enum 2. Class 3. Interface 4. Array 5. Tuple

12. Data Types in Typescript

Another great part of Typescript which makes it great it replacement for JavaScript is that it has all the Object Oriented Goodness in it. Primarily Typescript support the Classes, Object, Interface etc

Classes-
Classes is blueprint of the Object. In this case, that basically shows that how a Object would look like.class keyword is used to declare the class in typescript .Class can hold Constructor, Methods & properties.



```
class Bike {
    name:string;
    price:number;
    isElectric:boolean;

    constructor(name,price,isElectric){
        this.name=name;
        this.price=price;
        this.isElectric=isElectric;
    }

    bikeInfo():void{
        console.log("Bike is "+this.name+" price: "+this.price)
    }
}

let ktm200 = new Bike("KTM",3000,false);
ktm200.bikeInfo();
```

Object-
Object is an instance which contains set of key value pairs. The values can be scalar values or functions or even array of other objects. Objects are instances of Class In this case, the we have Class Bikes which is blueprint of the object that we have created ktm200 In this case, ktm200 is instance of class Bike

13. More on TypeScript

TypeScript is Case-Sensitive-

TypeScript is basically case-sensitive which means that it differentiates between Uppercase, Lowercase letters.

TypeScript Semicolons are optional-

Each line of Instruction forms to be statement.Semicolons are optional in Javascript and the same applies in TypeScript as well.

TypeScript Comments-

In this case, of TypeScript we can comment the code using two methods.

- Single Line Comment // Single Line Comment
- Multi-Line Comment /* Multi-LIne Comment */

TypeScript Type Checking-

In this Typescript, is a way in which before processing the type is being checked in this case. By doing so it ensures that code is reliable and works as it's expected to run.

TypeScript Prevention better than cure-

In this case what that basically means is that errors received at the compiled time are far more better than a the error which is received at runtime. Typescript compiler does a great job in going ahead and Verifying the code which is written in typescript and by doing so it ensures that if there are errors or conflicts which occur.

TypeScript Number is number-

Number Class is wrapper for the primitive type number. number which is present is used to present all the numeric type of values which include. (negative , positive, floating point... etc). In this case, when we declare a number by default the type of number is 64 bit floating point number.

TypeScript String is string-

String Class is wrapper for the primitive type string Data type . They are used to represent collection of , String Class has its own set of methods that can be used for String manipulation.

TypeScript Array-

Array is used to define Collection of Elements of the same type. We can store Data with the help of variables, but if we want to store ‘n’ of elements data we can use Array.Array can be edited once created but they cannot increase in size.

TypeScript Null & Undefined-

TypeScript has ways to express values for variables but they aren’t data type.

- **Null**- Null is Object , which defines there is no value.
- **Undefined**- Undefined is when variables is declared, but value is not defined.

TypeScript Keywords-

TypeScript has number of keywords which are present the same keywords cannot be used for naming of variables or functions or

break	as	any	switch
case	if	throw	else
var	number	string	get
module	type	instanceof	typeof
public	private	enum	export
finally	for	while	void
null	super	this	new

in	return	true	false
any	extends	static	let
package	implements	interface	function
new	try	yield	const
continue	do	catch	

TypeScript Lambda Function-

TypeScript Also support Lambda Function => Arrow Function.
Example: Arrow functions.
 let add = (a,b) => a + b;
add(4,5) // returns 9

TypeScript: Superset of JS

TypeScript as already discussed has most the Javascript function but .

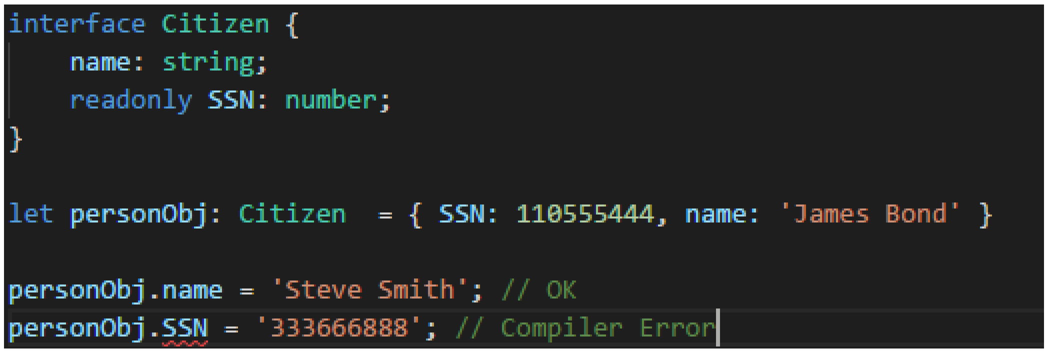
TypeScript: Operators

TypeScript has many operator which perform a specific set of action on the data. Data on which the operation is performed are operands. Operators are 8 types.

Type of Arithmetic	Arithmetic operators + - / * % Logical operators &&(And) || (OR) ! (Not) Relational operators > < >= <= == != Bitwise operators & | ^ ~ << >> >>> Assignment operators = Conditional operator ? expression1 : expression2 String operator string Type Operator typeof Instanceof Operator instanceof Negation Operator -

TypeScript: ReadOnly

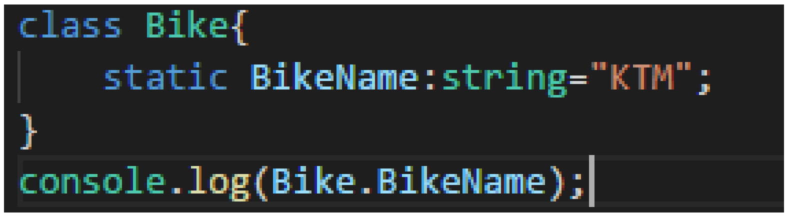
Typescript allows to set a property as read only which means This means that once a property is assigned a value, it cannot be changed!



```
interface Citizen {
    name: string;
    readonly SSN: number;
}

let personObj: Citizen  = { SSN: 110555444, name: 'James Bond' }

personObj.name = 'Steve Smith'; // OK
personObj.SSN = '333666888'; // Compiler Error
```

TypeScript Static

Typescript also support static data members in the class than can be created using the static keyword and accessed using Class Name & dot notation.



```
class Bike{
    static BikeName:string="KTM";
}
console.log(Bike.BikeName);
```

TypeScript Access Modifiers

TypeScript Access Modifier are used to restrict the access & make changes in the visibility of the code.

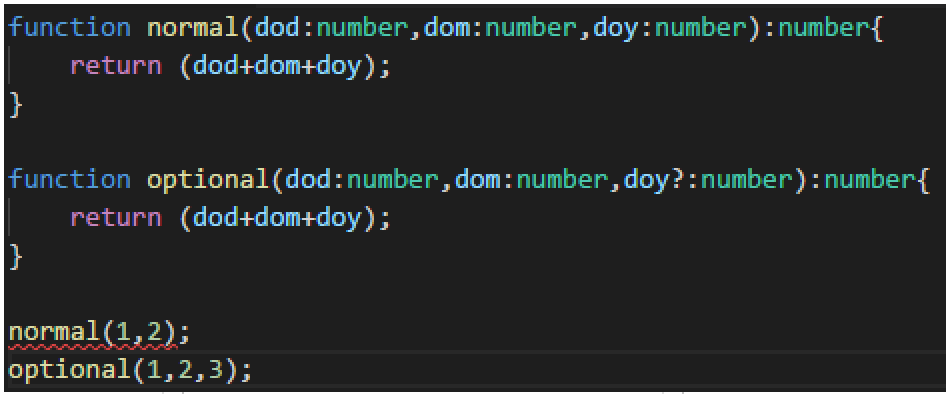
- Public- By Default access type is public for the data members in the class
- Private- Only Accessible Inside of the class and not outside.
- Protected- Only Accessible Inside of the class and only to the Derived Class which is present.

14. More on Functions

What is Functions?

Imagine, having this to love 40 lines of which does some amazing task, now in this case, i want to these same lines of codes again to be reused, Alright let's copy paste the code.!! Naah!! That would create millions lines of code if i'm required the reuse the same code.

Functions to the rescue...



```
function normal(dod:number,dom:number,doy:number):number{
    return (dod+dom+doy);
}

function optional(dod:number,dom:number,doy?:number):number{
    return (dod+dom+doy);
}

normal(1,2);
optional(1,2,3);
```

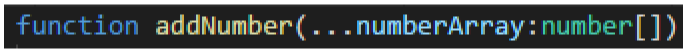
In this case, here how a declaration of Function looks like

Syntax:
function func_name (param1: typeofparam1, param2 :typeofparam2) : returntype
{
//body + logic
   return return_value;
}

Rest Parameters

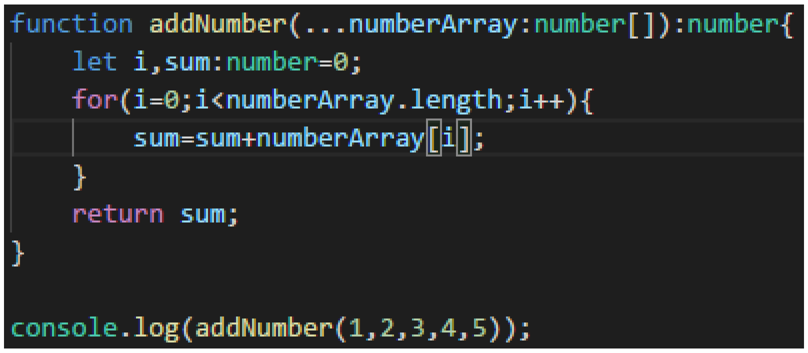
Rest parameters are more of the implementation of that of the var-args.
In this case we can pass any number of arguments but we need to make sure that the number of arguments passed need to be of the same type.

The Syntax of the function involves ... three dots having the name of identifier and the followed by the data of the array.



```
function addNumber(...numberArray:number[])
```

In this case, the addNumber is function in which we have used and array of numbers in which we have the freedom to specify how much can we provide.



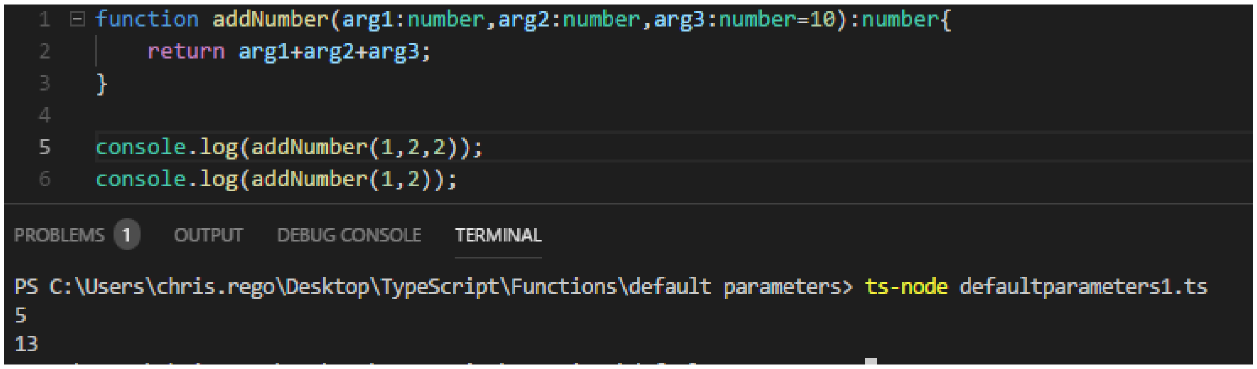
```
function addNumber(...numberArray:number[]):number{
    let i,sum:number=0;
    for(i=0;i<numberArray.length;i++){
        sum=sum+numberArray[i];
    }
    return sum;
}

console.log(addNumber(1,2,3,4,5));
```

Default Parameters

While Calling a function, we can certainly pass values to the function when needed, by passing values accordingly.On the other hands, Default Parameters we can specify default values to the functions without specifying them in the function call itself. If we don’t specify any value as arguments to the function where a default parameters are already present in that case, it goes ahead and applies the default values to the function. But on the other hands, we are passing a value where a default value is already present, then in that case the value which is passed overrides the default value which is present.

Let's Consider and example below,
The first instance when we call a function in that case, we specify all the three parameters what happens is that it overrides the default value and places the value which is present which in this case is 2.

The Second instance when we call a function in that case, we don’t specify the third parameter in this case, the default value which is present is used.
In this case which is 10



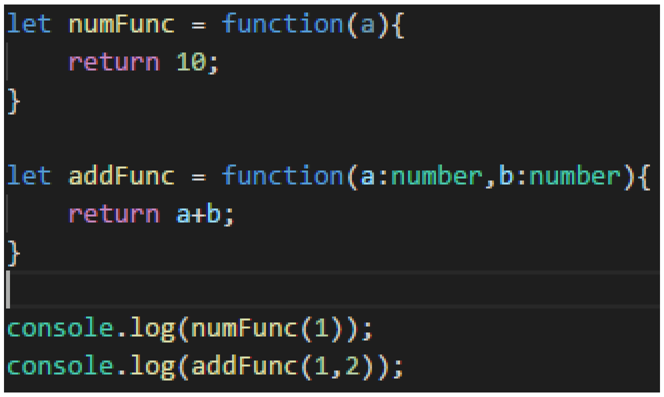
```
function addNumber(arg1:number,arg2:number,arg3:number=10):number{
    return arg1+arg2+arg3;
}

console.log(addNumber(1,2,2));
console.log(addNumber(1,2));
```

```
PS C:\Users\chris.rego\Desktop\TypeScript\Functions\default parameters> ts-node defaultparameters1.ts
5
13
```

Anonymous Functions

Anonymous Function, don't have function name on which they are directly accessible with the help of Anonymous Function. Anonymous can certainly be assigned to variable, So that with the help of the variable we can access the function later.

Example of Anonymous Function-



```
let numFunc = function(a){
    return 10;
}

let addFunc = function(a:number,b:number){
    return a+b;
}

console.log(numFunc(1));
console.log(addFunc(1,2));
```

In this case, we haven't declared in both the case the name of the function but rather we have called the function directly and assigned reference to the variables.
By using the variable names we are able to access the function indirectly.

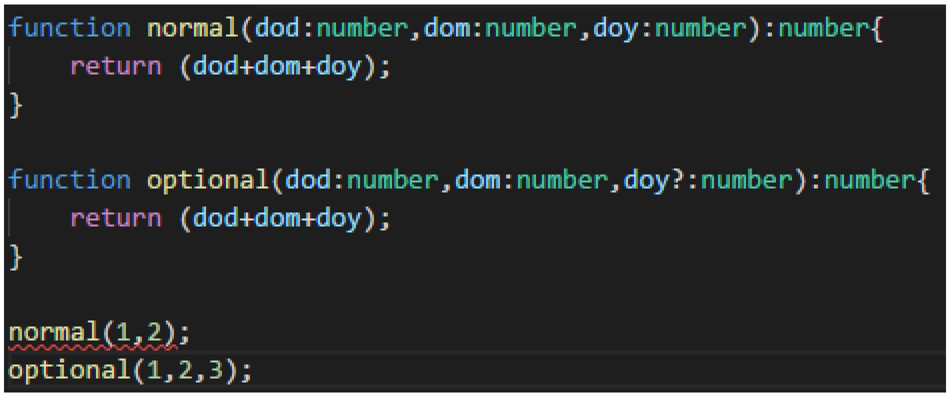
In order to access the function we have used the variables and used the function call expression ( ) , also by passing values to it.

Recursive Functions

Function that calls itself repeatedly a certain number of types until a specified condition is met to obtain

Optional Parameters

In this case, there are times when we don't want all the parameters of the function to be passed compulsorily.
This can be achieved using Optional parameters , In this case



```
function normal(dod:number,dom:number,doy:number):number{
    return (dod+dom+doy);
}

function optional(dod:number,dom:number,doy?:number):number{
    return (dod+dom+doy);
}

normal(1,2);
optional(1,2,3);
```

In this case, as we have two functions which are present one is a normal function so we need to compulsory pass all the params which are present.
But on the other we, have the optional function in which we have optional parameters specified in this are not always necessary when running the calling the function.
While in the normal case the function needs to have the parameters passed to it.

Function Overriding

In this case, we can go ahead and perform Function Overloading in which there are functions with the same name in the same scope by unique signature.

Signature consist of function name , number of args and type of args.

15. Object Oriented Programming

Inheritance

Inheritance, is one of the four pillars of Object Oriented Programming in this case, the Inheritance can be used in which one class inheritance the features from the other class. Inheritance helps to extend and make the code reusable among class. Implement Inheritance is present in TypeScript is easier and requires fewer lines of code, in comparison to it's compiled version of Javascript.

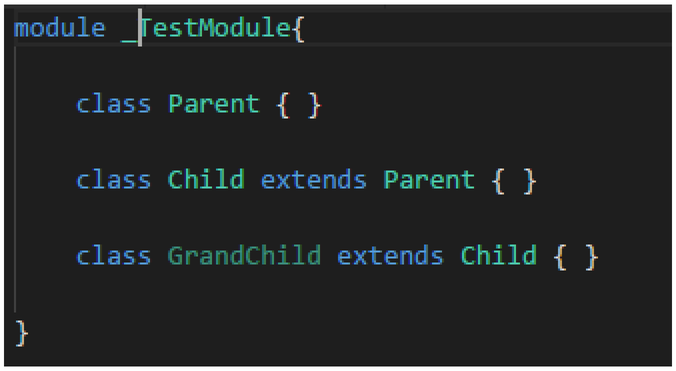
Inheritance goes ahead and helps us to get the properties & Methods of the Class. Inheritance basically has two classes, A class from which the features/methods are inherited from and the class which inherits the properties and methods.

Base Class/Parent Class-
Parent Class is the class which goes ahead & get inherited and provides the methods/features to the Child Class.

Derive Class/Child Class-
Derived Class which is present is used to go ahead and inherit features/methods from the Parent Class.

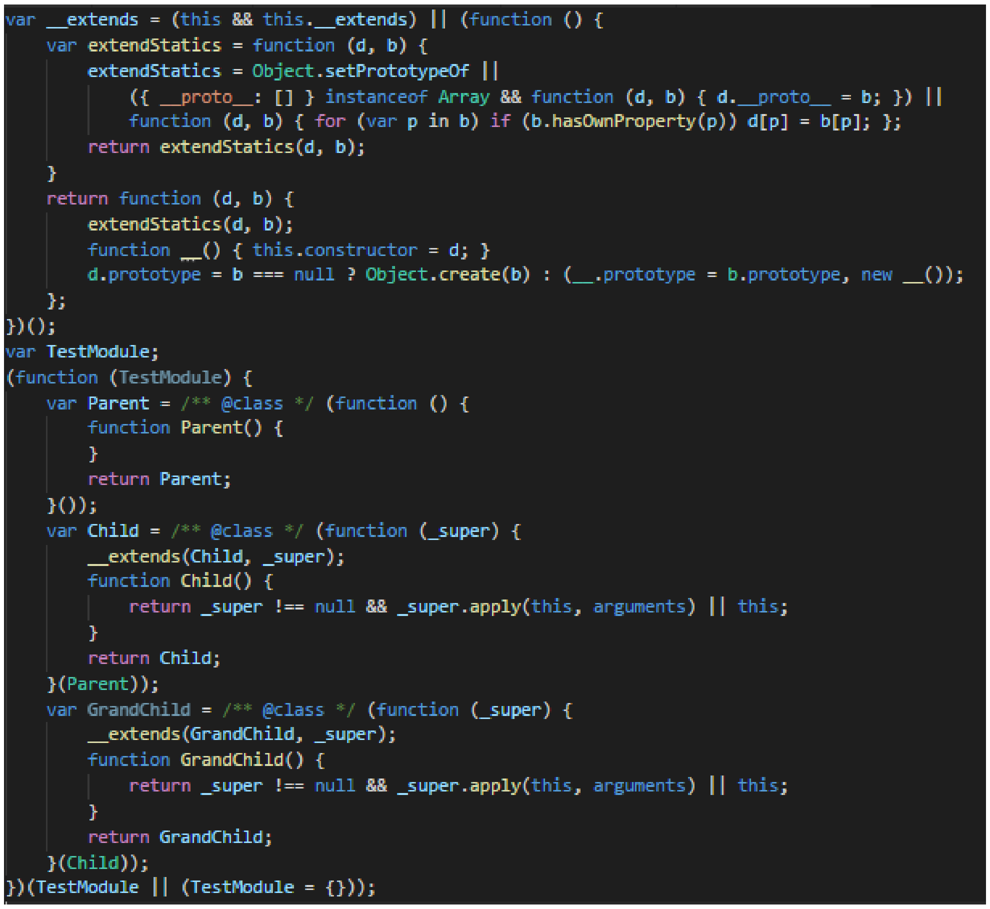
By Implementing Inheritance the child class, has access to the properties/methods of that of a parent/base class from which it is derived from.Base Class doesn't have access to thee Child Class features/methods. Inheritances implement when child 'extends' from the base class.

Here we can see it only takes a few lines code to implement Inheritance in TypeScript while the same isn't in the case of Javascript.As the Compiled version of TS which .js is more than 35 lines...



```
module _TestModule{

    class Parent { }

    class Child extends Parent { }

    class GrandChild extends Child { }

}
```

35 Lines of Equivalent code compiled into Javascript

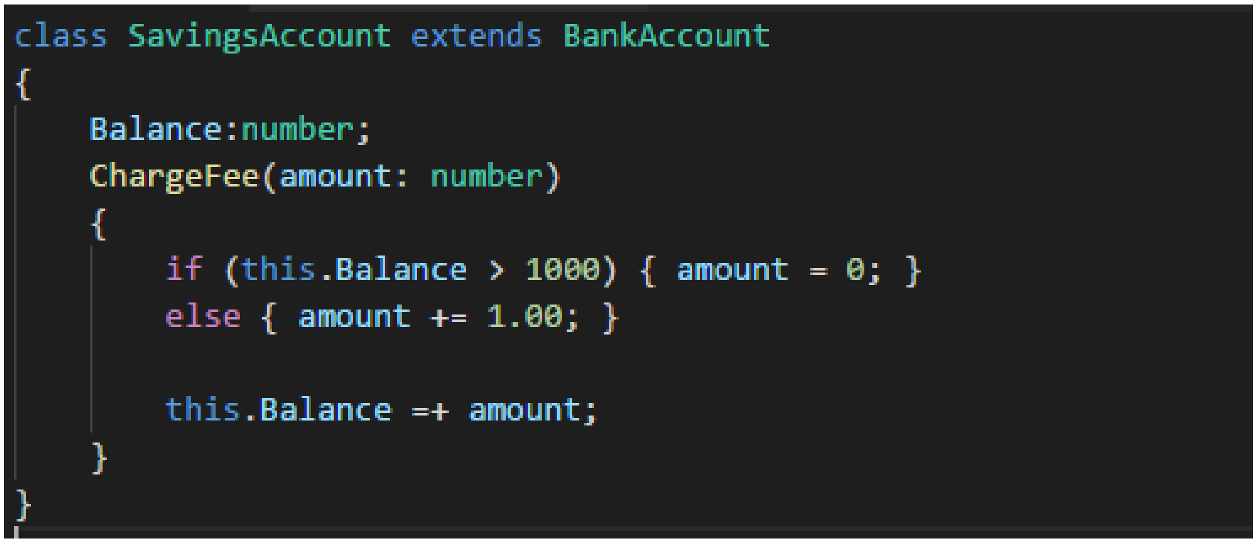


```
var __extends = (this && this.__extends) || (function () {
    var extendStatics = function (d, b) {
        extendStatics = Object.setPrototypeOf ||
            ({ __proto__: [] } instanceof Array && function (d, b) { d.__proto__ = b; }) ||
            function (d, b) { for (var p in b) if (b.hasOwnProperty(p)) d[p] = b[p]; };
        return extendStatics(d, b);
    }
    return function (d, b) {
        extendStatics(d, b);
        function __() { this.constructor = d; }
        d.prototype = b === null ? Object.create(b) : (__.prototype = b.prototype, new __());
    };
})();
var TestModule;
(function (TestModule) {
    var Parent = /** @class */ (function () {
        function Parent() {
        }
        return Parent;
    }());
    var Child = /** @class */ (function (_super) {
        __extends(Child, _super);
        function Child() {
            return _super !== null && _super.apply(this, arguments) || this;
        }
        return Child;
    }(Parent));
    var GrandChild = /** @class */ (function (_super) {
        __extends(GrandChild, _super);
        function GrandChild() {
            return _super !== null && _super.apply(this, arguments) || this;
        }
        return GrandChild;
    }(Child));
})(TestModule || (TestModule = {}));
```

Screenshot of javascript file which is generated from typescript.

16. Polymorphism

Polymorphism is TypeScript enables polymorphism via method overrides as you can see in the example below. The Withdraw and Deposit methods of the CheckingAccount and Savings Account classes derive from the parent BankAccount class. In the child classes, we can override these methods and add our own business logic customizations, such as waiving a fee if the balance is more than a specified amount.

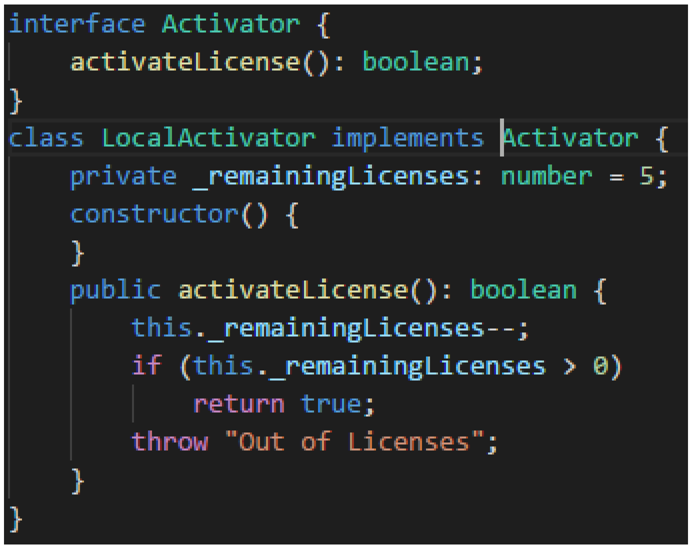


```
class SavingsAccount extends BankAccount
{
    Balance:number;
    ChargeFee(amount: number)
    {
        if (this.Balance > 1000) { amount = 0; }
        else { amount += 1.00; }

        this.Balance =+ amount;
    }
}
```

In this case, we override the Charge Fee that was originally part of the Fee interface. Polymorphism is an essential characteristic of OO programming, as code would be quite redundant without it.

17. Encapsulation

The concept of encapsulation in OOP allows us to define all of the necessary members for an object to function while hiding the internal implementation details from the application using the object. This is a very powerful concept for us to use because it allows us to change the internal workings of a class without breaking any of its dependent objects. This could come in the form of either private members or private method implementations. The contract that has been established remains the same and the functionality that it provides is consistent, however, improvements are able to be made. In the following code segment, you can see how hiding certain members from the calling application will be beneficial. Below is an Example of Encapsulation



```
interface Activator {
    activateLicense(): boolean;
}
class LocalActivator implements Activator {
    private _remainingLicenses: number = 5;
    constructor() {
    }
    public activateLicense(): boolean {
        this._remainingLicenses--;
        if (this._remainingLicenses > 0)
            return true;
        throw "Out of Licenses";
    }
}
```

18. Abstraction

In this case, Abstraction is concept in Object Oriented Programming whereby a class can specify how an inherited class should implement the class itself, including any abstract methods specified.The concept behind abstraction is that we are hiding the complexity of the implementation from inherited classes and we allow them to implement the functionality themselves.In TypeScript abstraction can be achieved by using the abstract keyword - which can be applied to both classes and methods specified in classes.

Let's go ahead and create an abstract class with an abstract method as well-

```
abstract class Warrior {
  readonly name: string;
  public weapon: string;
  constructor(name: string) {
    this.name = name;
  }

  abstract arm(weapon: string): void;
}
```

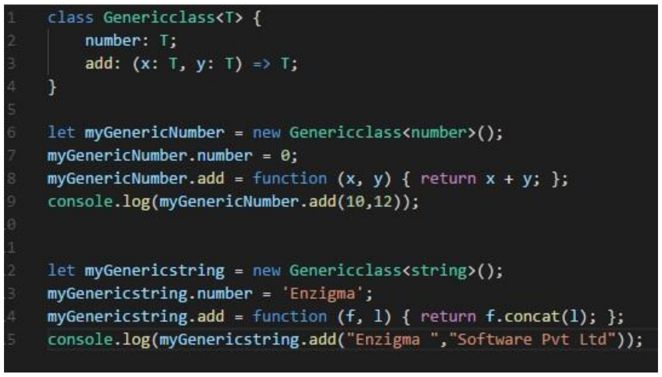
Now the above is abstract class, now we try to instantiate it directly it will give an error.

```
const john = new Warrior('John, the Nomad'); // Cannot create an
```

19. TypeScript: Generic Types

Generic Class-

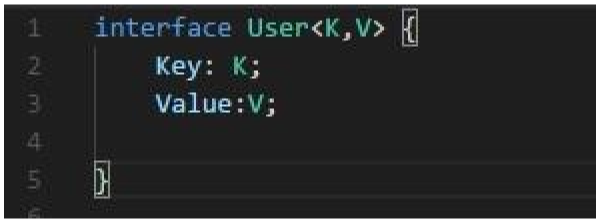
Generic classes have a generic type parameter list in angle brackets (<>) following the name of the class.



```
class Genericclass<T> {
    number: T;
    add: (x: T, y: T) => T;
}

let myGenericNumber = new Genericclass<number>();
myGenericNumber.number = 0;
myGenericNumber.add = function (x, y) { return x + y; };
console.log(myGenericNumber.add(10,12));


let myGenericstring = new Genericclass<string>();
myGenericstring.number = 'Enzigma';
myGenericstring.add = function (f, l) { return f.concat(l); };
console.log(myGenericstring.add("Enzigma ","Software Pvt Ltd"));
```

Generic Interface-

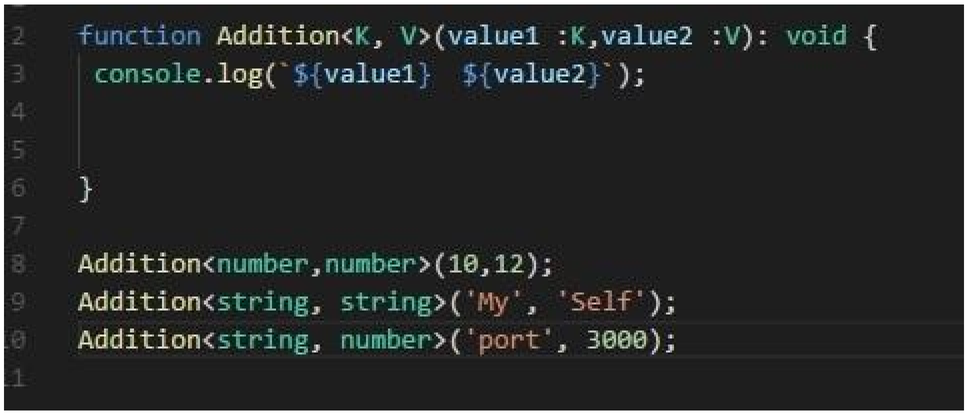
The generic type can also be used with the interface.



```
interface User<K,V> {
    Key: K;
    Value:V;

}
```

Generic function-

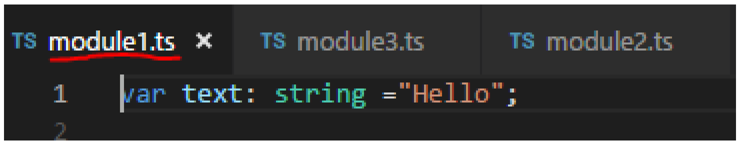
Functions are one way to create flexible, reusable functions. But before you start creating your own generic functions.



```
function Addition<K, V>(value1 :K,value2 :V): void {
 console.log(`${value1}  ${value2}`);


}

Addition<number,number>(10,12);
Addition<string, string>('My', 'Self');
Addition<string, number>('port', 3000);
```

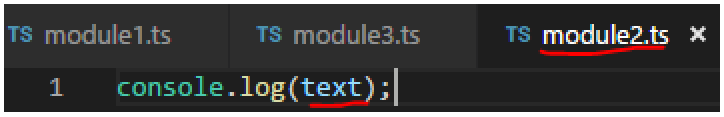
20. TypeScript Module

Let's say if we declare a variable than in that case that variable can be accessed globally as well, Well that seems to be security threat isn't it. Module to the rescue!, It basically allows the files that are local by modularize it and basically by exporting & importing when required and thereby securing it.



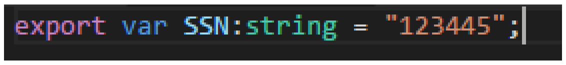
```
var text: string ="Hello";
```

In this case, we have module1.ts which is ts file which has text which can be accessed freely as global in any other file locally.



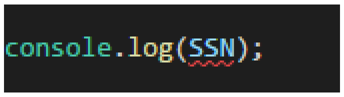
```
TS module1.ts   TS module3.ts   TS module2.ts ×
console.log(text);
```

In this case, we are able to access the file directly.



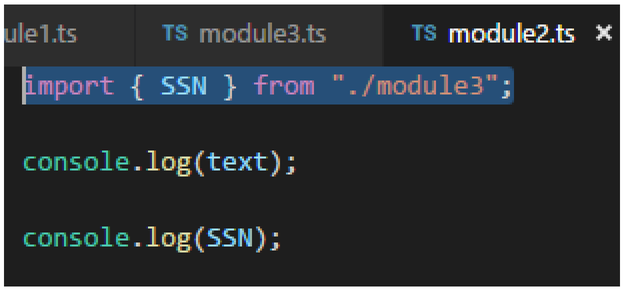
```
export var SSN:string = "123445";
```

Now above we have used export keywords and we try to access the file directly without import it will fail.



```
console.log(SSN);
```

In Order to access it we need to go ahead and import it.



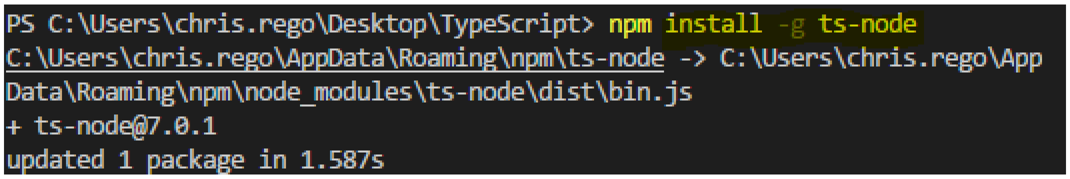
```
ule1.ts   TS module3.ts   TS module2.ts ×
import { SSN } from "./module3";

console.log(text);

console.log(SSN);
```

21. How about directly running the typescript file

We had discussed previously the method in which we first need to convert a typescript file and then later on we need to convert the file into typescript file.Previous method involved that we needed to convert the typescript file first into native .js file (using Typescript compiler) and then after conversion of the Typescript to Javascript.We can run the Javascript using node.

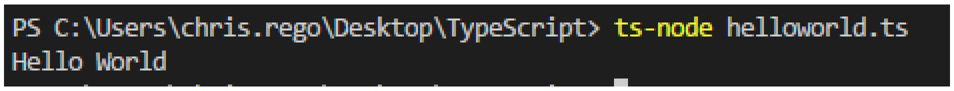
Far more simplest approach to run typescript code directly is by using, ts-node

We need to install npm package named ts-node. We can install it by running the below command



```
PS C:\Users\chris.rego\Desktop\TypeScript> npm install -g ts-node
C:\Users\chris.rego\AppData\Roaming\npm\ts-node -> C:\Users\chris.rego\App
Data\Roaming\npm\node_modules\ts-node\dist\bin.js
+ ts-node@7.0.1
updated 1 package in 1.587s
```

Command- npm install -g ts-node
After Running the above command ts-node gets installed.



```
PS C:\Users\chris.rego\Desktop\TypeScript> ts-node helloworld.ts
Hello World
```

Command- ts-node helloworld.ts
After Running the above command it goes ahead and runs the typescript file directly and does not generate and equivalent Javascript file for the same.

22. Footnotes

- @Namespace in Typescript
- @Details: tuples, array & never
- @ Deep Dive into Generics

23. Assignment TypeScript

- Before Starting the Assignment , we need to make sure you have the github account setup , Create a repo under your account "typescript-module"
- Create a branch named after "dev-yourname"
- Make sure all the code is committed to your respective branches.
- Create a console based Calculator App using TypeScript. Follow Object Oriented Paradigm and follow the best practises while writing the code.
  - The Application needs to be Console based and Menu Driven.
  - To get more details on how to take input refer to @link
  - Basic Operation that it should perform includes.
    - Addition / Subtraction
    - Multiplication / Division

- Follow a Modular approach while developing the Application, and encourage the use of Modules & Data Safety approach.
- Make Optimum usage of :
    - Constructor & Modules
    - Access Modifiers (private/protected)
      Inheritance (interface)