**T**ypeScript is a super-set of JavaScript. It lets you write JavaScript using static typing and class-based inheritance.

**Prerequisites for typescript:**

1. OOP concepts and
2. Basic JavaScript

**Installing and compiling typescript:**

1. First download and install node js

<https://nodejs.org/en/download/>

1. Install node modules using below command:

Npm install which will install the node modules locally or globally depends up on the command used.

1. Install typescript globally by using below command:

Npm install -g typescript //here g means global

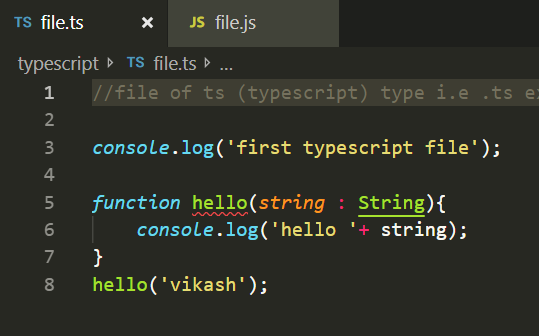
1. For compiling .ts file we use “tsc filename.ts”

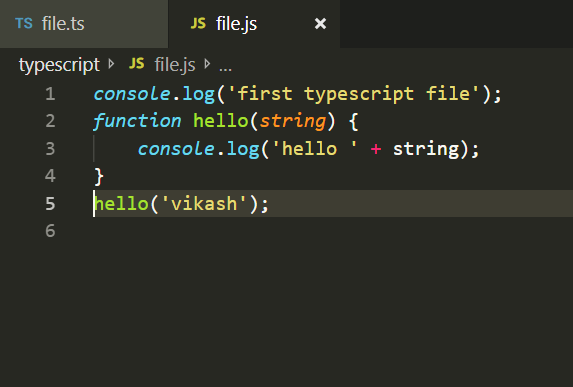
As we know typescript is typed superset of JavaScript, so whenever we write anything is .ts file tsc (transcompiler) will compile .ts file and change it in .js file extension with js format either its es5, es6 etc.

For ex:

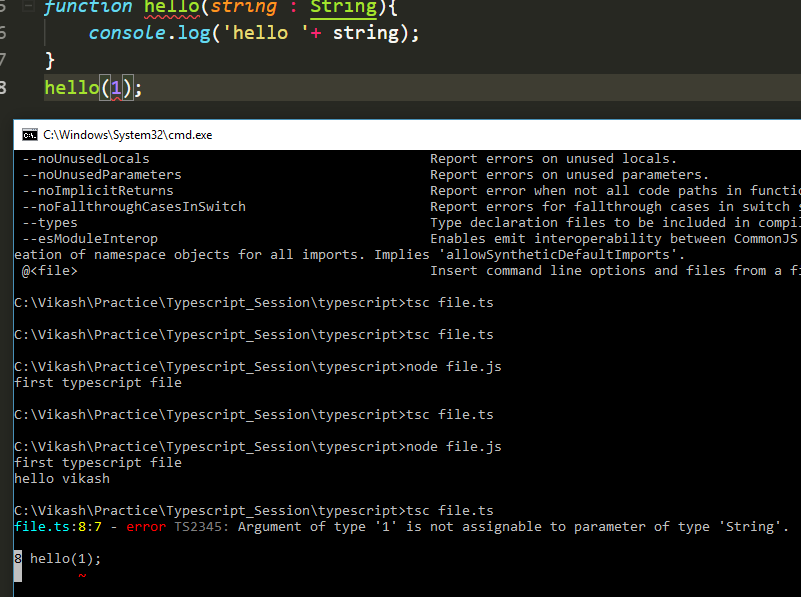
Below is the .ts file with name file.ts so once we run the command tsc file name

tsc file.ts then it will convert file.ts to file.js which will be final file





In the above example, for function hello we defined type as string, so if we are passing string as parameter then we need to pass only string but if we pass number instead of string will get the below error, because ts is typed language.



As we can see in the above screenshot we have defined hello function which accept string as parameter type, but we are passing number type which throws exception.

Function hello(string : String) typescript annotation{}

**Variable declaration in typescript:**

TypeScript support much the same types.

For ex:

*var* n : Number = 1;

in the above line of code we are saying the ‘n’ is a number type, so here we need to pass the value of n as number only, if we pass anything like string or Boolean then it will throw error.

After compiling the above line of code using tsc filename.ts command we get the following js:

*var* n = 1;

Now pass string instead of number for the above line of ts code like below:

*var* n : Number = 1;

n = 'vikash'

here we are passing string as the value of n which will throw error when we try to compile the above line of code. So, to handle this use ‘any’ instead of number. So when we use any it means ‘n’ will accept any kind of variable whatever we pass either its string, number or Boolean. for eg.

*var* n : *any* = 1;

n = "vikash"

in the above line of code, we have used “any”, so now whatever we pass we won’t get any type error.

NOTE: We should not use “any” type because it defeats the uses of type. The stricter we are with types the more useful they will be in preventing types error.

//number type

*var* n : Number = 1;

n = 'vikash'

//any type

*var* n : *any* = 1;

n = "vikash"

//string type

*var* n : String = 'hi'

// boolean type

*var* isMsgSent : Boolean = false;

// array

// for array the type should be same, if we mix type for an array then it will throw an error.

*var* name : String[] = ['vikash', 'shankar', 'Deepak'];

//for eg, below array will throw an error because we are using string with number

/\* var name : String[] = ['vikash', 'shankar', 'Deepak', 5]; \*/

// so to handle the above array and if we want to use mix type then we should use array type 'any' which will except any type of value either its a number or string

*var* name = any[] = ['vikash', 'shankar', 'Deepak', 5];

//enum type

//enum is use to store build in variables

*enum* Starks {vikash, shankar, deepak}

*var* dc : Starks = Starks.deepak;

//that's how we use enum

//void type

//void is type of type which is not any type,its equal to nothing.

*function* getName() : *void*{

*console*.log('winter is coming....')

}

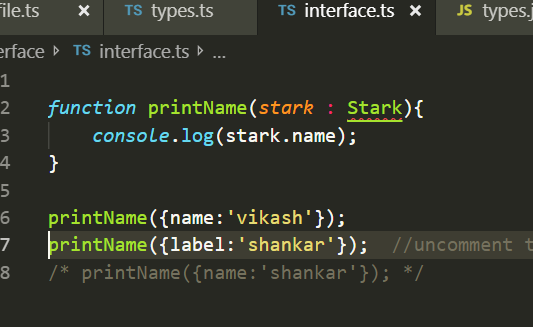
//the above function won't return anything

**Interface:**

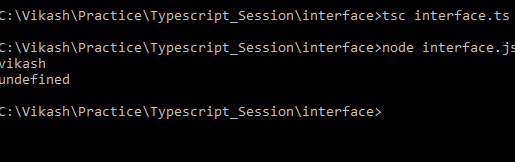
Interface is a blueprint for an object that tells typescript what the object is going to be.

Interfaces are yet another way for checking error as its been compiled in to JavaScript.

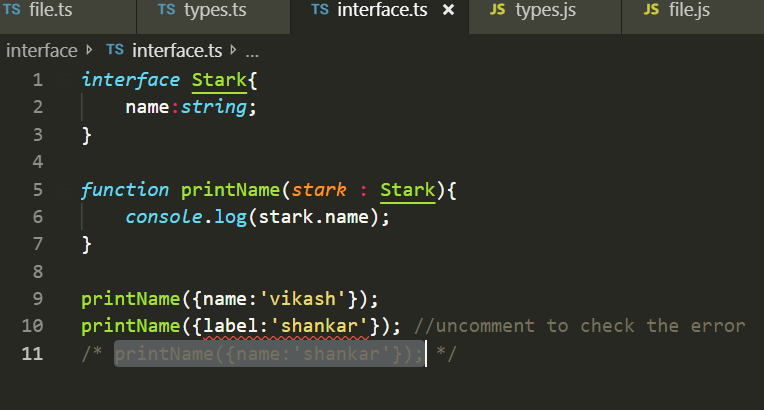
For eg:



When we compile the above code from ts to js it won’t throw any error but when we compile the js file using node filename.js command it will throw error saying undefined for second printName function.



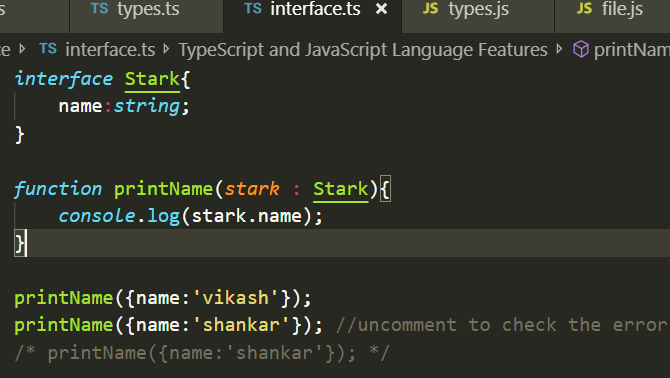
So to handle this kind of error will introduce interface.



In the above snippet we can see we introduced interface but still its showing some error on the label property, why?

Because we have defined a property name and trying to access label which is not present in our interface.

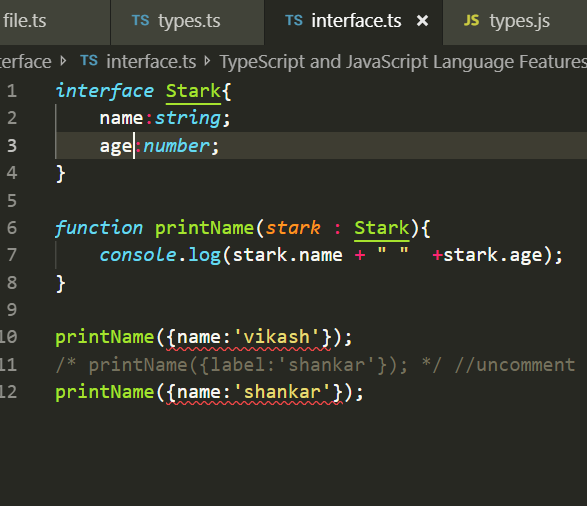
Label is not assignable to parameter of type stark, object literal may only specify known properties and “label” doesn’t exist in type “stark”.



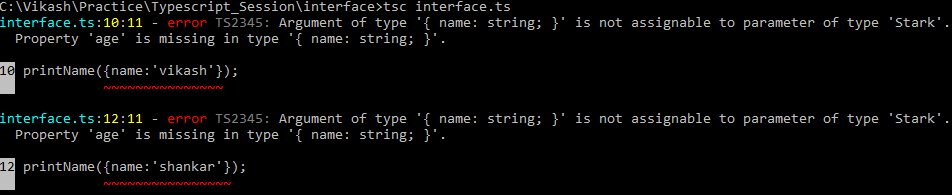
So now to handle this kind of error we should change property label with name.

So whatever we are defining inside interface is required parameters.

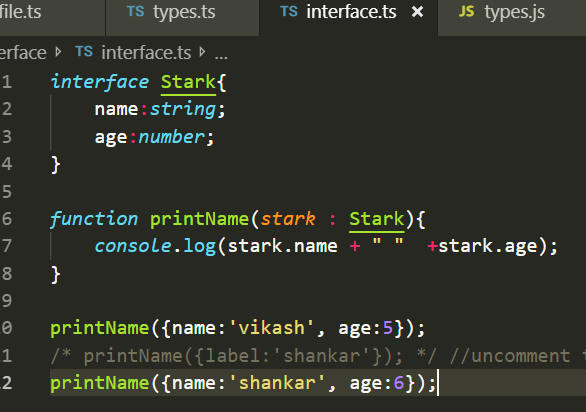
For eg:



The above code snippet will throw following error:

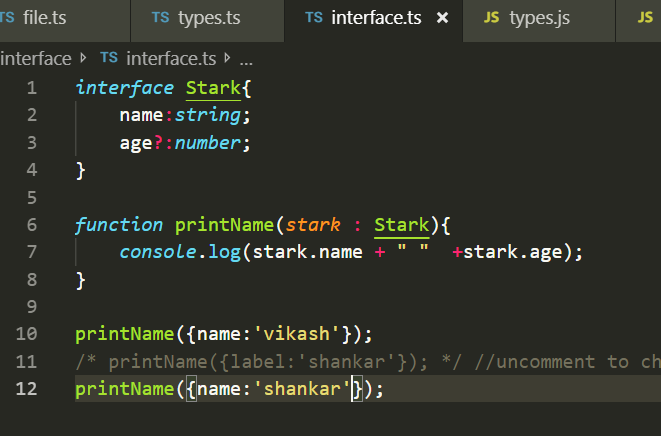


Here we defined a required parameter age inside stark but its missing in printName function. So to handle this we need to add property age in printName function like this:



How to make a parameter optional in interface ?

We can make any of the parameter optional by adding a question mark after the property name like below snippet:



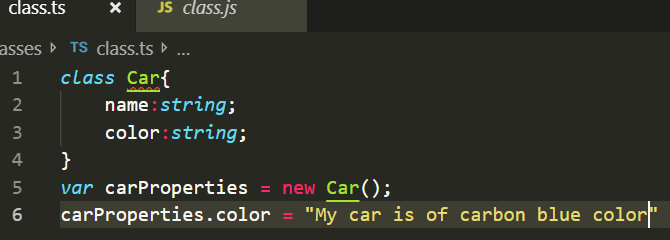
In the above code snippet we can see we made age as a optional property in stark parameter and printName now doesn’t show any error as earlier, means if we make any property as optional and won’t use inside the function then it won’t throw any error but it we won’t make it optional and won’t use it inside printName function then it will throw error stating property name is missing.

**Classes**

Classes are like interface however its more powerful than interface.

Class is also made up the blueprints of objects, classes are meant to be constructed or in other word have new instances of the class created.

For eg.

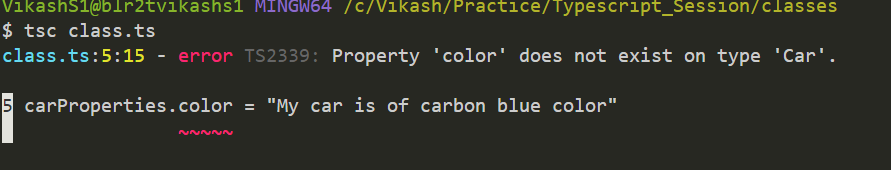


In the above code snippet ‘Car’ is a class with some properties like name and color,

In line 5 we have created an object called carProperties.

Here if we won’t define a property and try to access it then it will throw an error.

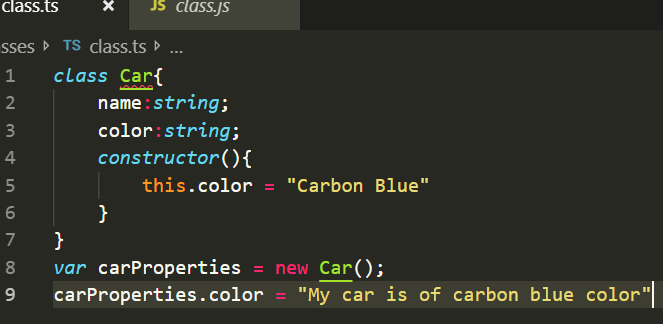
For e.g. It we won’t add color inside ‘Car’ class and try to access it in line 6 then it will throw an error like below:



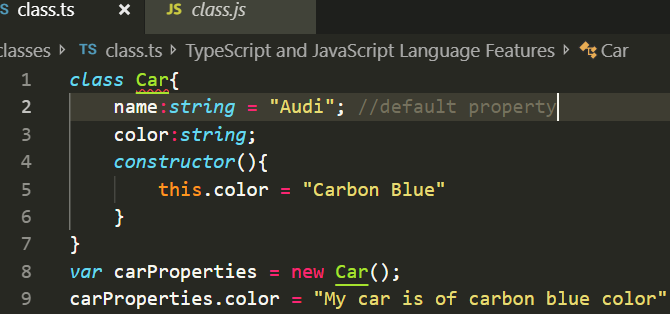
Like interface we cannot make any optional property for class using “**?**” .

Classes can also have **constructor** properties which is function when the class run.

Below is the snippet showing constructor properties in a class.



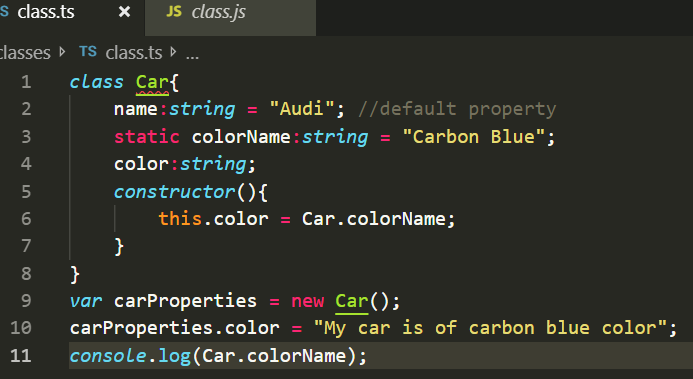
We can also give a default property in the property definition by putting a **“=”** after that like below snippet.



Here “**name: string = ‘Audi’**” is a default property.

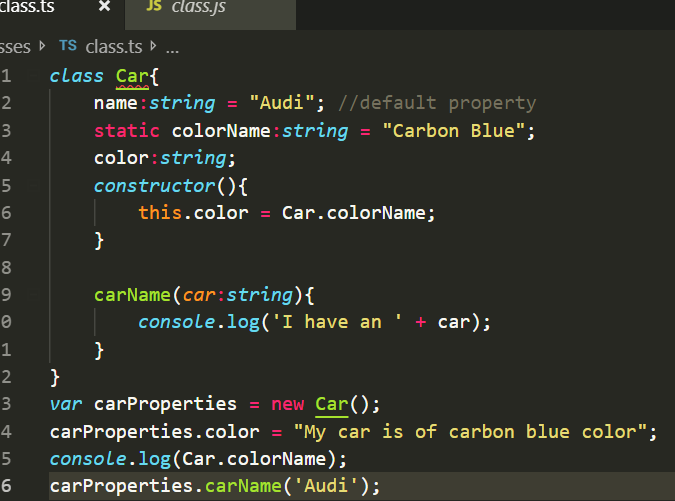
Classes can also have static properties; static property is accessible from the class blueprint which is nothing but “**Carbon Blue**” or the object with the capital letter “**Car**”.

For e.g.



Classes can also have methods which is not a constructor.

For e.g.



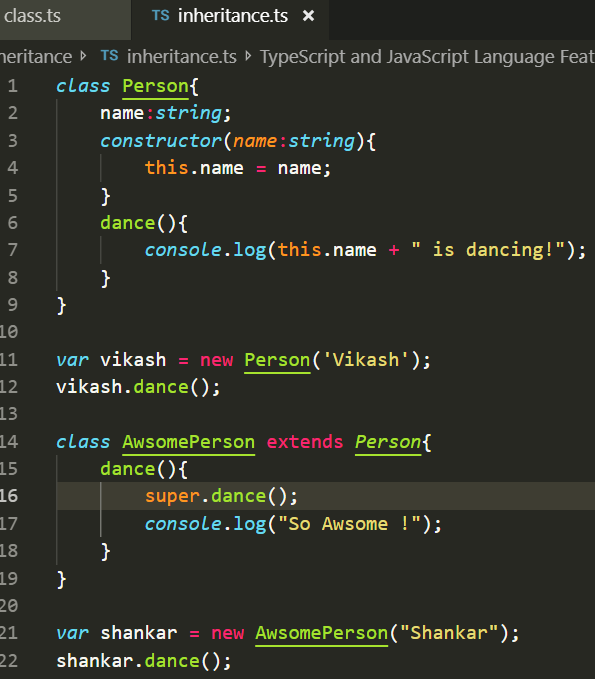
So in the above snippet we can see we have a method name **carName,** which can have arguments too (**car:string**).

Classes are very powerful and we can use them to define the behaviour of an object in depth.

**Inheritance:**

Inheritance is a way for objects and classes to gain access to the properties of classes that represents by other objects.

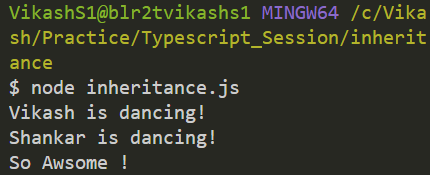
For e.g.



In the above code snippet, we are creating a class “**AwsomePerson**” by extending the class “**Person**” which is nothing but inheriting the feature from Person class.

We use keyword **extend**, for extending a class.

output for the above code snippet is :



Why ?

Here we are overriding the method “**dance()**”.

**super.dance() :**

super is a keyword for accessing the extended class’s method. So when we excute the code it will return the above output.

When we override a function it doesn’t automatically call the method of the extended class, so for that only we use super.

**Modules**

Modules are powerful way to share code between files, using module we use our file size small and dependencies would be clear.

So for sharing code between files first we need to create a module using

module name{} //name is the name of the module.

module calculation{ //here module name is calculation

export class classname{ //classname is the name what we are going to use in other file

}

}

File1.ts

*module* calculation{ //calculation is the name of the module

export *class* useful{ //useful is the class which we are exporting to other file

timesTwo(*n*:*number*){ //timesTwo is the method which will get executed and return value.

return n\*2;

}

}

}

So now we need to use this file “**file1.ts**” in to another ts file, meaning we want to use the code of “**file1.ts**” in to “**file2.ts**”, for that we need to use reference path inside “**file2.ts**”

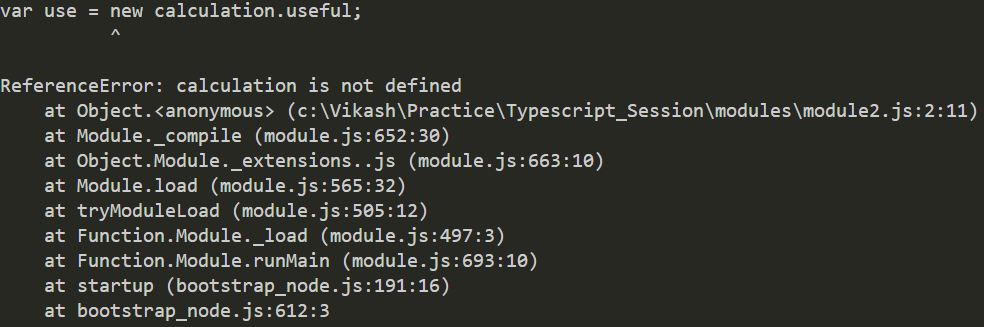
For e.g.

///<reference path="file1.ts" /> //this is how we reference the path for file1

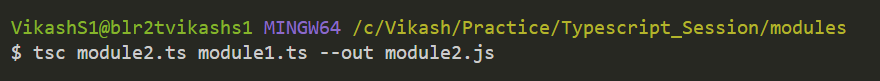
*var* use = new calculation.useful; //calculation is the modulename and useful is the class name

*console*.log(use.timesTwo(9)); //here is the object with the help of which we are accessing timestwo function.

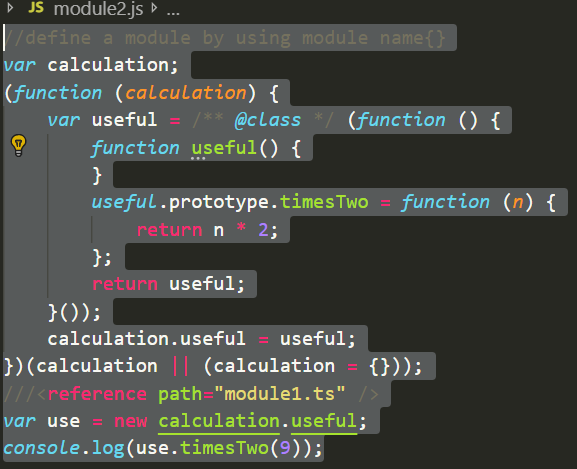
So after doing the above operations when we run “**file2.ts**” then we will get the following error:



So to handle the above error we need to combine two files like below and we need to use **out** flag.



The above command will create file2.js with combination of both the files like below:



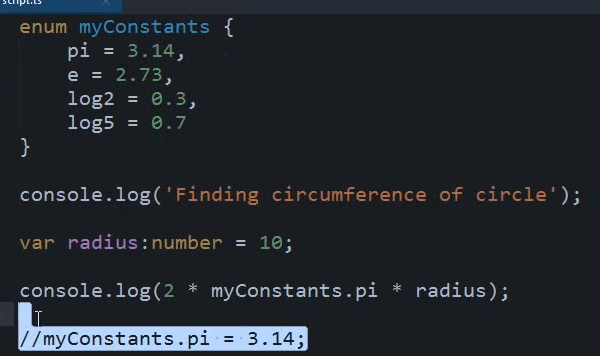
Then we need to execute command **node file2.js**, which is combination of **file1.ts** and **file2.ts.**

**Enum:**

Enum simply store bunch of numbers in our program, which cannot modified. But eventually get converted in to js file.

Enum are restricted to numbers only so we cannot have string inside Enum.

For e.g.



If we try to change the Enum constants value then it will throw error, like in below snippet we are trying to change the pi value from 3.14 to 4.

*enum* myConstants {

pi = 3.14,

e = 2.73,

log2 = 0.3,

log5 = 0.7

}

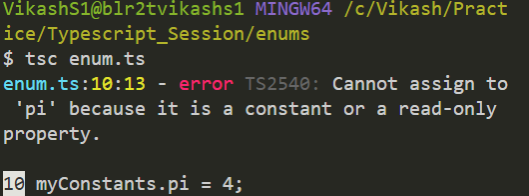
*console*.log("Finding circumference of circle");

*var* radius:*number* = 10;

*console*.log(2 \* myConstants.pi \* radius);

myConstants.pi = 4;

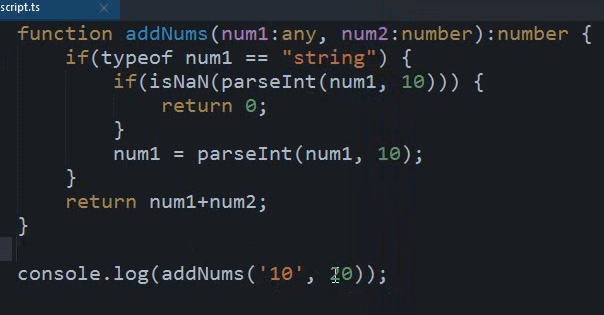
below are the listed error.



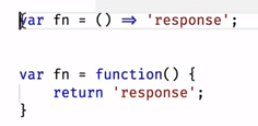
**Function**

Functions are the fundamental building block of any applications in JavaScript. They’re how you build up layers of abstraction, mimicking classes, information hiding, and modules. In TypeScript, while there are classes, namespaces, and modules, functions still play the key role in describing how to do things.

For e.g.



The output for above snippet will be 30.



1. This snippet which is JavaScript function

Var fn = function (){

return ‘response’

}

1. This snippet which is JavaScript arrow function:

Var fn = () => ‘response’

Snippet 2 is exactly the same as snippet one.

1. We can define type for return too, i.e. what my function is going to return either a number or a string or something else.

For e.g.

Function add () {return a + b + c + d}

add (1, 2, 3);

o/p: 6

for defining a return type, we should add type after function parenthesis

i.e. function add (): number {return a + b + c}, so this function will return only number.

1. Default and optional parameter in function.

For e.g.

Function add (a: number, b: number, c?) {}

In the above snippet c is optional parameter, it means either we pass it or not it doesn’t matter. But what if we want to make it default.

For e.g.

Function add (a: number, b: number, c = 0)

In the above snippet c is default param, so whenever we execute it will pass **0** for **c.**

1. Default with type:

For e.g.

Function add (a: number, b: number, c: number = 0){return a + b + c};

add (1, 2)

o/p: 3

but if we pass something else for c param then it will override 0 with passed value

add (1, 2, 3);

o/p: 6

So, in the above snippet, c is optional and default too, so either we pass it or not it won’t throw any error.

**Implicit type**

1. When we declare any variable and we don’t add the type of the variable then by default it going to define the type based on value we assigned to that variable.

For e.g.

Var a = 10;

Var b = true;

Var c = ”Hello”

In the above snippet” a” is number type, “**b**” is Boolean type and “**c**” is string type.

1. When we define a variable in one line and we declare its value in another line whatever the value is in that case it’s going to make the variable as type “**any**”.

For e.g.

Var a;

a = 10;

in the above code snippet, the type for a is going to be any, that’s how typescript behave.

**Note: same is going to apply for function too.**

**Miscellaneous:**

We can change the file name, it’s not necessary that if we create a file name say “test.ts” and when we execute it, it will return a js file with same name “test.js”.

so, for that we can use some commands like we used in below screenshot.

For e.g.

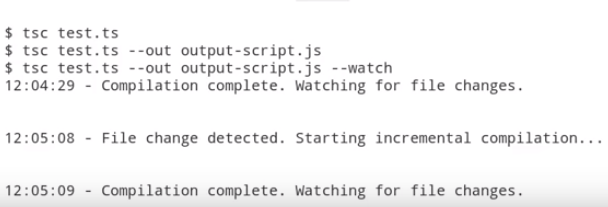
**tsc test.ts –out output-script.js**

above snippet will return a file name output-script.js for the file “test.ts”

Now what if we want to keep on watching for the changes what we are going to do in our “.ts” file, so for that we can use below command.

**tsc test.ts –out output-script.js –watch**

above command will keep on looking for the changes and keep on updated the output js file.



**Try tsc help in command line.**

**Grunt:** **https://www.udemy.com/typescript/learn/v4/t/lecture/2709262?start=0**

