***Typescript***

Adding exclamation mark (!): we that wont yield will not result in null

Converting string to number: just add +

Ex: +str

Important: Type Casing

In TypeScript, you work with types like string or number all the times.

**Important**: It is string and number (etc.), **NOT** String, Number etc.

**The core primitive types in TypeScript are all lowercase!**

**In JS and TS all numbers are float by default**

**Type inference : when we didn’t specify the result type of the result it will auto resolves to the type it assigned**

**Objects:**

**const obj = { name:string;**

**age:number } // notice semicolon**

Nested Objects & Types

Of course object types can also be created for**nested objects**.

Let's say you have this JavaScript **object**:

1. const product = {
2. id: 'abc1',
3. price: 12.99,
4. tags: ['great-offer', 'hot-and-new'],
5. details: {
6. title: 'Red Carpet',
7. description: 'A great carpet - almost brand-new!'
8. }
9. }

This would be the **type** of such an object:

1. {
2. id: string;
3. price: number;
4. tags: string[],
5. details: {
6. title: string;
7. description: string;
8. }
9. }

So you have an object type in an object type so to say.

Tuples: for fixed length and fixed position arrays

Role : [number, string]

We can now only assign number to the first index and string to the second index . Total length of the array is fixed.

Enum can contain any types in it

enum Role { ADMIN =true, ID=10}

Union types:

const id : number | string ;

Literal:

Confining value/s:

let roles : ‘ADMIN’ | ‘USER’;

roles can be either of the above values not other than that

Type aliases:

type combined = number | string;

// we can even use complex unions and assign it to some variable.

Type Aliases & Object Types

Type aliases can be used to "create" your own types. You're not limited to storing union types though - you can also provide an alias to a (possibly complex) object type.

For example:

1. type User = { name: string; age: number };
2. const u1: User = { name: 'Max', age: 30 }; // this works!

This allows you to avoid unnecessary repetition and manage types centrally.

For example, you can simplify this code:

1. function greet(user: { name: string; age: number }) {
2. console.log('Hi, I am ' + user.name);
3. }
5. function isOlder(user: { name: string; age: number }, checkAge: number) {
6. return checkAge > user.age;
7. }

To:

1. type User = { name: string; age: number };
3. function greet(user: User) {
4. console.log('Hi, I am ' + user.name);
5. }
7. function isOlder(user: User, checkAge: number) {
8. return checkAge > user.age;
9. }

undefined is also a type

function something() : void{ }

const result = something() // undefined

function something() : undefined{

return;} // Now we can use undefined

Function types :

Function is also a type

if we want to store a function pointer in variable we can specify the type of the variable as function that accepts smth and returns smth using arrow functions.

const Combined : (a: number, b: number)=> number;

callback functions can return something, even if the argument on which they're passed does NOT expect a returned value.

Never can be used to throw exceptions

never

tsconfig

if you use include you have to specify the files you want to compile.

By default node\_modules will be excluded.

if you use exclude then you have to explicitly specify node\_modules

sourceMap helps us with the debugging

Add a debug point in the browser source tab

“noEmitOnError” === if there are any errors in the ts files it will not generate the output files

“strict” if it is set to true all the strict options below it are true implicity.

if you want to change the one of the option remove the comment and configure it.

“strictMode” will add “use strict” in js files

ts will not throw any error for global variables because it will be in an impression of the variable may be used in other files

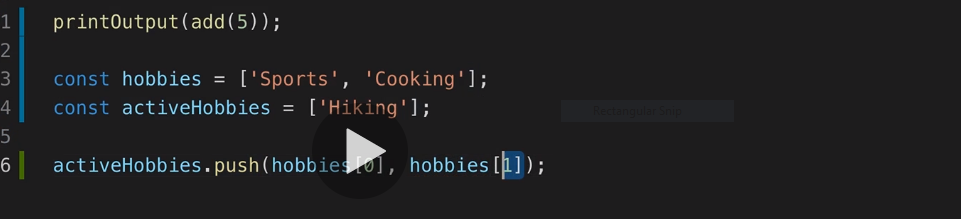
Var :

var has only two scopes : global and function scope.

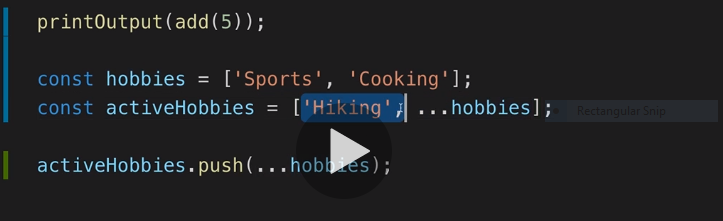
if variable is declared inside a conditional block then that will be available.

let and const : can have a extra scope which is block scope

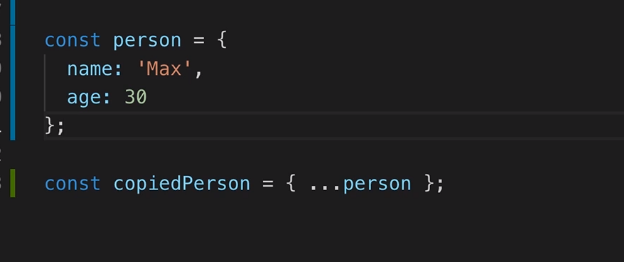
Arrays push: it cannot take object as an arg instead it can multiple values.



using of spread operator



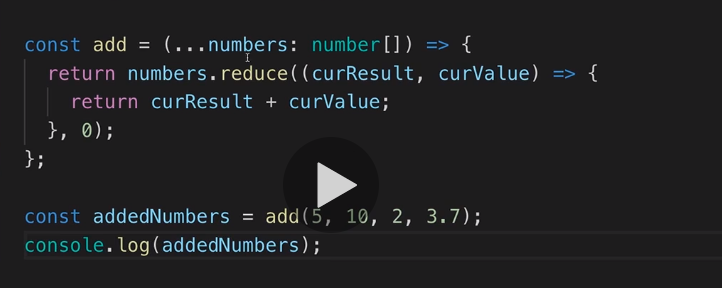
We can even use it to copy the objects



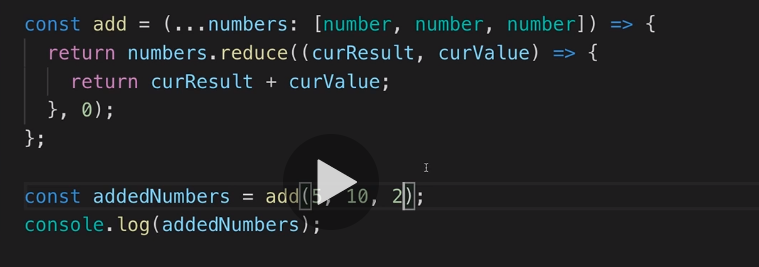
rest parameters :

Addl: reduce – it will take two parameters a function and index value for Arrays and it will return a single value.

The arg func will take two params currentResult and currentValue



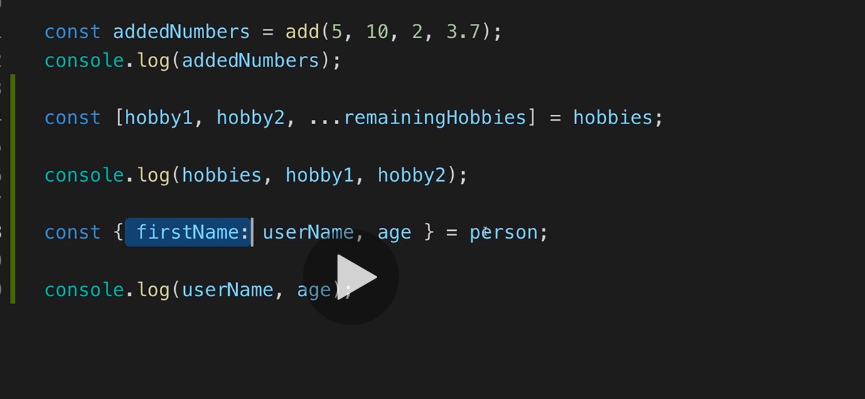
We can even use tuples



Array and object destructuring:

The order matters in array destructuring and we need to use [] square brackets.

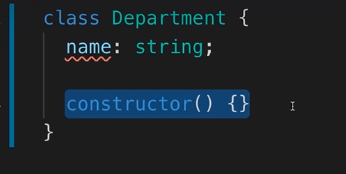
In object case order does’t matters and the destructuring keyword has to be the property name in the object. we can also rename it while destructuring.



Classes: fields and properties

while defining properties we need use semicolon

in case of objects we use comma // not sure

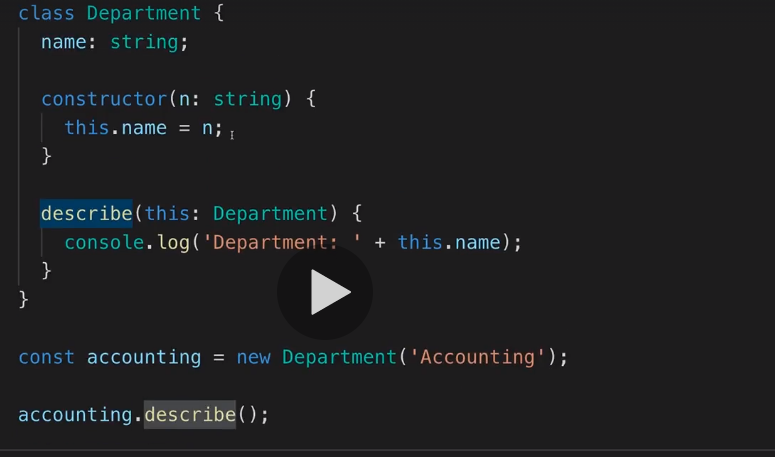


Usage of this:

We have to use this in the class methods while referring to the class variables…….

we add this as parameter while defining the method and we don’t have to pass the arg while calling it.

even this can be “type” here describe will only accepts object of Department

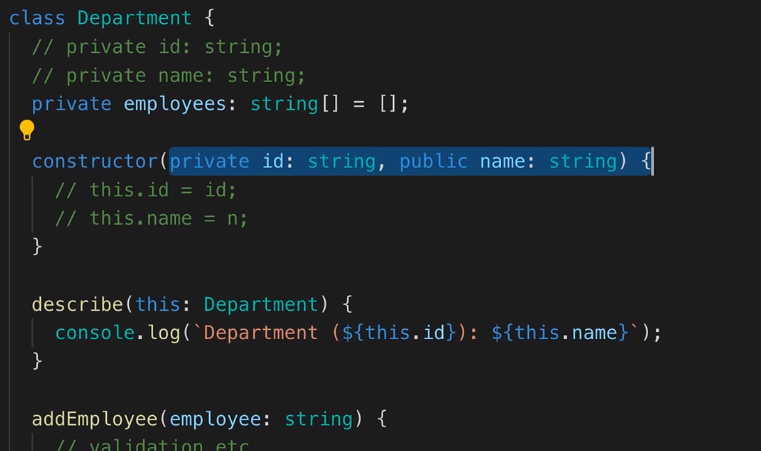


We can use private keyword for variables and methods.

so they won’t be accessible outside of the class.

by default public as modifier

Short hand initialization:



In the picture above we don’t have to define our properties twice in the outside of constructor and inside ( also assigning to this) .

We can avoid that by using short hand initialization. we can even add public and private also in the arg list

readOnly : once initialized the value cannot be changed . we cannot reassign a different value

Methods of a class called as methods only

typically be called a "method".

What's a class property?

A variable in a class.

Inheritance

you can only inherit from one class

child class will gets all the properties and methods of the base class

variables, methods and constructor

the private properties and methods will not be accessible to child class

we can use protected …

getter and setters :

get someValueGet(){return };

set someValueSet(val)….

While calling get we don’t need to call it as a method instead

obj.somValueGet

while calling set

obj.someValueSet = val;

Static:

constructor cannot be marked as static

static variables cannot be accessed by other instance methods by using this keyword instead we can use class name for accessing

Department.year

Abstract:

we can add abstract keyword in front of method and class

like java if class contains a abstract method then the class also a abstract class

and also method should not have implementation only return type

abstract doSmth(val: string) : void;

abstract classes cannot be instantiated.

Abstract classes can have methods with implementation

Singleton : using private constructor and static methods.

Interfaces

it will define the structure of the object

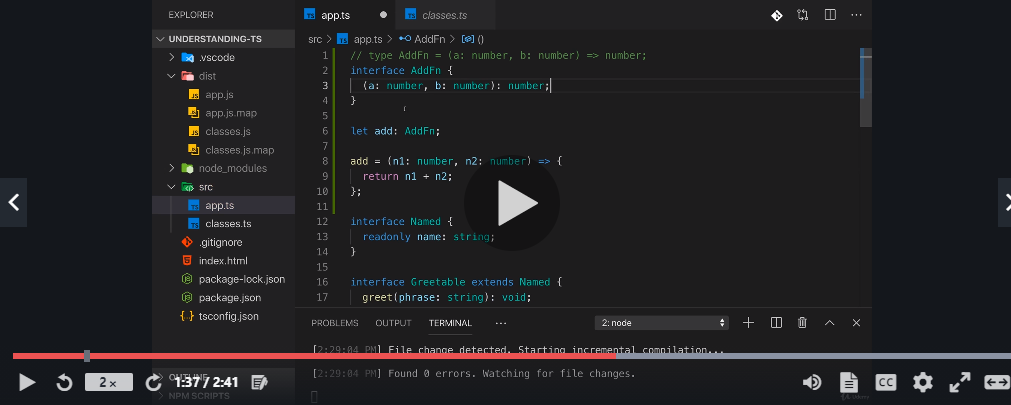
interfaces cannot have initializers like assigning a default value

Inheriting multiple interfaces.

we cannot add public or private modifiers in interface but we can add readOnly

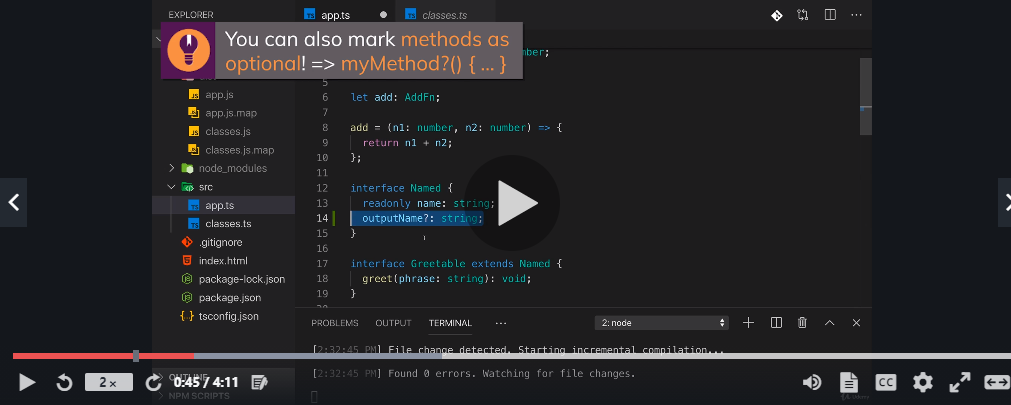
Extending interfaces by other interfaces.

Interface function types



Optional properties in interface and classes

by adding a ‘?’ after the variable



optional params in constructor will also work.

Interfaces describe objects (or function types) but can't store/ describe arbitrary types like union types.