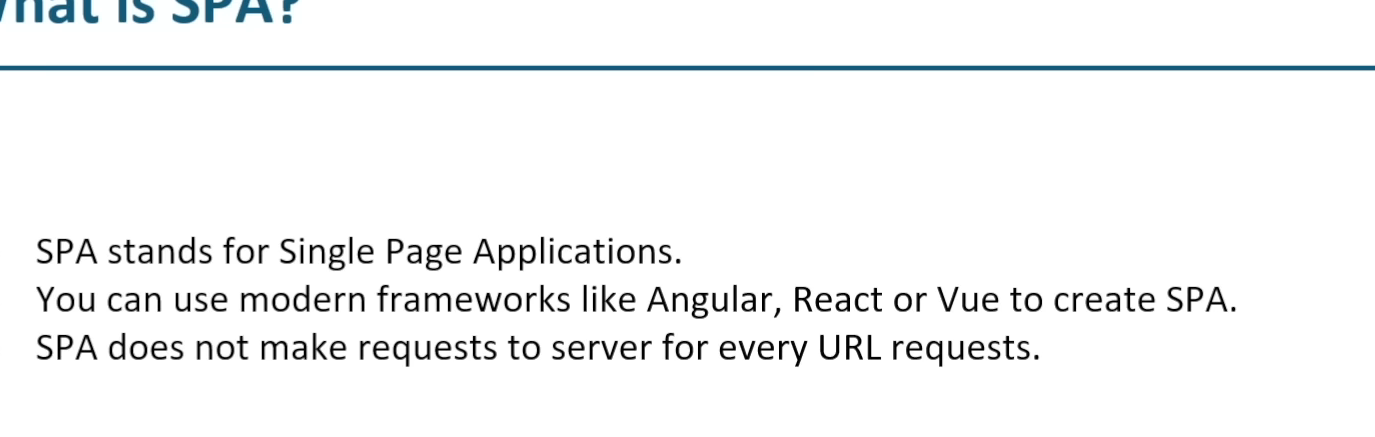
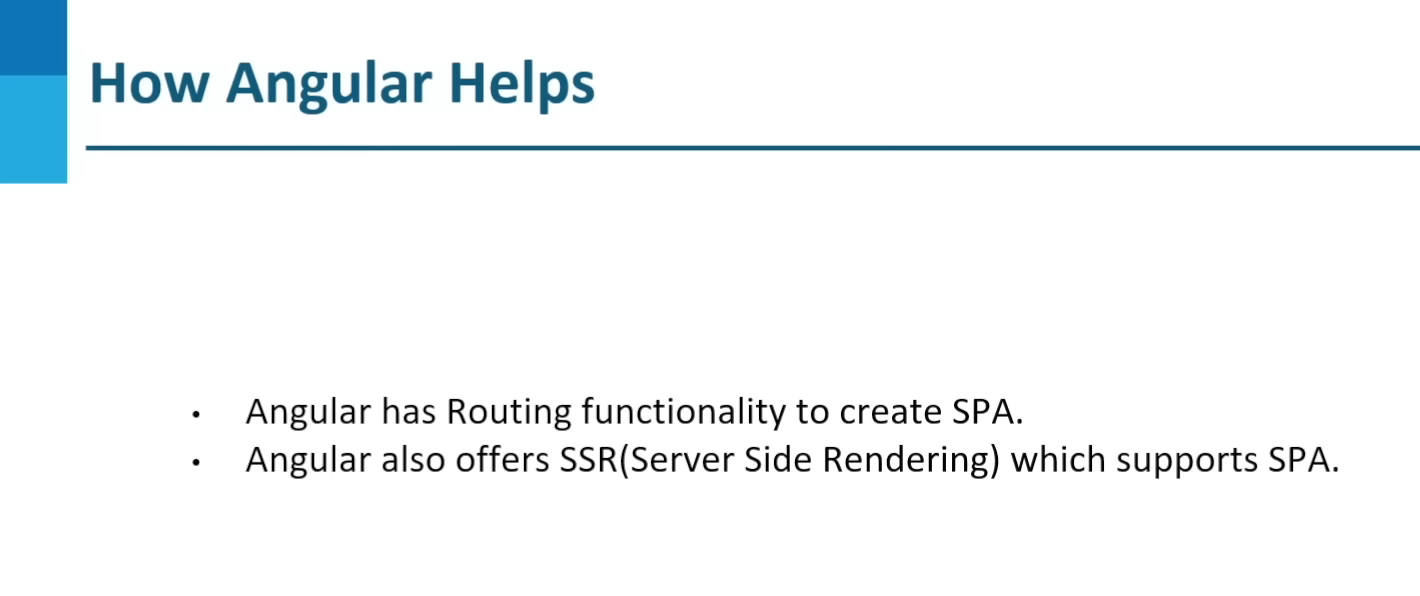
TYPESCRIPT & ANGULAR

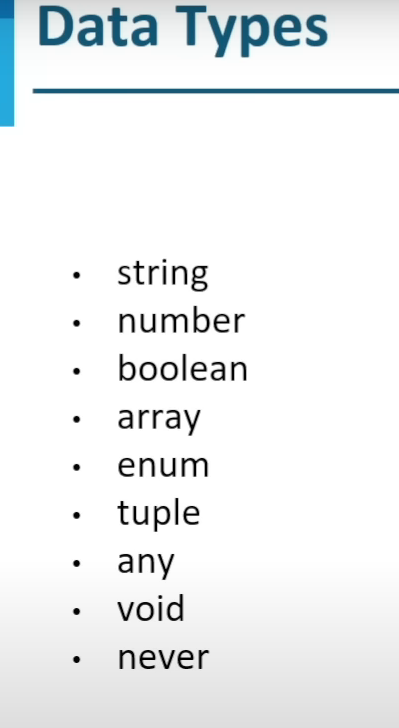
**#Topic: What is SPA**

1. Single page application
2. You can use lot of framework like angular react vue.js to create SPA
3. SPA doesn’t make e





# #TOPIC: DATA TYPES



Interface will not be the part of the production build it will be removed at runtime in TS

So we should go with the class in TS

#TUPLE EXAMPLE :

In arrays we didn’t get the compile time error if we try to access the out of bound index

But in tuple we get one error

Example of tuple is:

Let testNumber : [number,number];

function swapNumbers(n1 :number , n2:number):[number,number] {

  return [n2,n1];

}

swapNumber = swapNumbers(10,20);

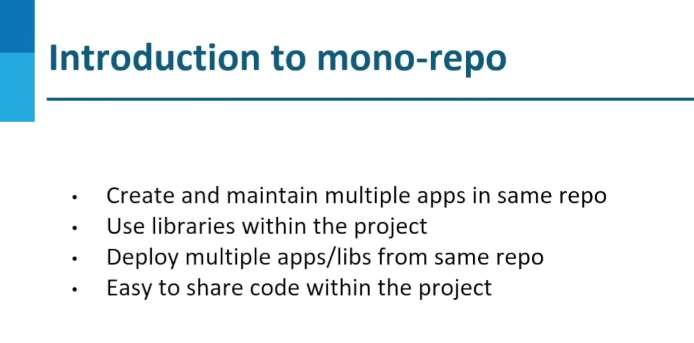
console.log(swapNumber[1]);

#TOPIC : DECORATORS IN ANGULAR:  
This can be applied on class to change the behavior of the class and method or property at runtime

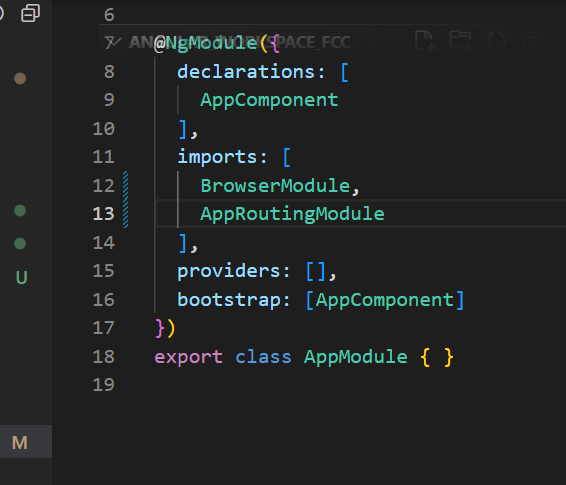
This is used internally in angular

Examples

#Pollyfils.ts fils : this file will help us backward compatibility with the browsers

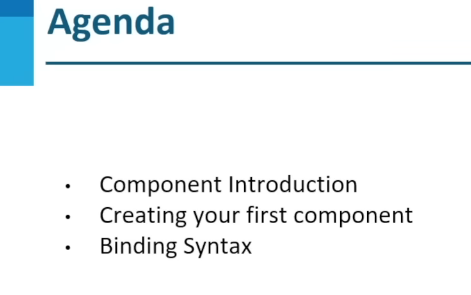


Angular module that needs to be updated in this



In imports array we need to define all the external module

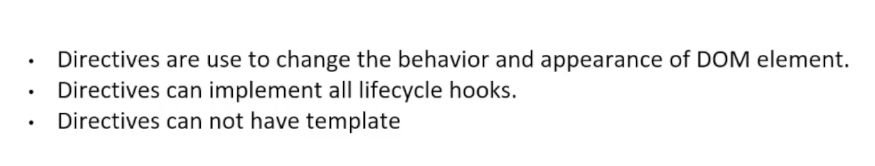
1. @NgModule – is a decorator modifies the behavior of our class
2. Bootstrap:[] when we run the angular which component loads first that we need to define into this bootstrap
3. In the App module we will define all the pipe and custom modules and routing



Binding Syntax :

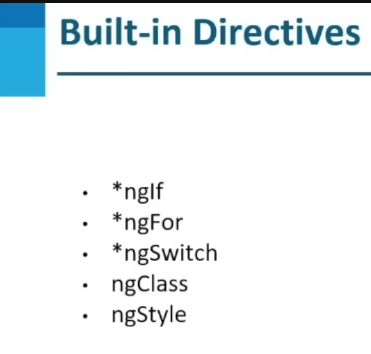
1. Interpolation
2. Property
3. Event Binding

Directives In angular :  
types of Directive



Types Of Directive :

1. Structural
2. Attribute



What is this star mean to differentiate between the Structural and attribute directive

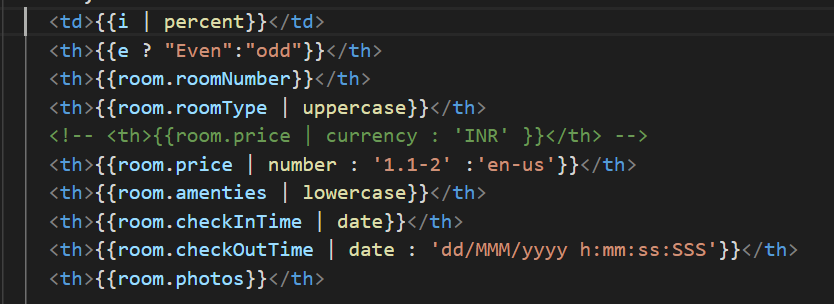
Which will change the dom and modified it this Is known as structural directives

[ngStyles] = “{‘color: predicate ? ‘green’:’red’}”;

We can define the condition and change DOM accordingly

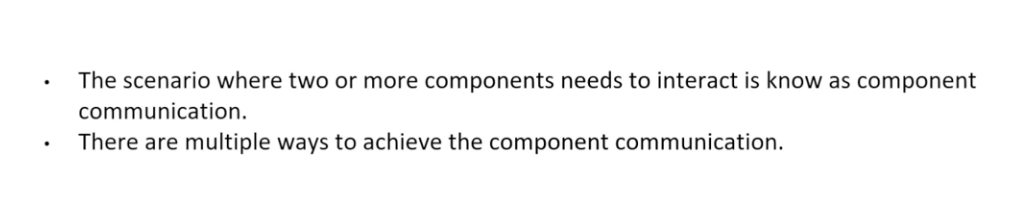
PIPES

* Pipes help us to transform the data from one form to another form
* It doesn’t change the value of the object



Different pipe to transform the data

# Component Communication



There are three ways to interact with other component :



To use the different component proper we can user either @Input and @Output

or we can use the @ViewChild to access the component value

Example :

 @ViewChild('name',{static:true}) name! : ElementRef;

  ngAfterViewInit(): void {

    console.log(this.name);

    this.name.nativeElement.innerText='Hamilton Hotel Companies'

  }

<app-room></app-room>

<div #name>

</div>

With the help of # we can make the dom variable from which we can access the tag of the html element !

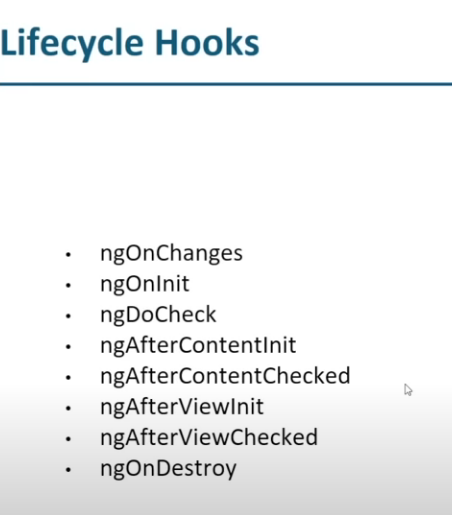
@ViewChild('user',{read:ViewContainerRef}) vcr! : ViewContainerRef;

  ngAfterViewInit(): void {

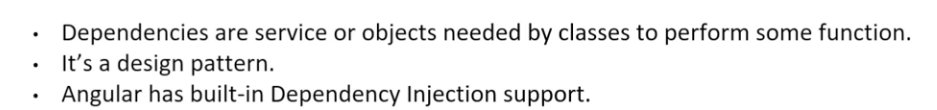
    const componentRef = this.vcr.createComponent(RoomComponent);

    componentRef.instance.numberOfRooms=15;

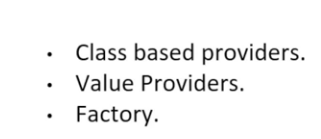
# Life Cycle Hooks :



# Dependency Injection



DI Provider :

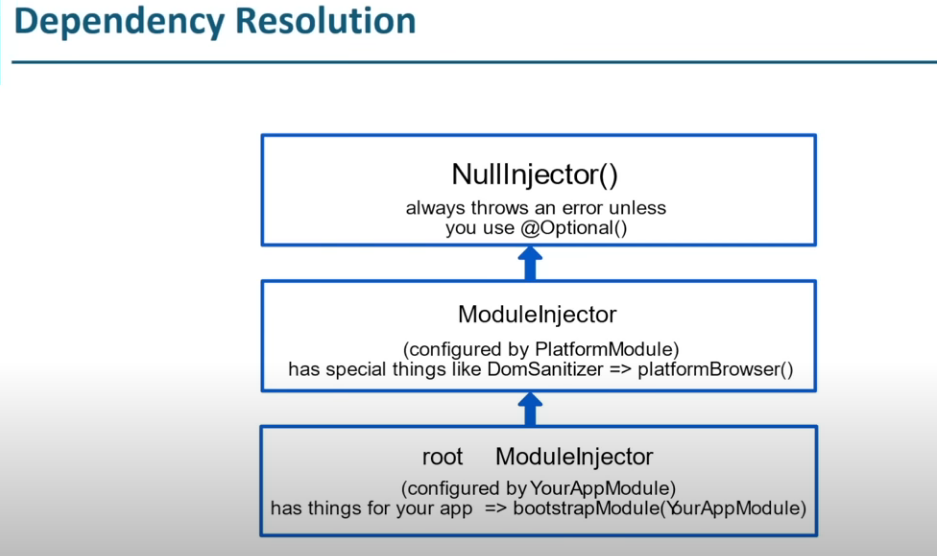


Always define the services in the Constructor () and define the service as private so that template should not be able to access that!!

@Injectable({

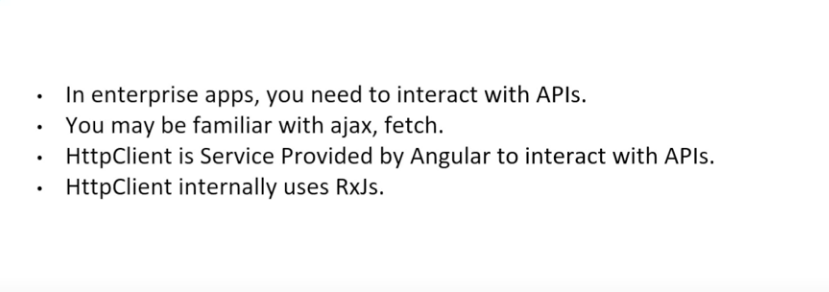
  providedIn: 'root'

})



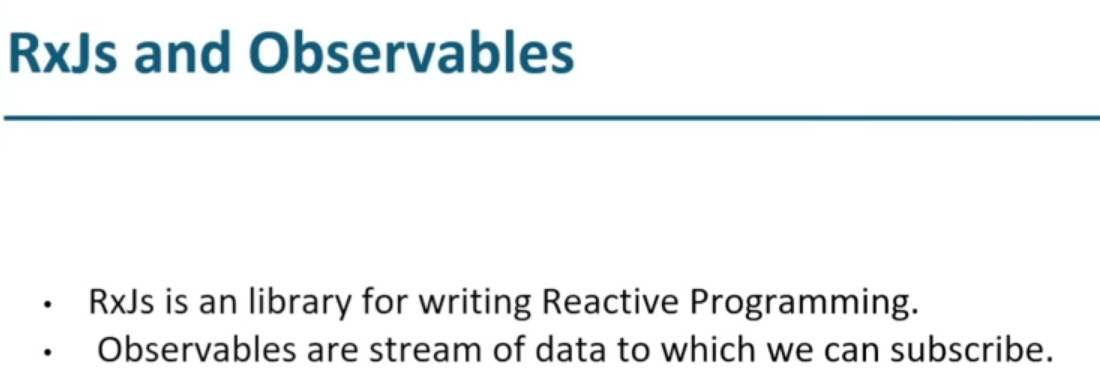
It will look for the service in the give diagram flow only

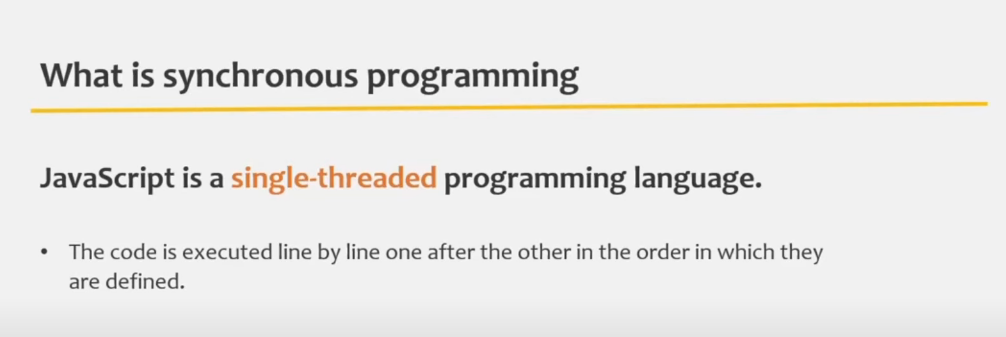
Angular Http and Observables



We can use Injector to inject the constants and Api end point and use it everywhere

With the help of this snippet code and We can also inject the Local And Session storage



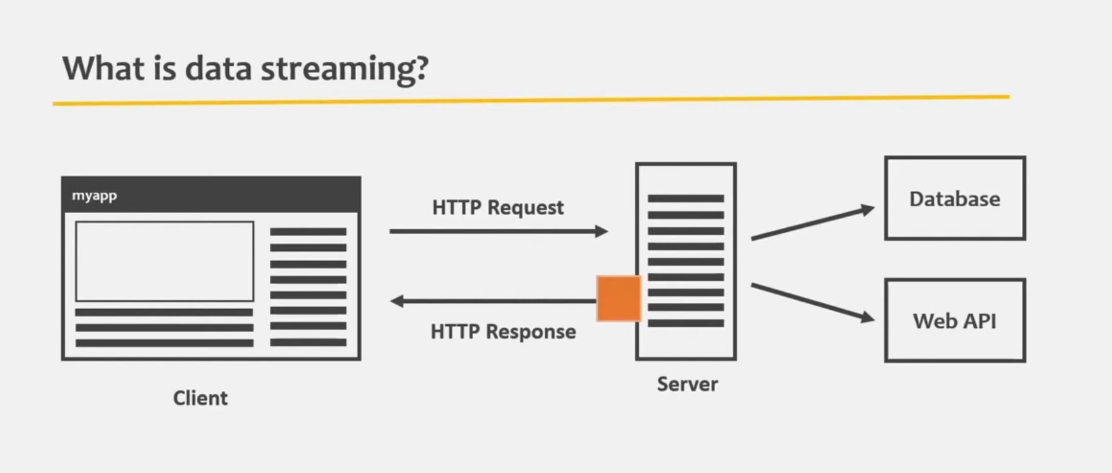


Synchronous code Is blocking code in nature

Async code it will not executed in that single thread And Asynchronous will run in the background

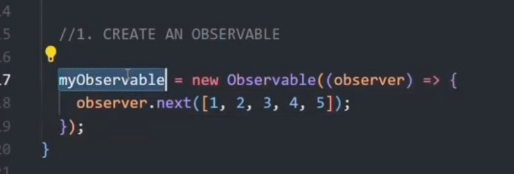
Without block the main thread !!

Promise & Observable is used to handle the Asynchronous data



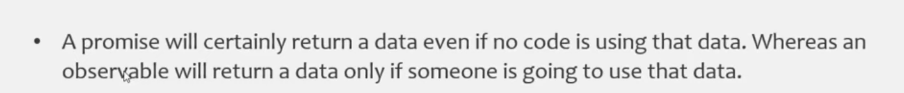
# Promise Vs Observable:

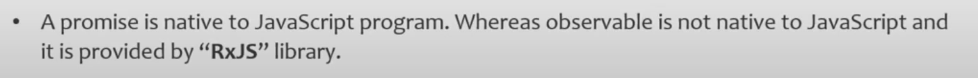
A promise can handle Stream of Asynchronous data. It always return a single value. On The other hand we can use observable to handle stream of async data. It can return multiple Values

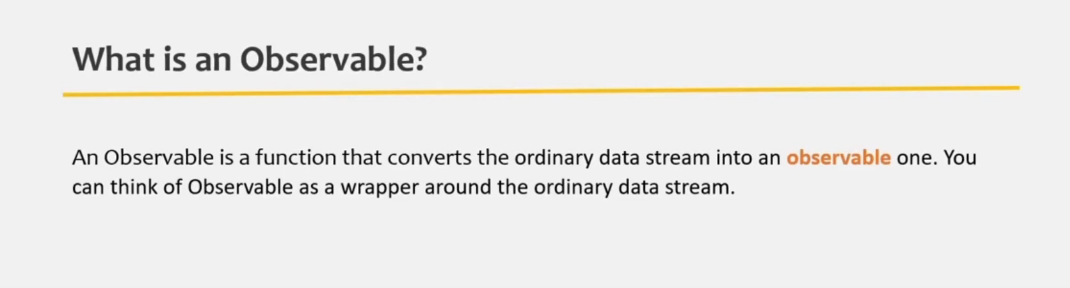


It will emit the data if and only if we subscribe the data for this observer

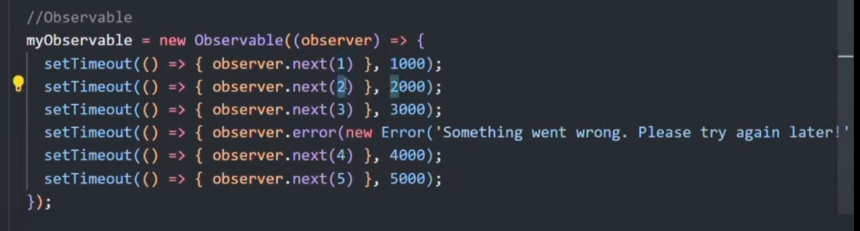
Whenever the observable emit the new data then the subscriber of that observer will notify about the event

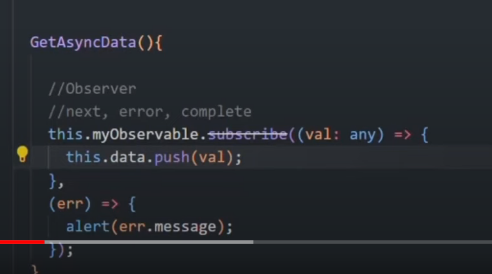






In Observable we have next , error, complete



Once the observable return the error it will stop the emiting data !!  


getAsyncData(){

this.myObservable.subscriber( (val : any) =>{

this.data.push(val)

},

// error

(err) =>{

Alert(err.message);

}

// Complete

() => {

Alert(‘data is stream’)

}

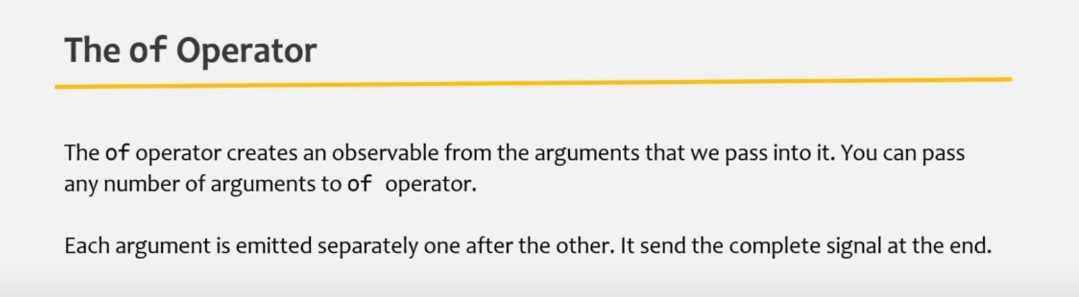
}

Once the error occurred this will not emit any data and it will not call the complete signal

In observable we have 3 things next, error,complete to chec

* Next will check what all are the data we are getting observer emiting the data so we are subscribing the new data from that observer
* Error will check if the in processing the async data we are getting something then it will stop emiting the val
* Complete stage is in use for checking the clean up activites or notification of completing the process.

# Of operator



Import { of } from ‘rxjs’;

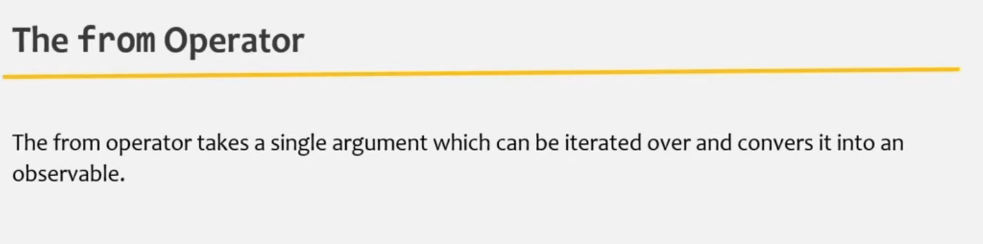
Array1 : = [];

Array2 = [];

myObservable = of(this.array1);

to pass the multiple value we will use OF operator

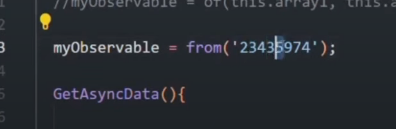
# From Operator:



It will take only one arg and it will be iterable argument

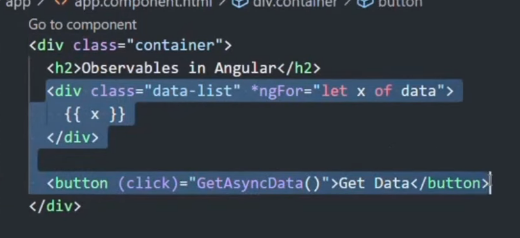
myObservable = from(this.array1);

Each element of array is emitted one after the but “of” operator emit the whole array without streaming the value.



Each character is emit one after the other one by one we will get the emitted value.

# FromEvent Operator:

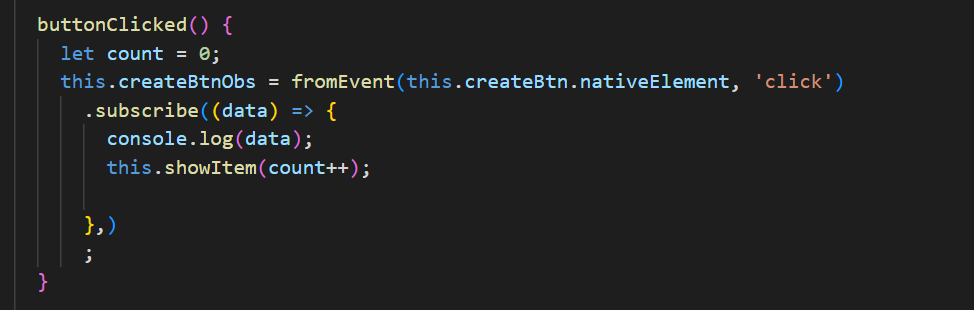


To use the template refrence variable we need to use the @ViewChild Decorator to access the template refrence variable

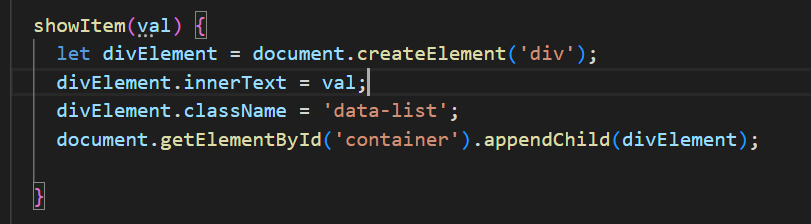
<button #createbtn> Create Button </button>

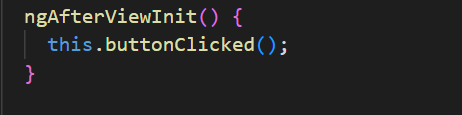
Component {

@ViewChild(‘createbtn’) createBtn : ElementRef;



This is how we are subscribe to the changes using formEvent() in that we have pass the 2 argument one is the target value and other one is the targeted event Which event we are going to perform on that targeted value !!





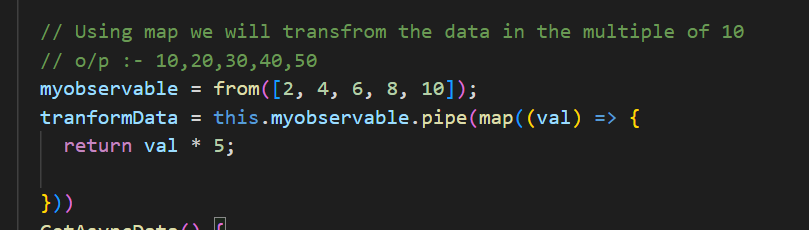
We have implement the AfterViewInit Interface this we used because once the component view is fully initialized then we can perform other activities onto the view like DOM manipulation

Or we need to do some changes or We need to load the child view in the same component

If we don’t Implement this and try to manipulate DOM element then nothing will happen.

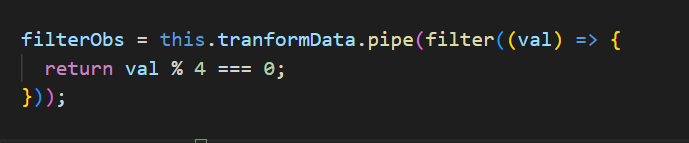
## Map Operator:

The map operator is used to transform data emitted by source observable in some way.



## Filter Operator:

Filter operator will give us the Boolean where it filter out the value and return the stream of data that we are passing



Why we using pipe is because we can chain multiple operator

So here we are doing the filter operation on the map operator

We can chain multiple operator together using PIPE

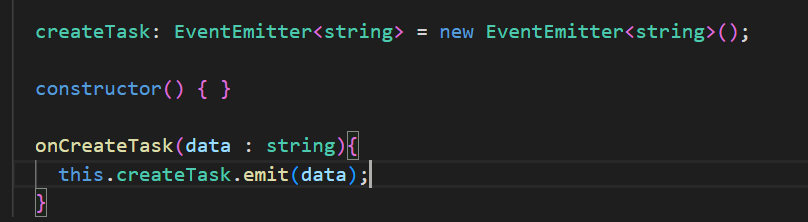
## Subject In RXJS:

A subject is a special type of observable that allows values to be multicasted to many observer

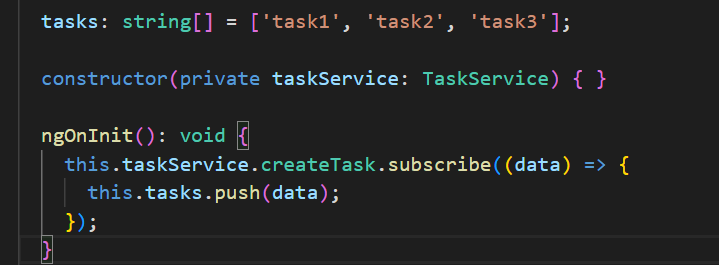
Subject Like EventEmitter.

Cross component Communication is easy

We Have created the One Task Service in that we call createTask method and give the data to the eventEmitter<string>



Now we will subscribe this event emitted data to another component and get the values what we are passing below is the code:

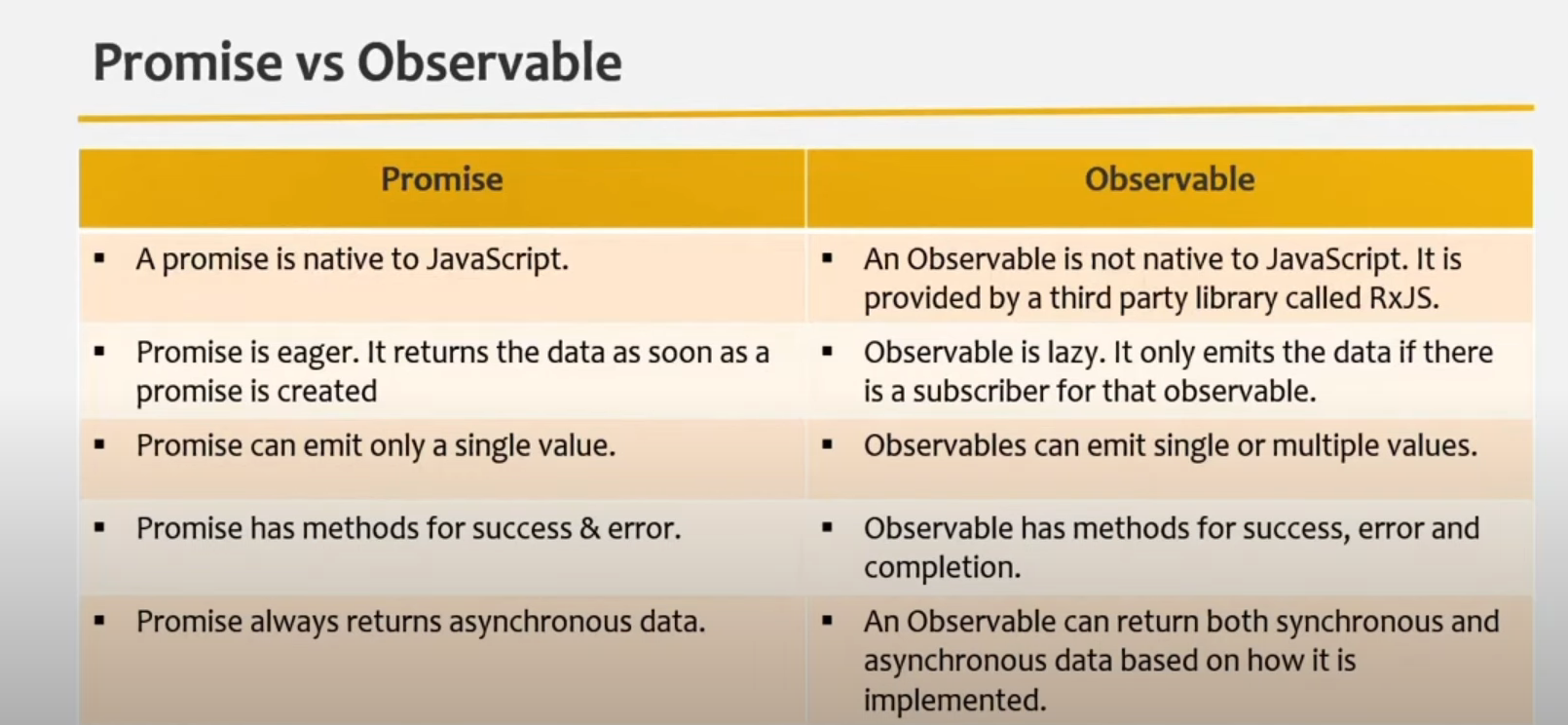


## Observable Vs Subject:

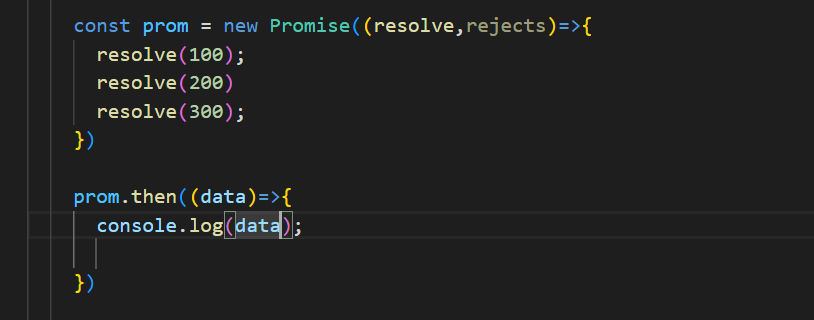
A subject is multicast(means it will give you the same value through out all the subscriber) and observable Is unicast(means give the different values)

To handle Asynchronous data we use Promise or Observable

But what is the main difference between them!!



If we have n number of data then Obseravable give all the multiple values but promise gives the single value if we have multiple value in the streams



Promise have 2 things success and error

Observable have 3 things success, error, Completion.

# Unsubscribe from an observable

Onservable emits some data when we subscribe it we will get the data if we unsubscribe

