1. What is meta data  
   data about data. Tells angular to process a class.  
   @NgModule, @Component…  
   .
2. Creating custom @ngmodule.  
    @NgModule({  
    declaration:[], componens,directives,pipes

Imports:[], other modules need to import.  
 exports:[], If you want other modules to use the componens,directives,pipes  
 providers:[], services, injection  
 entrycomponents:[], dynamically loaded needs to be declared here  
 bootstrap:[]  
})  
  
 1.Root Module

2.Feature Module

3.Shared Module -> There are many components, directives & pipes, which we may like to share across various modules. All these components should go into the shared module.

4.Core Module

1. What is properly binding  
     
   {{}} ->

<input type=”text” [value]=”title”/>

1. Event binding?  
     
   (click)=”testClick()”  
   testClick(){  
     
   }
2. Two way binding  
   forms module need to import at module.  
   [(ngModel)] = “username” (ngModelChange)=” username=$event”  
     
   [ngModel]="currentHero.name" is the syntax for one-way binding, while,

[(ngModel)]="currentHero.name" is for two-way binding, and the syntax is compound from:

[ngModel]="currentHero.name" and (ngModelChange)="currentHero.name = $event"

1. Router events (router start and end).
2. Lazy loading syntax.

{path: "admin", loadChildren: () => import('./admin/admin.module').then(m => m.AdminModule)},  
(or)  
{path: "admin", loadChildren:'./admin/admin.module#AdminModule'},  
  
Angular Preloading Strategy :- By Preloading the lazy loaded module, the user do not have to wait for the module to be downloaded as the module is already downloaded in the background

RouterModule.forRoot(routes, {preloadingStrategy: PreloadAllModules})

PreloadAllModules

This strategy will preload all the lazy loaded modules.

it is by default

RouterModule.forRoot(routes,

{

preloadingStrategy: NoPreloading

}  
  
Custom preloading strategy

{path: "admin", loadChildren:'./admin/admin.module#AdminModule',data: { preload: true, delay:5000 }},

import { Injectable } from '@angular/core';

import { Observable, of, timer } from 'rxjs';

import { flatMap } from 'rxjs/operators'

import { PreloadingStrategy, Route } from '@angular/router';

@Injectable()

export class CustomPreloadingStrategy implements PreloadingStrategy {

preload(route: Route, loadMe: () => Observable<any>): Observable<any> {

if (route.data && route.data['preload']) {

var delay:number=route.data['delay']

console.log('preload called on '+route.path+' delay is '+delay);

return timer(delay).pipe(

flatMap( \_ => {

console.log("Loading now "+ route.path);

return loadMe() ;

}));

} else {

console.log('no preload for the path '+ route.path);

return of(null);

}

}

}

providers: [CustomPreloadingStrategy],

1. HttpInterceptor.
2. ngFor trackBy.  
   Angular Trackby option improves the Performance of the ngFor if the collection has a large no of items and keeps changing. Learn why we need it and how to use it to improve the performance of the ngFor.

Hence even if we refresh the data from the back end, the unique id will remain the same and the list will not be rendered again.

It takes two arguments index,item.  
  
\*ngFor=”let user of Users”  
<li \*ngFor="let item of list; trackBy:identify">{{item.name}}</li>

identify(index, item){

return item.name;

}

1. Angular life cycle
2. Viewchild, viewcontext.
3. Pipes-> pure and impure  
     
    display value transformations that can declare in your html.  
   We use them to change the appearance of the data before presenting it to the user.  
     
   export class SqrtPipe implements PipeTransform {

transform(val : number) : number {

return Math.sqrt(val);

}

}

1. Async Pipe.

The async pipe allows us to subscribe to an Observable or Promise from the template and returns the value emitted. The async pipes subscribe to the observable when the component loads. It unsubscribes when the component gets destroyed.

1. Directive and types.  
     
   structural:- \*ngIf, \*ngFor ‘starts with \*’  
   attribute:- which change style [ngClass] -> [ngClass]=”{greenClass:title=== ‘abc’}”; [ngStyle]=”{backgroundColor:color}”  
   component:- normal component.(which has own html).  
     
   @Directive({  
    selector:’[test]’  
   })  
     
   export class directative{  
    constructor(public Eleref:ElementRef){}  
      
    ngOnInit(){  
    this.Eleref.nativeEement.style.backgroundColor = ‘green’;  
   }  
   }
2. What is ElementRef?  
     
   a wrapper around a native element inside of a view.  
   not to use:-  
   direct access to the dom can make xss attacks  
   normally native element or null if direct access to native element is not supported.

this.renderer.setProperty(this.divHello.nativeElement,'innerHTML',"Hello Angular"); // to avoid xss attacks.

1. What is HostBinding and HostListener?  
     
   @hostListener:- allows to handle events of host element  
     
    @HostListener(‘mouseenter’) mouseEnter(){

This.color = ‘yellow’;

}

@HostListener(‘mouseleave’) mouseLeave(){

This.color = ‘green’;

}

@hostBinding:- allows you set propery of the host element.  
  
@HostBinding(‘style.backgroundColor’) color;

this.color =’green’;

1. How to send data from parent to child?  
     
   @Input() name:string; // we can see data in ng oninit  
     
   <div [name]=”hellow World”></div> //propery binding
2. How to send data from child to parent?  
     
   @Output() detach :eventEmitter<any>= new eventEmitter<any>();  
     
    this.detach.emit(‘test’);  
     
   <div (detach)=”event($event)”></div>
3. What are services?  
     
   for writing business logic  
   services are singleton  
   services are injected by injector.  
   injectors rely on providers to inject the dependencies.  
     
   @Injectable({  
    providedIn: ‘root’ // from angular6 (Tree shakeable)  
   })
4. Dependency injection

Module level -> @NgModule or @Injectable  
Component level -> @Directive or @Component

1. What are observables  
     
   passing message b/w publisher and subscribers (any number of subscriable). (handling async). (send streem of data)  
   observable streem get closed when there is error or completed.  
     
   angular uses observable default in.  
   services  
   output data from child to parent.  
   routing and form modules user input.  
     
   Obserable.create((observer)=>{

observer.next(‘Hello’);  
 observer.error(‘error’);  
 observer.complete();  
}).subscribe((res)=>{  
 console.log(‘res’,res)  
},(err)=>{  
 console.log(‘err’,err);  
},()=>{  
 console.log(‘complete’);  
});

1. What are the advantages of observable over promises?  
     
   multiple data.  
   send data only there is subscribe.  
   can call unsubscribe when there is pending data.   
   operaters we can transform data.
2. How to use HttpService?  
     
   HttpClienModule in ngModule, HttpClient service in the constructor.
3. Angular life cycle?  
     
   ngOnChange :- @input of the parent change. Here we will know new and old value. It calls whenever change occur. (simple changes)

ngOnInit:- when the component is initialization. It calls only once.

ngDoCheck:- whenever binding the values from view to class and class to view.(detection change and to act on changes that angular can’t or won’t detect on its own). (We also looked at how to use key-value differs and Iterrable differs.). when ever we

send non-primitive data types. When every any value or event occur changes in parent or child this hook will execute.

ngAfterViewInit:- initializes the component view and child view.(complete view is ready). @viewChild and template refs we can access.  
ngAfterViewChecked:- checks the component view and child view changes.

ngAfterContentInit :- external content into the component view.

ngAfterContentChecked:- checks the content projected into the component.

ngDestory:-

1. What is ngContent?  
     
   <ng-content> tag places for that dynamic content parsed angular will replace form parent to child.  
     
   <ng-container ngProjectAs="headersub">

</ng-container>

1. @ContentChild (need to check template reference in content projection using <div>)  
     
   content child which can access the ng-content data from parent to child   
   in ngAftercontentinit life cycle.

content projection :- We cannot use that technique to pass the content which includes the HTML elements, CSS, etc to the child component.

<header><h1>Angular</h1></header> -> parent  
<ng-content select="header" ></ng-content> -> child.

<div class="content">One framework. Mobile & desktop.</div> ->parent

<ng-content select=".header" ></ng-content> -> child

1. @viewChild  
     
   Element reference we can access in ngAfterViewInit.
2. Routes  
     
   Finally, we will learn how to build the Angular application using the PathLocationStrategy (HTML5 routing ) HashLocationStrategy.(hash style Routing )  
   need to import RoutingModule  
     
   const routes:Routes =[  
    {path:””, redirectUrl:’/signin’ pathMatch:’full’}  
    {path:’signin’,component:SigninComponent}  
    {path:’home’,component:homeComponent,children:[  
    {path:’test’,component:TestComponent}  
    ]}  
   ]  
     
   RouterModule.forRoot(routes);  
     
   <router-outlet></router-outlet> // place at where routes to load.  
     
   to navigate one route to routerLink=”signin”  
   <li><a [routerLink]="['product/detail/1']">Product 1 Overview</a></li>

this.\_router.navigate(['product/detail/1'])

If the First segment of the route starts with “/“, then the path is considered to be Absolute path

If the First segment begins with “./” or it does not begin with a slash, then the path is considered to be the relative path.

And if the First segment begins with “. ./“, then the path is relative to the parent route. (one level up)

fragment: string

Sets the hash fragment for the URL.

Example:

The following code constructs the URL as “/home#top”

this.router.navigate(['/home'], { fragment: 'top' });  
  
this.router.navigate(['../Detail'], { relativeTo: this.activatedRoute });

this.router.navigate(['/products'], { queryParams: { page: 1 } });

Passes the query parameters of the current route to the next route

this.router.navigate(['/view'], { queryParams: { page: 2 },preserveQueryParams: true, queryParamsHandling: "merge" });

skipLocationChange: boolean

You can change the route, without changing the URL in the browser. This Navigates to a new URL without pushing a new state into history.

this.router.navigate(['/view'], { skipLocationChange: true });

router.navigateByUrl('/inbox/33/messages/44');  
router.navigate(['/inbox/33/messages/44']);  
router.navigate(['/team', 33, 'user', 11]);  
router.navigateByUrl(router.createUrlTree(commands, { relativeTo: route }), { relativeTo: route });  
this.router.navigate([‘sigin’]);   
this.router.navigate([‘login’],{queryParams:{returnUrl:state.url}}); // this.route,snapshot.queryParams.

<a [routerLink]="['product']" [queryParams]="{ page:2 }">Page 2</a>  
 this.router.navigate(['/product'], { queryParams: { page: pageNum } });

this.sub = this.Activatedroute.queryParamMap .subscribe(params => {

this.pageNum = +params.get('pageNum')||0;

});

this.Activatedroute.snapshot.queryParamMap.get('pageNum')||0;;

1. Angular Forms

Reactive Forms(also known as Model-driven forms)  
this.forms= new FormGroup({  
name: new FormControl(null,[validators.required]);  
});  
  
<form [formGroup]= “forms”>  
 <input formControlName=”name” type=”text”/>  
</form>  
formGroupName="address"

<div [formGroupName]="i">

</div>  
 this.skillsForm = this.fb.group({

name: '',

skills: this.fb.array([]) ,

});  
  
this.reactiveForm.valueChanges.subscribe(x => {

console.log(x);

})  
this.reactiveForm.statusChanges.subscribe(x => {

console.log(x);

})  
  
TEMPLATE DRIVEN FORMS  
 this. contactForm = {  
 name:’test67’  
 }  
<form #contactForm="ngForm">  
<input type="text" name="firstname" ngModel>  
</form>

ngModelGroup=”address”

<div formArrayName="skills">

<div \*ngFor="let skill of skills().controls; let i=index">

</div>

</div>

1. Pwa
2. Angular Pass Data to Route: Dynamic/Static

{ path: 'static', component: StaticComponent, data :{ id:'1', name:"Angular"}},

this.activatedroute.data.subscribe(data => {

this.product=data;

})  
dynamic data  
<a [routerLink]="['dynamic']" [state]="{ id:1 , name:'Angular'}">Dynamic Data</a>

this.router.navigateByUrl('/dynamic', { state: { id:1 , name:'Angular' } });  
   
<li><a [routerLink]="['home']" routerLinkActive="active">Home</a></li>

<li><a [routerLink]="['product']" [routerLinkActive]="['active']">Product</a></li>  
  
Exact matching

<a routerLink="/product" routerLinkActive="class1 class1" [routerLinkActiveOptions]="{exact:

true}">Product</a>

1. Angular Router Events

The Angular Router raises events when it navigates from one route to another route. It raises several events such as NavigationStart, NavigationEnd, NavigationCancel, NavigationError, ResolveStart, etc. You can listen to these events and find out when the state of the route changes. Some of the useful events are route change start ( NavigationStart ) and route change end ( NavigationEnd). In this tutorial, we learn what is router events are and how to listen to them using Example code.

this.router.events

.subscribe(

(event: NavigationEvent) => {

if(event instanceof NavigationStart) {

console.log(event);

}

});

1. Guards  
     
   Guards Order of execution:- CanDeactivate(),CanActivateChild(),CanActivate(),CanLoad(),Resolve()  
     
   These guards help us to secure the route or to perform some actions before navigating into a route or leaving the route  
   canActive  
     
   canActivate:[ AuthGuard]  
     
   export class AuthGuard implement CanActivate{  
    constructor(private router:Router){}  
    canActivate(route:ActivatedRouteSnapshot,state; state:RouterStateSnapshot): boolean{  
    return true  
    }  
   }  
     
   canActiveChild  
     
     
   canDeactive  
     
   canDeactivate(component:IDeactivateComponent,

                route: ActivatedRouteSnapshot,

                state: RouterStateSnapshot,

                nextState: RouterStateSnapshot) : Observable<boolean> | Promise<boolean> | boolean {

**return** component.canExit ? component.canExit() : **true**;

  }

CanLoad :- This guard works similar to CanActivate guard with one difference. The CanActivate guard prevents a particular route being accessed. The CanLoad prevents entire lazy loaded module from being downloaded, Hence protecting all the routes within that module.  
  
canLoad(route: Route): boolean {

    let url: **string** = route.path;

    console.log('Url:'+ url);

**if** (url=='admin') {

      alert('You are not authorised to visit this page');

**return** **false**;

    }

**return** **true**;

  }

Reslove :- You can use the guard to pre-fetch the data from the backend API, before activating the route  
  
 resolve: {products: ProductListResolveService}  
  
 resolve(route: ActivatedRouteSnapshot,

            state: RouterStateSnapshot): Observable<any> {

**return** **this**.productService.getProducts();

    }

1. Authentication (INTERCEPTER)  
     
   providers:[  
   {provide:HTTP\_INTERCEPTORS,useClass:BasicAuher,multi:true  
   }  
   ]  
     
   export class BasicAuher implements HttpInterceptor{  
    intercept(req:HttpRequest<any>,next:HttpHandler):Observable<HttpEvent<any>>{

req.clone({  
 setHeader:{  
 Authorization:`Basic`  
 }  
 })  
  
 next.handle(req);  
 }  
}

1. Change dectaction statey.  
   changeDetection:-  
     
   changeDetectionStrategy.Default - it checks all components when change occur  
   changeDetectionStrategy.OnPush – it not render if there is also change occur also in parent level.  
     
   changeDetectorRef:-  
     
    markForCheck():- explicitly mark the view as changed so that it can be checked again   
   detach():- not checked until it is reattached.  
   detechChanges: check this view and its children  
   reattach:- when we deached view to the chage detection tree.  
     
   with out onPush. In constructor we use “detach”(it works as same as onPush). To make change we need to use “detectChanges”.  
     
    with out onPush. In constructor we use “reattach”(it works as same as default).”. with onPush “reattech” we need to use markForCheck.

1. Component view provider.  
     
   If we want to have one instance of a service per component, and shared with all the component’s children, we configure it on the providers property on our component decorator. Child is a view children.

If we want to have one instance of a service per component, and shared with only the component’s view children and not the component’s content children, we configure it on the viewProviders property of our component decorator.

1. How to handle error in angular apis.
2. Ng-template, ng-content, ng-container  
     
   https://vibhas1892.medium.com/difference-between-ng-template-ng-container-and-ng-content-a1d264619655
3. Difference b/w navigate and navigatebyurl.  
   s  
   router.navigateByUrl('/inbox/33/messages/44');  
   router.navigate(['/inbox/33/messages/44']);  
   router.navigate(['/team', 33, 'user', 11]);  
   router.navigateByUrl(router.createUrlTree(commands, { relativeTo: route }), { relativeTo: route });  
   this.router.navigate([‘sigin’]);   
   this.router.navigate([‘login’],{queryParams:{returnUrl:state.url}}); // this.route,snapshot.queryParams.
4. Package.json and package.lock.json
5. Test cases.
6. Ng-switch

<div [NgSwitch] = "day">

<div \*NgSwitchCase = 0>Today is Sunday! You deserve a break</div>

<div \*NgSwitchCase = '1'><h3>Today is Monday, Hope you have productive Week</h3></div>

<div \*NgSwitchDefault>Uh oh! Somethings wrong</div>

</div>

1. What are the compilers available.

JIT(Just-in-Time) compilation   
the application compiles inside the browser during runtime(in deveplment phase)(it compail code in browser at run time)

AOT(Ahead-of-Time) compilation  
the application compiles during the build time (at time of exection only compiles and remove angular related files.) ng build –prod

#### **What is ViewEncapsulation and how many ways are there do to do it in Angular? (shadow dom)**

**Answer**: To put simply, ViewEncapsulation determines whether the styles defined in a particular component will affect the entire application or not. Angular supports 3 types of ViewEncapsulation:

* Emulated – Styles used in other HTML spread to the component (default) (only for that component) (unique attribute)
* shadowDom – Styles used in other HTML doesn’t spread to the component (it remove all 3rd party syles for that component)
* None – Styles defined in a component are visible to all components of the application (it allows all component. ) (remove unique attribute)

1. NgClass  
     
   **type one**

[class.my\_class] = "step === 'step1'"

**type two**

[ngClass]="{'my\_class': step === 'step1'}"

and multiple option:

[ngClass]="{'my\_class': step === 'step1', 'my\_class2' : step === 'step2' }"

**type three**

[ngClass]="{1 : 'my\_class1', 2 : 'my\_class2', 3 : 'my\_class4'}[step]"

**type four**

[ngClass]="step == 'step1' ? 'my\_class1' : 'my\_class2'"

1. Mvc arcticure.

Diagram

Description automatically generated

1. Difference b/w templae and reactive form.
2. Template-driven forms make use of the "FormsModule", while reactive forms are based on "ReactiveFormsModule".
3. Template-driven forms are asynchronous in nature, whereas Reactive forms are mostly synchronous.
4. In a template-driven approach, most of the logic is driven from the template, whereas in reactive-driven approach, the logic resides mainly in the component or typescript code. Let us get started by generating a component and then we'll update our form code

Tdf:-   
@ViewChild('f') courseForm: NgForm; // we need to import ngForm  
<form #f=”ngForm”></form>  
  
reactiveform:-  
 courseForm: FormGroup;  
this.courseForm = new FormGroup({

'courseName': new FormControl(null, Validators.required),

'courseDesc': new FormControl(null),

'courseAmount': new FormControl(null)

});  
<form [formGroup]="courseForm" (ngSubmit)="onSubmit()"></form>

1. List types of data binding.

String Interpolation {{}} (it binds context of data)

Property Binding, // [value]='myBlog'

Event Binding, ()

Two Way Data Binding. (combination of property binding & event binding)

1. DI in angular (Dependency Injection)  
     
   It allows us to inject dependencies into the Component, Directives, Pipes, or Services.  
   Dependency Injection is a design pattern that is used to resolve the dependencies in different classes or components. you can inject the service in a component's constructor as below.  
   **this**.productService=**new** ProductService(); // to get instance. With out di. Testing problem (when we added parameter here also we maintain)  
   <https://blog.angular-university.io/angular-dependency-injection/>  
     
   The Angular creates two Injector trees when the Application bootstraps. One is the ModuleInjector tree for the Modules and the other one is the ElementInjector tree which is for the Elements (Components & Directives etc).  
     
   Injectable & @Inject :-

The Injectable is a decorator, which you need to add to the consumer of the dependency. This decorator tells angular that it must Inject the constructor arguments via the Angular DI system.

The @Inject takes the Injector token as the parameter. The token is used to locate the dependency in the Providers.  
export class ProductService{

constructor(@Inject(LoggerService) private loggerService) {

this.loggerService.log("Product Service Constructed");

}

}

Class Provider : useClass

Value Provider: useValue

Factory Provider: useFactory

Aliased Class Provider: useExisting

1. ProvidedIn root, any & platform in Angular

root -> creates same instance for parent and lazy loading modules.  
any -> creates different instance for parent and lazy loading modules.

Platform-> create same instance for our different application’s.

1. What is a Module Loader

Module loader takes a group of modules with their dependencies and merges them into a single file in the correct order. This process is called as Module bundling.  
in our application we will include no.of js files using <scriptTag> they load one by one. Page speed and each file with different request separatly. To solve this problem we we use bundler all into single file with minify(remove spaces and comments).

Two types of module loader mainly systemjs and webpack(angularCli by default webpack).

1. What is platformBrowserDynamic

platformBrowserDynamic is the module, which is responsible for loading the Angular application in the desktop browser.

platformBrowserDynamic().bootstrapModule(AppModule) .catch(err => console.error(err));

1. Angular Directives  
     
   manipulate the DOM. You can change the appearance, behavior, or layout of a DOM element using the Directives.  
   a) Component Directive

b) Structural directives (ngFor,ngIf,ngSwitch) \*ngFor="let movie of movies; let i=index;"

c) Attribute directives (ngModel,ngClass,ngStyle)

1. ngStyle (to add inline style)

<some-element [ngStyle]="{'font-size': '20px'}">Set Font size to 20px</some-element>

<div [ngStyle]="{'background-color':status === 'error' ? 'red' : 'blue' }"></<div>

<p [ngStyle]="{'color': 'purple','font-size': '20px','font-weight': 'bold'}">Multiple styles</p>

1. custom directive  
     
   Create a custom directive using the @Directive decorator.

We will create both custom attribute directive & custom Structural directive.

How to setup selectors

Pass value to it using the @input.

How to respond to user inputs,

Manipulate the DOM element (Change the Appearance) etc.

1. What is an Angular Service

Service is a piece of reusable code with a focused purpose. A code that you will use in many components across your application

1. HTTPClient  
   The Angular HTTPClient makes use of observable. Hence it is important to understand the basics of it

1. Angular features  
   a) Since the Angular 7, we have new feature called conditional polyfill loading. Now Angular builds two script files, one for es2015 & another for es5. The es2015 (es6) is for modern browser and es5 is older browsers, which do not support the new features of es2015.

Note the nomodule attribute, which tells the modern browser to ignore the script and do not load it. Hence es5 scripts are not loaded in the modern browsers.  
b) The angular-cli.json was the configuration file in Angular 5 and before. It is now angular.json since the version Angular 6.

1. List upgrads of angular versions.

<https://www.ngdevelop.tech/angular/history/>

1. Template reference

Giving reference to a element in html to access.

<div #e>hello world</div>

Child to parent communication.

1. ViewChild and viewChildern

from child to parent communication, we can use. We need to import in parent component.

To access methods and property in child components and same component html elements also access dom .

1. <ng-container> vs <ng-template> vs <ng-content>

<ng-container> -> it will not appear tag in dom it eliminated . in a tag we need to use one structural directive. When no need to add additional tag that time we can use. Normally ng-container tag data will appear.

<ng-template> -> when we write any contact in this tag it will not show. If need to show this tag data we need to connect with any structural directive. To use structural directives we need to use ‘[]’. it will not appear tag in dom it eliminated.

<ng-content> -> when we need to share dynamic html template from patent to child we need to use this tag in child component.

1. Cold vs hot observables

Cold-> value which produces is created inside the observable. Each time the subscribe method called on cold observable the integer code executed every time. That means cold observable pushes value to single observer at a time. (unicast (interval , ajax)).

Hot-> the value producer exist outside the observable. Since the value outside the observable it can execute if there are no observers. Shared value allows multiple observers receive the same value at a same time. (multicast (fromEvent)).

1. Types of Rxjs subjects (multicast)

Subject is a class to create subject we need to use ‘new’ keyword. The Subjects are special observable which acts as both observer & observable.

Observable -> unicast. Inside an observable we need to send data by using ‘next’.

Subject-> to share data across multiple components same data.

Behaviour subject-> initial value we need to provide. requires an initial value and stores the current value and emits it to the new subscribers. Sends the initial value to the observer when subscribed before emitting the value. After emitting the values then the subject send the latest value to all observers.

replaySubject -> replays old value to new subscribers by emitting them when they first subscribe. (buffersize,windowtime).

bufferSize -> how many items are store in the buffer, default infinite.

windowTime -> amount of time to hold a value in buffer before removing it from the buffer.

ReplaySubject(3,2000)

asyncSubject -> only emit last emitted value. When execute completed.

1. Rxjs library

Observable is a class. To create of this type, we have provided a call back function. (next,error,complete)

Let obs = new observable((o)=>{  
 o.next(1);  
})

Operators :- of,from,create,fromEvent,Interval,Timer,map,filter,reduce,merge,join,forkjoin,debounce.

Creation operators :-

Of(6,7,8,9) :- it will sends individual data. It takes multiple inputs.

From([6,7,8,9]) :- it takes only one input. We need to give ittrable items. String,object,arrays it can ittrable.

Range(7,10) :- it will give range b/w those numbers. From 7 to next 10 numbers it will emit.

Interval :- from that duration it will emit the value.

fromEvent :- event information will send to handle observable form.

@viewchild(‘btn’) btn:any;

fromEvent(this.btn.nativeElement, ‘click’).subscribe((data)=> console.log(data));

pipeable operators :-

map :-

filter:-

forkjoin:-

debounceTime :- we need to use in pipe. debounceTime(5000).

Merge :- combain two observables and as single obserables.

Take :- only limited number only we will capture.

Pluck -> observable emitting data in form of object we need to take particular data in object. Pluck(‘address’,’street’).

Skip(2) -> it skip first 2 element observable emitted.

distincuntilChanged -> observable emitting data if previous and present data same. It will not send to subscribers.