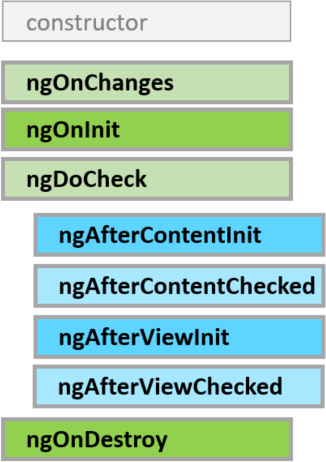
Life cycle Hooks

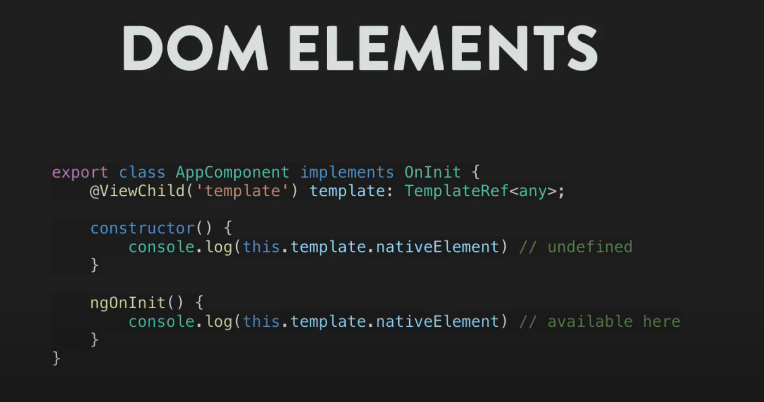
Life cycles hook are the functions which runs at specific point of components life.

Order of life cycle hooks:



Green is for both components and directives and blue are only for components

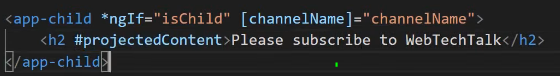
* **ngOnChanges** − When the value of a data bound property changes, then this method is called. When you are passing data from parent to child using data bound property, ngOnChanges in child runs. Note: this only runs whenever @Input property changes
* **ngOnInit** − A lifecycle hook that is called after Angular has initialized all data-bound properties of a component/directive. And you can use ngOnInit() method to handle any additional initialization tasks.
* **ngOnInit vs Constructor:**
  + constructor is typescript feature
  + constructor is mainly used for dependency injection.
  + ngOnInit is better place to write Logic
  + suppose we have parent component and child component then order of execution is parent constructor->child constructor->parentLifeCycle hooks> Child lifecycle hooks
  + so if there is some changes coming from parent to child and we written some logic in child constructor so it will not detect change so its good practice to write logic in ngOnInit or other life cycle hooks.
  + And also, we cannot access DOM element inside constructor because component is not yet fully initialized yet.



* **ngDoCheck** – It runs after ngOnChanges, and whenever some property of current component changes using two way binding etc. then ngDoCheck will run.
* **ngAfterContentInit** - This is called in response after Angular projects external content into the component's view.

whenever some HTML content is render in child from parent (content projection) using <ng-content></ng-content> tag, in order to access that content which we rendering using ng-content we can use ngAfterContentInit hook.







* **ngAfterContentChecked** − This is called in response to checks the content projection is completed or not.
* **ngAfterViewInit** − This is called in response after Angular initializes the component's views.

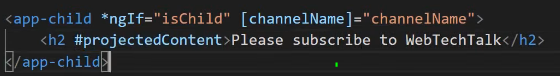
**e**.g when we access some html element using template reference variable and @viewChild() decorator, then access of element only gets after view is initialized so we can access using ngAfterViewInit Hook.

* **ngAfterViewChecked** − This is called in response to checks the component's views Initialization is completer or not
* **ngOnDestory -**  when component is distoryed then ngOnDestory is called, when you want to write some logic on component destroy e.g cleaning subscription, cleaning timers inside components etc. to avoid memory leaks

ng-content vs ng-container vs ng-template

* ng-content – whenever some HTML content is render in child from parent (content projection) using <ng-content></ng-content> tag, in order to access that content which we rendering using ng-content we can use ngAfterContentInit hook.



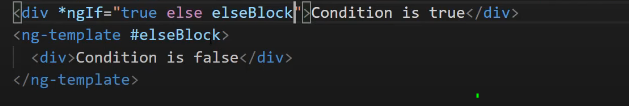




* ng-container – ng-container will help to group elements which do not interfere with styles we defined in css.
* ng-template – ngIf in html elements transform into ngtemplate behind the scene.



And it generally used for writing else condition.



@viewChild() -> this decorator used to get element reference from html and also use to transfer data from child to parent.

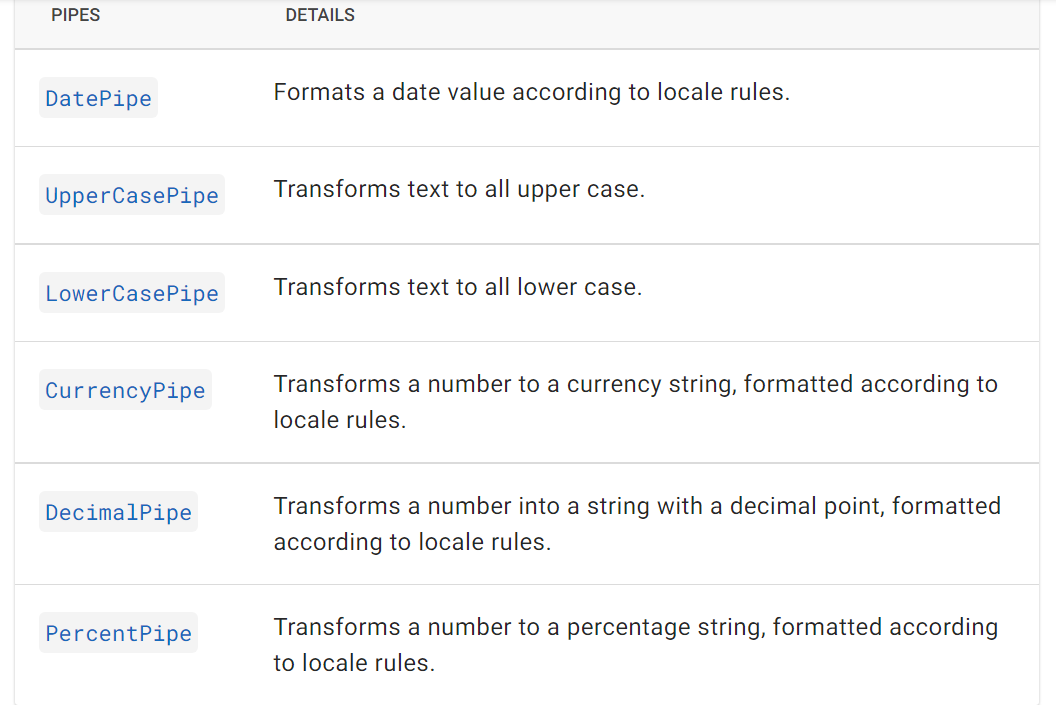
Pipes:

Pipes are simple functions to use in [template expressions](https://angular.io/guide/glossary#template-expression) to accept an input value and return a transformed value.

e.g <p>The hero's birthday is {{ birthday | date }}</p>

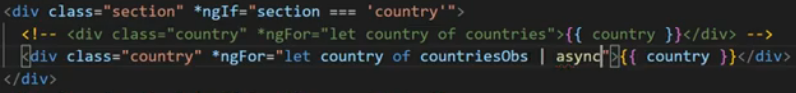
<p>The hero's birthday is {{ birthday | date:"MM/dd/yy" }} </p>

All types of datetime values displays the date in **‘MMM d, y’** format which is default Angular date format e.g‘Jun 15, 2019’



Async Pipe:

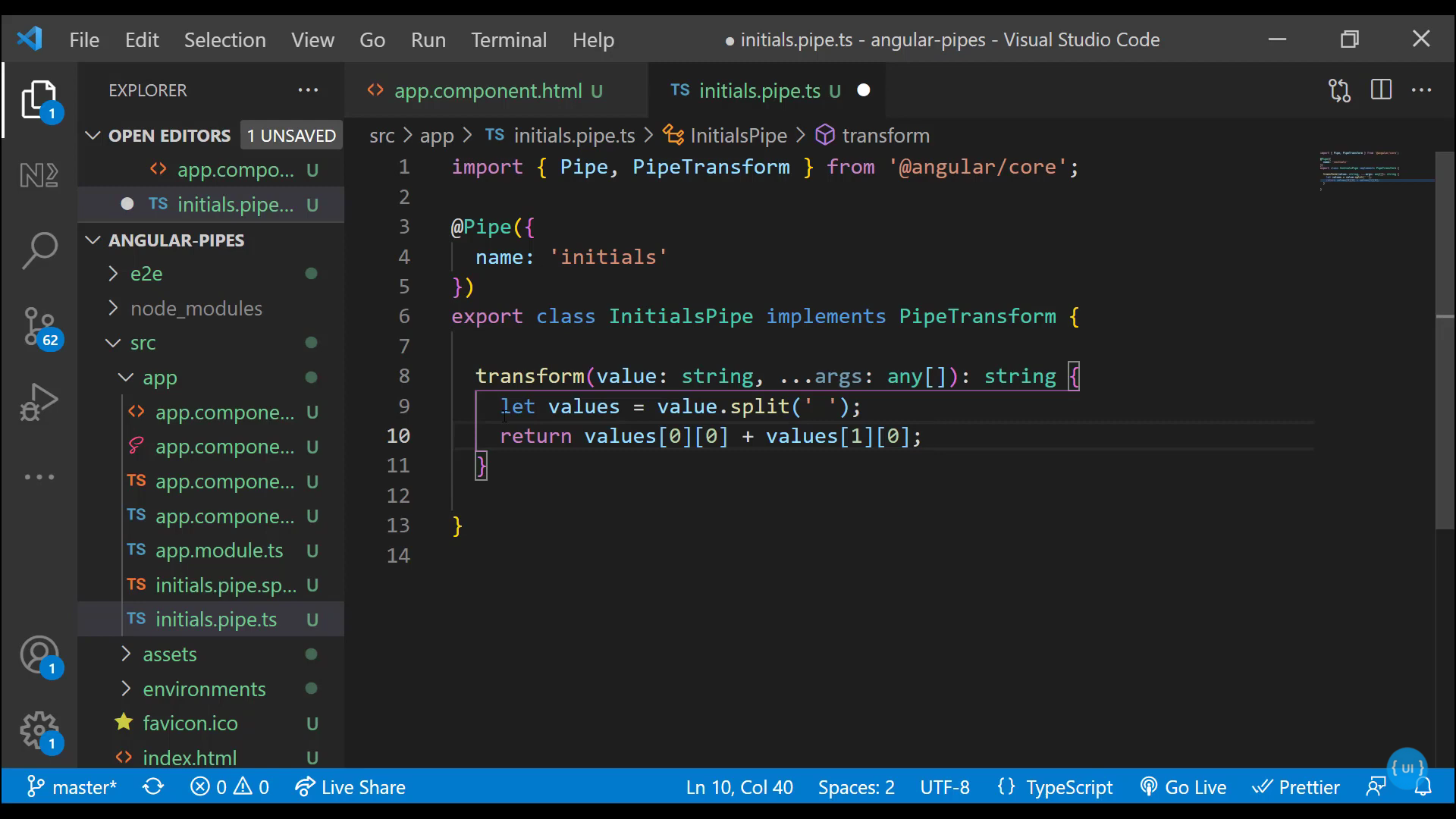
It is used subscribe observable in Template Itself and unsubscribe automatically on component Destroy.



Custom Pipes:

Looping through Observer which is defined in Service e.g and assigned to countriesObs variable in component.. below is Observer returned from service.

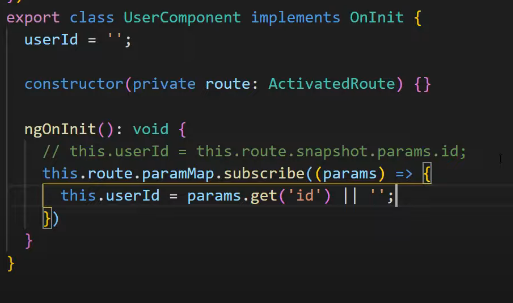
Custom pipes uses transform() method insidePipeTransform class





Activated Route vs Activated Route Snapshot:

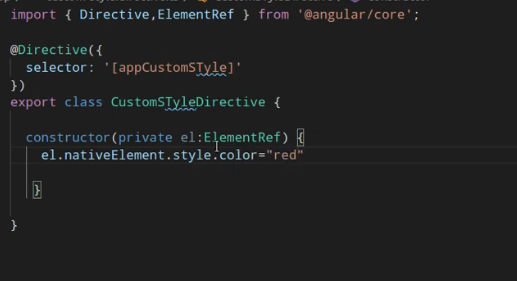
this.route.snapshot.params.id use to get router parameter when we load new Component, if router params is changing in same component then to access that params we use route.paramMap.subscribe() method because ngOninit only runs on 1st load



Directives:

Structural Directive: ngIf, ngFor, ngSwitch

Attribute Directive: ngStyle, ngClass



@ViewChild(‘ChildComponentName’,{static:false}) child

If child component have data variable

Then access child data variable by this.child.data

If we want to access template reference variable on some event then we pass element ref on event function

e.g <button (click)=”func(input)”></button>

To access html reference variable e.g <input type=’text’ #input>

To access html element without any event we need to use @ViewChild.

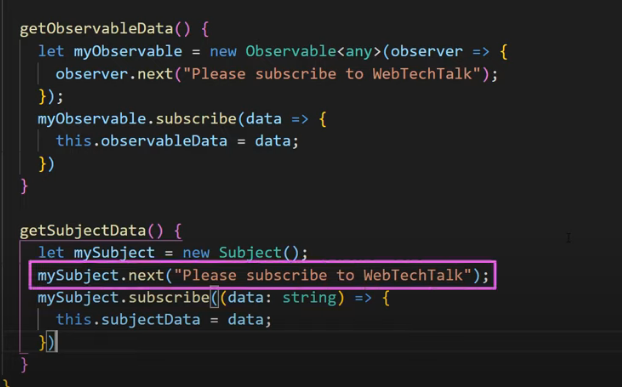
@ViewChild(‘input’,{static:true}) myCustomInput

ngOnInit(){

this.myCustomInput.nativeElement.focus()

}

Subject Vs Observerable



Suppose this above two function gets Invoked on button click,

Observables are lazy so it will Emit data only on subscribe, while subject data gets emitted on runtime, so we need to subscribe 1st before emitting to receive the data.

Subject are used to transfer data between components using mssgService.

Subject vs ReplaySubject vs Behaviour Subject

Subject: Subject will read data which is emitted only after subscribing.

Behaviour Subject: Behaviour Subject will keep track of last emitted value.

ReplaySubject: this will keep track of all previously emitted data and emit all data which in its memory.

ViewEncapsulation:

It is used to scope to styles, we define inside component decorator

3 types:

1. encapsulation: ViewEncapsulation.Emulated (default)
2. encapsulation: ViewEncapsulation.None - here current component styles will scope too global also.
3. encapsulation: ViewEncapsulation.ShadowDom - It will Ignore the global styles, but it will shadow the styles from the component where ViewEncapsulation is None.