Angular concepts

pipes :

Use pipes to transform strings, currency amounts, dates, and other data for display.
A pipe class must implement the PipeTransform interface.Pipes are defined using the pipe "|" symbol. Pipes can be provided with arguments by using the colon (:) sign.

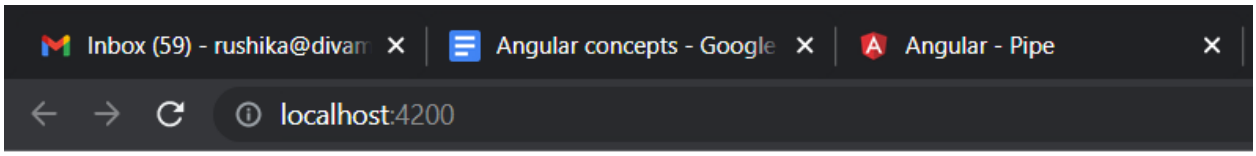
Built-in pipes:

- DatePipe: Formats a date value according to locale rules.
- UpperCasePipe: Transforms text to all upper case.
- LowerCasePipe: Transforms text to all lower case.
- CurrencyPipe: Transforms a number to a currency string, formatted according to locale rules.
- DecimalPipe: Transforms a number into a string with a decimal point, formatted according to locale rules.
- PercentPipe: Transforms a number to a percentage string, formatted according to locale rules.

{{ (true ? 'true' : 'false') | uppercase }}

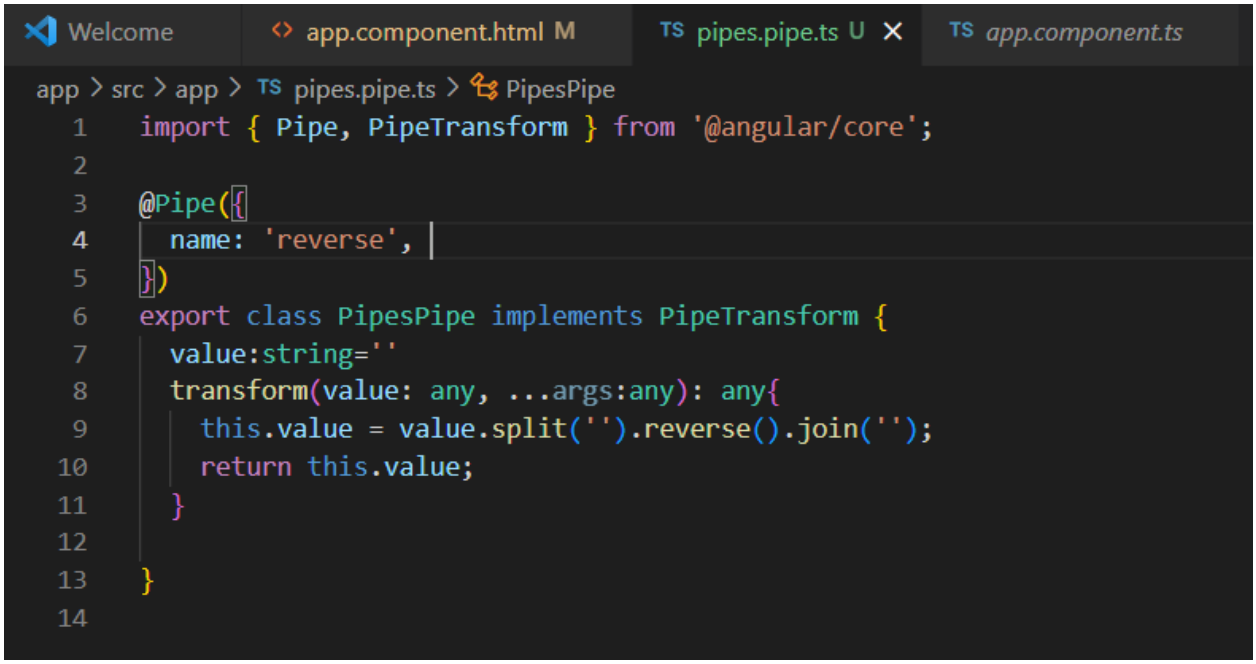
Custom pipes:

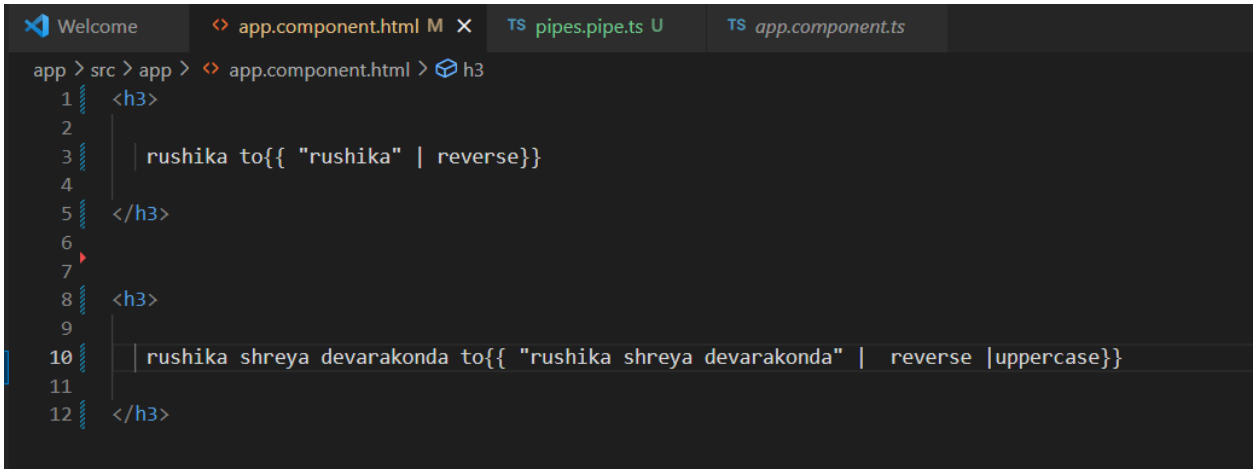
Create custom pipes to encapsulate transformations that are not provided with the built-in pipes. Then, use your custom pipe in template expressions, the same way you use built-in pipes—to transform input values to output values for display.



rushika toakihsur

rushika shreya devarakonda toADNOKARAVED AYERHS AKIHSUR





Observables:

Angular makes use of observables as an interface to handle a variety of common asynchronous operations.

Life cycle methods:

In Angular, components have a lifecycle, managed by Angular itself, that follows a series of phases from creation to destruction. The following is a list of some of the key lifecycle phases

ngOnChanges():

Respond when Angular sets or resets data-bound input properties. The method receives a SimpleChanges object of current and previous property values.

Called before ngOnInit() (if the component has bound inputs) and whenever one or more data-bound input properties change.

app.component.html M

onchange > src > app > app.component.html > button

```
<app-child [counter]="counter"></app-child>
    <button (click)="incrementCounter()">Increment Counter</button>
```

app.component.ts M

onchange > src > app > app.component.ts > AppComponent > incrementCounter

```
import { Component } from '@angular/core';

@Component({
  selector: 'app-root',
  templateUrl: './app.component.html',
  styleUrls: ['./app.component.css']
})
export class AppComponent {
  title = 'onchange';

  counter: number = 0;
  incrementCounter() {
    this.counter++;


  }


}

```

child.component.html U

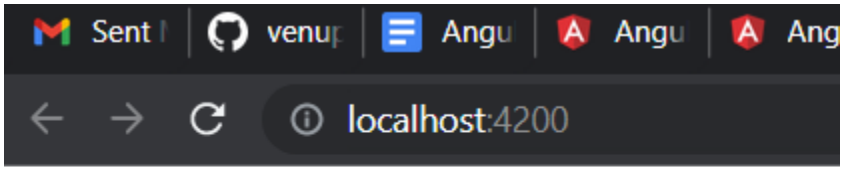
onchange > src > app > child > child.component.html > p

```
1   <p>Counter: {{ counter }}</p>
2
```

child.component.ts U

onchange > src > app > child > child.component.ts > ChildComponent

```
1   import { Component, Input,OnChanges ,OnInit} from '@angular/core';
2
3   @Component({
4     selector: 'app-child',
5     templateUrl: './child.component.html',
6     styleUrls: ['./child.component.css']
7   })
8   export class ChildComponent implements OnChanges {
9     @Input() counter: number=0 ;
10
11    ngOnChanges() {
12      console.log('onchange');
13      console.log('val: ', this.counter);
14    }
15
16
17
18  }
19
```



Counter: 0

Increment Counter

Oninit:

Initialize the directive or component after Angular first displays the data-bound properties and sets the directive or component's input properties. See details in Initializing a component or directive in this document.

Called once, after the first ngOnChanges(). ngOnInit() is still called even when ngOnChanges() is not (which is the case when there are no template-bound inputs).

child.component.html U | child.component.ts U

oninit > src > app > child > child.component.ts > ChildComponent > ngOnInit

```ts
import { Component,Input,OnInit } from '@angular/core';

@Component({
  selector: 'app-child',
  templateUrl: './child.component.html',
  styleUrls: ['./child.component.css']
})
export class ChildComponent implements OnInit{
  @Input() message: string='';

  ngOnInit() {
    console.log('oninit: ', this.message);
  }
}
```

app.component.html M | child.component.html U | child.compo

oninit > src > app > app.component.html > button

```html
<app-child [message]="message"></app-child>
<button (click)="change()">Change Message</button>
```

child.component.html U

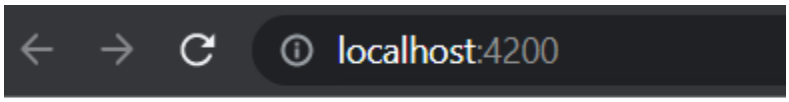
oninit > src > app > child > child.component.html > p

```html
<p>{{ message }}</p>
```

app.component.html M | app.component.ts M | chi

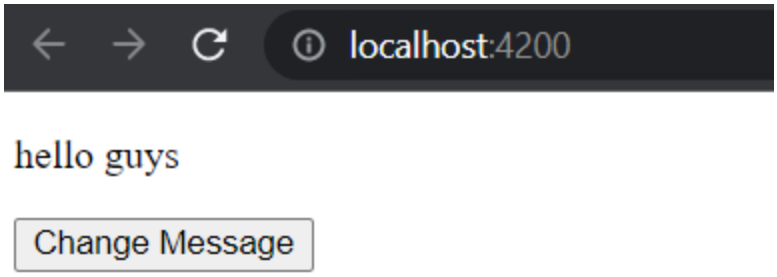
oninit > src > app > app.component.ts > ...

```ts
import { Component } from '@angular/core';

@Component({
  selector: 'app-root',
  templateUrl: './app.component.html',
  styleUrls: ['./app.component.css']
})
export class AppComponent {
  title = 'oninit';
  message = 'this is rushika shreya';

  change() {
    this.message = 'hello guys';
  }
}
```



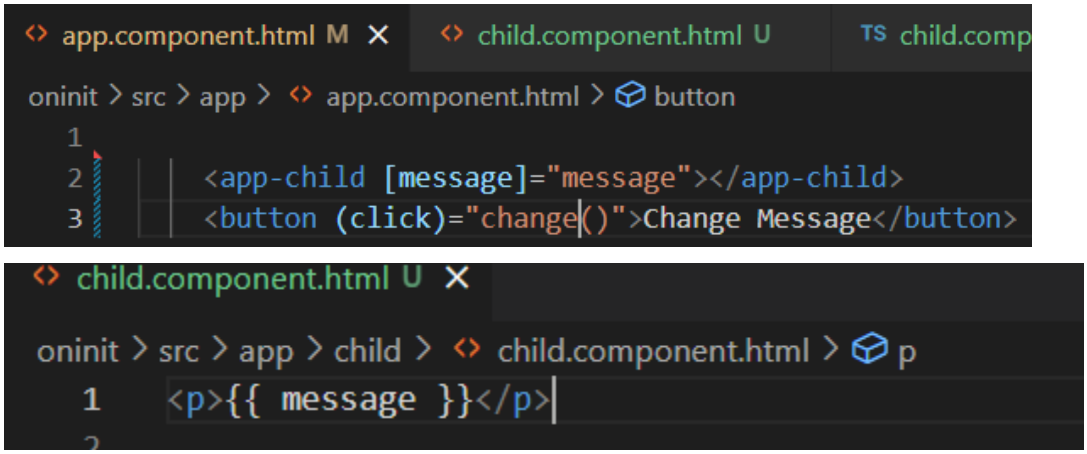
this is rushika shreya

Change Message



ngDoCheck():

Detects and acts upon changes that Angular can't or won't detect on its own. See details and example in Defining custom change detection in this document.Called immediately after ngOnChanges() on every change detection run, and immediately after ngOnInit() on the first run.



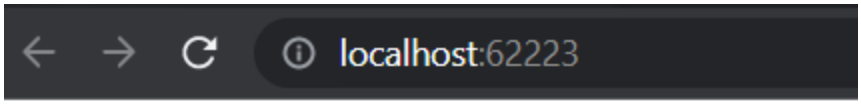
ngdocheck > src > app > child > TS child.component.ts > ChildComponent > ngDoCheck

```ts
import { Component ,Input, DoCheck } from '@angular/core';

@Component({
  selector: 'app-child',
  templateUrl: './child.component.html',
  styleUrls: ['./child.component.css']
})
export class ChildComponent {
  @Input() message: string='';

  ngDoCheck() {
    console.log('do check: ', this.message);
  }
}
```

TS child.component.ts oninit\... U | child.component.html U | TS child.component.ts ngdocheck\... U | TS

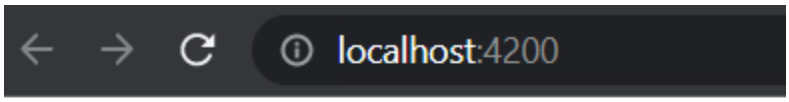
ngdocheck > src > app > TS app.component.ts > AppComponent > change

```ts
import { Component } from '@angular/core';

@Component({
  selector: 'app-root',
  templateUrl: './app.component.html',
  styleUrls: ['./app.component.css']
})
export class AppComponent {
  title = 'ngdocheck';
  message = 'this is rushika shreya';

  change() {
    this.message = 'currently on do check';
  }
}
```



currently on do check

Change Message



this is rushika shreya

Change Message

ngAfterContentInit():

Respond after Angular projects external content into the component's view, or into the view that a directive is in.See details and example in Responding to changes in content in this document.Called *once* after the first ngDoCheck().

ngaftercontentinit > src > app > child > TS child.component.ts > ...

```
import { Component,AfterContentInit, ContentChildren ,QueryList,ElementRef } from '@angular/core';

@Component({
  selector: 'app-child',
  templateUrl: './child.component.html',
  styleUrls: ['./child.component.css']
})
export class ChildComponent implements AfterContentInit{
  @ContentChildren('button') buttonChildren: QueryList<ElementRef>= new QueryList();;

  ngAfterContentInit() {
    console.log('aftercontentinit(): ["Button 1", "Button 2"]');
}
```

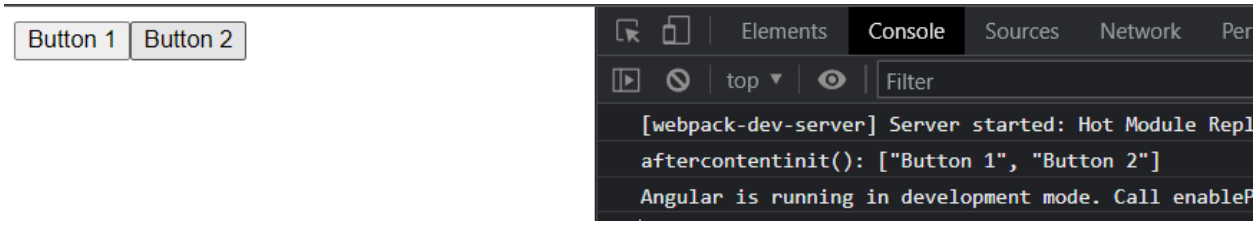
ngaftercontentinit > src > app > app.component.html > ...

```
1
2   <app-child>
3     <button>Button 1</button>
4     <button>Button 2</button>
5   </app-child>
```

TS app.component.ts

ngaftercontentinit > src > app > TS app.component.ts > ...

```
1   import { Component } from '@angular/core';
2
3   @Component({
4     selector: 'app-root',
5     templateUrl: './app.component.html',
6     styleUrls: ['./app.component.css']
7   })
8   export class AppComponent {
9     title = 'ngaftercontentinit';
10  }
11
```



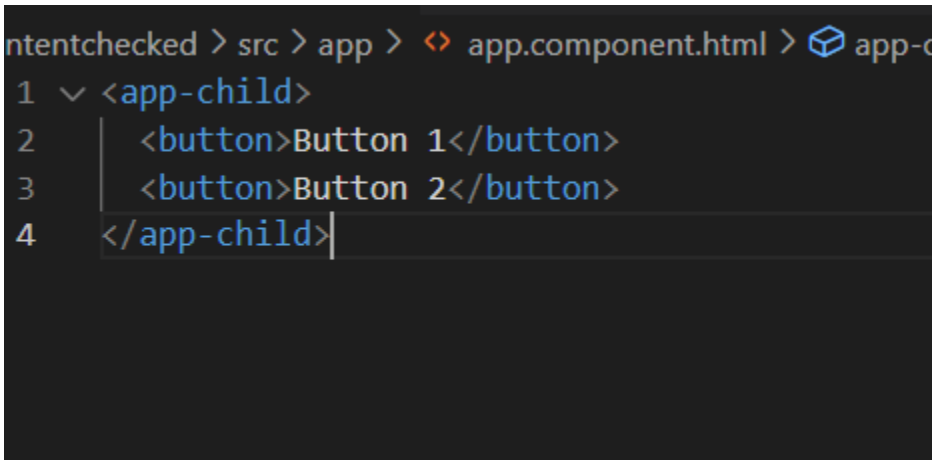
ngaftercontentinit > src > app > child > child.component.html > ng-content

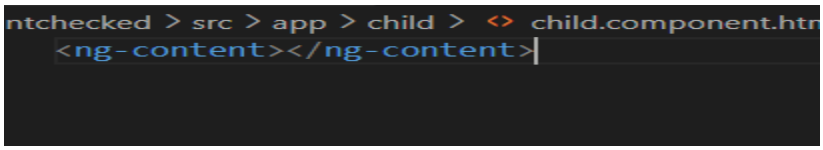
```
1   <ng-content></ng-content>
```

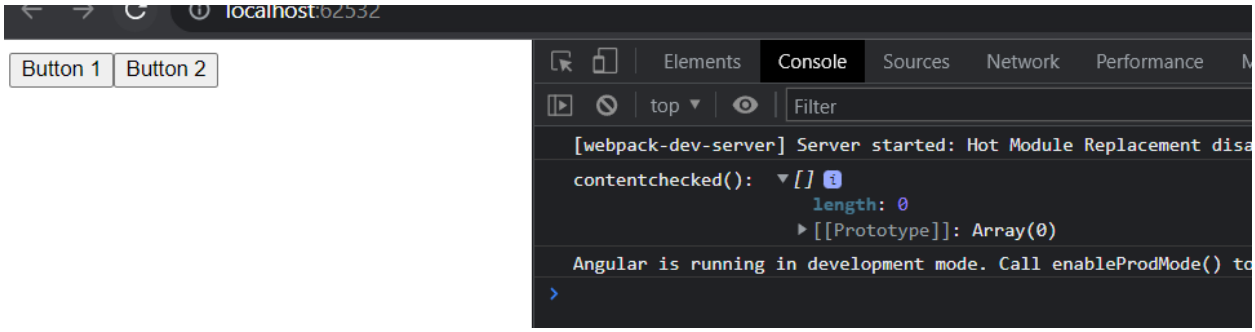
ngAfterContentChecked():

Respond after Angular checks the content projected into the directive or component.See details and example in Responding to projected content changes in this document.

- 







`ngAfterViewInit():`

Respond after Angular initializes the component's views and child views, or the view that contains the directive.See details and example in Responding to view changes in this document.

`ngAfterViewChecked():`

Respond after Angular checks the component's views and child views, or the view that contains the directive

`ngOnDestroy():`

Cleanup just before Angular destroys the directive or component. Unsubscribe Observables and detach event handlers to avoid memory leaks. See details in Cleaning up on instance destruction in this document.

afterviewinit > src > app > child > TS child.component.ts > ChildComponent > message

```ts
import { Component , AfterViewInit, AfterViewChecked, OnDestroy, Input } from '@angular/core';

@Component({
  selector: 'app-child',
  templateUrl: './child.component.html',
  styleUrls: ['./child.component.css']
})
export class ChildComponent implements AfterViewInit, AfterViewChecked, OnDestroy {
  @Input() message: string='';

  ngAfterViewInit() {
    console.log('Child Component ngAfterViewInit called with message: ', this.message);
  }

  ngAfterViewChecked() {
    console.log('Child Component ngAfterViewChecked called with message: ', this.message);
  }

  ngOnDestroy() {
    console.log('Child Component ngOnDestroy called with message: ', this.message);
  }
}
```

afterviewinit > src > app > TS app.component.ts > AppComponent > changeMes

```ts
import { Component } from '@angular/core';

@Component({
  selector: 'app-root',
  templateUrl: './app.component.html',
  styleUrls: ['./app.component.css']
})
export class AppComponent {
  title = 'afterviewinit';
  message = 'Initial Message';

  changeMessage() {
    this.message = 'Message Changed';
  }
}
```

afterviewinit > src > app > child > child.component.html > p

```
1  <p>{{message}}</p>
2
```

afterviewinit > src > app > app.component.html > button

```
1  <app-child [message]="message"></app-child>
2  <button (click)="changeMessage()">Change Message</button>
```