

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.

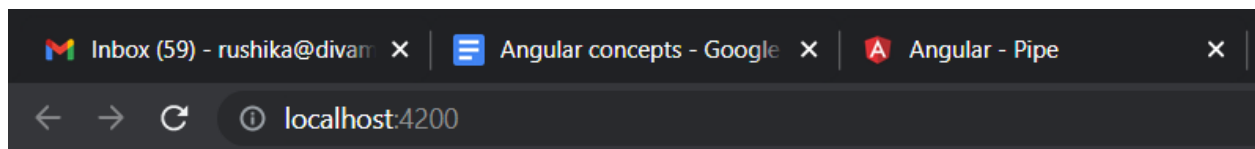
Built-in pipes:

- 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

```
Welcome    app.component.html M    TS pipes.pipe.ts U X    TS app.component.ts

app > src > app > TS pipes.pipe.ts > PipesPipe
1  import { Pipe, PipeTransform } from '@angular/core';
2
3  @Pipe({
4    name: 'reverse', |
5  })
6  export class PipesPipe implements PipeTransform {
7    value:string=''
8    transform(value: any, ...args:any): any{
9      this.value = value.split('').reverse().join('');
10     return this.value;
11   }
12
13 }
14
```

```
Welcome    app.component.html M X    TS pipes.pipe.ts U    TS app.component.ts

app > src > app > app.component.html > h3
1  <h3>
2
3    | rushika to{{ "rushika" | reverse}}
4
5  </h3>
6
7
8  <h3>
9
10   | rushika shreya devarakonda to{{ "rushika shreya devarakonda" | reverse | uppercase}}
11
12 </h3>
```

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 X
onchange > src > app > <> app.component.html > button
1 <app-child [counter]="counter"></app-child>
2 <button (click)="incrementCounter()">Increment Counter</button>
```

```
TS app.component.ts M X
onchange > src > app > TS app.component.ts > AppComponent > incrementCounter
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 = 'onchange';
10
11   counter: number = 0;
12   incrementCounter() {
13     this.counter++;
14   }
15
16 }
17
18
19 }
20
```

<> child.component.html U X

onChange > src > app > child > <> child.component.html > p

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

TS child.component.ts U X

onChange > src > app > child > TS 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
```

Sent | venu | Angu | Angu | Angu

← → ↻ ⓘ localhost:4200

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.

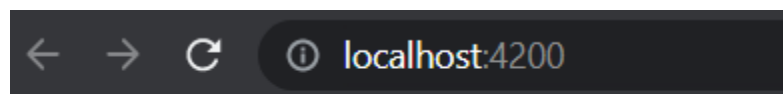
Called once, after the first ngOnChanges(). ngOnInit() is still called even when ngOnChanges() is not (which is the case when there are no template-bound inputs).

```
child.component.html U TS child.component.ts U X
oninit > src > app > child > TS child.component.ts > ChildComponent > ngOnInit
1 import { Component, Input, 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 OnInit{
9   @Input() message: string='';
10
11   ngOnInit() {
12     console.log('oninit: ', this.message);
13   }
14 }
15
```

```
app.component.html M X child.component.html U TS child.comp
oninit > src > app > app.component.html > button
1
2 <app-child [message]="message"></app-child>
3 <button (click)="change()">Change Message</button>
```

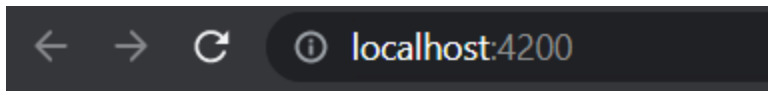
```
child.component.html U X
oninit > src > app > child > child.component.html > p
1 <p>{{ message }}</p>
2
```

```
<> app.component.html M    TS app.component.ts M X    <> chi
oninit > 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 = 'oninit';
10    message = 'this is rushika shreya';
11
12    change() {
13      this.message = 'hello guys';
14    }
15  }
16
```



this is rushika shreya

Change Message



hello guys

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.

```
<> app.component.html M X <> child.component.html U TS child.comp
oninit > src > app > <> app.component.html > button
1
2 | | <app-child [message]="message"></app-child>
3 | | <button (click)="change()">Change Message</button>
```

```
<> child.component.html U X
oninit > src > app > child > <> child.component.html > p
1 <p>{{ message }}</p>|
2
```

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

```
1  import { Component ,Input, DoCheck } 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 {
9    @Input() message: string='';
10
11    ngDoCheck() {
12      console.log('do check: ', this.message);
13    }
14  }
15
```

TS child.component.ts oninit\... U

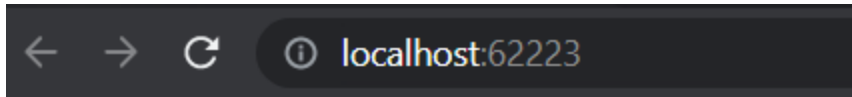
child.component.html U

TS child.component.ts ngdocheck\... U

TS

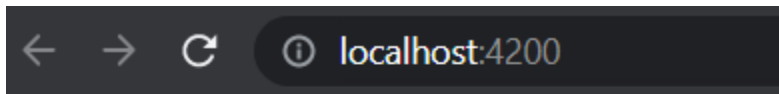
ngdocheck > src > app > TS app.component.ts > AppComponent > change

```
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 = 'ngdocheck';
10   message = 'this is rushika shreya';
11
12   change() {
13     this.message = 'currently on do check';
14   }
15 }
16
```

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 > ...
1  import { Component, AfterContentInit, ContentChildren, QueryList, ElementRef } 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 AfterContentInit {
9    @ContentChildren('button') buttonChildren: QueryList<ElementRef>= new QueryList();;
10
11    ngAfterContentInit() {
12      console.log('aftercontentinit(): ["Button 1", "Button 2"]');
13    }
14  }
```

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 X

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
```

Button 1 Button 2

Elements Console Sources Network Per

top Filter

[webpack-dev-server] Server started: Hot Module Replacement
aftercontentinit(): ["Button 1", "Button 2"]
Angular is running in development mode. Call enableProdMode() to enable production mode.

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

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

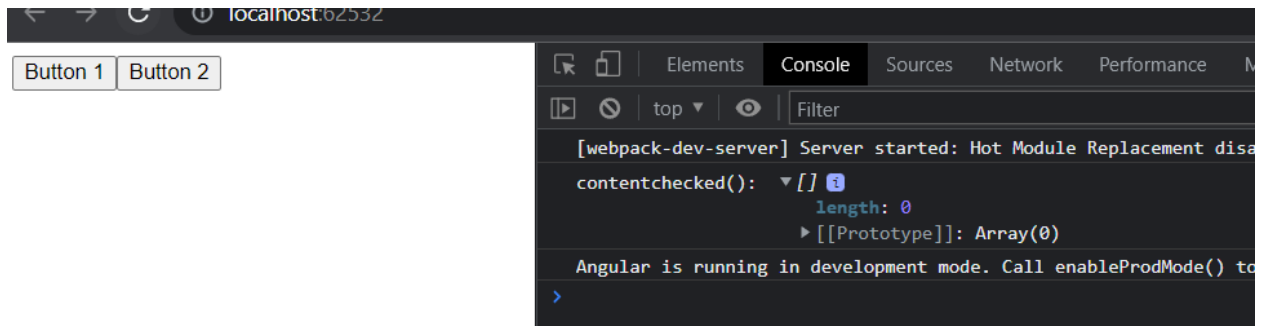
ngAfterContentChecked():

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.

```
contentchecked > src > app > child > TS child.component.ts > ...
1  import { Component, AfterContentChecked, ContentChildren, QueryList, ElementRef } 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 AfterContentChecked {
9    @ContentChildren('button') buttonChildren: QueryList<ElementRef> = new QueryList();
10    private previousButtonChildren: string[] = ["hey", "hi", "hello"];
11
12    ngAfterContentChecked() {
13      const currentButtonChildren = this.buttonChildren.toArray().map(buttonChild => buttonChild.nativeElement.textContent);
14      if (this.previousButtonChildren.toString() !== currentButtonChildren.toString()) {
15        console.log('contentchecked(): ', currentButtonChildren);
16        this.previousButtonChildren = currentButtonChildren;
17      }
18    }
19  }
20
```

```
contentchecked > src > app > <img alt="HTML icon" data-bbox="670 508 690 528"/> app.component.html >
1  <app-child>
2    <button>Button 1</button>
3    <button>Button 2</button>
4  </app-child>
```

```
ntchecked > src > app > child > <> child.component.html
<ng-content></ng-content>
```



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
1  import { Component, AfterViewInit, AfterViewChecked, OnDestroy, Input } 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 AfterViewInit, AfterViewChecked, OnDestroy {
9    @Input() message: string='';
10
11    ngAfterViewInit() {
12      console.log('Child Component ngAfterViewInit called with message: ', this.message);
13    }
14
15    ngAfterViewChecked() {
16      console.log('Child Component ngAfterViewChecked called with message: ', this.message);
17    }
18
19    ngOnDestroy() {
20      console.log('Child Component ngOnDestroy called with message: ', this.message);
21    }
22  }

```

```

app.component.html contentchecked...  app.component.html afterviewinit...
afterviewinit > src > app > TS app.component.ts > AppComponent > changeMes
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 = 'afterviewinit';
10   message = 'Initial Message';
11
12   changeMessage() {
13     this.message = 'Message Changed';
14   }
15 }
16

```

```
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>|
```

localhost:4200

Message Changed

Change Message

Elements Console Sources Network Performance Memory Application Security

top Filter

[webpack-dev-server] Server started: Hot Module Replacement disabled, Live Reloading enabled

Child Component ngAfterViewInit called with message: Initial Message

Child Component ngAfterViewChecked called with message: Initial Message

Angular is running in development mode. Call enableProdMode() to enable production mode.

2 Child Component ngAfterViewChecked called with message: Initial Message

Child Component ngAfterViewChecked called with message: Message Changed