**Chapter: 4. Pipes, Services and Dependency Injection**

**Objective:** Our main objective in this chapter is to learn below topics:

* Introduction to Pipes
* Types of Pipes
  + Pre-Defined pipes in Angular
    - Lowercase
    - uppercase
    - date
    - json
    - precent
    - slice
    - titlecase
    - decimal
    - async etc.
  + Custom Pipes in Angular
* Chaining Pipes by passing multiple options
* Pure and Impure Pipes
* Angular Services Introduction
* Dependency Injection
* Multiple Choice Questions
* FAQ/Lab assignments
* Summery about the chapter.

# Introduction to Pipes

1. Pipes are one of the very important features in Angular. It’s a simple way of transforming the values in an angular template.
2. You will find various built-in pipes in Angular, though you can also keep up with your Pipe.
3. The pipe is packed in a value or values and then it returns a certain value
4. All pipes in Angular are derived from “**PipeTransform**” base.
5. Angular support both built-in pipes and custom pipes as well.
6. The pipe can take in the parameter as well, and we can pass the parameters through the pipe. The parameters are being separated by the pipe.
7. Angular pipes are assigned to your data by using “|” pipe symbol.
8. Angular Pipes are by default “Pure” pipes. They will not change the value; they just defined a format for value.
9. If Pipe can change the state and value then it is “Impure” pipe.

**Syntax:**

{{ property | pipeName:options }}

**Angular Built-In Pipes:** Angular provides following built in pipes like

|  |  |  |
| --- | --- | --- |
| **Pipe** | **Name** | **Description** |
| **UpperCasePipe** | uppercase | It converts all letters into block letters.  Ex:  public product = {  Name: 'Samsung TV',  Price: 45000.50,  Mfd: new Date('2020-03-20')  };  {{product.Name | uppercase}} |
| **LowerCasePipe** | lowercase | It converts all letters into lowercase letters.  Ex:  public product = {  Name: 'Samsung TV',  Price: 45000.50,  Mfd: new Date('2020-03-20')  };  {{product.Name | lowercase}} |
| **TitleCasePipe** | titlecase | It capitalizes the first letter of every word.  Ex:  public product = {  Name: 'Samsung TV',  Price: 45000.50,  Mfd: new Date('2020-03-20')  };  {{product.Name | titlecase}} |
| **DecimalPipe** | number | It is used to display numeric value with thousands separator and fractions.  **It comprises of:**  Minimum-Integer-Digits  Minimum-Fraction-Digits  Maximum-Fraction-Digits  **Syntax:**  {{ data |number }}  {{ data | number: {minIntegerDigits}.{minFractionDigits}-{maxFractionDigits} }}  Ex:  price: 45000.50  {{price}} 🡪 45000.5  {{price|number}} 🡪 45,000.5  {{price|number:’5.4-4’}} 🡪 45,000.5000  {{price|number:’5.2-4’}} 🡪 45,000.50 |
| **CurrencyPipe** | currency | It is used to display numbers in a currency format. It comprises of thousands separator, fractions and a currency symbol.  Syntax:  {{data|currency:’currencyFormat’: ‘digitsInfo’}}  Currency Format: USD, INR, you can use literals.  Digits Info: {minInteger}.{minFraction}-{maxFraction}  Ex:  {{product.Price |currency:'INR'}}  {{product.Price |currency:'&#8377;'}} |
| **DatePipe** | date | It is used for displaying the date and time values in various date and time formats. You can use predefined formats for Date or your can define custom format.  **Pre-Defined Formats:**   * short * medium * long * full * shortDate * mediumDate * longDate * fullDate * shortTime * mediumTime * longTime * fullTime   Syntax:  product.Mfd = new Date(“2020-03-22”);  {{yourDate | date: ‘format’}}  {{product.Mfd | date:'shortDate'}}  **Custom Format:**  MM - 2 Digits Month  MMM - Short Month Name  MMMM - Long Month Name  dd - 2 Digits Date  d - 1 Digit Date  yy - 2 Digits Year  yyyy - 4 Digits Year  Ex:  {{product.Mfd | date:'MMM-dd-yyyy'}} |
| **PercentPipe** | percent | Transforms a number into a percentage string.  Syntax:  {{ value | percent:’digitsInfo’}}  Ex:  public product = {  Name: 'Samsung TV',  Price: 45000.50,  Mfd: new Date('2020-03-20'),  Sales: 0.259  };  {{product.Sales | percent:'2.2-2'}} |
| **SlicePipe** | slice | It creates a new Array or string containing a subset (sliced) of the elements.  It can extract values based on specified index and returns an array.  Syntax:  {{collection | slice:startIndex:endIndex }}  Ex:  public products = ['TV', 'Mobile', 'Shoe', 'Watch'];  <ol>  <li \*ngFor="let item of products | slice:1:3">  {{item}}  </li>  </ol> |
| **JsonPipe** | json | * It is used to convert the data into JSON. * You can access the form data, convert into JSON and send to any API Service.   Syntax:  {{ data | json }}  Ex:  public product = {  Name: 'Samsung TV',  Price: 45000.50,  Mfd: new Date('2020-03-20'),  Sales: 0.259  };  <pre>  {{product | json}}  </pre>  O/P:  {  "Name": "Samsung TV",  "Price": 45000.5,  "Mfd": "2020-03-20T00:00:00.000Z",  "Sales": 0.259  } |
| **KeyValuePipe** | keyvalue | * It is used to transform an object or map into an array of key and value pairs. * Without key and value pipe we have to use iterator over property using “in” operator.   Ex:  for(var property in object)  {  console.log(property + “:” + object[property]);  }   * KeyValue pipe allows to extract the key (property) and value from a map or array.   Syntax:  {{collection | keyvalue}}   * “key” returns the property. * “value” returns the value.   Ex:  **Pipedemo.component.ts**  export class PipedemoComponent {  public products = ['TV', 'Mobile', 'Shoe', 'Watch'];  public data:{[key:number]:string} = {1:'Samsung TV', 2:'Nike Casuals'};  }  **Pipedemo.component.html**  <div class="container-fluid">  <h2>Products Array </h2>  <ol class="list-unstyled">  <li \*ngFor="let item of products | keyvalue">  [{{item.key}}] {{item.value}}  </li>  </ol>  <h2>Products Map</h2>  <ol class="list-unstyled">  <li \*ngFor="let item of data | keyvalue">  {{item.key}} - {{item.value}}  </li>  </ol>  </div> |
| **I18nSelectPipe** | i18Select | * I18 is a community of Angular. * It designed a SelectPipe. * It is a Generic selector that can make decision dynamically according to the state or value, and define the result when the relative condition is matching. * In Early versions we have to depend on lot of iterations and condition.   Syntax:  {{value\_Expression | i18Select: mapping }} |

**Example with Pipes:**

Create a component with **PipesDemo** and write the following code

**Pipes.component.ts:**

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

@Component({

  selector: 'app-root',

  templateUrl: './app.component.html',

  styleUrls: ['./app.component.css']

})

export class AppComponent {

  students: any[] = [

    {ID:'nit101',Name:'Srikanth rallabandi',DOB:'8/19/1990',Gender:'Male',CourseFee:1234.56},

    {ID:'nit102',Name:'Haritha pavulauri',DOB:'7/25/1898',Gender:'Female',CourseFee:1234.56},

    {ID:'nit103',Name:'Naresh it',DOB:'7/24/1990',Gender:'Male',CourseFee:5897.56},

    {ID:'nit104',Name:'Ramu sri',DOB:'2/18/1985',Gender:'Male',CourseFee:1452.56},

    {ID:'nit105',Name:'Sai babu',DOB:'10/8/2002',Gender:'Female',CourseFee:3652.56},

    {ID:'nit106',Name:'Diyansh raj',DOB:'8/22/1989',Gender:'Male',CourseFee:8547.56},

  ];

  today:number = Date.now();

}

**Pipes.component.html:**

<h1>Example with Pipes</h1>

<table border="2">

  <tr>

    <th>Student ID</th>

    <th>Student Name</th>

    <th>DOB</th>

    <th>Gender</th>

    <th>Course Fee</th>

  </tr>

  <tr \*ngFor="let std of students">

    <td>{{std.ID | uppercase}}</td>

    <td>{{std.Name | myTitle: std.Gender | titlecase }}</td>

    <td>{{std.DOB | date:'shortDate' }}</td>

    <td>{{std.Gender | lowercase}}</td>

    <td>{{std.CourseFee | currency:'INR'}}</td>

  </tr>

</table>

<h2>{{today | date}}</h2>

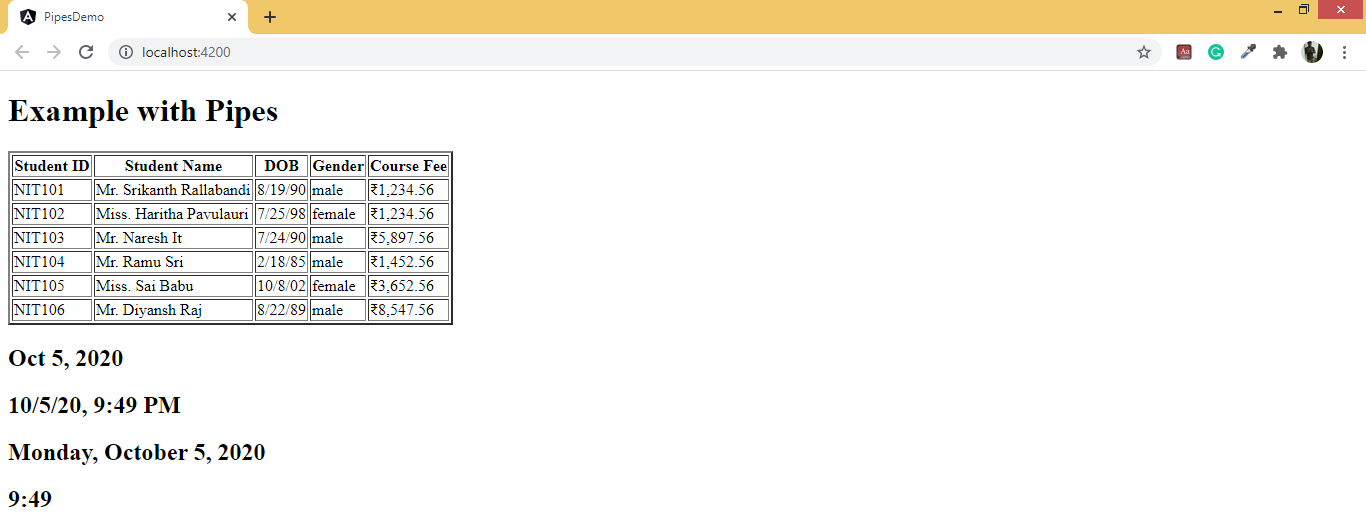
<h2>{{today | date :'short'}}</h2>

<h2>{{today | date :'fullDate'}}</h2>

<h2>{{today | date :'h:mm'}}</h2>

**Output:**

### 



### Custom Pipes in Angular

You can create a custom pipe in Angular. The most general way of defining the custom pipe is as below:

**Syntax:**

import { Pipe, PipeTransform } from '@angular/core';

@Pipe({name: 'nameofPipe'})

export class Pipeclass implements PipeTransform {

   transform(parameters): returntype{

//Implementation

 }

}

And here,

1. 'name' –name of the pipe.
2. Pipeclass– The custom Pipe is assigned class whose name is PipeClass.
3. Parameters – The parameters passed to the Pipe.
4. Returntype– Pipe return type.
5. Transform –the function of working through Pipe.

Now let us make a custom pipe that adds two numbers and then use that pipe in a component class.

**Step 1**: You need to first create a file known as **my.pipe.ts.**, by using Angular CLI command like **“ng g pipe My”**

**Step 2** : Now place the below code in the file **my.pipe.ts.**

import {Pipe, PipeTransform} from '@angular/core';

@Pipe({

    name:'myTitle'

})

export class MyPipe implements PipeTransform{

    transform(name:string,gender:string):string{

        if(gender.toLowerCase()=='male'){

            return "Mr. "+name;

        }

        else{

            return "Miss. "+name;

        }

    }

}

**Step 3**: Now place the pipe in the template like

**<h2>Welcome {{Naresh | myTitle}}</h2>**

# Chaining Pipes by passing multiple options

We can use numerous pipes together through chaining. And this helps in Scenarios where we can link more than one pipe which requires to be linked. The final output is going to be transformed through all the pipes in the chain.

The chain is triggered and applied to the element of the pipes in a series. For example, let's think of the chain pipe given below:

**Syntax:** {{today | date: “full" | lowercase}}

Here we have two chain pipes. The first one is the DatePipe which is applied to the today variable, and then the lowercase pipe is applied to the date. Let's write the whole code for the chainPipeComponent:

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

@Component({

 template: `

<h2>Chain Pipes</h2>

<p>Month is {{today | date:"dd-mm-yyyy" | uppercase}}

 `,

})

export class DateChainPipeComponent {

 today = new Date();

}

We can make use of slice for only printing the first four characters of the month as well. And that will be another layer in chain pipe.

**Angular Services Introduction**

1. Angular Services are a pre-defined business logic which can be reused in the application by injecting into any component.
2. Service is a collection of Factories.
3. Factory is a collection of related type of functions.
4. You can inject a factory directly into any component in order to use the functionality.
5. Factory uses **“Single Call Mechanism”.** Every time when you want to use a function you need an object to create. [Disconnected and Discrete]
6. Service uses a **“Single Ton Mechanism”.** Object is created only for the first request and the same object is used across any number of requests. [Connected and Continuous]
7. Angular Service uses **“Dependency Injection”** to inject a service into any component constructor. Instead of creating the service object using **“new”** operator.
8. Dependency Injection is an **“Application Design Pattern”.**
9. Angular has its own **DI framework**.
10. Angular uses DI framework to increase the application **efficiency and modularity.**
11. Dependencies are services or objects that a class needs to perform its function.
12. DI is a coding pattern in which a class asks for dependencies from external sources rather than creating them itself.
13. It allows to share information between classes.
14. In Angular, the DI framework provides declared dependencies to a class when that class is instantiated.
15. The DI framework lets you to supply data to a component from an injectable service class.
16. Technically service in Angular is a class, which comprises of a set of service methods.

**The syntax is as below for services:**

import { Injectable } from ‘@angular/core’

@Injectable()

export class DemoService

{

}

And with the help of the above code, you can always create a service. You can define variables as properties, add getters and setters, add any number of methods, add an init method, and so on.

**What is Injector?**

1. It is an object in Angular **“dependency-injection”** system.
2. It can find a names dependency in its cache or create a dependency using a configured provider.
3. **Injectors** are created for “**NgModules**” automatically as part of the “**bootstrap**” process.
4. Injector provides a singleton instance of a dependency and can inject the same instance into multiple components.
5. The injector provides a hierarchy so that the content can be uses for the parent and child components.
6. We can configure injector with different providers that can provide different implementations of the same dependency.

**What is Provider?**

1. Provider is an object that implements one of the “Provider” interfaces.
2. Provider defines how to obtain an injectable dependency associated with a DI token.
3. Injector uses a provider to create a new instance of dependency for a class.
4. Angular registers its own providers with every injector for services.
5. Angular provides different type of providers
   1. ValueProvider
   2. ClassProvider
   3. TypeProvider
   4. ConstructorProvider
   5. FactoryProvider etc..

**Syntax:**

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

@Injectable(

{

providedIn: 'root'

}

)

export class SampleService {

constructor() { }

}

**Injecting Services:**

1. You have to inject the service into any component.
2. We have to make sure that service is injected into component rather that creating a new instance.
3. You can tell angular to inject dependency in a components constructor by specifying a constructor parameter with the dependency type (service type).
4. Every parameter defined in constructor is accessible only within the constructor.
5. You can define an “Access Modifier” to specify the scope of parameter as “private, public or protected”.

**Syntax:**

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

import { SampleService } from '../sample.service';

@Component({

selector: 'app-sampleconsume',

templateUrl: './sampleconsume.component.html',

styleUrls: ['./sampleconsume.component.css']

})

export class SampleconsumeComponent implements OnInit {

constructor(private sampleservice: SampleService) {

}

ngOnInit(): void {

this.sampleservice.GetData();

}

}

**Example with Service:**

**Step-1:** Create a new Angular Application by using **“ng new ServiceDemo”**

**Step-2:** Create a new Service by using CLI command like **“ng g s Captcha –skipTests”** and write the following code

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

@Injectable({

    providedIn: 'root'

})

export class CaptchaService

{

    public GenerateCode() {

        let a = Math.random() \* 10;

        let b = Math.random() \* 10;

        let c = Math.random() \* 10;

        let d = Math.random() \* 10;

        let e = Math.random() \* 10;

        let f = Math.random() \* 10;

        let code = `${Math.round(a)} ${Math.round(b)} ${Math.round(c)} ${Math.round(d)} ${Math.round(e)} ${Math.round(f)}`;

        return code;

    }

}

**Step-3:** Create a **LoginComponent** and write the following code

**Login.component.ts:**

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

import { CaptchaService } from '../captcha.service';

@Component({

    selector: 'app-login',

    templateUrl: 'login.component.html',

    styleUrls: ['login.component.css']

})

export class LoginComponent {

    constructor(private captcha: CaptchaService){}

    public code = this.captcha.GenerateCode();

    public refreshClick() {

        this.code = this.captcha.GenerateCode();

    }

}

**Login.component.html:**

<div class="container-fluid">

    <div class="form-group">

    <label>User Name</label>

    <div>

        <input type="text" class="form-control">

    </div>

   </div>

   <div class="form-group">

    <label>Password</label>

    <div>

        <input type="password" class="form-control">

    </div>

   </div>

   <div class="form-group">

      <label>Verify Code</label>

      <div>

          {{code}} <button (click)="refreshClick()" class="btn"><span class="fa fa-sync"></span></button>

      </div>

      <div>

          <input type="text" class="form-control">

      </div>

   </div>

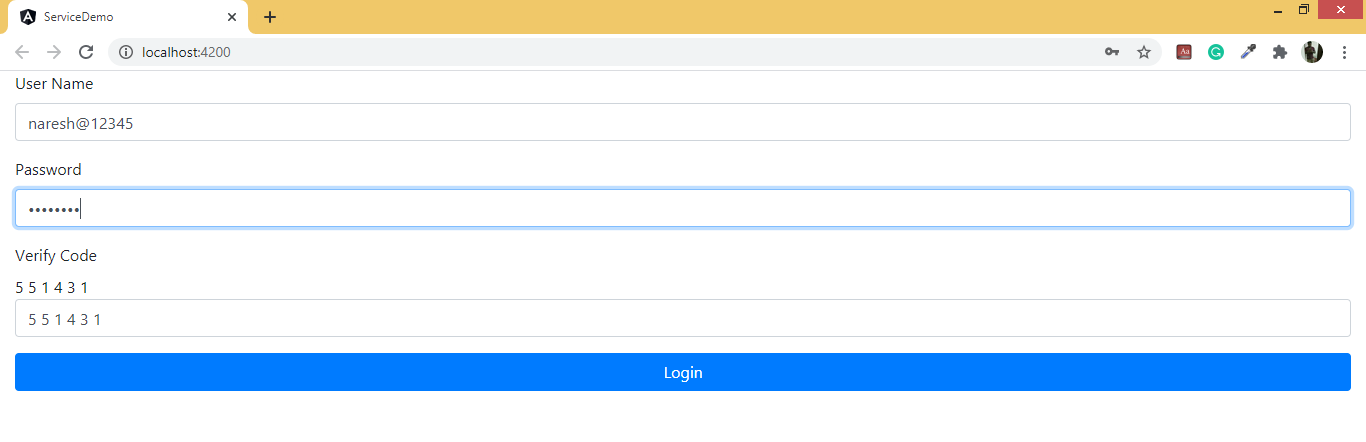
   <div class="form-group">

      <button class="btn btn-primary btn-block">Login</button>

   </div>

  </div>

**Step-4:** Register the **Service** and **Component** in **app.module.ts** and use the selector in **index.html** will get following **output**



# Multiple Choice Questions:

**1. Pipes are simple way of transforming the values in Angular. Is this true?**

a) Yes b) No

c) It depends upon the condition. d) Yes, it transforms the value into desired.

**2. Is pipes a way of customizing the output in Angular?**

a) Yes b) Yes, but not always

c) No d) Yes, it customizes the values.

**3. Does pipe decorate the output?**

a) Yes partially. b) Yes, to a very deep extent.

c) No d) Only filter pipe does that

**4. Services ensure:**

a) Code reusability b) Code simplification

c) Well-structured code d) Better cohesion and coupling

e) All of these

**5. Angular with typescript supports OOPS**

a) Yes fully b) Yes partially

c) No d) None of these

**6. Chaining of pipes is possible. Is this true?**

a) Yes, by applying multiple pipes

b) Yes, by applying multiple interconnected pipes.

c) Its possible only in Date pipe

d) Its possible only in currency pipes

**7. What is dependency injection?**

a) Dependency is an important application design pattern.

b) Through it, class asks for its dependencies from external sources.

c) It’s a coding pattern

d) All of these

**8. How can you inject an interface?**

a) Using injection token b) Using subject service pattern

c) Using pipes d) Using class

**9. Why dependency injection is important in Angular?**

a) For compiling. b) For lazy loading

c) For AOT d) All of these

**10. Which is more complex parent child communication or control parent communication?**

a) parent child communication b) control parent communication

c) Both

d) First is done with the help of @input and @Output or @ViewChild or @ViewChildren and second one is done through control value accessor interface or validator interface.

**11. Date pipes accept parameters.**

a) yes b) no c) yes, for displaying dates in various formats.

d) No, it always output the date in only one format.

**12. Async pipes can be used with all pipes.**

a) Yes b) No only with date pipes

c) No only with currency pipes d) Yes, except json pipe

# 

# FAQs / LAB Assignments:

1. Explain briefly Parameterized Pipes.

2. How can you chain multiple pipes?

3. What do you mean when you say you will chain pipes in Angular?

4. How can you create the custom pipes in Angular?

5. How can you create filter pipes in Angular?

6. How can you make custom filters in Angular?

7. How can you use filters with templates?

8. How can you pass arguments to filters in Angular?

9. How can you perform filtering in controllers using $filter();

10. How can you create custom pipes in Angular?

11. How can you pass arguments to pipes in Angular?

12. What are pure and impure pipes? Explain them briefly.

13. What is an async pipe?

14. What do you mean by services? Explain briefly various types of services.

15. What is dependency injection. Explain in brief.

16. How can you create data services in Angular?>

17. What is hierarchical services.

18. How can you use services for cross component communication?

19. What is a subject service pattern?

20. What is an injection token? Explain briefly.

# 

# Summary:

We studied following in this chapter:

1. We discussed Pipes in detail, and various types of built in pipes especially.
2. We discussed how to make custom pipes.
3. We also discussed how to make filter pipes.
4. We discussed how chaining pipes by using various parameters.
5. We discussed Pure and Impure Pipes.
6. We discussed services and types of services.
7. We discussed dependency injection.
8. Regarding nested components as well as nested controls, there are various use cases.
9. You must be now knowing what is event bubbling. Suppose the events are nested, they are processed hierarchically.
10. Suppose some service fetches the data from certain API on certain servers, and it renders it for showing in the view that same data. You can generate data service with the help of the below cli command.
11. You will note that the appmodule provider is never reached as the component itself has the provider. However, we can override the provider, and there are four ways of doing that.
    * + 1. Aliasing
        2. Class
        3. Factory
        4. Value
12. If a function does not possess an internal state it is a pure function. This means each of the operations which are performed is not being hindered by the state and given the same set of input options and also gives rise to the same deterministic output.
13. We discussed subject service injection.