Angular Pipes

- Pipe is used to transform data.
- Data comes to your Angular application from various sources.
- The data type of provider and the TypeScript types will not match.
- The data is not displayed in the same format how we are expecting.
- Pipe can transform the data and display in desired format.
- Angular pipes are used for formatting and filtering the data.
- Angular allows to create custom pipes and also provides pre-defined pipes.
- All Angular pipes are derived from "**PipeTransform**" base.
- Every pipe implements a functionality by using "transform()" method.
- Pipe related meta data is defined by using "@Pipe()" marker.

Syntax:

```
import { PipeTransform } from
‘@angular/core’;
@Pipe({
        name: ‘uppercase’
 })
export class UpperCasePipe implement
PipeTransform
{
         transform(value) {
                // transform your value
                // return the transformed value
                return value;
        }
}
{{ product.Name | uppercase }}
```

- Angular provides built-in pipes
    - AsyncPipe [Observable, Subscribers – RxJS – HttpServices]

    - **CurrencyPipe**
    - **DatePipe**
    - **DecimalPipe**
    - **I18PluralPipe**
    - **I18SelectPipe**
    - **JsonPipe**
    - **KeyValuePipe**
    - **LowerCasePipe**
    - **UpperCasePipe**
    - **TitleCasePipe**
    - **PercentPipe**
    - **SlicePipe**
- Angular Pipes are by default “**Pure**” pipes.
- They will not change the value; they just define a format for value.
- If a Pipe can change the state and value then it is “**Impure**” pipe.
- Pipe can accept arguments and can modify the functionality of pipe. [often referred as Parameterized pipes]

Syntax:

{{ data | pipe:option1:option2 | pipe }}

Pipe	Name	Description

UpperCasePipe	upper case	It converts all letters into block letters.
		EX:
		product = {
		Name: 'JBL Speaker',
		Price: 4500.50,
		Mfd: new Date('2020/02/10')
		};
		{{product.Name | uppercase}}

LowerCasePipe	lower case	It converts all letters into lowercase letter. Ex: product = { Name: 'JBL Speaker', Price: 4500.50, Mfd: new Date('2020/02/10')

		}; {{product.Name | lowercase}}

TitleCase Pipe	titlecase	It converts every word first char to uppercase.
		Ex:
		product = {
		Name: 'JBL Speaker',
		Price: 4500.50,
		Mfd: new Date('2020/02/10')
		};
		{{product.Name | titlecase}}

DecimalPipe	number	It is used to display numeric value with thousands separator and fractions. It comprises of following parameters: Minimum-Integer-Digits Minimum-Fraction-

		Digits
		Maximum-Fraction-Digits
		
		Syntax:
		{{data | number:
		{minIntegerDigits}:{minFractionDigits}-
		{maxFractionDigits} }}
		
		Ex:
		product = {
		Name: 'JBL Speaker',
		Price: 4500.50,
		Mfd: new Date('2020/02/10')
		};
		
		{{product.Price | number:’4.2-4’}}

Currency Pipe	currency	It is similar to decimal pipe but have a currency symbol to

		display.
		
		Syntax:
		{{data | currency: 'CurrencyFormat': 'digitsInfo'}}
		
		Currency Format you can define "USD, INR, etc."
		You can also define literals "₹"
		
		Ex:
		product = {
		Name: 'JBL Speaker',
		Price: 4500.50,
		Mfd: new Date('2020/02/10')
		};
		{{ product.Price | currency: 'INR' }}
		{{ product.Price |

		currency: ‘₹’ }}
DatePipe	date	It is used for displaying date and time value in various date and time formats. You can use predefined date formats: - short - medium - long - full - shortDate - mediumDate - longDate - fullDate - shortTime - mediumTime - longTime - fullTime Ex: product = {

		Name: 'JBL Speaker', Price: 4500.50, Mfd: new Date('2020/02/10') }; {{product.Mfd | date: 'shortDate'}} **You can also define custom format for date.** 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 Syntax: {{ data | date: 'MM-dd-

		yyyy’}} {{product.Mfd | date: 'MMMM-dd-yyyy'}}

PercentPipe	percent	Transforms a number into percentage string.
		
		Syntax:
		{{data | percent: ‘digitsInfo’ }}
		
		Ex:
		product = {
		Name: 'JBL Speaker',
		Price: 4500.50,
		Mfd: new Date('2020/02/10'),
		Sales: 0.259
		};
		{{product.Sales | percent:'2.2-2'}}

SlicePipe	slice	It creates a new array or string containing subset of the elements.

		It can extract values based on specified index and return an array. {{ collection | slice:startIndex:endIndex }} Ex: products = ['TV', 'Mobile', 'Speaker', 'Shoe', 'Shirt']; <li *ngFor="let item of products | slice:1:3"> {{item}}
JsonPipe	json	It converts the data into JSON format, so that can transport to server-side services.

		Ex:
		product = {
		Name: 'JBL Speaker',
		Price: 4500.50,
		Mfd: new Date('2020/02/10'),
		Sales: 0.259
		};
		
		<pre>
		{{product | json}}
		</pre>

KeyValue Pipe	keyvalue	It allows to read the properties and values from a collection. “Key” is used to access the keys or properties “Value” is used to access the values. You can configure on a collection like “Array or

		Map”.
		
		Ex:
		data:
		{[key:number]:string} = {
		1001: 'Samsung TV',
		1002: 'Nike Casuals'
		};
		products = ['TV', 'Mobile', 'Speaker', 'Shoe', 'Shirt'];
		
		 Array
		
		<li *ngFor="let item of products | **keyvalue**">
		[{{item.key}}] {{item.value}}

		  Map <li *ngFor="let item of data | **keyvalue**"> {{item.key}} - {{item.value}}
I18nSelectPipe	i18select	- i18 is community of Angular. - It designed SelectPipe. - It is a generic selector that can make decision dynamically according to the state or values and define result when the relative

		condition is matching. - In early version we have to depend on lot of iterations and conditions. Syntax: {{value_expression | i18select:mapping}} - It is an impure pipe. Ex: export class PipedemoComponent{ products = [{Name: 'Samsung TV', City: 'Delhi'}, {Name: 'Nike Casuals', City: 'Hyderabad'}, {Name: 'Mobile', City: 'Delhi'},

		{Name: 'Shirt', City: 'Goa'}
		];
		statusMessage = {
		'Hyderabad': 'Delivery in 2 Days',
		'Delhi': 'Delivery on same Day',
		'Mumbai': 'Not Deiverable',
		'other': 'Unknow - We Will Update'
		};
		}
		
		 Products Status
		<table class="table table-hover">
		<thead>
		<tr>

		<th>Name</th> <th>City</th> <th>Delivery Status</th> </tr> </thead> <tbody> <tr *ngFor="let item of products"> <td>{{item.Name}}</td> <td>{{item.City}}</td> <td>{{item.City | i18nSelect:statusMessage}}</td> </tr> </tbody> </table>
I18Plural Pipe	i18plural	- As per coding standards we use plural name of multiple items or collection and

		singular name for one object. Syntax: product = {}; products = []; - Plural Pipe can identify whether the object comprises of single or multiple value and define a plural name dynamically. - It can get collection count and display messages according to count. - It uses a map to verify the values. Syntax: {{collection.length | i18plural:keyValueCollection}}

Ex:

- **Go to “app.module.ts” and import following modules**

```
import { MatIconModule } from
'@angular/material/icon';
import { MatBadgeModule } from
'@angular/material/badge';

imports: [
MatIconModule,
   MatBadgeModule
 ],
```

- **Pluraldemo.component.ts**

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

@Component({
 selector: 'app-pluraldemo',
 templateUrl:
'./pluraldemo.component.html',
 styleUrls: ['./pluraldemo.component.css']
```

```
})
export class PluraldemoComponent{
  notifications = [];
  notificationsMap: {[key:string]:string} = {
    '=0': 'No Missed Calls', '=1': 'One Missed
Call', 'other': '# Missed Calls'
  };
  showCalls = false;
  ManagerClick(){
    this.notifications.push('Call From
Manager');
  }
  AdminClick(){
    this.notifications.push('Call from
Admin');
  }
  GetMissedCalls() {
    this.showCalls = true;
  }
}
```

- **Pluraldemo.component.html**

```
<div class="container-fluid">
```

```
  <h2>Plural Demo</h2>
  <div class="form-group">
    <button (click)="ManagerClick()">Call
From Manager</button>
    <button (click)="AdminClick()">Call
From Admin</button>
  </div>
      <mat-icon
matBadge="{{notifications.length}}">phon
e</mat-icon>
  <span style="cursor: grab;"
(click)="GetMissedCalls()">
    {{notifications.length |
i18nPlural:notificationsMap}}
  </span>
  <div *ngIf="showCalls" class="form-
group">
    <h3>Missed Calls</h3>
    <ul>
      <li *ngFor="let item of notifications">
        {{item}}
      </li>
```

```
    </ul>
  </div>
</div>
```

Custom Pipe

- You can create your own pipe that can transform data.
- To create a pipe, you need create a class that implement “PipeTranform”
- You have to configure the functionality by using “transform()”

Ex:

- Add a new folder “CustomPipes”
- Add a new file **Sentencecase.pipe.ts**

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

@Pipe(
```

```
  {name: 'sentencecase'}
)
export class SentenceCasePipe
implements PipeTransform {
  transform(str){
    let firstChar = str.charAt(0);
    let restChars = str.substring(1);
    let sentence = firstChar.toUpperCase()
+ restChars.toLowerCase();
    return sentence;
  }
}
```

- **Go to “app.module.ts” and register the pipe in declarations**

```
declarations : [
      SentenceCasePipe
]
```

- Apply for your content

```
msg  = “wELCome TO AGulaR”;

{{ msg | sentencecase }}
```

Try:

- **Create a pipe for sorting and printing the list of values from array.**
  Somecollection = [ ]

  <li *ngFor=”let item of somecollection | yourPipe”>
  sort()
  reverse()

Angular Services