Pipes are used to format the data before displaying it to the user. A pipe takes data as input and transforms it into the desired output.

**Syntax**:

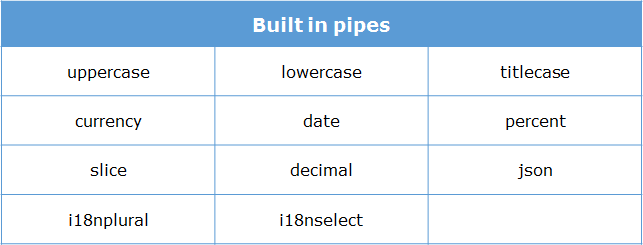
1. {{ expression | pipe }}

**Example**:

1. {{ "Technology" | uppercase }}

This will display TECHNOLOGY

Following table displays the  built-in pipes available:



Now let us learn about some important built-in pipes.

**uppercase**

This pipe converts the template expression into uppercase.

**Syntax**:

1. {{ expression | uppercase }}

**Example**:

1. {{ "Laptop" | uppercase }}

**Output**: LAPTOP

**lowercase**

This pipe converts the template expression into lowercase.

**Syntax**:

1. {{ expression | lowercase }}

**Example**:

1. {{ "LAPTOP" | lowercase }}

**Output**: laptop

**titlecase**

This pipe converts the first alphabet in each word of the given expression into an uppercase alphabet.

**Syntax**:

1. {{ expression | titlecase }}

**Example**:

1. {{ "product details" | titlecase }}

**Output**: Product Details

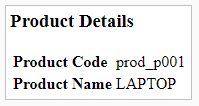
Let us understand the implementation of these pipes with a demo.

Highlights:

* Exploring built-in pipes
* Applying the pipes

**Demo Steps:**

**Problem Statement:**Displaying the product code in lowercase and product name in uppercase using built-in pipes. Output is as shown below:



1. Write the below code in **app.component.ts**
2. import { Component } from '@angular/core';
3. @Component({
4. selector: 'app-root',
5. templateUrl: './app.component.html',
6. styleUrls: ['./app.component.css']
7. })
8. export class AppComponent {
9. title: string = "product details";
10. productCode: string = "PROD\_P001";
11. productName: string = "Laptop";
12. }

1. Write the below code in **app.component.html**
2. <h3> {{ title | titlecase}} </h3>
3. <table style="text-align:left">
4. <tr>
5. <th> Product Code </th>
6. <td> {{ productCode | lowercase }} </td>
7. </tr>
8. <tr>
9. <th> Product Name </th>
10. <td> {{ productName | uppercase }} </td>
11. </tr>
12. </table>

1. Save the files and check the output in the browser

Let us continue our discussion on built-in pipes by taking a look at the built-in pipes which have parameters.

A pipe can also have optional parameters to change the output. To pass parameters, after a pipe name add a colon( : ) followed by the parameter value.

**Syntax**:

1. {{ value | pipename [: parametervalue] }}

A pipe can also have multiple parameters as shown below:

1. {{ value | pipename [: parametervalue1 [: parametervalue2] ] }}

**Note**: In the above syntax [ ] brackets are optional and can be removed.

# 1. currency

This pipe displays the currency symbol before the expression. By default, it displays currency symbol $

**Syntax**:

1. {{ value | currency [ : currencyCode [ : display [ : digitsInfo [ : locale ] ] ] ] }}

* **currencyCode** is of string type. It takes the code to display such as INR for the rupee, EUR for the euro, etc.
* **display**is the format for the currency indicator. It can take the following values:

'code' | 'symbol' | string | boolean

* 'code' - Shows the currency code (such as USD).
* 'symbol' - Shows the currency symbol (such as $)
* string - Use the given string value instead of a code or a symbol.
* boolean - (marked deprecated in v5): true for symbol and false for code.

**Note**:  Default is 'symbol'

* **digitInfo** is a string in the following format

{minIntegerDigits}.{minFractionDigits} - {maxFractionDigits}

* minIntegerDigits is the minimum integer digits to display. Default value is 1
* minFractionDigits is the minimum number of digits to display after fraction. Default value is 0
* maxFractionDigits is the maximum number of digits to display after fraction. Default value is 3

* **locale**is a string type param. A locale code for the locale format rules to use. When not supplied, uses the value of Locale\_ID which is *en-US* by default. It is optional and the default value is undefined.

**Examples:**

1. {{ 66600 | currency }} <!-- will display $66,600.00 --> <br>
2. {{ 66600 | currency : 'CAD' }} <!-- will display CAD$66,600.00 --> <br>
3. {{ 66600 | currency :'INR' }} <!-- will display ?66,600.00 --> <br>
4. {{ 66600 | currency :'INR' :'code' }} <!-- will display INR66,600.00 --> <br>
5. {{ 66600 | currency :'INR' :'symbol' }} <!-- will display ?66,600.00 --> <br>
6. {{ 66600 | currency :'INR' :'myOwnString' }} <!-- will display myOwnString66,600.00 --> <br>
7. {{ 66600 | currency :'INR' :'code' :'6.2-3'}} <!-- will display INR066,600.00 --> <br>
8. {{ 66600 | currency :'INR' :'code' :'1.3'}} <!-- will display INR66,600.000 --> <br>

# 2. percent

This pipe can be used to display the number as a percentage.

**Syntax**:

1. {{ expression | percent [:digitInfo [:locale] ] }}

The expression should be numeric and the value for **digitInfo**and **locale**is similar to currency pipe.

**Examples**:

1. {{ 0.1 | percent }} will display 10%
2. {{ 0.1 | percent:'2.2-3' }} will display 10.00%

# 3. number

This pipe can be used to format a number.

**Syntax**:

1. {{ expression | number [:digitInfo [:locale]] }}

The expression should be numeric and the value for **digitInfo**and **locale**is similar to currency pipe.

**Examples**:

1. {{ 25000 | number }} will display 25,000
2. {{ 25000 | number:'.3-5' }} will display 25,000.000

Let us take a look at some more built-in pipes

# 1. date

This pipe can be used to display the date in the required format.

**Syntax**:

1. {{ expression | date [:format [:timezone [:locale] ]] }}

The**expression**is a date or number in milliseconds

The **format**indicates in which form date/time should be displayed. Following are the pre-defined options for it.

* 'medium' :equivalent to 'MMM d, y, h:mm:ss a' (e.g. Jan 31, 2018, 11:05:04 AM)
* 'short': equivalent to 'M/d/yy, h:mm a' (e.g. 1/31/2018, 11:05 AM)
* 'long': equivalent to 'MMMM d, y, h:mm:ss a z' (e.g. January 31, 2018 at 11:05:04 AM GMT+5)
* 'full': equivalent to 'EEEE, MMMM d, y, h:mm:ss a zzzz' (e.g. Wednesday, January 31, 2018 at 11:05:04 AM GMT+05:30)
* 'fullDate': equivalent to 'EEEE, MMMM d, y' (e.g. Wednesday, January 31, 2018)
* 'longDate' : equivalent to 'MMMM d, y' (e.g. January 31, 2018)
* 'mediumDate' : equivalent to 'MMM d, y' (e.g. Jan 31, 2018)
* 'shortDate' : equivalent to 'M/d/yy' (e.g. 1/31/18)
* 'mediumTime' : equivalent to 'h:mm:ss a' (e.g. 11:05:04 AM)
* 'shortTime' :  equivalent to 'h:mm a' (e.g. 11:05 AM)
* 'longTime': equivalent to 'h:mm a' (e.g. 11:05:04 AM GMT+5)
* 'fullTime': equivalent to 'h:mm:ss a zzzz' (e.g. 11:05:04 AM GMT+05:30)

**Timezone** to be used for formatting. For example, ’+0430’ (4 hours, 30 minutes east of the Greenwich meridian) If not specified, the local system timezone of the end-users browser will be used.

**Examples**:

1. {{ "6/2/2017" | date }} will display Jun 2, 2017
2. {{ "6/2/2017, 11:30:45 AM" | date:'medium' }} will display Jun 2, 2017, 11:30:45 AM
3. {{ "6/2/2017, 11:30:45 AM" | date:'mmss' }} will display 3045{{ 90000000 | date }} will display Jan 2, 1970 – date pipe will start from Jan 1, 1970 and based on the given number of milliseconds, it displays the date

# 2. slice

This pipe can be used to extract a subset of elements or characters from an array or string respectively.

**Syntax**:

1. {{ expression | slice:start:end }}

The**expression**can be an array or string

**start**represents the starting position in an array or string to extract items. It can be

* A positive integer which will extract from the given position till the end
* A negative integer which will extract the given number of items from the end

**end** represents the ending position in an array or string for extracting items. It can be

* A positive number which returns all items before end index
* A negative number which returns all items before end index from the end of the array or string

**Examples**:

1. {{ ['a','b','c','d']| slice:2}} will display c,d
2. {{ ['a','b','c','d']| slice:1:3}} will display b,c
3. {{ 'Laptop Charger'| slice:3:6}} will display top
4. {{ 'Laptop Charger'| slice:-4}} will display rger
5. {{ 'Laptop Charger'| slice:-4:-2}} will display rg

Next, we will learn the use of the built-in JSON pipe.

# json

This pipe can be used to displays the given expression in the form of a JSON string. It is mostly used for debugging.

**Syntax**:

1. {{ expression | json }}

**Example**:

1. {{ {'productId':1234, 'productName':'Samsung Mobile'} | json }} will display {"productId":1234, "productName":"Samsung Mobile"}

# Chaining of pipes

Multiple pipes can be chained together to get a more **desirable** output. The final output will be displayed with all the pipes applied.

**Example:**

1. <p> Month is{{ "2018-11-11" | date | uppercase | slice:0:4 }} </p>

**Output:** Month is NOV

**Note**: Pipes can be chained only when they are compatible with each other. For example, we can't chain a json pipe with a date pipe.

Now let us learn how JSON pipe can be used with the help of a demo.

Highlights:

* Understanding JSON pipe
* Applying JSON pipe

**Demo Steps:**

**Problem Statement:** Apply JSON pipe to product object to display the following output:



1. Write the below code in **app.component.ts**
2. import { Component } from '@angular/core';
3. @Component({
4. selector: 'app-root',
5. templateUrl: './app.component.html',
6. styleUrls: ['./app.component.css']
7. })
8. export class AppComponent {
9. title: string = "product details";
10. product: Object = {
11. "productCode": "PROD\_P001", "productName": "Laptop", "productPrice": 25000,
12. "purchaseDate": "5/12/2017", "productTax": "0.1", "productRating": 4.53
13. };
14. }

1. Write the below code in **app.component.html**
2. <h3> Product Details in JSON Format </h3>
3. {{ product | json }}
4. Save the files and check the output in the browser

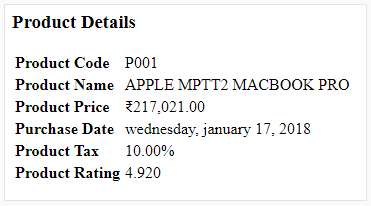
Now let us see how to use the built-in pipes with the help of a demo.

Highlights:

* Understanding Built-in Pipes
* Passing parameters to built-in pipes

**Demo Steps:**

**Problem Statement:**Apply built-in pipes with parameters to display product details. The output is as shown below:



1. Write the below code in **app.component.ts**
2. import { Component } from '@angular/core';
3. @Component({
4. selector: 'app-root',
5. templateUrl: './app.component.html',
6. styleUrls: ['./app.component.css']
7. })
8. export class AppComponent {
9. title: string = "product details"
10. productCode: string = "PROD\_P001";
11. productName: string = "Apple MPTT2 MacBook Pro"
12. productPrice: number = 217021;
13. purchaseDate: string = "1/17/2018"
14. productTax: string = "0.1";
15. productRating: number = 4.92;
16. }

1. Write the below code in **app.component.html**
2. <h3> {{ title | titlecase}} </h3>
3. <table style="text-align:left">
4. <tr>
5. <th> Product Code </th>
6. <td> {{ productCode | slice:5:9 }} </td>
7. </tr>
8. <tr>
9. <th> Product Name </th>
10. <td> {{ productName | uppercase }} </td>
11. </tr>
12. <tr>
13. <th> Product Price </th>
14. <td> {{ productPrice | currency: 'INR':'symbol' }} </td>
15. </tr>
16. <tr>
17. <th> Purchase Date </th>
18. <td> {{ purchaseDate | date:'fullDate' | lowercase}} </td>
19. </tr>
20. <tr>
21. <th> Product Tax </th>
22. <td> {{ productTax | percent : '.2' }} </td>
23. </tr>
24. <tr>
25. <th> Product Rating </th>
26. <td>{{ productRating | number:'1.3-5'}} </td>
27. </tr>
28. </table>

1. Write the below code in **app.module.ts**
2. import { NgModule } from '@angular/core';
3. import { BrowserModule } from '@angular/platform-browser';
4. import { AppComponent } from './app.component';
5. @NgModule({
6. imports: [BrowserModule],
7. declarations: [AppComponent],
8. providers:[],
9. bootstrap: [AppComponent]
10. })
11. export class AppModule {
12. }

1. Save the files and check the output in the browser

We have explored built-in pipes so far. But if we want to implement functionalities such as sorting, filtering, etc., we should go for custom pipes as there are no corresponding built-in pipes available

We can create our own custom pipe by inheriting PipeTransform interface

**PipeTransform**interface has a transform() method where we need to write custom pipe functionality

**Syntax**:

1. @Pipe({
2. name: 'pipename'
3. })
4. export class classname implements PipeTransform {
5. transform(value: any, ...args:any[]): any {
6. }
7. }

**transform()**method has two arguments, first one is the value of the expression passed to the pipe and the second is a variable "arguments". We can have multiple arguments based on the number of parameters passed to the pipe. The transform method should return the final value.

**Example**:

Let us create a custom pipe called **salutation** which should add **Mr.** or **Ms.** based on the parameter passed.

Create a pipe using the following CLI command

1. ng generate pipe salutation

This will create two files called salutation.pipe.ts to write custom pipe functionality and salutation.pipe.spec.ts for writing unit test cases for the pipe created. This command will also add the pipe to the root module to make it available to the entire module

**salutation.pipe.ts**

1. @Pipe({
2. name: 'salutation'
3. })
4. export class SalutationPipe implements PipeTransform {
5. transform(value: any, args?: any): any {
6. if(args == 'M'){
7. return "Mr."+value;
8. }
9. else if(args == 'F'){
10. return "Ms."+value;
11. }
12. else{
13. return value
14. }
15. }
16. }

**Line 1-3:** @Pipe decorator creates a pipe with the name **salutation**

**Line 4:** Inherit PipeTransform interface for custom pipe

**Line 6:** Overrides the transform method of the PipeTransform interface to write the functionality. This method stores the value passed into the first argument called value and the parameters of the pipe into the second argument called args.

**Line 7-15:** Based on the second argument value, functionality is implemented.

**app.component.ts**

1. import { Component } from '@angular/core';
2. @Component({
3. selector: 'app-root',
4. templateUrl: './app.component.html',
5. styleUrls: ['./app.component.css']
6. })
7. export class AppComponent {
8. username : string "JohnDavid";
9. }

**Line 9:** username property holds the value "JohnDavid"

**app.component.html**

1. {{ username | salutation : 'M' }}

**Line 1:** Render's username with 'Mr'

**app.module.ts**

1. import { BrowserModule } from '@angular/platform-browser';
2. import { NgModule } from '@angular/core';
3. import { FormsModule } from '@angular/forms';
4. import { AppComponent } from './app.component';
5. import { SalutationPipe } from './salutation.pipe';
6. @NgModule({
7. declarations: [
8. AppComponent,
9. SalutationPipe
10. ],
11. imports: [
12. BrowserModule,
13. FormsModule
14. ],
15. providers: [],
16. bootstrap: [AppComponent]
17. })
18. export class AppModule { }

**Line 10:** SalutationPipe is added to the declarations property to make it available to the entire module.

Let us learn how to create pipes with the help of a demo.