Angular Pipes

**Angular Pipes with Examples**

.

1. **What are Angular Pipes?**
2. **When and how to use Pipes in Angular Application?**
3. **Different types of Pipes with examples.**
4. 

**Why we need Angular Pipes?**

As we already know every web application starts with a simple task: first get the data, then transform the data into some format, and finally, show the formatted data to the users.

Getting the data is very simple, you can create a local variable or a complex type to hold the data or even you may get the data from APIs.

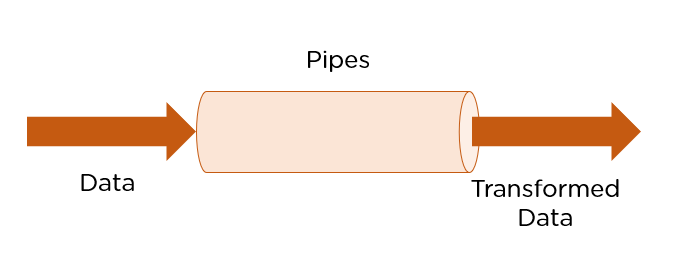
Once you get the data, then you could show the raw data as it is to the end-user, but that will not make a good user experience. To get a good user experience we need to modify the raw data into some specific format and in such cases, Angular Pipes plays an important role.

**What are Pipes in Angular Application?**

The Angular Pipe takes the raw data as input and then transforms the raw data into some desired format. So in simple words, we can say that the angular pipes transform the data into a specific format before displaying them to the end-users.

Using the Pipe (|) operator,

Angular Pipes transform the output. You can think of them as makeup rooms where they beautify the data into a more desirable format. They do not alter the data but change how they appear to the user.



Technically, pipes are simple functions designed to accept an input value, process, and return a transformed value as the output. [Angular](https://www.simplilearn.com/tutorials/angular-tutorial/what-is-angular) supports many built-in pipes. However, you can also create custom pipes that suit your requirements. Some salient features include:

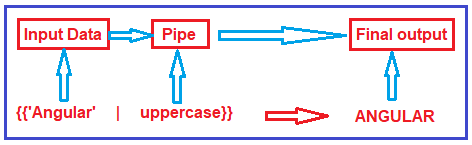
we can apply the pipes features to any of the property in angular application. There are so many built-in pipes provides by Angular Framework such as

* lowercase,
* uppercase,
* titlecase, decimal,
* date, percent
* , currency etc.
* It is also possible to create custom pipes.

**Syntax to use Pipes in Angular Application:**

1. **Pipes are defined using the pipe “|” symbol.**

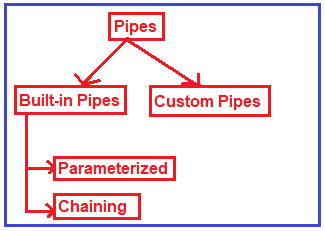
Syntaxes are written inside the HTML. The syntax to use the Angular Pipe is given below. To apply a pipe on a property, you need to use the pipe operator “|”.



**Types of Pipes in Angular:**

The Angular Framework divided the Pipes into two types i.e. Built-in Pipes and Custom Pipes. Further Built-in Pipes are divided into two types i.e

. Parameterized and chaining as shown in the below image.



# What Are Angular Pipes? How Are They Implemented?

## What Angular Pipes can do?

1. Pipes can be chained with other pipes.
2. Pipes can be provided with arguments by using the colon **(:) sign.**
3. **What is the difference between pure and impure pipes?**

Pure Pipes:

* 1. Input parameters value determine the output so if input parameters don’t change the output doesn’t change.
  2. Can be shared across many usages without affecting the output result.
  3. Pure pipes are pure functions that are easy to test.

Impure Pipes:

* 1. Cannot use the input value to determine if the output will change.
  2. Cannot be shared because the internal state can be affected from outside.

1. What is a pure pipe?
2. What is an async pipe?
3. What kind of data can be used with async pipe?
4. How do you create a custom pipe?
5. How does async pipe prevents memory leeks?

Some commonly used predefined Angular pipes are:

## Using Built-in Angular Pipes

Pipes are a great way to encapsulate and share a common display-value transformation.

1. **AsyncPipe**
2. **CurrencyPipe**
3. **DatePipe**
4. **DecimalPipe**
5. **JsonPipe**
6. **PercentPipe**
7. **LowerCasePipe**
8. **UpperCasePipe**
9. **SlicePipe**
10. **TitleCasePipe**

## ****Parameterizing the pipe in Angular****

We can also pass a parameter to the pipe. You can write the HTML code like this to pass the parameter.

*<!-- app.component.html -->*

<h1>

Krunal's birthdate is {{ birthday | date:"dd/MM/yyyy" }}

</h1>

Save the file, and you can see inside the browser that the page has a different date format as defined here.

## ****Chaining pipes****

We can chain pipes together and create useful combinations. For example, we can use the uppercase or lowercase pipe in our example.

*<!-- app.component.html -->*

<h1>

Krunal's birthdate is {{ birthday | date | uppercase }}

</h1>

Now, your date is in uppercase letters.

## ****Pure and impure pipes****

There are two categories of pipes:

1) pure

2) impure.

By default, the pipes in Angular are Pure. Every pipe you have seen so far has been pure like built-in pipes. You can make the pipe impure by setting the pure flag to false.

### ****Pure pipes****

Angular executes the pure pipe only when it detects the absolute change to an input value.  The pure change is either changing the primitive input value (String, Number, Boolean,  Symbol) or a changed object reference (Date, Array, Function, Object).

### ****Impure pipes****

Angular executes the contaminated pipe during every component change detection cycle. Thus, the impure pipe is called often, as often as every keystroke or mouse-move.

When writing a custom pipe in Angular you can specify whether you define a pure or an impure pipe:

@Pipe({  
 name: 'filterPipe',   
 pure: false/true <----- here (default is `true`)  
})  
export class FilterPipe {}

# Pure pipe

A pure pipe is only called when Angular detects a change in the value or the parameters passed to a pipe

@Pipe({  
 name: 'filterPipe',   
 pure: true   
})  
export class FilterPipe {}

# Impure pipe

An impure pipe is called for every change detection cycle no matter whether the value or parameter(s) changes.

@Pipe({  
 name: 'filterPipe',   
 pure: false  
})  
export class FilterPipe

**Example to understand Angular Pipes:**

Let us see an example to understand pipes. First we will see the output without pipes and then we will see the output with pipes.

**Modify app.component.ts file:**

Please modify the app.component.ts file as shown below. Here, we have created one student array with some dummy data that we want to show in the web page.

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

**@Component({**

**selector: 'app-root',**

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

**styleUrls: ['./app.component.css']**

**})**

**export class AppComponent {**

**title = 'DataBinding';**

**greet(){**

**event?.stopPropagation();**

**alert("innerelement");**

**}**

**greet1(){**

**alert("OuterElement");**

**}**

**students: any[] = [**

**{**

**ID: 'std101', FirstName: 'Preety', LastName: 'Mahesh',**

**Branch: 'CSE', DOB: '1988-09-09', Gender: 'Female'**

**,Fee:198756**

**},**

**{**

**ID: 'std102', FirstName: 'Anurag', LastName: 'Kumar',**

**Branch: 'ETC', DOB: '1989-09-08', Gender: 'Male'**

**,Fee:178906**

**},**

**{**

**ID: 'std103', FirstName: 'Priyanka', LastName: 'Rani',**

**Branch: 'CSE', DOB: '1992-04-09', Gender: 'Female',Fee:34216**

**},**

**{**

**ID: 'std104', FirstName: 'lakshman', LastName: 'Sharma',**

**Branch: 'ETC', DOB: '1990-09-06', Gender: 'Female',Fee:34567**

**},**

**{**

**ID: 'std105', FirstName: 'Siva', LastName: 'Satapathy',**

**Branch: 'CSE', DOB: '1991-02-07', Gender: 'Male',Fee:89956**

**}**

**];**

**}**

**Modify app.component.html file:**

Please modify **app.component.html**file as shown below. As you can see at the moment we are not using any pipes.

**<table** border="1"**>**

**<thead>**

**<tr>**

**<th>**Student ID**</th>**

**<th>**Name**</th>**

**<th>**DOB**</th>**

**<th>**Gender**</th>**

**<th>**Course Fee**</th>**

**</tr>**

**</thead>**

**<tbody>**

**<tr** \*ngFor='let student of students'**>**

**<td>**{{student.ID}}**</td>**

**<td>**{{student.Name}}**</td>**

**<td>**{{student.DOB}}**</td>**

**<td>**{{student.Gender}}**</td>**

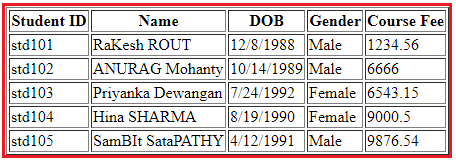
**<td>**{{student.CourseFee}}**</td>**

**</tr>**

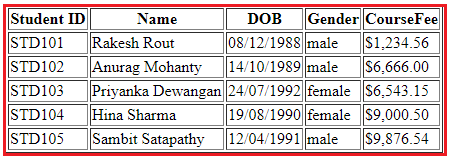
**</tbody>**

**</table>**

Now, if you browse the application, then you will get the following output in the browser.



As you can see in the above image, the data is not that user friendly. Let us discuss how we want to display the above data. We want to display the Student ID in upper case character and Name using the title case i.e. first character of every word in uppercase. Again, we want to display display the Date of Birth in **MM/DD/YYYY** format. We can achieve this by using the date pipe. Again we want to display the Gender in lower case and finally, we want to display the course Fee using the $ sign as shown in the below image.



**How can we achieve the above output?**

In order to achieve the desired output, we are going to use the following built-in pipes.

1. **lowercase**: This is used to convert the characters into lower case.
2. **uppercase**: This is used to convert the characters into upper case.
3. **titlecase**: This built-in pipe is used to convert the first character in each word to upper case.
4. **date**: This pipe is used to convert a date to some specific format.
5. **currency**: this pipe is used to convert number to currency with currency symbol.

So, modify the **app.component.html** file as shown below to use the required built-in pipes to get the desired output.

**<table** border="1"**>**

**<thead>**

**<tr>**

**<th>**Student ID**</th>**

**<th>**Name**</th>**

**<th>**DOB**</th>**

**<th>**Gender**</th>**

**<th>**CourseFee**</th>**

**</tr>**

**</thead>**

**<tbody>**

**<tr** \*ngFor='let student of students'**>**

**<td>**{{student.ID | uppercase}}**</td>**

**<td>**{{student.Name | titlecase}}**</td>**

**<td>**{{student.DOB | date:'dd/MM/yyyy'}}**</td>**

**<td>**{{student.Gender | lowercase}}**</td>**

**<td>**{{student.CourseFee | currency:'USD':true}}**</td>**

**</tr>**

**</tbody>**

**</table>**

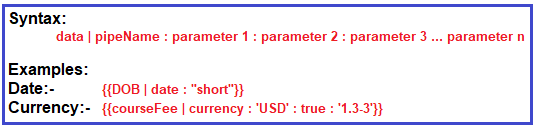
Now, run the application and you should get the output as expected.

**Angular Parameterized Pipes with Examples**

In this article, I am going to discuss **Angular Parameterized Pipes** in detail. Please read our previous article where we discussed the basics of [**Angular Pipes**](https://dotnettutorials.net/lesson/angular-pipes/). At the end of this article, you will understand what exactly Angular Parameterized Pipes are and when and how to use these pipes in Angular Application?

**What are Angular Parameterized Pipes?**

In Angular, we can pass any number of parameters to the pipe using a colon (:) and when we do so, it is called Angular Parameterized Pipes. The syntax to use Parameterized Pipes in Angular Application is given below.



**Date Pipe:**

Let us understand the Parameterized Date pipes with some examples. When you worked with any real-time applications, then you need to display the date time data in different formats. Here, I am going to show you some of the formats then I will provide you the link from where you will get all the available data formats.

**Step1: Modify app.modules.ts file**

Open **app.module.ts** file and then copy and paste the following code in it. Here we setting AppComponent as our startup file.

**import** **{** BrowserModule **}** from '@angular/platform-browser';

**import** **{** NgModule **}** from '@angular/core';

**import** **{** FormsModule **}** from '@angular/forms';

**import** **{** AppRoutingModule **}** from './app-routing.module';

**import** **{** AppComponent **}** from './app.component';

@NgModule**({**

declarations: **[**

AppComponent,

**]**,

imports: **[**

BrowserModule,

AppRoutingModule,

FormsModule

**]**,

providers: **[]**,

bootstrap: **[**AppComponent**]**

**})**

**export** **class** AppModule **{** **}**

**Step2: Modify app.component.ts file**

Open **app.component.ts** file and then copy and paste the following code in it. Here we simply create a variable i.e. today to hold the current data. As you can see you can use **Date.now()** to get the current date in typescript.

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

@Component**({**

selector: 'app-root',

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

styleUrls: **[**'./app.component.css'**]**

**})**

**export** **class** AppComponent **{**

today: **number** = Date.now**()**;

**}**

**Step3: Modify app.component.html file**

Open **app.component.html** file and then copy and paste the following code in it. As you can see, here we are using the parameterized data pipe to show different date formats.

**<p>**Date Pipe : {{today | date}}**</p>**

**<p>**Full Date : {{today | date:'fullDate'}}**</p>**

**<p>**Mediate Date : {{today | date:'medium'}}**</p>**

**<p>**Short Date : {{today | date:'short'}}**</p>**

**<p>**Date (dd/MM/yyyy) : {{today | date:'dd/MM/yyyy'}}**</p>**

**<p>**Time : {{today | date:'h:mm a z'}}**</p>**

**<p>**Medium Time : {{today | date:'mediumTime'}}**</p>**

**Step4: Modify index.html file**

Finally open **index.html** file and then copy and paste the following code in it.

<!doctype html>

**<html** lang="en"**>**

**<head>**

**<meta** charset="utf-8"**>**

**<title>**MyAngularApp**</title>**

**<base** href="/"**>**

**<meta** name="viewport" content="width=device-width, initial-scale=1"**>**

**<link** rel="icon" type="image/x-icon" href="favicon.ico"**>**

**</head>**

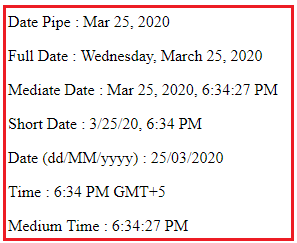
**<body>**

**<app-root></app-root>**

**</body>**

**</html>**

With the above changes in place nor run the application and you should the date in different formats in the web page as shown in the below image.



Please visit the following link for the complete list of angular date pipes.

[**https://angular.io/api/common/DatePipe**](https://angular.io/api/common/DatePipe)

**Currency Pipe:**

The Angular Currency Pipe is used to transforms a number to a currency string, formatted according to locale rules that determine group sizing and separator, decimal-point character, and other locale-specific configurations. Let us understand this with an example.

**Step1: Modify app.component.ts**

Open **app.component.ts** file and then copy and paste the following code in it. Here, we just created one property of type number.

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

@Component**({**

selector: 'app-root',

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

styleUrls: **[**'./app.component.css'**]**

**})**

**export** **class** AppComponent **{**

salary: **number** = 456723.50;

**}**

**Step2: Modify app.component.html file**

Open **app.component.html** file and then copy and paste the following code in it.

**<p>**Currency USD in Symbol : {{salary | currency:'USD':true}}**</p>**

**<p>**Currency INR in Symbol : {{salary | currency:'INR':true}}**</p>**

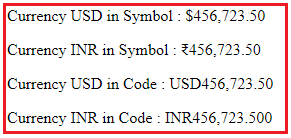
**<p>**Currency USD in Code : {{salary | currency:'USD':false:'4.2-2'}}**</p>**

**<p>**Currency INR in Code : {{salary | currency:'INR':false:'1.3-3'}}**</p>**

**Let us understand the above code.**

1. The first parameter is the currency Code (i.e. USD or INR)
2. The second parameter is boolean – True to display the currency symbol where as false to display the currency code.
3. The third parameter (‘1.3-3’ or ‘4.2-2’) specifies the number of integer and fractional digits.

Now save the changes and have a look at the browser and you should get the following output.



**Percentage Pipe Example**

**Appcomponent.html**

<table>

    <thead>

        <tr>

            <th>ID</th>

            <th>First Name</th>

            <th>Last Name</th>

            <th>Branch</th>

            <th>DOB</th>

            <th>Gender</th>

            <th>FEE</th>

            <th>dicountpercentage</th>

            <th>Discount</th>

    </thead>

    <tbody>

        <tr \*ngFor='let student of students'>

            <td>{{student.ID}}</td>

            <td>{{student.FirstName |titlecase}}</td>

            <td>{{student.LastName |titlecase}}</td>

            <td>{{student.Branch|titlecase}}</td>

            <td>{{student.DOB | date:'yyyy/MM/dd'}}</td>

            <td>{{student.Gender|uppercase}}</td>

            <td>{{student.Fee|currency:'INR':true}}</td>

            <td>{{0.1|percent}}</td>

            <td>{{(student.Fee)\*(0.1)|currency:'INR':true}}</td>

        </tr>

        <tr \*ngIf="!students || students.length==0">

            <td colspan="7">

                No Students to display

            </td>

        </tr>

    </tbody>

</table>

App.component.ts

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

@Component({

  selector: 'app-root',

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

  styleUrls: ['./app.component.css']

})

export class AppComponent {

  title = 'DataBinding';

  greet(){

    event?.stopPropagation();

    alert("innerelement");

  }

  greet1(){

    alert("OuterElement");

  }

  students: any[] = [

    {

    ID: 'std101', FirstName: 'Preety', LastName: 'Tiwary',

    Branch: 'CSE', DOB: '1988-09-09', Gender: 'Female'

    ,Fee:198756,discount:0

    },

    {

    ID: 'std102', FirstName: 'Anurag', LastName: 'Mohanty',

    Branch: 'ETC', DOB: '1989-09-08', Gender: 'Male'

    ,Fee:178906

    },

    {

    ID: 'std103', FirstName: 'Priyanka', LastName: 'Dewangan',

    Branch: 'CSE', DOB: '1992-04-09', Gender: 'Female',Fee:34216

    },

    {

    ID: 'std104', FirstName: 'Hina', LastName: 'Sharma',

    Branch: 'ETC', DOB: '1990-09-06', Gender: 'Female',Fee:34567

    },

    {

    ID: 'std105', FirstName: 'Sambit', LastName: 'Satapathy',

    Branch: 'CSE', DOB: '1991-02-07', Gender: 'Male',Fee:89956

    }

    ];

}

**Creating Angular Custom Pipe**

**Let us understand the need of Angular Custom Pipe with an example.**

Suppose you want to display the students detail in a web page. So, let us first create the student data in the AppComponent.

**Modify app.component.ts file:**

Open **app.component.ts** file and then copy and paste the following code in it. As you can here we simply created an student array.

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

@Component**({**

selector: 'app-root',

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

styleUrls: **[**'./app.component.css'**]**

**})**

**export** **class** AppComponent **{**

students: **any[]** = **[**

**{**

ID: 'std101', Name: 'Rakesh Rout',

DOB: '12/8/1988', Gender: 'Male', CourseFee: 1234.56

**}**,

**{**

ID: 'std102', Name: 'Anurag Mohanty',

DOB: '10/14/1989', Gender: 'Male', CourseFee: 6666.00

**}**,

**{**

ID: 'std103', Name: 'Priyanka Dewangan',

DOB: '7/24/1992', Gender: 'Female', CourseFee: 6543.15

**}**,

**{**

ID: 'std104', Name: 'Hina Sharma',

DOB: '8/19/1990', Gender: 'Female', CourseFee: 9000.50

**}**,

**{**

ID: 'std105', Name: 'Sambit Satapathy',

DOB: '4/12/1991', Gender: 'Male', CourseFee: 9876.54

**}**

**]**;

**}**

Let us show these student data in the web page.

**Modify app.student.html file:**

Open app.student.html file and then copy and paste the following code in it. As you can here we have applied some built-in pipes to format the data.

**<table** border="1"**>**

**<thead>**

**<tr>**

**<th>**Student ID**</th>**

**<th>**Name**</th>**

**<th>**DOB**</th>**

**<th>**Gender**</th>**

**<th>**CourseFee**</th>**

**</tr>**

**</thead>**

**<tbody>**

**<tr** \*ngFor='let student of students'**>**

**<td>**{{student.ID | uppercase}}**</td>**

**<td>**{{student.Name }}**</td>**

**<td>**{{student.DOB | date:'dd/MM/yyyy'}}**</td>**

**<td>**{{student.Gender | lowercase}}**</td>**

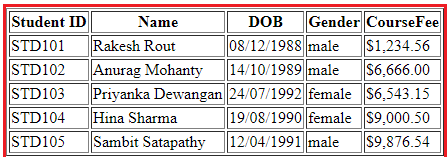
**<td>**{{student.CourseFee | currency:'USD':true}}**</td>**

**</tr>**

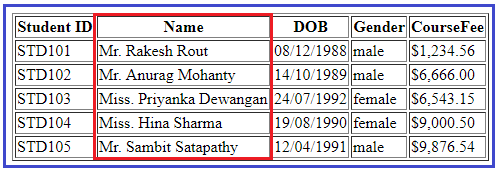
**</tbody>**

**</table>**

At this moment if you run the application, then you will get the following output in the browser.



Now, the requirement changes, now they want to show the title depending on the gender of the student I.e. we need to add Mr. or Miss. prefixed before the name of the student as shown in the below image.

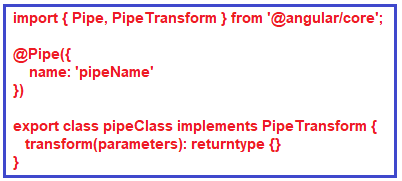


**How we can achieve this?**

We can achieve this very easily by creating an angular custom pipe.

**How to create Angular Custom Pipe?**

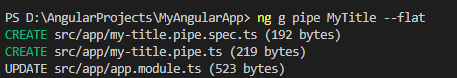
In order to create a custom pipe in angular, you have to apply the @Pipe decorator to a class which you can import from the Angular Core Library. The @Pipe decorator allows you to define the pipe name that you will use within the template expressions. The syntax to create a pipe in angular is given below.



**Note:**The transform method will decide the input types, the number of parameters and the output type.

**Creating Angular Custom Pipe using Angular CLI:**

Let say we want to create a custom pipe with the name MyTitle. In order to create a custom MyTitle pipe open a new terminal and type **ng g pipe MyTitle –flat** and press enter as shown in the below image.



Once you type **ng g pipe MyTitle –flat** and press enter, it will take some time and create two files (**my-title.pipe.ts** and **my-title.pipe.spec.ts**) within the app folder. Along the way, it also update the app.module.ts file.

**Modify my-title.pipe.ts file:**

Now, open my-title.pipe.ts file and then copy and paste the following code in it.

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

@Pipe**({**

name: 'myTitle'

**})**

**export** **class** MyTitlePipe implements PipeTransform **{**

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

**if** **(**gender.toLowerCase**()** == "male"**)**

**return** "Mr. " + name;

**else**

**return** "Miss. " + name;

**}**

**}**

**Understanding above code:**

* First, we import the **Pipe**decorator and **PipeTransform**interface from the Angular core Library.
* Then we decorated the “**MyTitlePipe**” class with the **Pipe**decorator so that this class will become an Angular pipe.
* **We then set the name property of the pipe decorator to myTitle so that we can use this name (myTitle) on any HTML page where we want this pipe functionality.**
* The **MyTitlePipe**class implements the **PipeTransform**interface and that interface has one method called **transform**() and here we implement that method.
* As you can see in the above code, the **transform**method takes 2 parameters (name and gender). The name parameter will receive the name of the student whereas the gender parameter will receive the gender of the student. The method returns a string i.e. Mr. or Miss. prefixed to the name of the student depending on their gender.

**Registering the Custom Pipe in Angular Application:**

Before using the custom MyTitlePipe, first we need to register it in the app.module.ts file.

If you are creating it using Angular CLI, then the angular framework will automatically register the pipe. To make sure, let us modify the app.module.ts file as shown below

. Here, first we need to import the MyTitlePipe and then we need to include it in the “declarations” array of NgModule decorator.

**import** **{** BrowserModule **}** from '@angular/platform-browser';

**import** **{** NgModule **}** from '@angular/core';

**import** **{** FormsModule **}** from '@angular/forms';

**import** **{** AppRoutingModule **}** from './app-routing.module';

**import** **{** AppComponent **}** from './app.component';

**import** **{** MyTitlePipe **}** from './my-title.pipe';

@NgModule**({**

declarations: **[**

AppComponent,

MyTitlePipe,

**]**,

imports: **[**

BrowserModule,

AppRoutingModule,

FormsModule

**]**,

providers: **[]**,

bootstrap: **[**AppComponent**]**

**})**

**export** **class** AppModule **{** **}**

**Using the Custom Pipe in Angular Application:**

**How to Create Custom Pipes**

To create a Custom Pipe, first, You need to follow these steps

1. Create a pipe class

(or)

Create pipe with following command

## ng generate pipe pipename

Use ng generate pipe followed by pipe name command to create custom pipes in angular.

ng generate pipe custom

// Output

CREATE src/app/custom.pipe.spec.ts (187 bytes)

CREATE src/app/custom.pipe.ts (217 bytes)

UPDATE src/app/app.module.ts (2931 bytes)

The command will create a file named custom.pipe.ts along with sample code to implement custom pipe at application root level.

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

@Pipe({

name: 'custom'

})

export class CustomPipe implements PipeTransform {

transform(value: unknown, ...args: unknown[]): unknown {

return null;

}

}

1. Decorate the class with @pipe decorator.

## Description[link](https://angular.io/api/core/Pipe#description)

A pipe class must implement the [PipeTransform](https://angular.io/api/core/PipeTransform) interface. For example, if the name is "myPipe", use a template binding expression such as the following:

content\_copy{{ exp | myPipe }}

The result of the expression is passed to the pipe's transform() method.

A pipe must belong to an NgModule in order for it to be available to a template. To make it a member of an NgModule, list it in the declarations field of the [NgModule](https://angular.io/api/core/NgModule) metadata.

## Options

| Name [mode\_edit](https://github.com/angular/angular/edit/master/packages/core/src/metadata/directives.ts?message=docs(core)%3A%20describe%20your%20change...#L609-L614)[code](https://github.com/angular/angular/tree/13.3.0/packages/core/src/metadata/directives.ts#L609-L614) |
| --- |
| The pipe name to use in template bindings. Typically uses [lowerCamelCase](https://angular.io/guide/glossary" \l "case-types) because the name cannot contain hyphens. |
| name: string |
|  |

| Pure [mode\_edit](https://github.com/angular/angular/edit/master/packages/core/src/metadata/directives.ts?message=docs(core)%3A%20describe%20your%20change...#L615-L626)[code](https://github.com/angular/angular/tree/13.3.0/packages/core/src/metadata/directives.ts#L615-L626) |
| --- |
| When true, the pipe is pure, meaning that the transform() method is invoked only when its input arguments change. Pipes are pure by default. |
| pure?: Boolean |
| If the pipe has internal state (that is, the result depends on state other than its arguments), set pure to false. In this case, the pipe is invoked on each change-detection cycle, even if the arguments have not changed. |

1. Give a name to the pipe in the name meta data of the @pipe decorator. We will use this name in the template.
2. The pipe class must implement the PipeTransform interface. The interfaces contain only one method transform.

# PipeTransform

INTERFACE

An interface that is implemented by pipes in order to perform a transformation. Angular invokes the transform method with the value of a binding as the first argument, and any parameters as the second argument in list form.

interface [PipeTransform](https://angular.io/api/core/PipeTransform) {

[**transform**(value: any, ...args: any[]): any](https://angular.io/api/core/PipeTransform#transform)

}

## Usage notes

In the following example, TruncatePipe returns the shortened value with an added ellipses.

simple\_truncate.ts

content\_copyimport {[Pipe](https://angular.io/api/core/Pipe), [PipeTransform](https://angular.io/api/core/PipeTransform)} from '@angular/core';

@[Pipe](https://angular.io/api/core/Pipe)({name: 'truncate'})

export class TruncatePipe implements [PipeTransform](https://angular.io/api/core/PipeTransform) {

transform(value: string) {

return value.split(' ').slice(0, 2).join(' ') + '...';

}

}

1. The first parameter to the transform method is the value to be transferred.
2. The transform method must transform the value and return the result. You can add any number of additional arguments to the transform method.
3. Declare the pipe class in the Angular Module (app.module.ts)
4. Use the custom pipe just as you use other pipes.

Modify the **app.component.html** file as shown below. Notice that we are passing the student gender as an argument for the gender parameter to our custom pipe. The Student name gets passed automatically.

**<table** border="1"**>**

**<thead>**

**<tr>**

**<th>**Student ID**</th>**

**<th>**Name**</th>**

**<th>**DOB**</th>**

**<th>**Gender**</th>**

**<th>**CourseFee**</th>**

**</tr>**

**</thead>**

**<tbody>**

**<tr** \*ngFor='let student of students'**>**

**<td>**{{student.ID | uppercase}}**</td>**

**<td>**{{student.Name | myTitle:student.Gender}}**</td>**

**<td>**{{student.DOB | date:'dd/MM/yyyy'}}**</td>**

**<td>**{{student.Gender | lowercase}}**</td>**

**<td>**{{student.CourseFee | currency:'USD':true}}**</td>**

**</tr>**

**</tbody>**

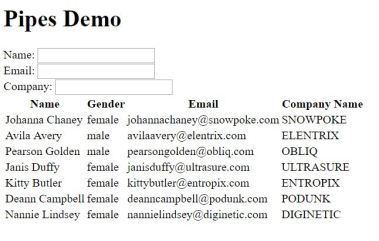
**</table>**

With the above changes in place, now run the application and you should the output as expected.

# Angular – Pipes passing multiple filters to Pipes

but what if there is requirement to pass multiple parameters to Pipes in Angular 2,below example demonstrates the same.

Below is the grid for displaying list of persons and as you type in the something in multiple filters like name, email and company, that filter criteria will get applied to that persons data and filter the grid items.





For creating custom pipes you need to create class that implements PipeTransform Interface, this class is then decorated with @Pipe decorator. From the PipeTranform interface you need to implement transform method,  for passing multiple filter parameters to the pipe in the transform method you need to add those parameters like shown below in the code snippet. (***nameSearch: string, emailSearch: string, companySearch: string***). Transform method implementation then can decide the filtering based on the parameters passed, you can implement your own custom logic over here.

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

@Pipe({

name: 'personSearch'

})

export class PersonSearchPipe implements PipeTransform {

transform(items: Array, nameSearch: string, emailSearch: string, companySearch: string){

if (items && items.length){

return items.filter(item =>{

if (nameSearch && item.name.toLowerCase().indexOf(nameSearch.toLowerCase()) === -1){

return false;

}

if (emailSearch && item.email.toLowerCase().indexOf(emailSearch.toLowerCase()) === -1){

return false;

}

if (companySearch && item.company.toLowerCase().indexOf(companySearch.toLowerCase()) === -1){

return false;

}

return true;

})

}

else{

return items;

}

}

}

You need to register the PersonSearchPipe to component module like show below

import { BrowserModule } from '@angular/platform-browser';

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

import { FormsModule } from '@angular/forms';

import { HttpModule } from '@angular/http';

import { AppComponent } from './app.component';

import { PersonSearchPipe } from './personSearch.pipe';

@NgModule({

declarations: [

AppComponent,

PersonSearchPipe

],

imports: [

BrowserModule,

FormsModule,

HttpModule

],

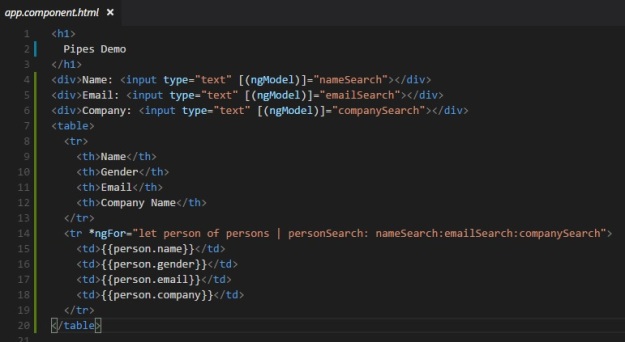
providers: [],

bootstrap: [AppComponent],

})

export class AppModule { }

In the HTML for the component add the filter like shown below



**Example-Custom pipes in angular 13**

**Angular 13 Custom Filter Search Pipe Example Tutorial**

Angular custom filter search pipe tutorial, In this tutorial, we will explain how to create a custom filter search pipe in the angular application using angular pipe and Bootstrap.

620.5K

Ukrainian Drone Company Volunteers Its Tech to War Effort

Pipes are handy and help you format the data before displaying it in the View. Pipe are declared with the help of the | symbol.

This symbol defines the Pipe Operator in angular. Throughout this Angular Search Filter tutorial, you will use the pipe symbol to filter the data.

Bootstrap is a robust front-end framework applied to build modern web and mobile applications. It reduces the pain of writing code from scratch for HTML and CSS, most importantly open-source.

Thus, we will use Bootstrap UI in this comprehensive angular search filter example.

**Angular 13 Custom Filter Search Pipe Example**

* Create Angular Project
* Generate Custom Pipe
* Create Custom Filter Search Pipe
* Add Search Filter in AppModule
* Add Dummy Data
* Implement Filter Search with Custom Pipe
* Run Angular Application

**Create Angular Project**

Angular CLI is a command-line interface tool recommended to initialize, develop likewise scaffold the angular application.

npm install -g @angular/cli

Further, use the following command to start installing the new angular app:

ng new ng-form-validaiton

Next, get into the project root:

cd ng-form-validaiton

**Disable Strict Angular TypeStrict Errors**

The latest version of Angular comes with strict mode, you have to manually disable the strict mode you can set “strict”: false, "noImplicitReturns": false and "strictTemplates": false inside the compilerOptions and angularCompilerOptions in **tsconfig.json** file.

/\* To learn more about this file see: https://angular.io/config/tsconfig. \*/

{

  "compileOnSave": false,

  "compilerOptions": {

    "baseUrl": "./",

    "outDir": "./dist/out-tsc",

    "forceConsistentCasingInFileNames": true,

    "strict": false,

    "noImplicitOverride": true,

    "noPropertyAccessFromIndexSignature": true,

    "noImplicitReturns": false,

    "noFallthroughCasesInSwitch": true,

    "sourceMap": true,

    "declaration": false,

    "downlevelIteration": true,

    "experimentalDecorators": true,

    "moduleResolution": "node",

    "importHelpers": true,

    "target": "es2017",

    "module": "es2020",

    "lib": [

      "es2020",

      "dom"

    ]

  },

  "angularCompilerOptions": {

    "enableI18nLegacyMessageIdFormat": false,

    "strictInjectionParameters": true,

    "strictInputAccessModifiers": true,

    "strictTemplates": false

  }

}

**Generate Custom Pipe in Angular**

Open the terminal, use the below command to generate the custom search filter pipe:

ng generate pipe search-filter

**Create Custom Filter Search Pipe**

The above command manifested the two files inside the app/ directory, hence open **search-filter.pipe.ts** file on top of that append the following code:

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

@Pipe({

name: 'searchFilter'

})

export class SearchFilterPipe implements PipeTransform {

transform(value: any, args?: any): any {

if(!value)return null;

if(!args)return value;

args = args.toLowerCase();

return value.filter(function(data){

return JSON.stringify(data).toLowerCase().includes(args);

});

}

}

TypeScript

COPY

**Register Search Filter in AppModule**

In the subsequent step, you have to add search filter, and register within the @NgModule’s declarations array. Thereupon, open and update the **app.module.ts** file:

import { BrowserModule } from '@angular/platform-browser';

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

import { AppRoutingModule } from './app-routing.module';

import { AppComponent } from './app.component';

import { FormsModule } from '@angular/forms';

import { SearchFilterPipe } from './search-filter.pipe';

@NgModule({

declarations: [

AppComponent,

SearchFilterPipe

],

imports: [

BrowserModule,

AppRoutingModule,

FormsModule

],

providers: [],

bootstrap: [AppComponent]

})

export class AppModule { }

**Add Dummy Data**

Open **app.component.ts** file, similarly define the Students array on top of that add the student’s test data.

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

@Component({

selector: 'app-root',

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

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

})

export class AppComponent {

public searchFilter: any = '';

constructor(){}

Students = [{

"id": 1,

"name": "Nathaniel Graham",

"email": "nathaniel.graham@example.com"

},

{

"id": 2,

"name": "Avery Adams",

"email": "avery.adams@example.com"

},

{

"id": 3,

"name": "Mario Stevens",

"email": "mario.stevens@example.com"

},

{

"id": 4,

"name": "Constance Beck",

"email": "constance.beck@example.com"

},

{

"id": 5,

"name": "Jimmie Little",

"email": "jimmie.little@example.com"

},

{

"id": 6,

"name": "Avery Matthews",

"email": "avery.matthews@example.com"

},

{

"id": 7,

"name": "Pat Sutton",

"email": "pat.sutton@example.com"

},

{

"id": 8,

"name": "Danny Crawford",

"email": "danny.crawford@example.com"

},

{

"id": 9,

"name": "Pearl Mccoy",

"email": "pearl.mccoy@example.com"

},

{

"id": 10,

"name": "Flenn Wallace",

"email": "flenn.wallace@example.com"

}

]

}

**Implement Filter Search with Custom Pipe**

In this step, you will implement the angular custom pipe and filter the data. Hence, open and update the **app.component.html** file.

<div class="container mt-5" style="max-width: 550px;">

<div class="form-group">

<input type="text" class="form-control mb-4" placeholder="Search" [(ngModel)]="query" id="listSearch">

</div>

<div class="card mb-1" \*ngFor="let data of Students | searchFilter: query">

<div class="card-body">

<h5 class="card-title">{{data.name}}</h5>

<h6 class="card-subtitle mb-2 text-muted">{{data.name}}</h6>

<p class="card-text">{{data.email}}</p>

</div>

</div>

</div>

**Run Angular Application**

In addition to to test the application, Consequently execute the following command to start the angular development server:

ng serve

Add the following url in the browser, and execute the below command:

http://localhost:4200

**Conclusion**

We have created a custom search pipe to filter results in Angular dynamically.

This step-by-step guide explained everything about creating a custom search filter pipe in the angular app using angular pipe and Bootstrap framework.

Class example for Custom Search Filter

**App.component.ts**

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

@Component({

  selector: 'app-root',

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

  styleUrls: ['./app.component.css']

})

export class AppComponent {

  title = 'pipes';

  public query:string =null ;

  students: any[] = [

    {

    ID: 'std101', Name: 'Rakesh Rout',

    DOB: '12/8/1988', Gender: 'Male', CourseFee: 1234.56

    },

    {

    ID: 'std102', Name: 'Anurag Mohanty',

    DOB: '10/14/1989', Gender: 'Male', CourseFee: 6666.00

    },

    {

    ID: 'std103', Name: 'Priyanka Dewangan',

    DOB: '7/24/1992', Gender: 'Female', CourseFee: 6543.15

    },

    {

    ID: 'std104', Name: 'Hina Sharma',

    DOB: '8/19/1990', Gender: 'Female', CourseFee: 9000.50

    },

    {

    ID: 'std105', Name: 'Sambit Satapathy',

    DOB: '4/12/1991', Gender: 'Male', CourseFee: 9876.54

    }

    ];

    }

Create a pipe

Ng g pipe searchfilter

Searchfileter.pipe.ts

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

@Pipe({

  name: 'searchfilter'

})

export class SearchfilterPipe implements PipeTransform {

  transform(value: any, args?: any): any {

    if(!value)return null;

    if(!args)return value;

    args = args.toLowerCase();

    return value.filter(function(data)

    {

        return JSON.stringify(data).toLowerCase().includes(args);

    });

}

  }

App.module.ts

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

import { BrowserModule } from '@angular/platform-browser';

import { AppComponent } from './app.component';

import { CustomPipePipe } from './custom-pipe.pipe';

import { FormsModule } from '@angular/forms';

import { SearchfilterPipe } from './searchfilter.pipe';

@NgModule({

  declarations: [

    AppComponent,

    CustomPipePipe,

    SearchfilterPipe

  ],

  imports: [

    BrowserModule,

    FormsModule

  ],

  providers: [],

  bootstrap: [AppComponent]

})

export class AppModule { }

<!DOCTYPE html>

<html>

<head>

    <title>

        filter pipe</title>

</head>

<body>

    <div>

        <input type="text" [(ngModel)]="query" id="serachstudent" placeholder="search">

    </div>

    <table border="1 ">

        <thead>

            <tr>

                <th>Student ID</th>

                <th>Name</th>

                <th>DOB</th>

                <th>Gender</th>

                <th>CourseFee</th>

            </tr>

        </thead>

        <tbody>

            <tr \*ngFor='let student of students | searchfilter:query'>

                <td>{{student.ID | uppercase}}</td>

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

                <td>{{student.DOB | date:'dd/MM/yyyy '}}</td>

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

                <td>{{student.CourseFee | currency:'USD ':true}}</td>

            </tr>

        </tbody>

    </table>

</body>

</html>

App.component.html

table {

    color: #369;

    font-family: Arial, Helvetica, sans-serif;

    font-size: large;

    border-collapse: collapse;

}

td {

    border: 1px solid #369;

    padding: 5px;

}

th {

    border: 1px solid #369;

    padding: 5px;

}

Custom Pipe

Example-2

how to create an Angular Custom Pipe. The [Pipes](https://www.tektutorialshub.com/angular/angular-pipes/) are a great way to transform the appearance of elements in the template. The [Angular](https://www.tektutorialshub.com/angular-tutorial/)comes with some great built-in pipes like Date pipe, Currency pipe, and Number pipe, etc. But if these pipes do not cover your needs, then we can create our own [pipe in Angular](https://www.tektutorialshub.com/angular/angular-pipes/).

must implement the PipeTransform interface. We also decorate it with @pipe decorator. Give a name to the pipe under name metadata of the @pipe decorator. Finally, we create the transform method, which transforms given value to the desired output.

Applies to: Angular 2 to the latest edition of i.e. Angular 8. Angular 9, Angular 10, Angular 11, Angular 12, Angular 13

## How to Create Custom Pipes

To create a Custom Pipe, first, You need to follow these steps

1. Create a pipe class
2. Decorate the class with @pipe decorator.
3. Give a name to the pipe in the name meta data of the @pipe decorator. We will use this name in the template.
4. The pipe class must implement the PipeTransform interface. The interfaces contain only one method transform.
5. The first parameter to the transform method is the value to be transferred. The transform method must transform the value and return the result. You can add any number of additional arguments to the transform method.
6. Declare the pipe class in the Angular Module (app.module.ts)
7. Use the custom pipe just as you use other pipes.

Now let us create a Temperature converter pipe, which converts temperature from Celsius to Fahrenheit and vice versa.

## Temparature Convertor Custom Pipe Example

Create a new Angular application. We are using bootstrap 4 for styling. Hence open the index.html and add the following

|  |  |
| --- | --- |
| 1  2  3 | <link rel="stylesheet" href="https://maxcdn.bootstrapcdn.com/bootstrap/4.0.0/css/bootstrap.min.css" integrity="sha384-Gn5384xqQ1aoWXA+058RXPxPg6fy4IWvTNh0E263XmFcJlSAwiGgFAW/dAiS6JXm" crossorigin="anonymous"> |

Create a new file temp-convertor.pipe.ts. Under the folder src/app. Copy the following code and paste it.

|  |  |
| --- | --- |
|  | import {Pipe, PipeTransform} from '@angular/core';    @pipe({      name: 'tempConverter'  })  export class TempConverterPipe implements PipeTransform {      transform(value: number, unit: string) {          if(value && !isNaN(value)) {              if (unit === 'C') {                  var temperature = (value - 32) /1.8 ;                  return temperature.toFixed(2);              } else if (unit === 'F'){                  var temperature = (value \* 1.8 ) + 32                  return temperature.toFixed(2);              }          }          return;      }  } |

Let us look at the code in details

We need to import the Pipe & PipeTransform libraries from Angular. These libraries are part of the Angular Core

|  |  |
| --- | --- |
| 1  2  3 | import {Pipe, PipeTransform} from '@angular/core'; |

We decorate TempConverterPipe class with @pipe decorator. The[@pipe decorator](https://angular.io/api/core/Pipe) is what tells Angular that the class is a pipe. the decorator expects us to provide a name to the pipe. We have given it as tempConverter. This is the name we must use in the template to make use of this pipe.

Our class must implement the PipeTransform interface.

|  |
| --- |
| @pipe({      name: 'tempConverter'  })  export class TempConverterPipe implements PipeTransform {      } |

The PipeTransform interface defines only one method transform. The interface definition is as follows.

|  |
| --- |
| interface PipeTransform {    transform(value: any, ...args: any[]): any  } |

The first argument value is the value, that pipe needs to transform. We can also include any number of arguments. The method must return the final transformed data.

The following is Our implementation of the transform method. The first is Value and the second is the Unit. The unit expects either C (Convert to Celsius) or F ( convert to Fahrenheit). It converts the value received to either to Celsius or to Fahrenheit based on the Unit.

|  |
| --- |
| export class TempConverterPipe implements PipeTransform {        transform(value: number, unit: string) {          if(value && !isNaN(value)) {              if (unit === 'C') {                 var temperature = (value - 32) /1.8 ;                 return temperature.toFixed(2);              } else if (unit === 'F'){                 var temperature = (value \* 1.8 ) + 32                 return temperature.toFixed(2);              }          }          return;      }    } |

### Declare the Pipe

Before using our pipe, we need to tell our component, where to find it. This is done by first by importing it and then including it in declarations array of the AppModule.

|  |  |
| --- | --- |
|  | import { BrowserModule } from '@angular/platform-browser';  import { NgModule } from '@angular/core';  import { FormsModule } from '@angular/forms';  import { HttpModule } from '@angular/http';    import { AppComponent } from './app.component';    import {TempConverterPipe} from './temp-convertor.pipe';    @NgModule({      declarations: [AppComponent,TempConverterPipe],      imports: [BrowserModule,FormsModule,HttpModule],      bootstrap: [AppComponent]  })  export class AppModule { } |

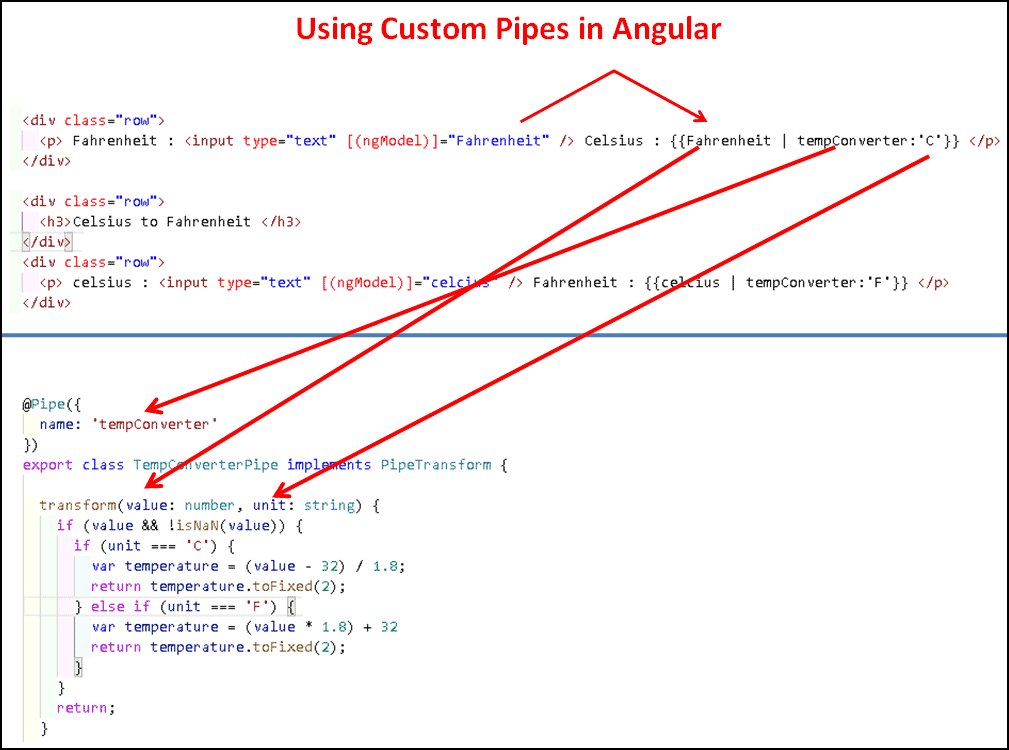
### Using the Custom Pipe

The custom pipes are used in the same as the Angular built-in pipes are used. Add the following HTML code to your app.component.html file

|  |  |
| --- | --- |
| 5 | <div class='card'>    <div class='card-header'>      <p>{{title}} </p>    </div>    <div class="card-body">        <div class="row">        <h3>Fahrenheit to Celsius </h3>      </div>      <div class="row">        <p> Fahrenheit : <input type="text" [(ngModel)]="Fahrenheit" />        Celsius : {{Fahrenheit | tempConverter:'C'}} </p>      </div>        <div class="row">        <h3>Celsius to Fahrenheit </h3>      </div>      <div class="row">        <p> celsius : <input type="text" [(ngModel)]="celcius" />         Fahrenheit : {{celcius | tempConverter:'F'}} </p>      </div>    </div>  </div> |

We use our pipe as follows. Fahrenheit is sent to the tempConverter as the first argument value. We use the | to indicate that the tempConverter is a pipe to angular. The C after the colon is the first argument. You can pass more than argument to the pipe by separating each argument by a : colon.

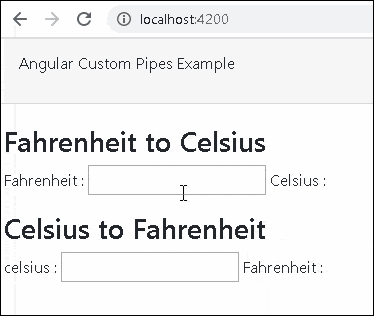
|  |  |
| --- | --- |
|  | {{Fahrenheit | tempConverter:'C'}} |

Using the TempConverter Pipe in Template

app.component code

|  |  |
| --- | --- |
|  | import { Component } from '@angular/core';  import { FormsModule } from '@angular/forms';      @Component({    selector: 'app-root',    templateUrl: './app.component.html',    styleUrls: ['./app.component.css']  })      export class AppComponent  {      title: string = 'Angular Custom Pipe Example' ;      celcius: number;      Fahrenheit: number;  } |

Run the application and test it. I

Angular Custom Pipe Example

# Multiple custom pipes

## Example[#](https://riptutorial.com/angular/example/30433/multiple-custom-pipes#example)

Having different pipes is a very common case, where each pipe does a different thing. Adding each pipe to each component may become a repetitive code.

It is possible to bundle all frequently used pipes in one Module and import that new module in any component needs the pipes.

breaklines.ts

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

/\*\*

\* pipe to convert the \r\n into <br />

\*/

@Pipe({ name: 'br' })

export class BreakLine {

transform(value: string): string {

return value == undefined ? value :

value.replace(new RegExp('\r\n', 'g'), '<br />')

.replace(new RegExp('\n', 'g'), '<br />');

}

}

uppercase.ts

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

/\*\*

\* pipe to uppercase a string

\*/

@Pipe({ name: 'upper' })

export class Uppercase{

transform(value: string): string {

return value == undefined ? value : value.toUpperCase( );

}

}

pipes.module.ts

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

import { BreakLine } from './breakLine';

import { Uppercase} from './uppercase';

@NgModule({

declarations: [

BreakLine,

Uppercase

],

imports: [

],

exports: [

BreakLine,

Uppercase

]

,

})

export class PipesModule {}

my.component.ts

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

@Component({

selector: 'my-component',

template: `{{ value | upper | br}}`

})

export class MyComponent {

public value: string;

}

my.module.ts

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

import { BrowserModule } from '@angular/platform-browser';

import { MyComponent } from './my.component';

import { PipesModule} from './pipes.module';

@NgModule({

imports: [

BrowserModule,

PipesModule,

],

declarations: [

MyComponent,

],

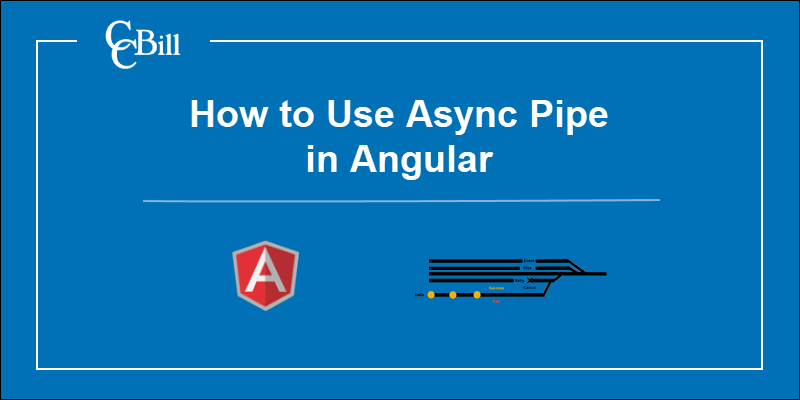
})

## Introduction

Built-in Angular Pipes are easily implemented within the Angular template syntax and are practical for handling common formatting tasks.

An Async Pipe allows you to detect changes and propagate asynchronous events directly in the template without changing the data’s underlying value.

Find out how to**use Async Pipes to subscribe to Observables and Promises**.



## What is an Async Pipe?

An Async Pipe is a built-in Angular feature that allows you to subscribe and automatically unsubscribe from objects. When subscribed to an Observable or Promise, the Async Pipe creates a copy of the latest emitted output, modifies its format, and displays the resulting value directly in the view. The Async Pipe uses a straightforward syntax:

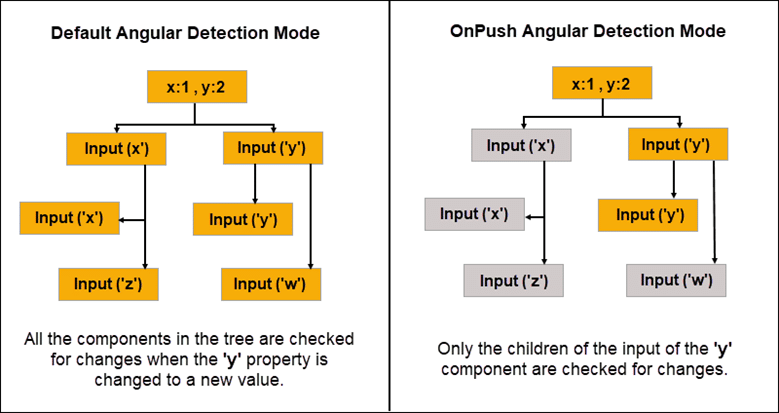
**{{ obj\_expression | async }}**

You can also use Async Pipes to avoid unnecessary Angular change detection runs and apply checks only to Observables that receive new values.

## When to Use Async Pipes?

By default, Angular runs change detection on all components before updating the DOM. This process can potentially drain system resources and negatively affect your app.

Using Async Pipes with the OnPush change detection strategy can improve web application performance. By setting the **ChangeDetector** class to <strong>OnPush</strong>, only an Observable that registers a new value needs to go through the change detection process.



The OnPush mode uses the Async Pipe to inform Angular that the component only tracks value changes originating from its parent. Angular does not need to check the component if there were no changes registered in the parent component.

Additionally, an Async Pipe streamlines Angular’s change detection process by subscribing and automatically unsubscribing from a component precisely at the end of its life cycle. You no longer need to unsubscribe from an Observable or Promise manually. Async Pipes guarantee that redundant subscriptions do not remain open after the component is destroyed and result in a potential memory leak.

### Using Async Pipes with Observables

You can subscribe to an Observable object to track changes and pass functions that execute on specific events. An Async Pipe automates this process by consuming the values from the observable data stream and exposing the resolved values for binding.

In this example, an Async Pipe is used with the **\*ngFor** directive to resolve an observable to an array type.

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

import {Observable, of} from 'rxjs';

@Component({

selector: 'async-observable-pipe',

template: `<ul><li \*ngFor="let d of uList | async">{{d}}</li></ul>`

})

export class AppComponent {

uList: Observable<number[]>;

constructor() {

this.uList = this.getData();

}

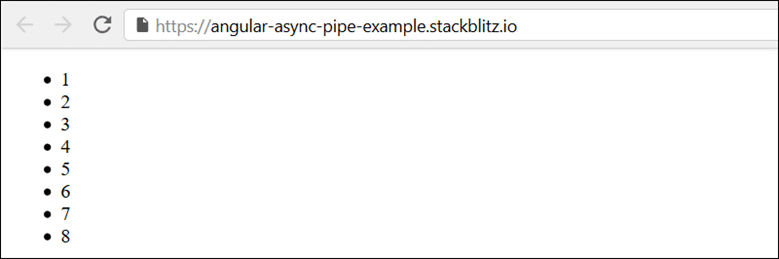
getData(): Observable<number[]> {

return of([1,2,3,4,5,6,7,8]);

}

}

The Async Pipe maintains the subscription to the Observable and continues to deliver values in real-time.



Once the observed component is destroyed, the Async Pipe categorically unsubscribes from the Observable.

### Using Async Pipes with Promises

You can use the Promise function to resolve a value asynchronously. Promises are limited to a one call cycle, either a **resolve** call with a single fulfillment value or a **reject** call with a single error message.

We need to evaluate the resulting value asynchronously and display it directly. You can use the result to call an API that:

1. Returns a promise.
2. Passes that promise into the binding using the Async Pipe.
3. Returns the new value and displays it in the view.

@Component({

selector: 'async-promise-pipe',

template: `<div>

<code>promise|async</code>:

<button (click)="clicked()">{{ arrived ? 'Reset' : 'Resolve' }}</button>

<span>Wait for it... {{ greeting | async }}</span>

</div>`

})

export class AsyncPromisePipeComponent {

greeting: Promise<string>|null = null;

arrived: boolean = false;

private resolve: Function|null = null;

constructor() {

this.reset();

}

reset() {

this.arrived = false;

this.greeting = new Promise<string>((resolve, reject) => {

this.resolve = resolve;

});

}

clicked() {

if (this.arrived) {

this.reset();

} else {

this.resolve!('hi there!');

this.arrived = true;

}

}

}

If a promise needs to return a value on an event such as a user click, the promise is going to resolve on the first click. Observables offer more flexibility and can perform the same function for multiple user clicks.

## Conclusion

You now have a better understanding of Async Pipes and how to use them to subscribe and unsubscribe to Observables and Promises.

Implementing Async Pipes in the right use cases makes the Angular change detection process more efficient and ultimately improve end-user experience.

### Pure and Impure Pipes in Angular

When you [create a custom pipe in Angular](https://www.netjstech.com/2020/12/custom-pipe-angular-example.html) there is one more attribute of @Pipe decorator which you can assign a value as true or false, that attribute is **pure**. In this tutorial we’ll see what are pure and impure pipes in Angular and what are the differences between pure and impure pipes.

@Pipe({

name: 'myCustomPipe',

pure: false/true

})

export class MyCustomPipe {

}

*By default, pipes are defined as pure* so you don't explicitly need to assign value of pure as true.

**Table of contents**

1. [Pure pipes in Angular](https://www.netjstech.com/2020/12/pure-impure-pipes-angular.html#AngularPurePipe)
2. [Impure pipes in Angular](https://www.netjstech.com/2020/12/pure-impure-pipes-angular.html#AngularImpurePipe)
3. [Pure and impure pipe Angular example](https://www.netjstech.com/2020/12/pure-impure-pipes-angular.html#AngularPureImpurePipe)

### Pure pipes in Angular

Angular pipes use data binding, if the data is a primitive input value like String or Number or an object reference such as Date or Array, Angular executes the pipe whenever it detects a change for the input value or reference.

Pure pipes in Angular (which is also default) are executed only when Angular detects a pure change to the input value. A pure change is either a change to a primitive input value (such as String, Number, Boolean, or Symbol), or a changed object reference (such as Date, Array, Function, or Object).

A pure pipe must use a pure function. A pure function always return the same output for the same input. For example following function to add number is a pure function calling it multiple times with argument 2 and 3 gives the same result 5.

addPure(a, b) {

return a + b;

};

*With a pure pipe, Angular ignores changes within objects. If you add an element to an existing array that won’t be considered by a pure pipe*, because checking a primitive value or object reference is much faster than performing a deep check for differences within objects.

### Impure pipes in Angular

If you want pipe to be executed after a change within an object, such as a change to an element of an array, you need to define your pipe as impure to detect impure changes.

An impure pipe is called for every change detection cycle which could slow down your app drastically so be very careful with implementing an impure pipe in Angular.

### Pure and impure pipe Angular example

In the example we’ll create an Angular app to display user data. We’ll create a custom pipe to filter the data on firstName.

**Custom pipe class (filter.pipe.ts)**

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

@Pipe({

name:'filter'

})

export class FilterPipe implements PipeTransform{

transform(value: any, name: string) {

if(name === ''){

return value;

}

return value.filter((user) => user.firstName.startsWith(name));

}

}

**Component Class (app.component.ts)**

In the component class there is an array users storing user details. There is also a property nameString used for [two way binding](https://www.netjstech.com/2020/04/angular-two-way-data-binding-with-example.html) with the input value in the template for the search.

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

@Component({

selector: 'app-root',

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

})

export class AppComponent{

nameString = '';

users = [{

firstName: 'John',

lastName: 'Doe',

dept: 'Finance',

salary: 5000,

doj: new Date('2015-12-11')

},

{

firstName: 'Amy',

lastName: 'Watson',

dept: 'HR',

salary: 8000,

doj: new Date('2013-07-23')

},

{

firstName: 'Shrishti',

lastName: 'Sharma',

dept: 'IT',

salary: 10000,

doj: new Date('2019-10-20')

}

]

}

**Template (app.component.html)**

<div class="container">

<h1>User Details</h1>

<span>Search </span><input type="text" [(ngModel)]="nameString">

<br/><br/>

<table class="table table-sm table-striped m-t-4">

<thead class="thead-dark">

<tr>

<th>First Name</th>

<th>Last Name</th>

<th>Department</th>

<th>Salary</th>

<th>Joining Date</th>

</tr>

</thead>

<tbody>

<tr \*ngFor="let user of users | filter:nameString">

<td>{{user.firstName}}</td>

<td>{{user.lastName}}</td>

<td>{{user.dept}}</td>

<td>{{user.salary}}</td>

<td>{{user.doj | date:'dd/MM/yyyy'}}</td>

</tr>

</tbody>

</table>

</div>

In the template there is an input element with a two way binding done for the “nameString” property in the component.

As you can see filter pipe is used here with in the [ngFor directive](https://www.netjstech.com/2020/04/angular-ngfor-directive-with-examples.html" \t "_blank). Since it is bound with the users array so that is passed as the value. Parameter passed with the pipe here is nameString which is the value of the search string.

Here is a screen shot of how you can use this filter pipe.

**Adding pipe declaration to the AppModule**

If you have created the Pipe class yourself then you do need to add it to the AppModule class youself too. Add the class in the declarations array.

@NgModule({

declarations: [

AppComponent,

FilterPipe

],

...

...

})

export class AppModule { }

Now to show what is the difference between pure pipe and impure pipe in Angular let’s add two methods in the component class one to add a new user to the already existing array of users and another to change the reference of the array.

addUser(){

this.users.push({

firstName: 'Alia',

lastName: 'Bajaj',

dept: 'Finance',

salary: 6000,

doj: new Date('2016-09-16')

})

}

reset(){

this.users = this.users.slice()

}

Two buttons are also added to the template to call these methods.

<button type="button" class="btn btn-success m-3" (click)="addUser()">Add User</button>

<button type="button" class="btn btn-success" (click)="reset()">Reset</button>

With these changes you can see that when the filtering is done on ‘A’ and you click on Add User button that change won’t be reflected in the filtered result because pure pipe ignores changes within objects. If you click on reset now which changes the array reference itself then the change is reflected.

Another way to make the changes in the array reflected when filtering is to make the pipe impure. You can test that by making the following change in the FilterPipe.

@Pipe({

name:'filter',

pure: false

})

export class FilterPipe implements PipeTransform{

...

...

}

With this change when you click on Add User button that change will be reflected in the filtered result as FilterPipe is an impure pipe now and it will be executed after a change within an object.