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.

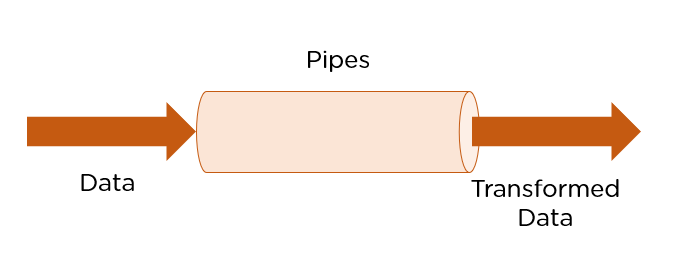
Once you get the data, then you could show the raw data as it is to the end-user, but that will not make a good user experience. To get a good user experience we need to modify the raw data into some specific format and in such cases,

Angular Pipes plays an important role.in transforming raw data

**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.

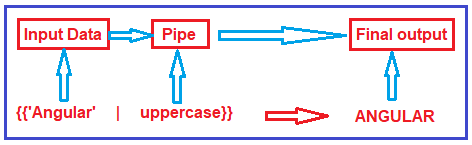
**Using the Pipe (|) operator, it is located above the enter key in the keyboard**

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.

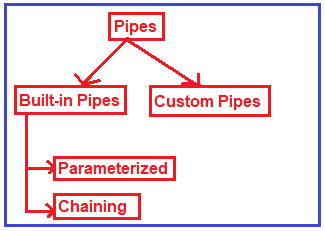
**Syntax to use Pipes in Angular Application:**

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



**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?

## Table of Contents

[What are Angular Pipes?](https://www.simplilearn.com/tutorials/angular-tutorial/angular-pipes#what_are_angular_pipes)

[Using Built-in Angular Pipes](https://www.simplilearn.com/tutorials/angular-tutorial/angular-pipes#using_builtin_angular_pipes)

[Creating Custom Pipes](https://www.simplilearn.com/tutorials/angular-tutorial/angular-pipes#creating_custom_pipes)

[Pure and Impure Pipes](https://www.simplilearn.com/tutorials/angular-tutorial/angular-pipes#pure_and_impure_pipes)

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:

1. Pipes can be chained with other pipes.
2. Pipes can be provided with arguments by using the colon (:) sign.

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

**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 **{**

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

**{**

ID: 'std101', Name: 'RaKesh ROUT',

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

**}**,

**{**

ID: 'std102', Name: 'ANURAG Mohanty',

DOB: new Date(4, 12,1991), Gender: 'Male', CourseFee: 6666.00

**}**,

**{**

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

DOB: new Date(4, 12,1991), Gender: 'Female', CourseFee: 6543.15

**}**,

**{**

ID: 'std104', Name: 'Hina SHARMA',

DOB: new Date(4, 12,1991), Gender: 'Female', CourseFee: 9000.50

**}**,

**{**

ID: 'std105', Name: 'SamBIt SataPATHY',

DOB: new Date(4, 12,1991), Gender: 'Male', CourseFee: 9876.54

**}**

**]**;

**}**

**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>**

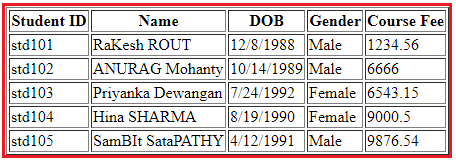
**<td>**{{student.CourseFee}}**</td>**

**</tr>**

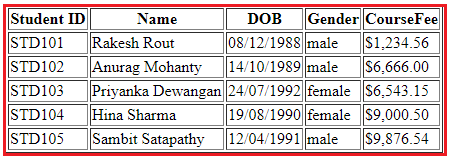
**</tbody>**

**</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>**

<table>

  <thead>

      <tr>

        <td>index</td>

          <th>ID</th>

          <th>First Name</th>

          <th>Last Name</th>

          <th>Branch</th>

          <th>DOB</th>

          <th>Gender</th>

          <th>isFirst</th>

            <th>isLast</th>

            <th>isOdd</th>

            <th>isEven</th>

            <th>studentfee</th>

      </tr>

  </thead>

  <tbody>

      <tr \*ngFor='*let* *student* *of* *students*; *let* i=*index*;*let* isFirst=*first*;*let* isLast=*last*;*let* *isEven* = *even*; *let* *isOdd* = *odd*'>

        <td>{{i}}</td>

          <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>

          <td>{{isFirst}}</td>

          <td>{{isLast}}</td>

          <td>{{isOdd}}</td>

          <td>{{isEven}}</td>

          <td>{{student.studentfee}}</td>

      </tr>

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

          <td *colspan*="7">

              No Students to display

          </td>

      </tr>

    </tbody>

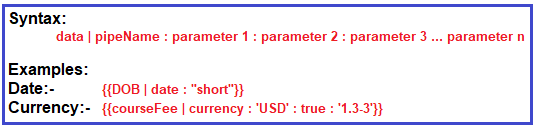
Now, run the application and you should get the output as expected.

**Angular Parameterized Pipes with Examples**

, 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>**

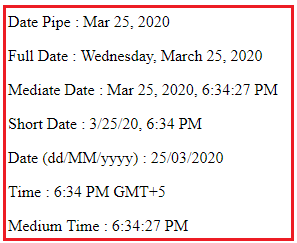
**<body>**

**<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>**

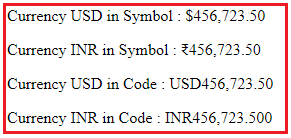
**<p>**Currency USD in Code : {{salary | currency:'USD':false:'4.2-2'}}**</p>**

**<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.



**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>**

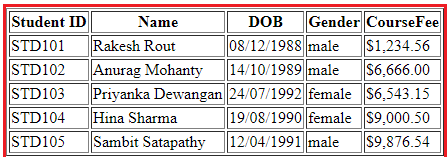
**<td>**{{student.CourseFee | currency:'USD':true}}**</td>**

**</tr>**

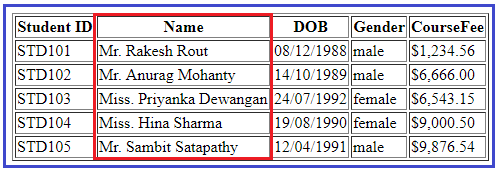
**</tbody>**

**</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.

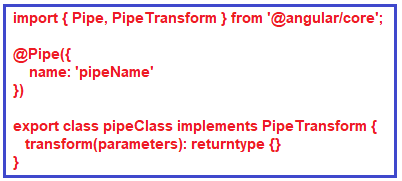


**How we can achieve this?**

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

**How to create Angular Custom Pipe?**

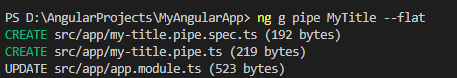
In order to create a custom pipe in angular, you have to apply the @Pipe decorator to a class which you can import from the Angular Core Library. The @Pipe decorator allows you to define the pipe name that you will use within the template expressions. The syntax to create a pipe in angular is given below.



**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:**

1. First, we import the **Pipe**decorator and **PipeTransform**interface from the Angular core Library.
2. Then we decorated the “**MyTitlePipe**” class with the **Pipe**decorator so that this class will become an Angular pipe.
3. 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.
4. The **MyTitlePipe**class implements the **PipeTransform**interface and that interface has one method called **transform**() and here we implement that method.
5. 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:**

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>**

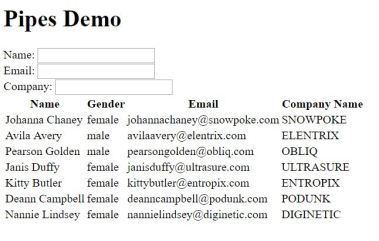
**</table>**

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

],

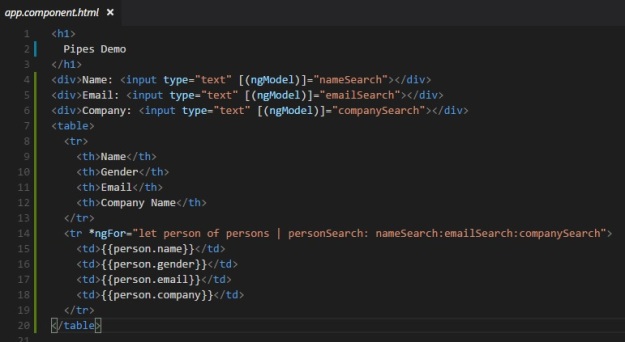
providers: [],

bootstrap: [AppComponent],

})

export class AppModule { }

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

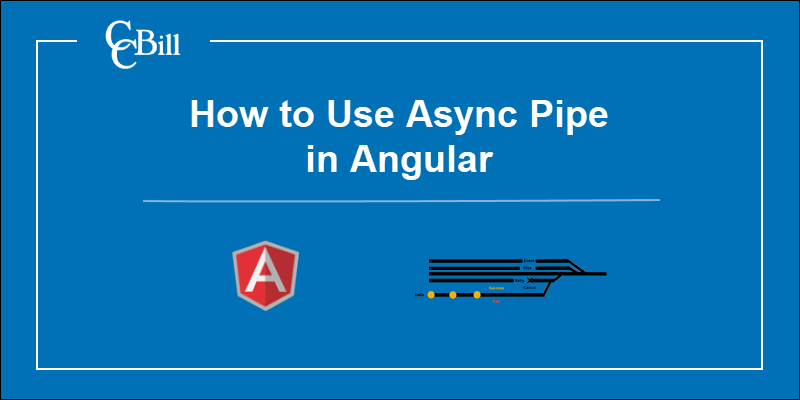


## Introduction

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:

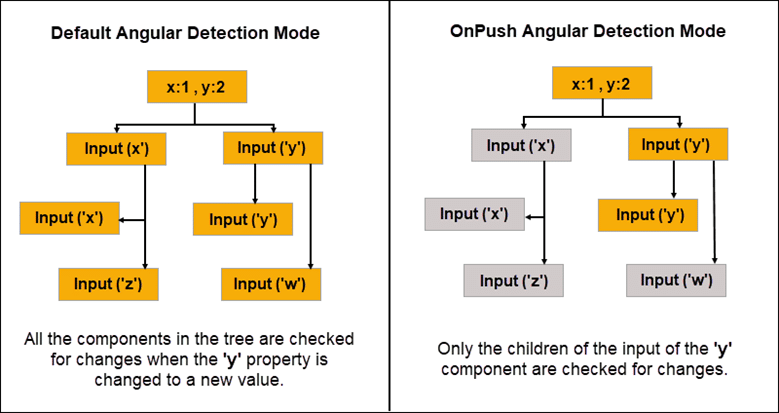
**{{ obj\_expression | async }}**

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();

}

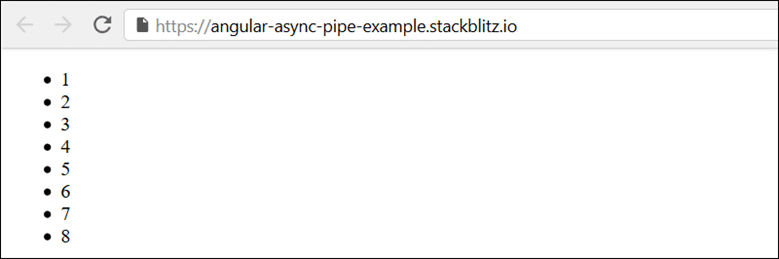
getData(): Observable<number[]> {

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.