**🔄 1. Angular Pipes Overview**

A **pipe** in Angular transforms data in templates. It’s used with the | symbol.

{{ value | pipeName:arg1:arg2 }}

**✅ 2. Pure vs Impure Pipes**

| **Feature** | **Pure Pipe** | **Impure Pipe** |
| --- | --- | --- |
| **Default** | Yes | No (pure: false in @Pipe) |
| **Called When** | On **reference change** | On **every change detection** |
| **Performance** | High (better performance) | Low (can slow down app) |
| **Use Case** | Simple, stateless transformations | Data that changes frequently, like filtering an array |

**🔧 3. Using Built-in Pipes**

| **Pipe Name** | **Description** | **Example** |
| --- | --- | --- |
| date | Formats a date | `{{ today |
| uppercase | Converts to uppercase | `{{ 'hello' |
| lowercase | Converts to lowercase | `{{ 'HELLO' |
| currency | Formats currency | `{{ price |
| percent | Formats number as percentage | `{{ 0.25 |
| json | Converts object to JSON string | `{{ object |
| slice | Slices string/array | `{{ name |
| async | Unwraps Observable or Promise | `{{ observableData |

Here’s a **complete Angular demo using the async pipe**, which is one of the most powerful **built-in pure pipes** in Angular.

**✅ Scenario: Async Pipe with Observable Timer**

We will:

* Create an Observable that emits values over time
* Use the async pipe in the template to subscribe and display data
* Show how it **automatically unsubscribes**

**🔧 File Structure**

src/

├── app/

│ ├── app.component.ts

│ ├── app.component.html

│ └── app.module.ts

**1️⃣ app.component.ts**

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

import { Observable, interval } from 'rxjs';

import { map } from 'rxjs/operators';

@Component({

selector: 'app-root',

templateUrl: './app.component.html'

})

export class AppComponent {

time$: Observable<string>;

constructor() {

// Emits the current time every second

this.time$ = interval(1000).pipe(

map(() => new Date().toLocaleTimeString())

);

}

}

**2️⃣ app.component.html**

<div style="padding: 20px;">

<h2>⏰ Async Pipe Demo</h2>

<p>Current Time: {{ time$ | async }}</p>

</div>

**3️⃣ app.module.ts**

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

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

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

@NgModule({

declarations: [

AppComponent

],

imports: [

BrowserModule

],

providers: [],

bootstrap: [AppComponent]

})

export class AppModule { }

**🚀 How to Run**

1. Create a new Angular app:

ng new async-pipe-demo

cd async-pipe-demo

1. Replace contents of src/app with the files above.
2. Run the app:

ng serve

Go to <http://localhost:4200>

**🎯 What You’ll Learn**

* The async pipe **subscribes to the observable** and renders the emitted value.
* It **auto-unsubscribes** when the component is destroyed (prevents memory leaks).
* You don’t need to call subscribe() or unsubscribe() manually in the component.

**🧠 Where Async Pipe is Used in Real Apps**

| **Use Case** | **Example** |
| --- | --- |
| Data from HTTP calls | http.get(...).pipe(...) |
| Reactive forms or search | FormControl.valueChanges |
| WebSocket or event stream | webSocketSubject.asObservable() |
| Real-time timers or clocks | interval(), timer() |

Here's a **full working Angular app** setup that demonstrates both a **custom pure pipe** and an **impure pipe**, with a UI to observe the **performance difference**.

**✅ App Overview**

**📦 Features:**

* Custom **pure pipe**: PureFilterPipe
* Custom **impure pipe**: ImpureFilterPipe
* An input box for filtering a large list
* A live counter showing pipe invocations
* Clear visual performance impact

**🛠️ Folder & File Structure**

src/

├── app/

│ ├── pipes/

│ │ ├── pure-filter.pipe.ts

│ │ └── impure-filter.pipe.ts

│ ├── app.component.ts

│ ├── app.component.html

│ └── app.module.ts

**1️⃣ pure-filter.pipe.ts**

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

@Pipe({

name: 'pureFilter',

pure: true

})

export class PureFilterPipe implements PipeTransform {

callCount = 0;

transform(items: string[], search: string): string[] {

this.callCount++;

console.log('Pure Pipe called:', this.callCount);

if (!search) return items;

return items.filter(item => item.toLowerCase().includes(search.toLowerCase()));

}

}

**2️⃣ impure-filter.pipe.ts**

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

@Pipe({

name: 'impureFilter',

pure: false

})

export class ImpureFilterPipe implements PipeTransform {

callCount = 0;

transform(items: string[], search: string): string[] {

this.callCount++;

console.log('Impure Pipe called:', this.callCount);

if (!search) return items;

return items.filter(item => item.toLowerCase().includes(search.toLowerCase()));

}

}

**3️⃣ app.component.ts**

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

@Component({

selector: 'app-root',

templateUrl: './app.component.html'

})

export class AppComponent {

search = '';

items = Array.from({ length: 1000 }, (\_, i) => `Item ${i + 1}`);

// For view refresh triggers

forceChange = 0;

triggerChange() {

this.forceChange = Math.random();

}

}

**4️⃣ app.component.html**

<div style="padding: 20px;">

<h1>Angular Pipes Performance Demo</h1>

<input type="text" [(ngModel)]="search" placeholder="Search..." />

<button (click)="triggerChange()">Trigger Change Detection</button>

<h2>Pure Pipe Result</h2>

<ul>

<li \*ngFor="let item of items | pureFilter:search">{{ item }}</li>

</ul>

<h2>Impure Pipe Result</h2>

<ul>

<li \*ngFor="let item of items | impureFilter:search">{{ item }}</li>

</ul>

</div>

**5️⃣ app.module.ts**

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

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

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

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

import { PureFilterPipe } from './pipes/pure-filter.pipe';

import { ImpureFilterPipe } from './pipes/impure-filter.pipe';

@NgModule({

declarations: [

AppComponent,

PureFilterPipe,

ImpureFilterPipe

],

imports: [

BrowserModule,

FormsModule

],

providers: [],

bootstrap: [AppComponent]

})

export class AppModule { }

**🚀 Run the App**

Make sure you have Angular CLI installed:

npm install -g @angular/cli

Create the project and paste the code:

ng new pipe-demo-app

cd pipe-demo-app

Replace the generated src/app contents with the files above.

Then run:

ng serve

Visit <http://localhost:4200> in your browser.

**👀 What You'll Observe**

* Open DevTools (Console tab)
* Type something in the search box
* Click "Trigger Change Detection"
* **Pure pipe** only logs on input change
* **Impure pipe** logs every time change detection runs (even unrelated)

**✅ Real-World Usage of Impure Pipes in Angular**

While **impure pipes** should generally be avoided due to their performance cost, there **are valid real-world use cases** where an impure pipe makes sense—especially when:

* You want to reflect **dynamic changes** in objects/arrays that do **not change by reference**
* You need to show **live updates** from a mutable object
* You're not using OnPush strategy or immutable data structures

**🔍 Real-World Use Case Examples**

**1️⃣ Filtering a Mutable List in Real Time (Without Changing Reference)**

Suppose you have a **live-updating list** (e.g., chat messages, stock prices, or sensor data) where the array itself stays the same, but its contents change.

**✨ Use Case: Chat Application**

messages = [

{ id: 1, text: 'Hi', seen: false },

{ id: 2, text: 'Hello', seen: true }

];

// message objects are mutated (not replaced)

setTimeout(() => {

this.messages[0].seen = true; // object mutated

}, 2000);

If you use a **pure pipe**, Angular won't re-evaluate the list because the reference hasn't changed. But with an **impure pipe**, Angular will check the values on every change detection cycle.

**2️⃣ Internationalization (i18n) or Localization with Dynamic Language Switching**

In an app where the user can change the language dynamically (e.g., dropdown), and the translation service **mutates data in-place**, an impure pipe can pick up the change automatically.

@Pipe({ name: 'translate', pure: false })

export class TranslatePipe implements PipeTransform {

constructor(private i18nService: I18nService) {}

transform(key: string): string {

return this.i18nService.translate(key); // re-run on each cycle

}

}

**⚠️ Note:**

* A better practice is to **change the keymap reference** on language switch, but impure pipe works here if service doesn’t use immutable patterns.

**3️⃣ Real-Time Dashboard (e.g., IoT Data, Monitoring Tools)**

In apps where data (like CPU usage, temperature, etc.) updates in place without changing object reference, an impure pipe can re-render transformed output without requiring explicit change detection logic.

**4️⃣ Custom Currency Formatter That Depends on User Settings**

Imagine the user can change **locale**, **currency style**, or **number precision** without changing the object being formatted.

@Pipe({ name: 'dynamicCurrency', pure: false })

export class DynamicCurrencyPipe implements PipeTransform {

constructor(private userSettings: UserSettingsService) {}

transform(value: number): string {

const locale = this.userSettings.locale;

const currency = this.userSettings.currency;

return new Intl.NumberFormat(locale, {

style: 'currency',

currency

}).format(value);

}

}

If user changes settings, pipe will automatically reflect new format.

**🧠 When Should You Use an Impure Pipe?**

✅ Use when:

* You're **mutating data** instead of replacing it
* You want a **declarative** template and avoid extra logic in components
* You **accept the performance trade-off** for better DX

🚫 Avoid when:

* You’re using \*ngFor or rendering large data sets
* You can handle it in component or via observables
* You're using OnPush + immutable pattern

Below is a **working Angular demo** that shows how an **impure pipe** reacts to **live-updating object mutations** — something a **pure pipe would not catch**.

**✅ Scenario: Live Scoreboard with Mutable Data**

We'll simulate a sports scoreboard where player scores are updated every second **without changing the array reference**.

**🔧 File Structure**

src/

├── app/

│ ├── pipes/

│ │ └── live-score.pipe.ts

│ ├── app.component.ts

│ ├── app.component.html

│ └── app.module.ts

**1️⃣ live-score.pipe.ts (Impure Pipe)**

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

@Pipe({

name: 'liveScore',

pure: false // impure pipe to reflect mutations

})

export class LiveScorePipe implements PipeTransform {

transform(players: any[], threshold: number): any[] {

// Filter only those with score >= threshold

return players.filter(p => p.score >= threshold);

}

}

**2️⃣ app.component.ts**

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

@Component({

selector: 'app-root',

templateUrl: './app.component.html'

})

export class AppComponent implements OnInit {

players = [

{ name: 'Alice', score: 0 },

{ name: 'Bob', score: 0 },

{ name: 'Charlie', score: 0 }

];

threshold = 0;

ngOnInit() {

// Simulate live score updates (object mutation)

setInterval(() => {

const randomIndex = Math.floor(Math.random() \* this.players.length);

this.players[randomIndex].score += Math.floor(Math.random() \* 10);

}, 1000);

}

}

**3️⃣ app.component.html**

<div style="padding: 20px;">

<h2>🏀 Live Scoreboard (Impure Pipe)</h2>

<label>Score Threshold:

<input type="number" [(ngModel)]="threshold" />

</label>

<ul>

<li \*ngFor="let player of players | liveScore:threshold">

{{ player.name }} - {{ player.score }}

</li>

</ul>

</div>

**4️⃣ app.module.ts**

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

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

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

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

import { LiveScorePipe } from './pipes/live-score.pipe';

@NgModule({

declarations: [

AppComponent,

LiveScorePipe

],

imports: [

BrowserModule,

FormsModule

],

providers: [],

bootstrap: [AppComponent]

})

export class AppModule { }

**🚀 Run It**

1. **Create a new Angular app**:

ng new impure-pipe-demo

cd impure-pipe-demo

1. Replace src/app with the files above.
2. Run:

ng serve

Visit: <http://localhost:4200>

**🔍 What You’ll Observe**

* The players' scores **update live**, even though the array reference never changes.
* The **impure pipe** picks up internal mutations and re-filters.
* Try increasing the **threshold** dynamically — only qualifying players will display.