# Chapter 6: Pipes

Pipes are simple, reusable functions that **transform data in your templates** before displaying it. Instead of changing data in your component, pipes let you format, filter, or manipulate data inline, keeping your templates clean and readable.

## 1. Built-in Pipes: Ready-to-use Transformers

Angular comes with a bunch of useful pipes out of the box.

|  |  |
| --- | --- |
| Pipe | Purpose |
| date | Format dates |
| currency | Format numbers as currency |
| uppercase | Convert string to uppercase |
| lowercase | Convert string to lowercase |
| json | Display object as JSON string |
| async | Subscribe to Observables or Promises |

### Example: Using date and uppercase pipes

<p>Today's date: {{ today | date:'fullDate' }}</p>

<p>User name: {{ userName | uppercase }}</p>

### Simple built-in pipe example

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

@Component({

standalone: true,

selector: 'app-pipe-demo',

template: `

<h2>Built-in Pipes Demo</h2>

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

<p>Price: {{ price | currency:'USD':'symbol' }}</p>

<p>Username (uppercase): {{ userName | uppercase }}</p>

`

})

export class PipeDemoComponent {

today = new Date();

price = 1234.56;

userName = 'angularUser';

}

### date Pipe

Transforms Date objects into human-readable strings.

{{ today | date:'shortDate' }}

|  |  |
| --- | --- |
| Format | Output (e.g., June 21, 2025 at 3:00 PM) |
| 'short' | 6/21/25, 3:00 PM |
| 'medium' | Jun 21, 2025, 3:00:00 PM |
| 'long' | June 21, 2025 at 3:00:00 PM GMT+5:30 |
| 'fullDate' | Saturday, June 21, 2025 |
| 'shortTime' | 3:00 PM |
| 'mediumDate' | Jun 21, 2025 |
| Custom Format | 'dd/MM/yyyy' → 21/06/2025 |

**Tip**: Use custom format strings like 'yyyy-MM-dd HH:mm:ss' for full control.

### currency Pipe

Formats a number as currency.

{{ amount | currency:'INR':'symbol':'1.2-2' }}

|  |  |
| --- | --- |
| Argument | Purpose |
| 'USD', 'EUR', 'INR' | Currency code |
| 'symbol' | $, ₹, €, etc. |
| 'symbol-narrow' | Shorter version, like $ or ₹ |
| 'code' | Shows USD, INR, etc. |
| '1.2-2' | Min 1 digit, 2 min/2 max decimal digits |

Example:

{{ 99.5 | currency:'EUR':'symbol':'1.0-0' }} → €100

### number Pipe

Formats numbers (without currency symbol).

{{ 1234.56 | number:'1.2-2' }} → 1,234.56

|  |  |
| --- | --- |
| Format | Description |
| '1.0-0' | No decimals |
| '1.2-2' | 2 decimal places |
| '3.1-4' | Pad with leading digits if needed |

### percent Pipe

Formats numbers as percentages.

{{ 0.35 | percent }} → 35%

{{ 0.75 | percent:'1.0-1' }} → 75.0%

### uppercase and lowercase

Transform case of text.

{{ 'Angular' | uppercase }} → ANGULAR

{{ 'Angular' | lowercase }} → angular

### slice Pipe

Returns a slice of an array or string.

{{ 'AngularPipes' | slice:0:7 }} → Angular

{{ items | slice:1:3 }} → second and third item

### json Pipe

Displays an object as formatted JSON.

<pre>{{ user | json }}</pre>

### async Pipe

Subscribes to Observables or Promises and returns the latest value.

<p>{{ user$ | async }}</p>

* Automatically unsubscribes on destroy.
* Works with BehaviorSubject, Observable, or Promise.

## 2. Custom Pipes: Your Own Data Transformers

Sometimes, you need a transformation that Angular doesn’t provide. That’s when **custom pipes** shine.

### What’s a Custom Pipe?

A class with a @Pipe() decorator and a transform() method where you put your logic.

### When to create one?

* Formatting phone numbers
* Filtering or sorting arrays
* Translating text
* Truncating long strings

### How to create a custom pipe?

#### CLI command:

ng generate pipe yourPipeName --standalone

Example:

ng generate pipe truncate --standalone

### Example: Truncate pipe (shorten text with “...”)

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

@Pipe({

standalone: true,

name: 'truncate'

})

export class TruncatePipe implements PipeTransform {

transform(value: string, limit = 20): string {

if (!value) return '';

return value.length > limit ? value.substring(0, limit) + '...' : value;

}

}

### Using your custom pipe

<p>{{ longText | truncate:10 }}</p>

### Complete example you can try:

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

import { TruncatePipe } from './truncate.pipe';

@Component({

standalone: true,

selector: 'app-truncate-pipe-demo',

template: `

<h2>Custom Pipe: Truncate</h2>

<p>Original: {{ text }}</p>

<p>Truncated (10 chars): {{ text | truncate:10 }}</p>

`,

imports: [TruncatePipe]

})

export class TruncatePipeDemoComponent {

text = 'This is a very long text that needs to be shortened.';

}

## 3. Pure vs Impure Pipes

* **Pure pipes** run only when the input changes (efficient, preferred).
* **Impure pipes** run every change detection cycle (use cautiously, e.g., for async data).

## 4. Component Architecture Best Practices

### Folder Structure Suggestions

Organize by feature for clarity and scalability:

/src/app

/feature1

feature1.component.ts

feature1.service.ts

feature1.module.ts (optional)

/shared

components/

directives/

pipes/

### Shared Modules and Imports

Since Angular 19 encourages standalone components, create shared folders with reusable standalone components, directives, and pipes, then import them where needed.

### Reusable Standalone Components

Design components so they don’t depend on specific parents and can be dropped anywhere, increasing reusability.

### Using inject() Over Constructor DI

Angular 19 encourages using inject() inside functional or class components to get dependencies, simplifying tests and reducing boilerplate.

Example:

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

import { SomeService } from './some.service';

export class MyComponent {

service = inject(SomeService);

}

## 6. Content Projection: <ng-content>

Allows you to pass markup into components, similar to “slots” or “children” in React.

### Basic usage

<my-card>

<p>This content goes inside the card!</p>

</my-card>

my-card component template:

<div class="card">

<ng-content></ng-content>

</div>

### Named slots

<my-card>

<h1 ng-content-select="header"></h1>

<p ng-content-select="body"></p>

</my-card>

### Complete content projection example:

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

@Component({

standalone: true,

selector: 'app-card',

template: `

<div class="card">

<div class="card-header">

<ng-content select="[header]"></ng-content>

</div>

<div class="card-body">

<ng-content select="[body]"></ng-content>

</div>

<div class="card-footer">

<ng-content select="[footer]"></ng-content>

</div>

</div>

`,

styles: [`

.card { border: 1px solid #ddd; padding: 1rem; border-radius: 5px; }

.card-header { font-weight: bold; }

.card-footer { font-size: small; color: gray; }

`]

})

export class CardComponent {}

Usage:

<app-card>

<h2 header>Title Here</h2>

<p body>Body content goes here...</p>

<small footer>Footer text</small>

</app-card>