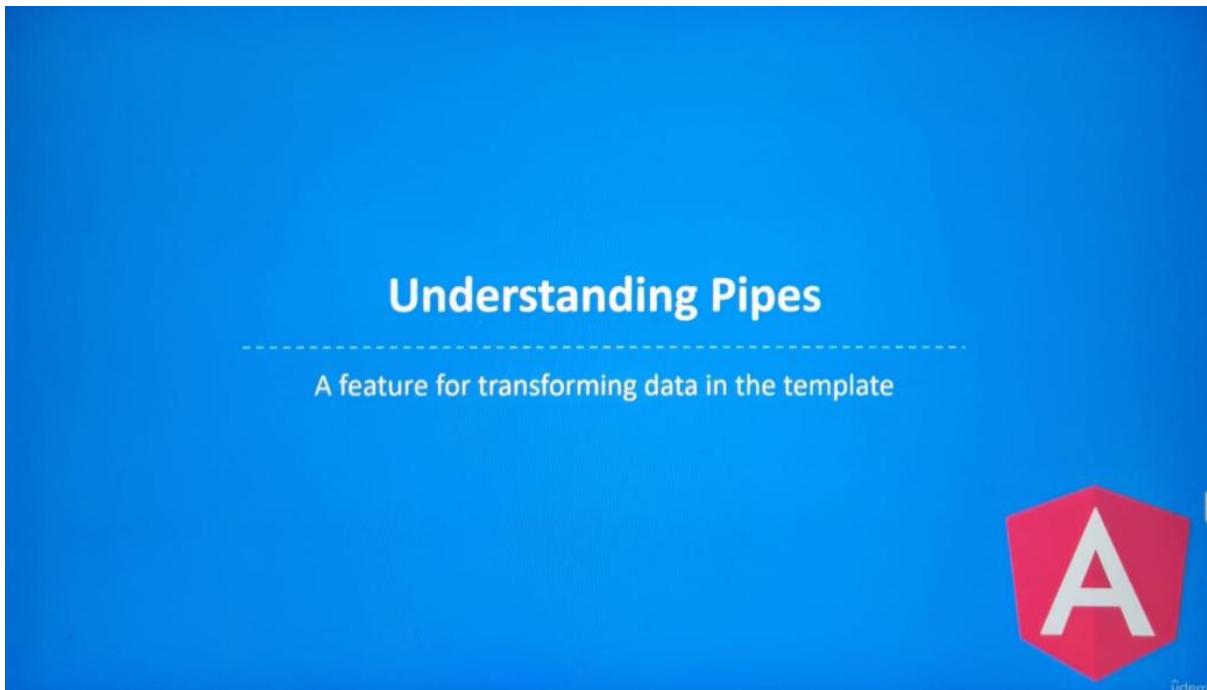


Understanding Pipes

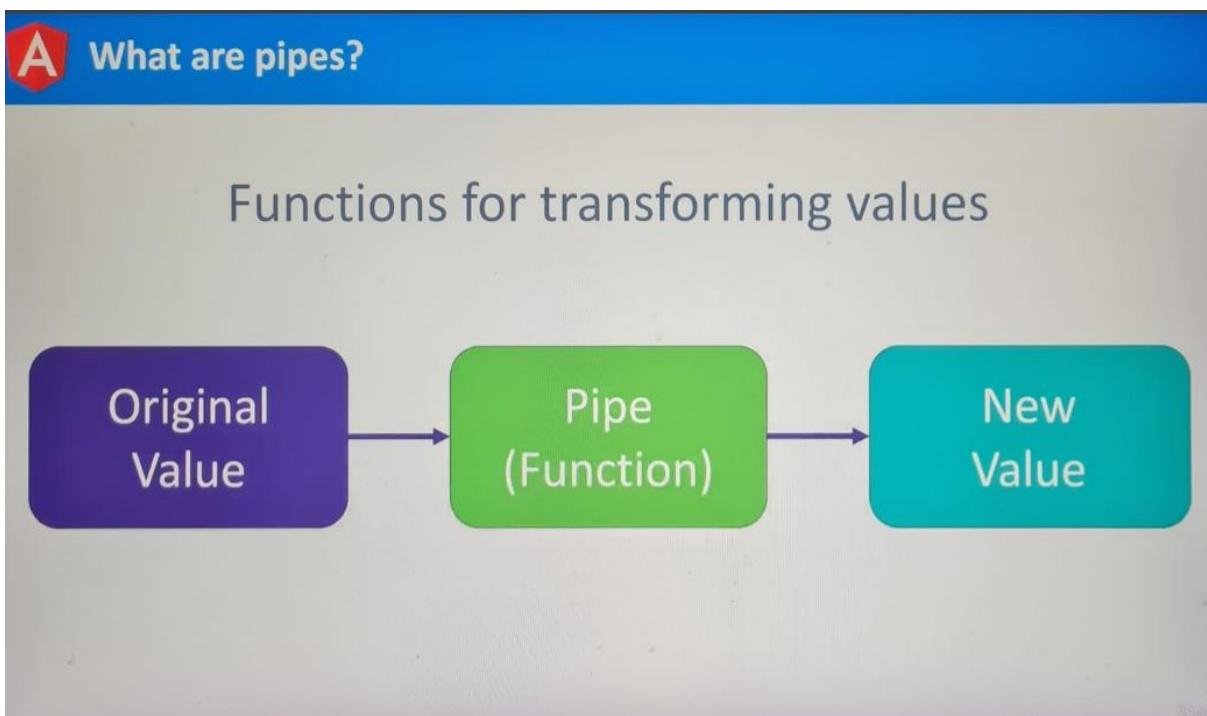


In this section, we are going to start focusing on transforming content.

Angular provides two features for helping us transform content:

1. Pipes
2. Directives

We will start with **Pipes**. They're the simplest to learn.



A Pipe is a function for transforming a value in the template. We pass in a value to a pipe function. The pipe will return a new value. This value will be rendered in the templates.

You may be thinking, how is this different from defining a method in a class?

Pipes are not bound to a specific component. We can define a pipe once, use it everywhere.

If we are creating functions for changing a value in the templates, we should use pipes.

It removes clutter from our component class. This way, our component can focus on handling other logic. Another feature worth noting about pipes is that they don't change the original value. The purpose of a pipe is to transform the value for the templates. The original value will remain the same in the class. It can be convenient to use a pipe instead of the alternative.

For example, let's say we are storing the price of a product. We may want to store this value as a number. However, we may want to prefix the price with a currency symbol in the templates. Even though it's one character, we would need to change the property type to a string. Currency symbols are not supported in the number type. One solution would be to create two properties to store the price. However, this duplication can clutter our code. Instead of storing the same property twice, we can create a pipe for adding the currency symbol in the templates. The property would remain a number in the class.

Pipes are helpful because they help us change the appearance of a value for the template without changing the original value.

Pipes Basic

In this lecture, we're going to start exploring some **pipes**. Angular comes with a couple of **pipes built into the framework**. We have the option of **creating custom pipes** too.

We will start with **Angular Pipes**. They should cover most cases.

List of pipes offered by Angular: <https://v17.angular.io/api?type=pipe>

<https://angular.dev/guide/templates/pipes#>

Above the list of **pipes**, Angular will tell us which package contains the **pipes**:



Built-in Pipes

Angular includes a set of built-in pipes in the `@angular/common` package:

Name	Description
<code>AsyncPipe</code>	Read the value from a <code>Promise</code> or an RxJS <code>Observable</code> .
<code>CurrencyPipe</code>	Transforms a number to a currency string, formatted according to locale rules.
<code>DatePipe</code>	Formats a <code>Date</code> value according to locale rules.
<code>DecimalPipe</code>	Transforms a number into a string with a decimal point, formatted according to locale rules.
<code>I18nPluralPipe</code>	Maps a value to a string that pluralizes the value according to locale rules.
<code>I18nSelectPipe</code>	Maps a key to a custom selector that returns a desired value.
<code>JsonPipe</code>	Transforms an object to a string representation via <code>JSON.stringify</code> , intended for debugging.
<code>KeyValuePipe</code>	Transforms Object or Map into an array of key value pairs.
<code>LowerCasePipe</code>	Transforms text to all lower case.
<code>PercentPipe</code>	Transforms a number to a percentage string, formatted according to locale rules.
<code>SlicePipe</code>	Creates a new Array or String containing a subset (slice) of the elements.
<code>TitleCasePipe</code>	Transforms text to title case.
<code>UpperCasePipe</code>	Transforms text to all upper case.

By default, the **common package** is **not imported** into Angular project. If we want **pipes** to be available in our app, we need to **register this package**. According to this description, we can register this package through a module.

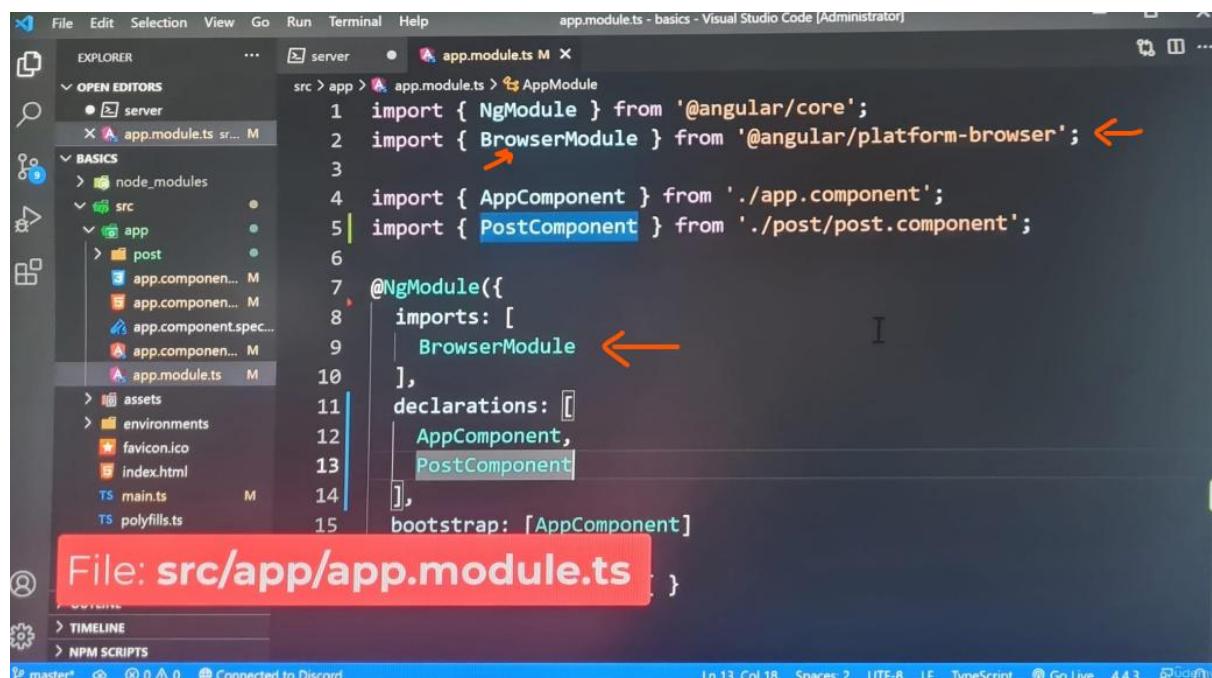
These **pipes** are exported from two modules:

1. CommonModule
2. BrowserModule

However, reading further (In Documentation), the description tells us, these modules are registered for new projects.

Let's check out the app.module.ts [src/app/app.module.ts]:

Note: Since I am using the standalone architecture, I won't be having this file.



```

File: src/app/app.module.ts
  1 import { NgModule } from '@angular/core';
  2 import { BrowserModule } from '@angular/platform-browser'; ←
  3
  4 import { AppComponent } from './app.component';
  5 import { PostComponent } from './post/post.component';
  6
  7 @NgModule({
  8   imports: [
  9     BrowserModule ←
 10   ],
 11   declarations: [
 12     AppComponent,
 13     PostComponent
 14   ],
 15   bootstrap: [AppComponent]
}

```

As we can see, the **BrowserModule** has been registered. Angular's **pipes** are readily available in our components.

Let's use our first **pipe**. It won't be tricky at all.

The syntax for running a value through a pipe is simple.

The first **pipe** we will learn is called **TitleCasePipe**.

TitleCasePipe will capitalize the first letter of each word in a string. Any other character will be lowercase. The goal of the **TitleCasePipe** is to output a title. It works for names too. This pipe can be helpful for providing consistent formatting for titles and names.

Let's apply this **pipe** to our name in the app component:

Inside this class, we are going to convert the letters in the name to **lowercase letters**. It will give us an opportunity to test if the **pipe** is working:

TS app.ts 1, M X

```

basics > src > app > TS app.ts > App > name
1 import { Component } from '@angular/core';
2 import { RouterOutlet } from '@angular/router';
3 import { Post } from './post/post';
4
5 @Component({
6   selector: 'app-root',
7   imports: [RouterOutlet, Post],
8   templateUrl: './app.html',
9   styleUrls: ['./app.css'],
10 })
11 export class App {
12   protected title = 'basics';
13
14   protected name = 'daniel kandalaft'; ← lowercase
15   protected imageURL = 'https://picsum.photos/id/237/500/500';
16
17   getName() {
18     return this.name;
19   }
20
21   changeImage(e: KeyboardEvent) {
22     this.imageURL = (e.target as HTMLInputElement).value;
23   }
24
25   logImg(event: string) {
26     console.log(event);
27   }
28 }
29

```

Next, let's open the app template file:

Inside the first paragraph tag, we are going to update the expression. We can apply a **pipe** by adding the **pipe character 'J'**, for this example, we will apply the **titleCasePipe → | titlecase**.

Pipes are applied after the expression has been evaluated. Whenever a value gets evaluated from the expression or we passed onto the pipe, the pipe will return a formatted value. The formatted value will be outputted onto the page.

```

<input (keyup)="changeImage($event)" [value]="imgURL" />
<app-post [img]="imgURL" (imgSelected)="logImg($event)">
    <p>Some caption</p>
</app-post>
<p>Hello {{ name | titlecase }}</p>
<p>Hello {{ getName() }}</p>
<p>{{ 15 + 13 }}</p>

```

Title Case Pipe

```

<app-post [img]="imgURL" (imgSelected)="logImg($event)">
    <p>Some caption</p>
</app-post>
<p>Hello {{ name | titlecase }}</p>
<p>Hello {{ getName() }}</p>
<p>{{ 15 + 13 }}</p>

```

No pipe found with name 'titlecase'. ngtsc(-998004)
app.ts(8, 28): Error occurs in the template of component App.

View Problem (Alt+F8) No quick fixes available

Maybe because I don't have module.ts

I don't have app.module.ts on my new Angular 20.0.5, since I am using the standalone architecture as recommended, as a result, it seems that **BrowserModule** is not registered.

ChatGPT Solution:

I am using Angular 20.0.5 with standalone architecture, No pipe found with name 'titlecase'.ngtsc(-998004)

ChatGPT said:

Great question! You're using **Angular 20.0.5 with the standalone architecture**, and you got this error:

```

pgsql
CopyEdit
No pipe found with name 'titlecase'.ngtsc(-998004)

```

✓ Why this error happens

In **standalone components, pipes and directives are not available globally** like they are in module-based Angular. You must **explicitly import** them in the `imports` array of the component that uses them.

The `titlecase` pipe is part of **Angular's `CommonModule`**, which must be imported to use built-in pipes like:

- `titlecase`
 - `uppercase`
 - `lowercase`
 - `ngIf`
 - `ngFor`
 - etc.
-

Solution

Import `CommonModule` in your component:

```
ts
CopyEdit
import { Component } from '@angular/core';
import { CommonModule } from '@angular/common';

@Component({
  selector: 'app-post',
  standalone: true,
  imports: [CommonModule], // 👈 add this
  template: `
    <p>{{ 'hello world' | titlecase }}</p>
  `
})
export class Post {}
```

Summary

Error

No pipe found with name
'`titlecase`'

Fix

Import `CommonModule` in the component's `imports` array

My Solution to the problem:

The screenshot shows a code editor window with the file `app.ts` open. The code is written in TypeScript and defines an `App` component. The imports section includes `@Component`, `CommonModule`, `RouterOutlet`, and `Post`. The `@Component` decorator specifies the selector as `'app-root'`, imports `RouterOutlet`, `Post`, and `CommonModule`, and provides the template URL as `'./app.html'` and style URL as `'./app.css'`. The `App` class has protected properties `title`, `name`, and `imgURL`, and methods `getName`, `changeImage`, and `logImg`. Two red annotations are present: a red arrow points to the `CommonModule` import at line 2, and a red arrow points to the `RouterOutlet` import at line 8.

```
TS app.ts 1, M X
basics > src > app > TS app.ts > ...
1 import { Component } from '@angular/core';
2 import { CommonModule } from '@angular/common'; ←
3 import { RouterOutlet } from '@angular/router';
4 import { Post } from './post/post';
5
6 @Component({
7   selector: 'app-root',
8   imports: [RouterOutlet, Post, CommonModule], ↓
9   templateUrl: './app.html',
10  styleUrls: ['./app.css'],
11})
12 export class App {
13   protected title = 'basics';
14
15   protected name = 'daniel kandalaft';
16   protected imgURL = 'https://picsum.photos/id/237/500/500';
17
18   getName() {
19     return this.name;
20   }
21
22   changeImage(e: KeyboardEvent) {
23     this.imgURL = (e.target as HTMLInputElement).value;
24   }
25
26   logImg(event: string) {
27     console.log(event);
28   }
29 }
30
```

app.html M X

basics > src > app > app.html > ...

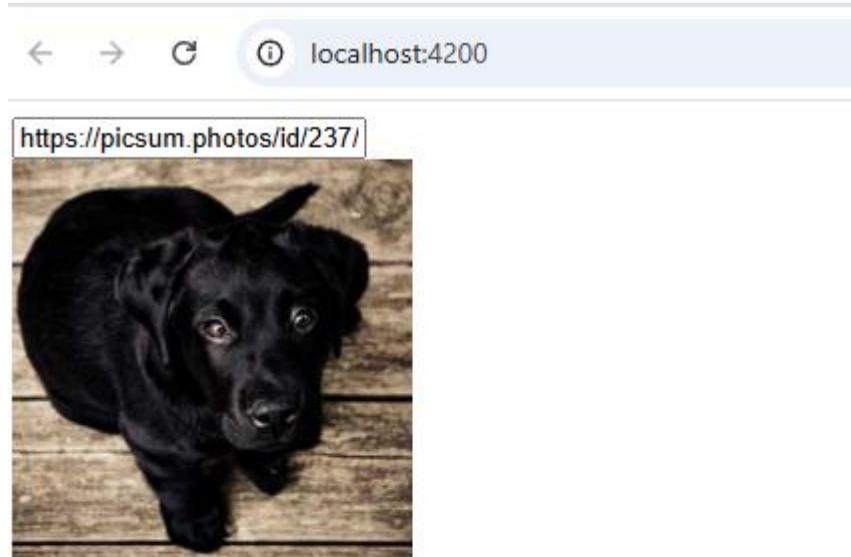
Go to component

```

1  <input (keyup)="changeImage($event)" [value]="imgURL" />
2
3  <app-post [img]="imgURL" (imgSelected)="logImg($event)">
4    <p>Some caption</p>
5  </app-post>
6  <p>Hello {{ name | titlecase }}</p>
7  <p>Hello {{ getName() }}</p>
8  <p>{{ 15 + 13 }}</p>
9
10

```

No problem



Some caption

Hello Daniel Kandalaft

Hello daniel kandalaft

28

The name has been properly title cased, perfect.

You may notice the name below hasn't been changed. If you can recall, the second expression will use a method returning the property. **Pipes do not directly modify properties.**

This is very important to remember, the purpose of a pipe is to transform the outputs. The value in the class will remain unchanged. Pipes are incredibly helpful for formatting a value before rendering it on the page. The value for the name property will always be lowercase, even after running it through a pipe.