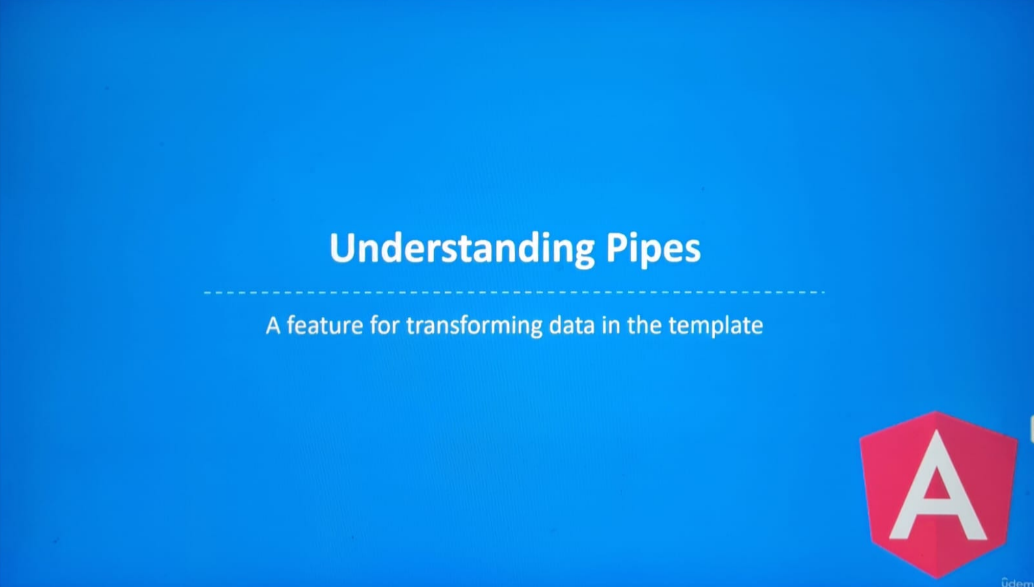
Understanding Pipes

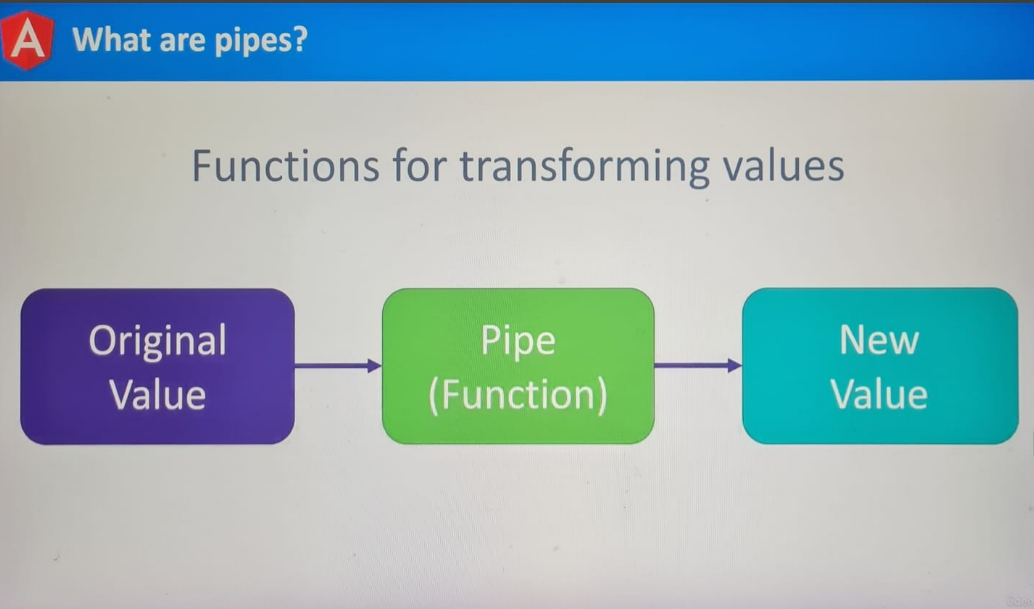


In this section, we are going to start focusing on transforming content.

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

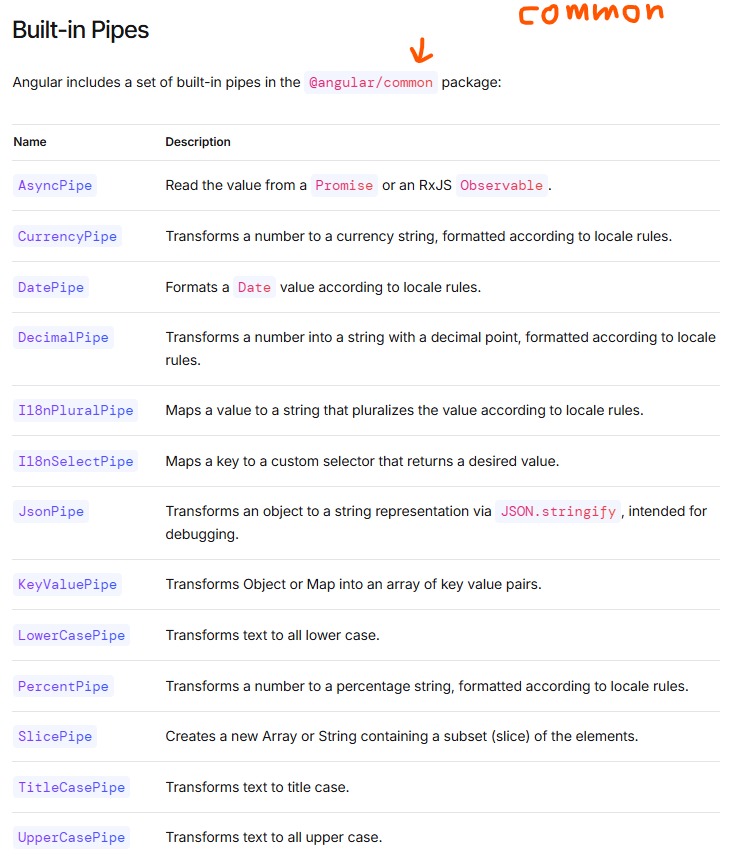
In this lecture, we’re going to start exploring some **pipes**. Angular comes with a couple of **pipes built into the framework**. We have the option of **creating custom pipes** too.

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#](https://angular.dev/guide/templates/pipes)

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



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.

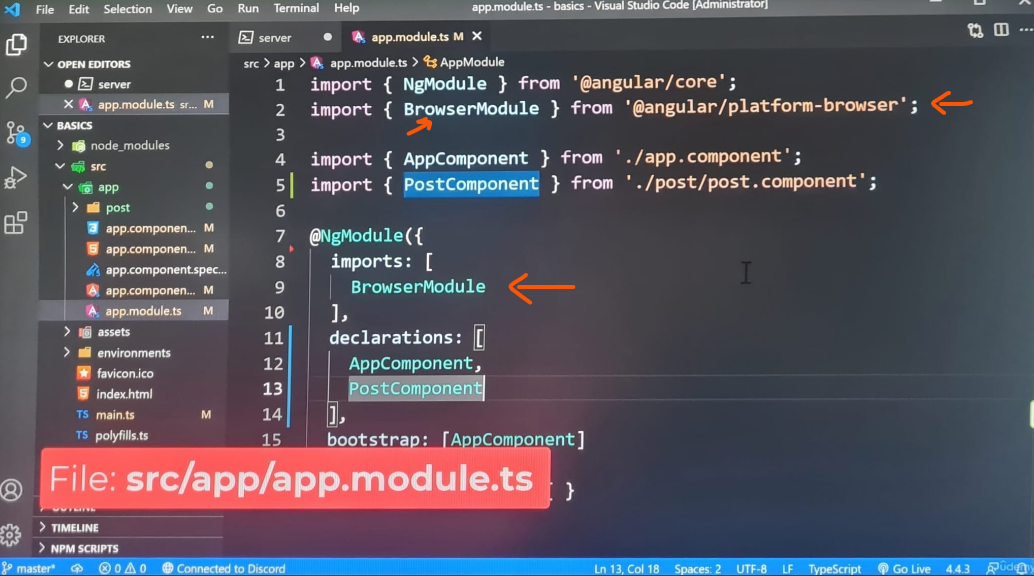
These **pipes are exported from two modules:**

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.**



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 **TitlCasePipe** 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:

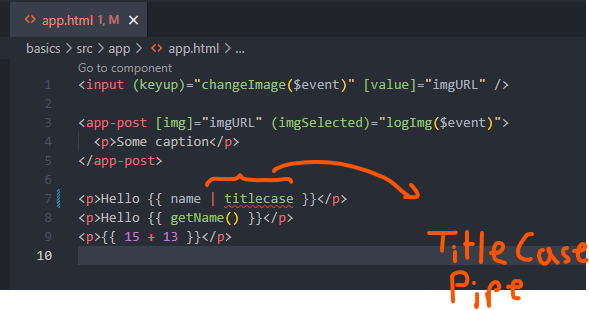
Inside this class, we are going to convert the letters in the name to **lowercase letters**. It will give us an opportunity to test if the **pipe** is working:

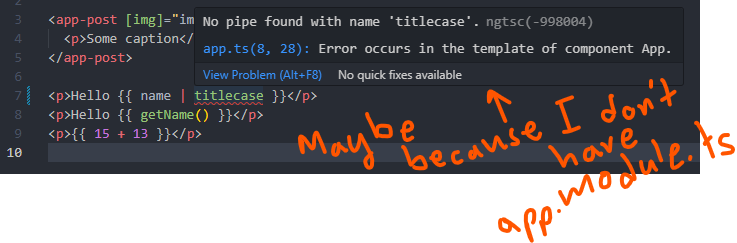


Next, let’s open the app template file:

Inside the first paragraph tag, we are going to update the expression. We can apply a **pipe** by adding the **pipe character ‘|’** , 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.**





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>

`

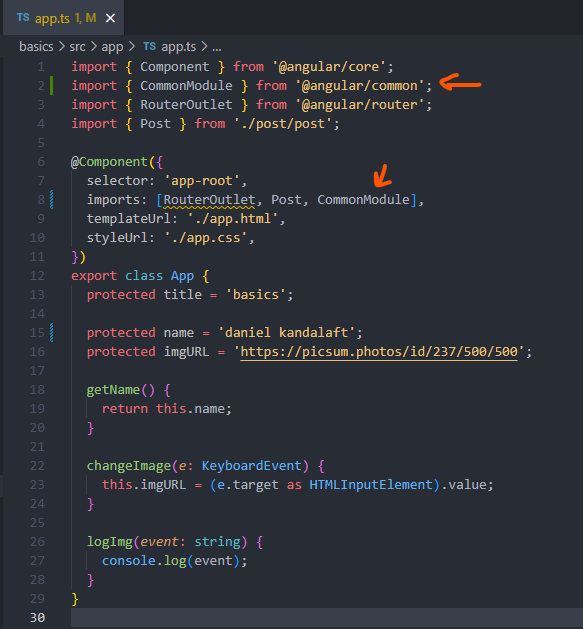
})

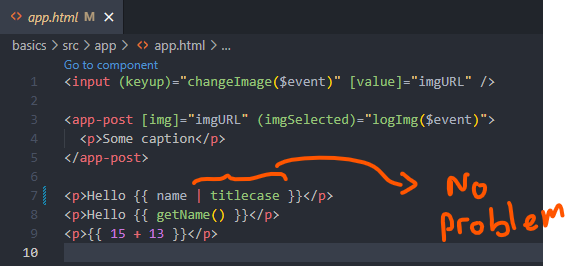
export class Post {}

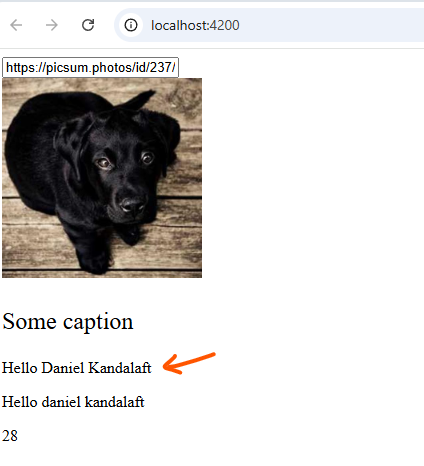
**✅ Summary**

| **Error** | **Fix** |
| --- | --- |
| No pipe found with name 'titlecase' | Import CommonModule in the component’s imports array |

**My Solution to the problem:**

****





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.**