**Pipes Overview**

We'll create a new project and get a basic overview of pipes.

**We'll cover the following**

* [Creating a new project](https://www.educative.io/module/page/El5jyzfkAngPpgpAB/10370001/5017901150502912/6661986949005312#Creating-a-new-project)
* [What are pipes?](https://www.educative.io/module/page/El5jyzfkAngPpgpAB/10370001/5017901150502912/6661986949005312#What-are-pipes)

The next thing we’ll be focusing on are **pipes**. In this section, you’ll learn what pipes are. Before we get started, we’ll be working with a new project.

**Creating a new project**

If you’re running code locally, then you’ll need to run the following command:

ng new pipes

During the setup process, you’ll be asked to configure the project. Go with the default settings.

## What are pipes?

A pipe is a function for formatting a value. It’s a feature in templates that we can use to modify the output of an expression. It’s possible to use pipes outside of a template, but it’s rare to do so.

Angular comes with some built-in pipes. You can view the list of pipes available in the documentation: <https://angular.io/api?type=pipe>.

It’s also possible to develop custom pipes. We’ll be looking at both built-in and custom pipes.

**Pipe Basics**

Learn about the basics of pipes.

**We'll cover the following**

* [TitleCasePipe](https://www.educative.io/module/page/El5jyzfkAngPpgpAB/10370001/5017901150502912/5084777482289152#TitleCasePipe)

We’ll start things off by learning how to use pipes. Using pipes isn’t tricky at all. We’ll be up and running in less than a few minutes.

Angular comes with some pipes by default. The team has developed pipes for the most common situations. A full list of pipes provided by Angular can be found [here](https://angular.io/api?type=pipe).

**TitleCasePipe**

The first pipe we’ll use is the TitleCasePipe. This pipe will capitalize the first letter of each word in a string. Any other characters are lowercase. The goal of the TitleCasePipe is to output a title. It also works very well for names.

This can come in handy when you have an input where the user may forget to capitalize some words or names.

In the app.component.ts file, we’ll add a new property called name, to the class.

export class AppComponent {  
  name = 'john doe';  
}

The value of the name property is a string in all lowercase letters. Typically, the value may come from a database or input. For demonstration purposes, we’re going to keep things as simple as possible. We’ll hardcode the values ourselves.

In the app.component.html template file, we’ll clear out the default template and replace it with the following:

<p>{{ name | titlecase }}</p>

The output will be John Doe. A pipe can be inserted using the pipe character (|). This may depend on the layout of your keyboard, but the pipe character can typically be found by holding shift and pressing the key above the enter key. This is followed up by the name of the pipe we’d like to use.

Angular will run the expression first. The value from the expression will be passed to the pipe, which is a function. The pipe will return the formatted value, and we get the output.

The main purpose of a pipe is to help you transform output without having to modify the property in the class. The value for the name property will **always** be john doe, even after running it through a pipe. Pipes will simply transform the output in the template.

**Pipe Parameters**

We'll learn how to use pipe parameters in this lesson.

**We'll cover the following**

* [Using pipe parameters](https://www.educative.io/module/page/El5jyzfkAngPpgpAB/10370001/5017901150502912/5145365512192000#Using-pipe-parameters)
* [Predefined date formats](https://www.educative.io/module/page/El5jyzfkAngPpgpAB/10370001/5017901150502912/5145365512192000#Predefined-date-formats)

We’ll look at another pipe defined by Angular called the DatePipe. The DatePipe will help you format dates into various formats. It’s a versatile pipe because we have the freedom to use different formats.

The DatePipe requires the date to be a Date() object, time in milliseconds from 1970, or an ISO string. We must make sure the value is in the required format before we use the DatePipe. Otherwise, it won’t work.

In the app.component.ts file, we’ll create a property called todaysDate. Its value will be a new instance of the Date object.

export class AppComponent {  
  name = 'john doe';  
  todaysDate = new Date();  
}

In the app.component.html template, we’ll add a <p> tag with the following content:

<p>{{ todaysDate | date }}</p>

By default, this will output the following:

Apr 26, 2020

The output may be slightly different because the date pipe reacts differently based on your environment or when you’re taking this course.

**Using pipe parameters**

By itself, the DatePipe is capable of formatting the date without our help. It was able to understand the current date because we provided it with an instance of the Date object.

In some cases, we may want the date in a specific format. Since **pipes are functions**, they have parameters for modifying the behavior of the function. Parameters can be added by adding a colon after the name of the pipe, followed by the value.

The date pipe’s parameter is the format we wish to use for the date. On the documentation page for the date pipe, there’s a section called **Custom format options**. Here’s a link to the page: <https://angular.io/api/common/DatePipe#custom-format-options>.

Under the format column, we can use the characters as placeholders for the actual value. The date pipe will take care of replacing the placeholders with the appropriate value. For example, let’s say we want to output only the month and day. We could update the pipe in the template to the following:

<p>{{ todaysDate | date: 'MMMM d' }}</p>

This will output the following:

April 26

This may be different for some of you, depending on when you’re taking this course.

**Predefined date formats**

One last thing, in the documentation for the date pipe, there’s a section called **Predefined format options**. If you don’t feel like setting the format yourself, you can use a predefined set of format options. For example, let’s say we want to use the short format. We could update the expression to the following:

<p>{{ todaysDate | date: 'short' }}</p>

The pipe will still be able to transform the output even though we’re not passing in a format with placeholders. It will notice we want to use one of its inner predefined format options.

# Currency Pipe and Decimal Pipe

We'll learn how to use the currency pipe and decimal pipe in this lesson.

**We'll cover the following**

* [Currency pipe](https://www.educative.io/module/page/El5jyzfkAngPpgpAB/10370001/5017901150502912/6136757696856064#Currency-pipe)
  + [Switching currencies](https://www.educative.io/module/page/El5jyzfkAngPpgpAB/10370001/5017901150502912/6136757696856064#Switching-currencies)
* [Decimal pipe](https://www.educative.io/module/page/El5jyzfkAngPpgpAB/10370001/5017901150502912/6136757696856064#Decimal-pipe)

We’ll be exploring two pipes in this lesson: the currency pipe, and the decimal pipe. Both are used for formatting numeric values.

## Currency pipe

The currency pipe will format a number by adding a currency symbol before it and adding decimal values if necessary. In the app.component.ts component class file, we’ll add a property called cost.

export class AppComponent {  
  name = 'john doe';  
  todaysDate = new Date();  
  cost = 2000;  
}

This will hold the cost of some imaginary product. We can update the app.component.html template to output the price with the correct currency symbol.

<p>{{ cost | currency }}</p>

This will output the following:

$2,000.00

You may see a different currency because the currency pipe will check the OS and language of your system. It will attempt to output the currency associated with your region.

As you can see, the currency pipe was able to add the correct currency symbol, add a comma, and add a decimal value.

### Switching currencies

The currency pipe has a parameter for changing the currency. The documentation for the currency pipe states that we can use an ISO 4217 code. Here’s a list of valid codes: <https://en.wikipedia.org/wiki/ISO_4217>.

Let’s say we want to switch the currency to the Japanese Yen; we can update the expression to the following:

<p>{{ cost | currency: 'JPY' }}</p>

This will output the following:

¥2,000

Perfect! One thing to keep in mind is that the currency pipe will not perform a conversion on your value. It will merely format it. It’s essential to perform a currency conversion before outputting it. 2000 Yen is not equal to 2000 USD.

## Decimal pipe

The next pipe we’ll look at is the DecimalPipe. It’s very similar to the CurrencyPipe because it deals with numeric values. We can use it to help format numbers by adding/removing decimal values.

For example, let’s say we want to output the temperature. Sometimes we may need to calculate the temperature, which may result in a decimal value. Typically, it’s not common to output the temperature with a decimal value. We want to strip it out if that’s the case.

In the app.component.ts component class file, we’ll add a property called temperature.

export class AppComponent {  
  name = 'john doe';  
  todaysDate = new Date();  
  cost = 2000;  
  temperature = 25.3;  
}

We want to remove the .3 from the value. We can do this in JavaScript, but, for demonstration purposes, we’ll look at how to remove it with the DecimalPipe.

In the app.component.html template, we’ll add the following:

<p>{{ temperature | number: '1.0-0' }}</p>

This will output the following:

25

The number pipe has a parameter, which is the format for the number. On the left side of the dot is the minimum number of digits that should be present in the number. For example, let’s say we updated the parameter to 5.0-0; the output will be the following:

00,025

The number pipe will add 0’s to the number because there are less than five digits in the number.

To the right of the dot, we can set the minimum and the maximum number of digits for the decimal value, respectively. In our example, we’re saying the number can’t have any decimal values. The number pipe will strip out any decimal values present in the number.

Let’s say we updated the parameter to 1.2-5 This will tell the number pipe to output a decimal value with at least two digits. If the value has more than five digits for the decimal value, they will be stripped out.

For example, if we had the temperature property set to 25.123456, then the output would be the following:

25.12345

The goal for this part of the lesson was to output the temperature without decimal values, so we’ll revert the parameter to 1.0-0.

# JSON Pipe

In this lesson, we'll learn how to use the JSON pipe.

There’s one last pipe I want to go over before we dive into creating custom pipes. Angular has a pipe called the JsonPipe. Here’s the link to the documentation: <https://angular.io/api/common/JsonPipe>.

According to the description, it’s useful for debugging. Sometimes you will want to see an object output on the template to inspect it.

Let’s create a property, called pizza, in the app.component.ts component class file.

export class AppComponent {  
  name = 'john doe';  
  todaysDate = new Date();  
  cost = 2000;  
  temperature = 25.3;  
  pizza = {  
    toppings: ['pepperoni', 'mushroom'],  
    size: 'Large'  
  };  
}

It will be an object with two properties: toppings and size. Let’s try to output the object.

<pre>{{ pizza }}</pre>

This will not work. We’ll end up with the following output:

[object Object]

If we want to output an object, we’ll need to use the json pipe, as shown here:

<pre>{{ pizza | json }}</pre>

This will change the output to the following:

{  
  "toppings": [  
    "pepperoni",  
    "mushroom"  
  ],  
  "size": "Large"  
}

The json pipe is convenient for debugging an object. It doesn’t serve much purpose outside of this.

**Custom Pipes**

We'll learn how to create custom pipes in this lesson.

**We'll cover the following**

* [Creating a pipe](https://www.educative.io/module/page/El5jyzfkAngPpgpAB/10370001/5017901150502912/4835738903904256#Creating-a-pipe)
* [Reviewing the pipe file](https://www.educative.io/module/page/El5jyzfkAngPpgpAB/10370001/5017901150502912/4835738903904256#Reviewing-the-pipe-file)
* [Doubling a number](https://www.educative.io/module/page/El5jyzfkAngPpgpAB/10370001/5017901150502912/4835738903904256#Doubling-a-number)
* [Adding a parameter](https://www.educative.io/module/page/El5jyzfkAngPpgpAB/10370001/5017901150502912/4835738903904256#Adding-a-parameter)

It’s time to create a custom pipe. If none of the pipes defined by Angular are suitable, then a custom pipe might be the solution you’re looking for.

**Creating a pipe**

The angular CLI has a command for helping us generate the necessary files and code for a pipe. In the command line, run the following command:

ng generate pipe double

The ng generate pipe <name> command will create two files in the src/app directory: double.pipe.spec.ts and double.pipe.ts. The double.pipe.spec.ts file is for testing the pipe. The double.pipe.ts file is where we’ll write the logic for the pipe.

**Note:** The app.module.ts file was modified. We’ll be ignoring this change. We’ll review modules in another section.

**Reviewing the pipe file**

Here are the contents of the double.pipe.ts file:

1

2

3

4

5

6

7

8

9

10

11

12

13

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

@Pipe({

  name: 'double'

})

export class DoublePipe implements PipeTransform {

  transform(value: unknown, ...args: unknown[]): unknown {

    return null;

  }

}

At the top of the file, we’re importing the Pipe decorator and the PipeTransform interface.

The job of the @Pipe decorator is to help Angular identify the class being exported as a pipe.

The job of the PipeTransform interface is to help us properly write a pipe. It will ensure the class we’re writing has a method, called transform(), and that it’s returning a value. While not required, it is helpful for debugging. We’ll be able to catch an error with our class before running it in the browser.

In the class, we have a method called transform(). This is where we can transform the value. There are two parameters. The value parameter is the value to format, and the args parameter is an array of pipe parameters. In the method, we’re returning a value. It’s required that we return a value since pipes are supposed to transform a value.

**Doubling a number**

The pipe we’ll create will double a number. We’ll update the transform() method to the following:

transform(value: number, ...args: unknown[]): number {  
  return value \* 2;  
}

We’re updating the value parameter to accept only number types. We’re also saying the value returned by the method will be the number type. In the function, we’re multiplying the value parameter by 2.

The pipe is ready. We can use it in our template like any other pipe. We’re going to apply the pipe to the cost property. In the app.component.html template, we’ll add the following:

<p>{{ cost | double }}</p>

This will output 4000.

**Adding a parameter**

We have a pretty straightforward pipe. Let’s make it more dynamic by adding some parameters.

Back in the double.pipe.ts file, the transform() method has a second parameter called args. It will represent an array of parameters in the pipe. We can have an unlimited number of parameters. Usually, we’ll want to update the type annotation of this parameter because unknown doesn’t help with debugging.

We’ll create two parameters. The first parameter will determine how many times the number should be doubled. The second parameter will allow the developer to flip the number from positive to negative.

We’ll update the transform() method to the following:

1

2

3

4

5

6

7

8

9

10

11

12

13

14

15

export class DoublePipe implements PipeTransform {

  transform(value: number, loop: number = 1 , flip: boolean = false): number {

    for (let i = 0; i < loop; i++) {

      value = value \* 2;

    }

    if (flip) {

      value = value \* -1;

    }

    return value;

  }

}

The loop parameter will be a number type with a default value of 1. We’re using a default value in case the developer doesn’t provide one. The second parameter is flip with a boolean type. Its default value is false.

We’re using both parameters in the function. The loop will check how many times the value should be doubled. By default, the loop will run one time.

Then, we have a conditional block checking if the flip parameter is true. If it is, we will flip the value from positive to negative.

Lastly, we’re returning the value. If we run the code in the browser, the interpolation will continue to output 4000 because we aren’t using the pipe parameters.

Let’s update the app.component.html template to use the pipe parameters.

<p>{{ cost | double: 2 : true }}</p>

We can add multiple parameters to a pipe by separating each parameter with a colon. At the end of the day, pipes are just regular functions. You can think of the colon as the comma in a function’s parameters. This will output -8000.

**Taking Things a Step Further**

In this lesson, we'll look at what else we can do with pipes.

**We'll cover the following**

* [Using pipes outside of interpolation](https://www.educative.io/module/page/El5jyzfkAngPpgpAB/10370001/5017901150502912/6307406562197504#Using-pipes-outside-of-interpolation)
* [Multiple pipes](https://www.educative.io/module/page/El5jyzfkAngPpgpAB/10370001/5017901150502912/6307406562197504#Multiple-pipes)

At this point, you should feel pretty confident with pipes. We’ve looked at a lot of examples for pipes. We’re going to take things a step further by looking at what else we can do with pipes.

**Using pipes outside of interpolation**

We’re not limited to using pipes in interpolation. We can use it in directives too. There is one slight change we’ll have to make if we want to do so.

In the app.compontent.html template, we’ll add the following:

<p \*ngIf="(cost | double) > 3000">Show me!</p>

In this example, we’re using the ngIf directive to toggle the visibility of the <p> element if the cost property is greater than 3000. The value for the cost property is 2000. However, we can use the double pipe to double its value. If we want to use the double pipe, we’ll need to wrap the value with a pair of parentheses. This will tell Angular to run the cost property through the double pipe before running the comparison.

This will result in the <p> element appearing in the template.

**Multiple pipes**

We can use multiple pipes if we’d like. All we have to do is separate the pipes with another pipe character (|). For example, let’s say we want to use the double and number pipe on the temperature property. Here’s how we would do that:

<p>{{ temperature | double | number: '1.2-2' }}</p>

This will output 50.60. Pipes will run from left to right. The temperature property will run through the double pipe first, followed by the number pipe. Be aware of this, as you may receive an unexpected value if you’re not careful with the order of the pipes.