In a real-world application, we may receive data in multiple forms, but some of these could in complex formats which are not easy for normal users to understand. For example Date object shows to date in a format like this

*Sat Aug 03 2019 19:48:11 GMT+0530 (India Standard Time)*

But it’s better to have something like this:

*Saturday, 03 Aug 2019 07:50 PM*

**What are pipes in Angular?**

Angular provides some helpful filters known as Pipes which makes it very easy to format or transform the data value according to our needs.

Pipes are used with a **Pipe (|)** character, it takes an input and returns a desired formatted output. Simple Right?

**How to use a Pipe?**

Pipes can be easily used in HTML templates. For example, convert big date object into readable formate.



|  |  |
| --- | --- |
| 1  2 | lastLoggedInTime = new Date(2018, 5, 25); |

**Without Pipe**



|  |  |
| --- | --- |
| 1  2  3  4  5 | <div>      Last Logged in @ {{lastLoggedInTime}}      <!-- OUTPUT  Last Logged in @ Mon Jun 25 2018 11:48:11 GMT+0530 (India Standard Time) -->  </div> |

**With *date* Pipe**



|  |  |
| --- | --- |
| 1  2  3  4  5 | <div>      Last Logged in @ {{lastLoggedInTime | date}}      <!-- OUTPUT  Last Logged in @ Jun 25, 2018 -->  </div> |

**Adding Parameters in Pipes**

Any number of the optional parameter(s) can be added in pipes by adding a **(:)** character. Like in date pipe which we used above can be used as follows:



|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9 | {{lastLoggedInTime | date:'shortDate' }}  //6/25/18    {{lastLoggedInTime | date:'fullDate' }}  //Monday, June 25, 2018    {{lastLoggedInTime | date:'M/d/yy, h:mm a' }}  //6/25/18, 11:48 AM |

**Type of Built-in Pipes in Angular**

Angular provides a [number of Pipes](https://angular.io/api?type=pipe):

**DatePipe**, **CurrencyPipe**, **AsyncPipe**, **DecimalPipe**, **JsonPipe**, **KeyValuePipe**, **LowerCasePipe**, **PercentPipe**, **SlicePipe**, **TitleCasePipe**, **UpperCasePipe**

* To apply a pipe on a bound property use the pipe character " | "  
  <td>{{employee.code | uppercase}}</td>
* We can also chain pipes<td>{{employee.dateOfBirth | date:'fullDate' |uppercase }}</td>
* Pass parameters to pipe using colon " : "  
  <td>{{employee.annualSalary | currency:'USD':true:'1.3-3'}}</td>

<td>{{employee.dateOfBirth | date:'fullDate'}}</td>

<td>{{employee.dateOfBirth | date:'dd/MM/y'}}</td>

<td>{{employee.dateOfBirth | date:'dd/MM/y'}}</td> the parameter we specified we want the date format to be dd/mm/yyyy

<td>{{employee.annualSalary | currency:'USD':true:'1.3-3'}}</td>

1. The first parameter is the currencyCode
2. The second parameter is boolean - True to display currency symbol, false to display currency code
3. The third parameter ('1.3-3') specifies the number of integer and fractional digits

**Date Pipes**

1. <div>
2. <h4>Date Pipe Example</h4>
3. <b>Date format:</b> My joining Date is {{joiningDate | date}}.<br/>
4. <b>Specific Date Format:</b> My joining Date is {{joiningDate | date:"dd/MM/yyyy"}}<br/>
5. <b>Full Date Format:</b> My joining Date is {{joiningDate | date:"fullDate"}}<br/>
6. <b>Short Time format:</b> My joining Date is {{joiningDate | date:"shortTime"}}<br/>
7. <b>Medium Date format:</b> My joining Date is {{joiningDate | date:"mediumDate"}}<br/>
8. </div>

**Uppercase Pipe**

1. <div>
2. <h4>Upper Case Pipe Example</h4>
3. My name is {{name | uppercase}}!
4. </div>

**Lowercase Pipe**

1. <div>
2. <h4>Lower Case Pipe Example</h4>
3. My name is {{name | lowercase}}!
4. </div>

**Percentages/Number Pipe**The percent and decimal are new pipes in Angular . These pipes take arguments which provide information about decimal points in a digit i.e. how many integers and fraction numbers to be display in a formatted output. We need to pass the argument in format {minIntegerDigits}.{minFractionDigits}-{maxFractionDigits}.

1. <div>
2. <h4>Decimals/Percentages/Number Pipe Example</h4>
3. Grade : {{grade | percent:'.3'}} <br/>
4. Rating : {{rating | number:'3.1-2'}}
5. </div>

**Currency Pipe**  
Currency pipe is similar to currency filter in AngularJS 1.x. The Angular 2 Currency pipe also accepts parameters. We can pass a currency symbol as a parameter in the Currency pipe.

1. <div>
2. <h4>Currency Pipe Example</h4>
3. <b>Euro: </b> My Laptop price is {{priceEuro | currency:'EUR':**true**}} <br/>
4. <b>USD:</b>My Laptop price is {{priceUsd | currency:'USD':**true**}}
5. </div>

**Slice Pipe**Slice pipe is very similar to “limitto” filter in Angular 1.x but the order of the parameters is reversed. The first parameter of this pipe is start index and the second parameter is limit.

1. <div>
2. <h4>Slice Pipe Example</h4>
3. <b>Without Slice:</b> {{description}}<br/>
4. <b>With Slice:</b>{{description | slice:0:10}}...
5. </div>

**JSON Pipe**  
JSON pipe is very similar to “JSON” filter in Angular 1.x. It converts the JjavaScript object to JSON string and displays that on screen.

1. <div>
2. <h4>Json Pipe Example</h4>
3. {{ testArray | json}}
4. </div>

**How to create a custom Pipe?**

There is a number of Built-in Pipes available, sometimes we may want to transform values in custom formats. Let’s check how we can create our own Pipes in Angular application.

Create a new class which will import *Pipe* class and have **@Pipe** decorator with meta-information *name*

Run following*ng* command in CLI to generate a new pipe named ‘**foo**‘

|  |  |
| --- | --- |
| 1  2 | $ ng g pipe foo |

Above command will add *FooPipe* in **app.module.ts** file’s declaration array.

The **foo.pipe.ts** file will look like this:



|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13 | import { Pipe, PipeTransform } from '@angular/core';    @Pipe({    name: 'foo'  })  export class FooPipe implements PipeTransform {      transform(value: any, ...args: any[]): any {      return null;    }    } |

Change the **transform** method with following to return a modified value.



|  |  |
| --- | --- |
| 1  2  3  4 | transform(value: string, footext?: string): string {      return footext+"\_"+value+"\_"+footext;    } |

Now we can use this Pipe in the template:



|  |  |
| --- | --- |
| 1  2  3 | {{someFooText | foo:'Freaky' }}  //OUTPUT Freaky\_MyFooText\_Freaky |

**Multiple Pipes by Chaining**

Multiple Pipes can be used on single value with the chaining of Pipe character as follows:



|  |  |
| --- | --- |
| 1  2 | {{someFooText | foo:'Freaky' | foo2 | foo3:'what':'when' }} |

The basic definition or the structure of creating custom pipe is given below.

1. **import** {
2. Pipe,
3. PipeTransform
4. } from '@angular/core';
5. @Pipe({
6. name: 'name of pipe used in markup'
7. })
8. **export** **class** NameofClass **implements** PipeTransform {
9. transform(value: string, args: string[]): any {
10. //Play with value and argument and return the result
11. **return** value;
12. }
13. }
15. // This is just a sample script. Paste your real code (javascript or HTML) here.
17. **if** ('this\_is' == /an\_example/) {
18. of\_beautifier();
19. } **else** {
20. **var** a = b ? (c % d) : e[f];
21. }

**Point to remember while defining pipes**

* The Pipe is a class that contains "pipe" metadata.
* The Pipe class contains method "transform", which is implemented from “PipeTransform” interface. This method accepts the value and optionally accepts the arguments and converts it to the required format.
* We can add the required argument to the "transform" method.
* The @pipe decorator is used to declare Pipe and it is defined in core Angular library, so we need to import this library. This decorator allows us to define the name of Pipe, which is used in HTML markup.
* The “transform” method can return any type of value. If our Pipe’s return type is decided on run time, we can use “any,” otherwise, we can define specific types like number or string.