1) What is Routing?

**Reference URLs:**

<https://www.geeksforgeeks.org/routing-in-angular-9-10/>

[What is Routing and Nested Routing in Angular 9/8 ? - GeeksforGeeks](https://www.geeksforgeeks.org/what-is-routing-and-nested-routing-in-angular-9-8/?ref=rp)

Routing in Angular allows the users to create a single-page application with multiple views and allows navigation between them. Users can switch between these views without losing the application state and properties.

**Reference URL:** <https://www.geekboots.com/story/difference-between-activatedroute-and-activatedroutesnapshot-in-angular>

**ActivatedRouteSnapshot**

It contains the information about a route associated with a component loaded in an outlet at a particular moment in time. ActivatedRouteSnapshot can also be used to traverse the router state tree.

**ActivatedRoute**

It contains the information about a route associated with a component loaded in an outlet. An ActivatedRoute can also be used to traverse the router state tree.

**ActivatedRoute vs ActivatedRouteSnapshot**

Since ActivatedRoute can be reused, ActivatedRouteSnapshot is an immutable object representing a particular version of ActivatedRoute. It exposes all the same properties as ActivatedRoute as plain values, while ActivatedRoute exposes them as observables.

ActivatedRoute requires that you subscribe. Which requires that you unsubscribe. Which requires that you implement OnDestroy. This is a lot of overhead for a static route.

If you use ActivatedRouteSnapshot and have a parameter in your route definition like product/:id, then you will not get any new ID if the user changes them or your page does. Snapshot means that it was when OnInit ran, this was the state it was in at that point in time. So, any changes will be ignored.

ActivatedRouteSnapshot is an immutable data structure.

ActivatedRouteSnapshot is representing the state of the router at a particular moment in time.

ActivatedRoute is similar to ActivatedRouteSnapshot, except that it represents the state of the router changing over time.

Every node in ActivatedRouteSnapshot tree knows about the "consumed" URL segments, the extracted parameters, and the resolved data.

When using snapshot - doesn't show the right ID when clicking. So your statement here is incorrect.

Use the Snapshot if you only need the initial value of the parameter once during the component's initialization, and don't expect the URL to change while the user is still on that same component.

2) What are Custom Pipes & How will you create it?

**Reference URL:** <https://www.tektutorialshub.com/angular/angular-custom-pipes/>

The [Pipes](https://www.tektutorialshub.com/angular/angular-pipes/) are a great way to transform the appearance of elements in the template. The [Angular](https://www.tektutorialshub.com/angular-tutorial/)comes with some great built-in pipes like Date pipe, Currency pipe, and Number pipe, etc. But if these pipes do not cover your needs, then we can create our own [pipe in Angular](https://www.tektutorialshub.com/angular/angular-pipes/).

## How to Create Custom Pipes

To create a Custom Pipe, first, You need to follow these steps

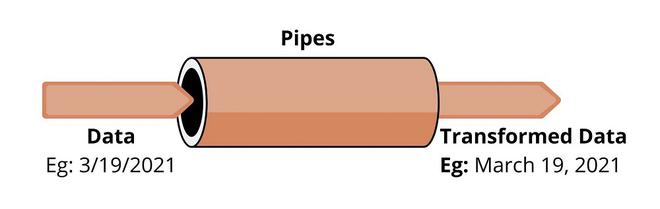
1. Create a pipe class
2. Decorate the class with @pipe decorator.
3. Give a name to the pipe in the name meta data of the @pipe decorator. We will use this name in the template.
4. The pipe class must implement the PipeTransform interface. The interfaces contain only one method transform.
5. The first parameter to the transform method is the value to be transformed. The transform method must transform the value and return the result. You can add any number of additional arguments to the transform method.
6. Declare the pipe class in the Angular Module (app.module.ts)
7. Use the custom pipe just as you use other pipes.

**Pure and Impure Pipes:**

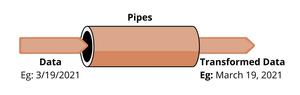
**Reference URL:** <https://www.geeksforgeeks.org/explain-pure-and-impure-pipe-in-angular/#:~:text=Pure%20pipes%20optimize%20the%20angular,value%20has%20changed%20or%20not>.

**What are Angular Pipes?**

**Angular Pipes**are a way to transform the format of output data for display. The data can be strings, currency amounts, dates, etc. Pipes are simple functions that *accept an input and return a transformed value* in a more technical understanding. They do not alter the data but change into the required format to display in the browser. Angular provides many built-in pipes for typical data transformation. You can also create custom pipes if you want to do custom transformation.



*Angular Pipes*



*Angular Pipes*

**Features:**

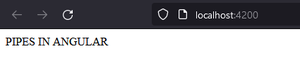
* Pipes are defined using the pipe “|” symbol.
* We can apply two formats to single data by chaining pipes. A chained pipe is when the output of the first pipe is input to the next pipe.
* Pipes can have optional parameters using the colon(:) symbol to fine-tune the output.

**Example:**We will use a built-in pipe [uppercase](https://angular.io/api/common/UpperCasePipe) that transforms text to all upper cases.

* Javascript

|  |
| --- |
| import { Component } from '@angular/core';    @Component({    selector: 'app-root',    template: `{{ title | uppercase}}`,    styleUrls: ['./app.component.css']  })  export class AppComponent {    title = 'pipes in angular';  } |

**Output:**



**Pure and Impure Pipes:**In Angular, Pipes are of two types.

* Pure Pipes
* Impure Pipes

**Pure Pipes:**

* Pure pipes in angular are the pipes that execute when it detects a *pure change* in the input value.
* A pure change is when the [change detection](https://angular.io/guide/glossary#change-detection) cycle detects a change to either a primitive input value (such as String, Number, Boolean, or Symbol) or object reference (such as Date, Array, Function, or Object). Let’s see an example,
* HTML

|  |
| --- |
| <div> {{ user | myPipe }} </div> |

* Javascript

|  |
| --- |
| import { Component } from '@angular/core';    @Component({    selector: 'app-root',    templateUrl: 'app.component.html',    styleUrls: ['./app.component.css']  })  export class AppComponent {    user = { name:'Rohit', age: 23};  } |

here, **user** is an object and **myPipe** is a custom Pipe.

* myPipe will execute if it detects a change in the user’s object reference.
* What will happen if I try to change the property of users? Will the myPipe executes?
* Javascript

|  |
| --- |
| user.age = 19; |

* The answer is no. myPipe will not execute even if the property has changed because the object reference is still the same. To detect this change, Angular will have to do a deep check for differences within the object. Also, this change is not pure; the pipe ignores the changes within objects.
* A single instance of pipe is used all over the component.
* A pure must use a pure function.
* A pure function does not depend on any state, data, or change during the execution. In other words, given the same arguments, a pure function should always return the same output.
* Javascript

|  |
| --- |
| function add(a, b){      return a + b   } |

* In the above example, If the argument values are not changed, the function will return the same output. So it does not depend on state change.
* By default, pipes in angular are pure pipes. Custom pipes can be defined as pure pipes by turning the pure flag to *true*of the @Pipe decorator.
* Javascript

|  |
| --- |
| @Pipe({    name: 'purePipe',    pure: true  })  export class PurePipe {} |

**Impure Pipes:**

* Impure pipes in angular are the pipes that execute when it detects an *impure change* in the input value.
* An impure change is when the [change detection](https://angular.io/guide/glossary#change-detection) cycle detects a change to composite objects, such as adding an element to the existing array. Let’s take the same example that we took earlier.
* HTML

|  |
| --- |
| <div> {{ user | myPipe }} </div> |

* Javascript

|  |
| --- |
| import { Component } from '@angular/core';    @Component({    selector: 'app-root',    templateUrl: 'app.component.html',    styleUrls: ['./app.component.css']  })  export class AppComponent {    user = { name:'Rohit', age: 23};  } |

* Javascript

|  |
| --- |
| user.age = 19; |

* For the above example, if we want our custom pipe to detect changes within our user object then we have to make the **pure** attribute to false.
* Javascript

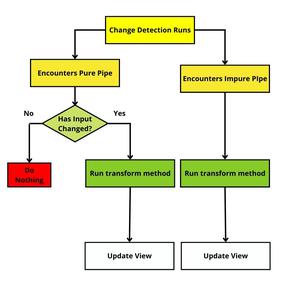
|  |
| --- |
| @Pipe({    name: 'myPipe',    pure: false  })  export class MyPipe {...} |

* Now, myPipe will execute on every change.
* Impure pipes are required because angular ignores changes to composite objects.
* Impure pipes execute every time angular detects any changes regardless of the change in the input value.
* It uses the impure function.
* An impure function depends on any state, data, or change during the execution and may not return the same result if the same inputs are passed into the respective function. Let’s take a simple function as an example
* Javascript

|  |
| --- |
| var discount = 15    function calculateAmount(price){      return total + (price - discount);  } |

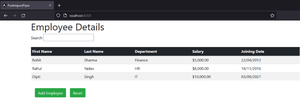
* The above function does not return the same output every time. Because the function return value depends on the external variable called discount. It may vary based upon the value.
* Multiple instances are created for impure pipes.
* [Async pipe](https://angular.io/guide/pipes#unwrapping-data-from-an-observable) is an example of an Impure pipe.

**How Angular Handles Pure and Impure Pipes:**The following diagram describes how angular handles Pure and Impure pipes.



*How pure and impure pipes are handled by angular*

**Pure and Impure Pipes Example:**In this example, we will create an Angular app to display employees’ data. We will create a custom pipe to understand the working of pure and impure pipes. The button “Add Employee” is used to add an employee to the list and “Reset” is used to clear the textbox.



**Custom pipe (search.pipe.ts)**

* Javascript

|  |
| --- |
| import { Pipe, PipeTransform } from '@angular/core';    @Pipe({    name: 'search'  })  export class SearchPipe implements PipeTransform {      transform(value: any, name: string) {      if(name === ''){        return value;      }      return value.filter((employee) => {      employee.firstName.startsWith(name)     });    }  } |

**search** pipe is used to filter the list based on the text added in the input box.

**Component class (app.component.ts)**

* Javascript

|  |
| --- |
| import { Component } from '@angular/core';    @Component({    selector: 'app-root',    templateUrl: './app.component.html',    styleUrls: ['./app.component.css']  })  export class AppComponent {    nameString = '';    employees = [{        firstName: 'Rohit',        lastName: 'Sharma',        dept: 'Finance',        salary: 5000,        doj: new Date('2012-04-22')      },      {        firstName: 'Aditi',        lastName: 'Mishra',        dept: 'Sales',        salary: 6000,        doj: new Date('2016-09-16')      },      {        firstName: 'Dipti',        lastName: 'Singh',        dept: 'IT',        salary: 10000,        doj: new Date('2021-09-03')      }    ]      addUser(){      this.employees.push({        firstName: 'Rahul',        lastName: 'Yadav',        dept: 'HR',        salary: 8000,        doj: new Date('2016-11-19')      })    }      reset(){      this.employees = this.employees.slice()    }  } |

In the component class, there is an array **employees**

for storing employees data. The function **addUser()**

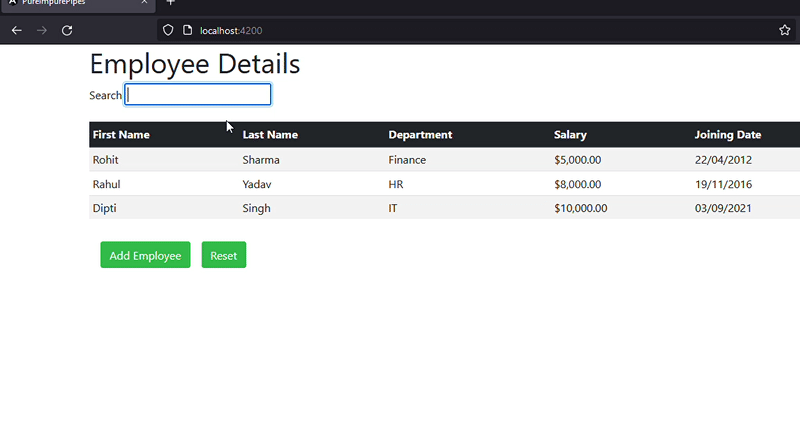
is used to add employees in the list.

**Template (app.component.html)**

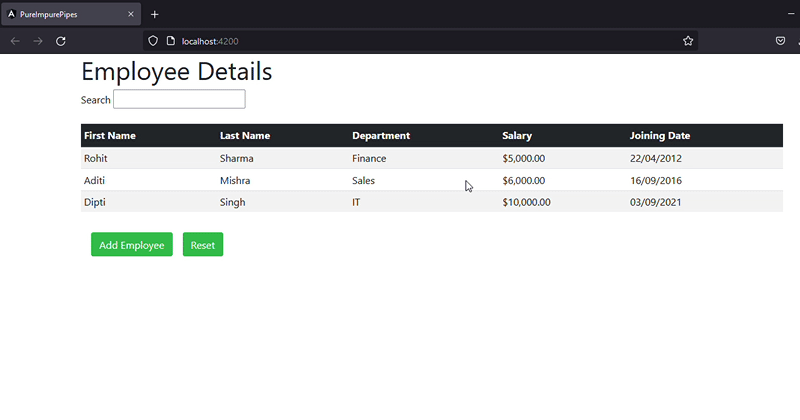
* HTML

|  |
| --- |
| <div class="container">    <h1>Employee Details</h1>    <span>Search </span>    <input type="text" [(ngModel)]="nameString">    <br/><br/>    <table class="table table-sm table-striped m-t-4">      <thead class="thead-dark">        <tr>        <th>First Name</th>        <th>Last Name</th>        <th>Department</th>        <th>Salary</th>        <th>Joining Date</th>      </tr>    </thead>    <tbody>      <tr \*ngFor="let employee of employees | search:nameString">        <td>{{employee.firstName}}</td>        <td>{{employee.lastName}}</td>        <td>{{employee.dept}}</td>        <td>{{employee.salary | currency}}</td>        <td>{{employee.doj | date:'dd/MM/yyyy'}}</td>      </tr>    </tbody>    </table>    <button type="button" class="btn btn-success m-3"            (click)="addUser()">      Add Employee    </button>    <button type="button" class="btn btn-success"            (click)="reset()">      Reset    </button>  </div> |

**Output:**The output below shows the working search pipe in the Angular app.



Now, let’s explore the pure and impure pipe working in our app. Now we will try to add an employee while we are filtering the data. By default, custom pipes are pure pipes. Therefore, the below output is an example of a pure pipe.



Here, We try to add an employee having the first name with the same starting letter to check if it

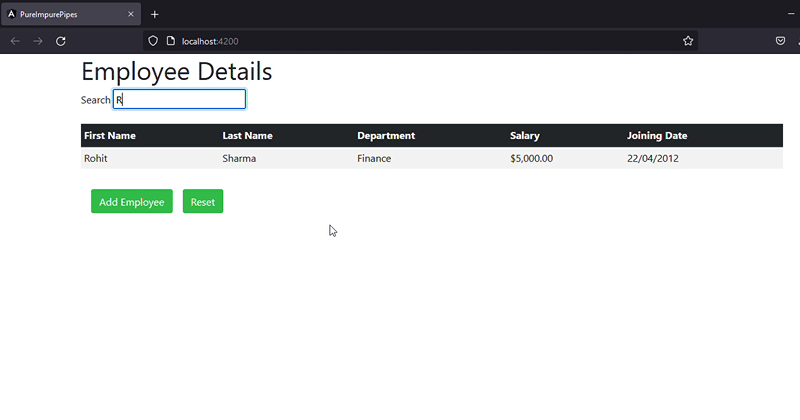
appears in the filtered data.

The filtered list doesn’t update on clicking the Add Employee button because pure pipe ignores changes within objects. The filtered list is updated by clicking the Reset button because it changes the array reference.

If we use impure pipes, the change will be reflected by just clicking the Add Employee button. Impure pipes now detect change within objects. Let’s see how it is implemented.

* Javascript

|  |
| --- |
| import { Pipe, PipeTransform } from '@angular/core';    @Pipe({    name: 'search',    pure: false  })  export class SearchPipe implements PipeTransform {      transform(value: any, name: string) {      if(name === ''){        return value;      }      return value.filter((employee) => {      employee.firstName.startsWith(name)     });    }    } |



### Difference between Pure and Impure pipes

| **Pure pipe** | Impure pipe |
| --- | --- |
| The pipe is executed only when it detects a change in primitive value or object reference | The pipe is executed on every change detection cycle irrespective of the change in the input value. |
| A single instance is created. | Multiple instances are created |
| It uses pure function | It uses an impure function |
| Pure pipe optimizes application performances. | Impure pipe may slow down your application |

**Conclusion:**

* Pure pipes detect changes only when there is a change to primitive types or object reference.
* Pure pipes optimize the angular change detection cycle because checking primitive values or object references is much faster than detecting a change within an object.
* Impure pipes execute every time the change detection runs regardless of input value has changed or not. So, now the code runs on every mouse or keyboard event; this may overuse system resources. If the pipe is heavy, it will degrade the application’s performance.
* Therefore, Be careful while using impure pipes.

3) What is template driven forms ?

Template Driven Forms rely on directives in the template to create and manipulate the underlying object model. Template-driven forms use two-way data binding to update the data model in the component as changes are made in the template and vice versa. They are useful for adding a simple form to an app, such as an email list signup form. They're straightforward to add to an app, but they don't scale as well as reactive forms. If you have very basic form requirements and logic that can be managed solely in the template, template-driven forms could be a good fit.

4) What are reactive forms ?

Reactive forms provide a model-driven approach to handling form inputs whose values change over time. They provide direct, explicit access to the underlying forms object model. Compared to template-driven forms, they are more robust: they're more scalable, reusable, and testable.

Please refer the following URL for more information about Template Driven and Reactive Forms:

<https://angular.io/guide/forms-overview>

5) Can you tell the difference between template driven and reactive forms?

**Reference URLs:**

<https://www.pluralsight.com/guides/difference-between-template-driven-and-reactive-forms-angular>

<https://blog.angular-university.io/introduction-to-angular-2-forms-template-driven-vs-model-driven/#:~:text=Template%20Driven%20Forms%20are%20based,powerful%20and%20easier%20to%20use>.

1) Template-driven forms make use of the "FormsModule", while reactive forms are based on "ReactiveFormsModule".

2) Template-driven forms are asynchronous in nature, whereas Reactive forms are mostly synchronous.

3) In a template-driven approach, most of the logic is driven from the template, whereas in reactive-driven approach, the logic resides mainly in the component or typescript code.

4) Reactive Forms are a better default choice for new applications, as they are more powerful and easier to use.

5) The Template Driven approach is very familiar to AngularJs developers and is ideal for easy migration of AngularJs applications into Angular.

6) The Reactive approach removes validation logic from the template, keeping the templates cleaner.

7) Reactive forms are easier to use in general and support better more advanced use cases via its Observable-based API.

6) When we should reactive forms and when to use template driven forms?

**Reference URLs:**

[Angular - Introduction to forms in Angular](https://angular.io/guide/forms-overview)

<https://www.ibrahima-ndaw.com/blog/5-reasons-to-use-reactive-form/>

If you have very basic form requirements and logic that can be managed solely in the template, template-driven forms could be a good fit.

If forms are a key part of your application, use reactive forms as they are more predictable, reusable, scalable and testable.