**In Angular 19, Pipes are a powerful tool for transforming data within your templates.** They allow you to format, filter, and manipulate data before displaying it to the user.

**Key Concepts:**

* **Data Transformation:** Pipes take an input value and transform it into a desired output.
* **Template Expressions:** Pipes are used within template expressions, typically with the pipe operator (|).
* **Built-in Pipes:** Angular provides several built-in pipes for common tasks:
  + **DatePipe:** Formats dates (e.g., {{ date | date:'shortDate' }}).
  + **CurrencyPipe:** Formats numbers as currency (e.g., {{ price | currency:'USD' }}).
  + **DecimalPipe:** Formats numbers with a specific number of decimals.
  + **PercentPipe:** Formats numbers as percentages.
  + **UpperCasePipe:** Converts strings to uppercase.
  + **LowerCasePipe:** Converts strings to lowercase.
  + **SlicePipe:** Extracts a portion of an array or string.
  + **JsonPipe:** Converts an object to a JSON string.
* **Custom Pipes:** You can create your own custom pipes to perform specific data transformations.

**Example:**

TypeScript

import { Component } from '@angular/core';

@Component({

selector: 'app-my-component',

template: `

<p>Today is: {{ today | date:'fullDate' }}</p>

<p>Price: {{ price | currency:'USD':'1.2-2' }}</p>

`

})

export class MyComponent {

today: Date = new Date();

price: number = 19.99;

}

In this example:

* {{ today | date:'fullDate' }} uses the DatePipe to format the today date.
* {{ price | currency:'USD':'1.2-2' }} uses the CurrencyPipe to format the price with the USD currency symbol and two decimal places.

**Creating a Custom Pipe:**

1. **Generate the Pipe:** Use the Angular CLI to generate a new pipe:

Bash

ng generate pipe myCustomPipe

1. **Implement the Pipe:**

TypeScript

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

@Pipe({

name: 'myCustomPipe'

})

export class MyCustomPipe implements PipeTransform {

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

// Implement your custom transformation logic here

return value.toUpperCase();

}

}

1. **Use the Custom Pipe in your Template:**

HTML

<p>{{ 'hello' | myCustomPipe }}</p>

**Benefits of Using Pipes:**

* **Improved Readability:** Pipes make your templates cleaner and more concise.
* **Reusability:** Custom pipes can be reused across multiple components.
* **Maintainability:** Encapsulates data transformation logic, making your code easier to maintain.

By effectively using built-in and custom pipes, you can enhance the presentation and user experience of your Angular applications.

Examples:

**1. Reverse Pipe**

* **Purpose:** Reverses the order of characters in a string.

TypeScript

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

@Pipe({

name: 'reverse'

})

export class ReversePipe implements PipeTransform {

transform(value: string): string {

return value.split('').reverse().join('');

}

}

* **Usage:**

HTML

<p>Original: {{ myText }}</p>

<p>Reversed: {{ myText | reverse }}</p>

**2. Highlight Search Term Pipe**

* **Purpose:** Highlights the occurrences of a search term within a string.

TypeScript

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

@Pipe({

name: 'highlight'

})

export class HighlightPipe implements PipeTransform {

transform(value: string, args: string): string {

if (!args) {

return value;

}

const term = args.toLowerCase();

return value.replace(new RegExp(term, 'gi'), '<span class="highlight">$&</span>');

}

}

* **Usage:**

HTML

<p>{{ myText | highlight: searchTerm }}</p>

* **CSS:**

CSS

.highlight {

background-color: yellow;

}

**3. Filter Pipe**

* **Purpose:** Filters an array of objects based on a given property and search term.

TypeScript

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

@Pipe({

name: 'filter'

})

export class FilterPipe implements PipeTransform {

transform(items: any[], searchText: string): any[] {

if (!searchText) {

return items;

}

searchText = searchText.toLowerCase();

return items.filter(item =>

JSON.stringify(item).toLowerCase().includes(searchText)

);

}

}

* **Usage:**

HTML

<ul>

<li \*ngFor="let item of items | filter: searchTerm">{{ item.name }}</li>

</ul>

**Key Considerations:**

* **Performance:** For complex pipe transformations, consider using pure pipes (by adding the pure: false option to the @Pipe decorator) to ensure that the pipe is executed whenever any input value changes.
* **Testing:** Write unit tests for your custom pipes to ensure they function correctly and produce the expected output.

These examples demonstrate the versatility of custom pipes in Angular. You can create a wide range of custom pipes to transform data in various ways, making your templates more concise, readable, and maintainable.