6 Pipes

Pipes, built-in pipes, pure vs impure

Pipe

- A pipe is a transformer that can be applied in the view
- Takes in data as input and transformers it do desired output

```
{{ value | pipe1 | pipe2:param1:param2:param3:… | … }}
```

  - Using the pipe (|) operator
  - Uses colons (:) to separate between parameters
  - Chaining to other pipes for useful combinations

Built-in Pipe

- LowerCasePipe
- CurrencyPipe
- DatePipe
- DecimalPipe
- JsonPipe
- PercentPipe
- AsyncPipe
- ...

Built-in Pipe

LowerCasePipe

- Puts text in lower case

UpperCasePipe

- Puts text in upper case

```
<div>
   <p>In lowercase: '{{value | lowercase}}'</p>
   <p>In uppercase: '{{value | uppercase}}'</p>
</div>
```

Title → title

Title → TITLE

Built-in Pipe

DecimalPipe

- Takes a number and show it in certain format

```
expression | number[:digitInfo]
```

- digitInfo: format is described in three values
- `{a}.{b}-{c}`
  - a : minimum number of integer digits (default 1)
  - b : minimum digits after fraction (default 0)
  - c : maximum digits after fraction (default 3)

```
<p>{{pi | number:'3.5-5'}}</p>
```

→ 003,14159

a=3 b=c=5

Built-in Pipe

CurrencyPipe

- Show a decimal in currency format

```
expression | currency[:currencyCode[:display[:digitInfo[:locale]]]]
```

  - currencyCode: like EUR, USD,... (ISO 4217 currency code)
  - display: 'symbol', 'symbol-narrow' or 'code'
  - digitInfo: same as with DecimalPipe (default 1,2-2)
  - locale: e.g. 'fr'

```
<p>A: {{0.259 | currency:'EUR':'code'}}</p>
```

0.259 → EUR 0.26

```
<p>B: {{1700.3495 | currency:'EUR':'symbol':’5.2-2'}}</p>
```

1700.3495 → € 01,700.35

Built-in Pipe

DatePipe

- Formats date value to a string based on format

```
expression | date[:format[:timezone[:locale]]]
```

  - expression: date object or number (ms since UTC epoch)
  - format: which date/time component to include

```
<p>A: {{today | date:'fullDate'}}</p>
```

Tuesday, September 20, 2022

```
<p>B: {{today | date:'shortTime'}}</p>
```

8:49 AM

```
<p>C: {{today | date:'h:mm:ss a z'}}</p>
```

8:49:19 AM GMT+1

Built-in Pipe

JsonPipe

- Transforms inputs using JSON.stringify()
  - Useful for debugging

```
<p>{{object | json}}</p>
```

Using pipe from TS code

- Import the built-in pipe class from @angular/common

```
import { DecimalPipe } from '@angular/common';
```

- Inject in constructor

```
export class MyComponent {
            constructor(private pipe:DecimalPipe){}
}
```

- Call the **transform()** function

```
var formattedNumber = this.pipe.transform(12,'4.1-4');
```

Custom pipe

- Custom pipes to transform towards own requirements
- Defining pipes
  - Needs pipe metadata, with the **@Pipe** decorator
  - *@Pipe* take the ***name*** of the pipe as parameter
  - Implements ***PipeTransform*** interface with *transform* method
  - ***transform*** method takes input value and array of extra parameters

Custom pipe (Example)

1. Define pipe

```
@Pipe ({
   name: 'multiplier'
})
export class MultiplierPipe implements PipeTransform {

   transform(value: number, multiply: string): number {
         let mul = parseFloat(multiply);
         return mul * value
   }
}
```

Custom pipe (Example)

2. Declare in module

```
import { MultiplierPipe } from ‘./multiplier.pipe';

@NgModule({
  imports: [BrowserModule, FormsModule],
  declarations: [AppComponent, MultiplierPipe],
  bootstrap: [AppComponent]
})
export class AppModule { }
```

- If the pipe is a part of a module, and you want to use it in other modules, add it also to providers to be part of the dependency injection

Custom pipe (Example)

3. Use the pipe

```
import {
    Component
} from '@angular/core';
@Component ({
    selector: 'my-app’,
    template: '<p>Multiplier: {{2 | multiplier:’10’}}</p>'
})
export class AppComponent { }
```

Pure VS Impure

Pure functions

- Always gives the same result for the same input
- Does not rely on external state

```
add(a: number, b: number): number { //pure function
  return a + b;
}

test() {
  let x = 4, y = 3, result: number;
  result = this.add(4, 3);
  result = this.add(x, y); //the same result, because x === 4 and y === 3

  x = 2;
  result = this.add(x,y); //might be a different result, because x !== 4
}
```

Pure VS Impure

Impure functions

- Does NOT necessarly gives the same result for the same input
- Might rely on external state

```
temp = 0;
add(a: number, b: number): number { //impure function
  this.temp += a + b;
  return this.temp;
}

test() {
  let x = 4, y = 3, result: number;
  result = this.add(4, 3); //7
  result = this.add(x, y); //14 different result, despite x === 4 and y === 3
}
```

Change detection

- Any asynchrounous event in Angular triggers change detection
  - Mouse, keyboard event, timers

- Angular try to minimize the change detection
  - Pipe transformations are considered **pure** by default
  - It will not do change detection when pipe inputs have not changed

- Pure pipes are better for performance
  - Most pipes are pure (DatePipe, DecimalPipe, ...)

Impure pipe

- Impure can be activated by setting **pure** property to false
- An impure pipe makes Angular's change detection system check output of the pipe on each cycle

```
@Pipe({
    name: 'translate',
    pure: false
})
```

- **Example**: A custom translation pipe is impure
  - It relies on external state
  - Resource file changes when the current language change

LAB 6

Using pipes