ARS

Motivation

- Daten im Front-End filtern
- Flexible und wiederverwendbare Darstellung der Daten, ohne diese dabei zu verändern

ARS

Lernziele

- Sie benennen den Aufbau und die Bestandteile einer Pipe.
- Sie erklären, wie Pipes in Templates verwendet werden.
- Sie beschreiben wie Pipes getestet werden.

ARS

Anwendungsfall

- **Häufig:** Transformation eines Wertes für die Anzeige
- **Beispiel:** Date-Objekt als String

Tue May 23 2017 15:28:56 GMT+0200 (Mitteleuropäische Sommerzeit)

Benutzer bevorzugen jedoch: 23.05.2017

- **Pipes lösen folgende Probleme:**
  - Wiederkehrende Ausgabe von Daten des gleichen Typs (aber u.U. aus verschiedenen Datenquellen)
  - Lediglich die Anzeige soll beeinflusst werden, i.e. Daten bleiben unverändert

ARS

Angular – Pipe

- Syntax vergleichbar mit Unix-Pipes
- **Beispiel**: Transformation eines Datums via `DatePipe`

```
birthday = new Date(1988, 3, 15); // April 15, 1988
<p>The hero's birthday is {{ birthday | date }}</p>
```

ARS

Pipes

DecimalPipe	Formatierung einer Zahl basierend auf lokalen Regeln.
JsonPipe	Wandelt einen Wert in ein JSON-String um.
SlicePipe	Selektiert einen Teilbereich einer Liste oder String.
CurrencyPipe	Formatierung einer Währungszahl basierend auf lokalen Regeln.
DatePipe	Formatierung des Datums basierend auf lokalen Regeln.
UpperCasePipe	Wandelt einen Wert in Großbuchstaben um.
LowerCasePipe	Wandelt einen Wert in Kleinbuchstaben um.
PercentPipe	Interpretiert und stellt eine Zahl als Prozentwert dar

ARS

Pipes

Parametrisieren von Pipes:

```
<tr *ngFor="let birthday of birthDates | date:'fullDate'">
```

Verketten von Pipes:

```
{{ birthday | date | uppercase }}
```

Zwei Arten von Pipes:

- Pure (Default): Ausführung nur bei Änderung eines primitiven Datentypen, oder Änderung der Objektreferenz
- Impure: Ausführung während jedem Change „Detection Lifecyle“-Schritt einer Komponente

ARS

Benutzerdefinierte Pipes

- Bestandteile
  - @Pipe-Dekorator mit Namen der Pipe
  - Pipe-Klasse, welche das Interface PipeTransform implementiert

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

@Pipe({name: 'customPipe'})
export class ACustomPipe implements PipeTransform {

  transform(value: number): number { ... }

}
```

- In der Transform-Funktion werden die Eingaben umgewandelt

ARS

Benutzerdefinierte Pipes

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

@Pipe({name: 'toThePowerOf'})
export class ToThePowerOfPipe implements PipeTransform {

  // value:    Wert der transformiert werden soll
  // exponent: Optionaler Parameter
  // returns:  Transformierter Rückgabewert
  transform(value: number, exponent: string): number {
    let exp = parseFloat(exponent);
    return Math.pow(value, isNaN(exp) ? 1 : exp);
  }
}
```

```
<p>Super power boost: {{2 | toThePowerOf: 10}}</p>

=> 1024
```

ARS

Zusammenfassung

Um was ging es in diesem Modul?

- Angular Pipes
  - Aufbau
  - Nutzen
  - Selbst Pipefunktionalität erstellen

Wozu brauche ich das? Was will ich damit machen?

- Trennung von Daten aus der Logikschicht und deren Darstellung in Templates (Separation of concerns)
- Konsistente, wiederverwendbare Darstellung von Daten in der ganzen Applikation
- Darstellung von Daten automatisiert mit Unit Tests verifizieren

ARS

Kontrollfragen

- Welchen Zweck erfüllen Pipes in Angular?
- Welche eingebauten Pipes existieren bereits in Angular?
- Wie können Pipes verkettet werden?
- Wie werden Argumente an Pipes übergeben?