Angular
Pipes

1 Pipes

Angular stellt einige sogenannter Pipes zur Verfügung, mit denen man Objekte vor der eigentlichen Verwendung verändern kann.
Dabei gibt es zwei Hauptanwendungsgebiete:

- Die Ausdrücke können **Collections** sein. In diesem Fall wird die Collection ähnlich wie bei LINQ gefiltert, sortiert,… Diese Variante findet bei **`*ngFor`** ihren Einsatz.
- Die Ausdrücke können aber auch **Strings** sein, in dem Fall wird die Ausgabe entsprechend den Angaben formatiert.

1.1 Allgemeine Struktur

Pipes werden allgemein mit **|** an Ausdrücke angehängt und verändern diese Ausdrücke vor dem Rendering.
Allgemein sieht die Struktur so aus:

```
expression | filternameA:parameter1:…:parameterN | filternameB:p1:…:pN
```

Die Parameter können dabei je nach Pipe auch entfallen.

1.2 Vordefinierte Pipes

Einige Pipes sind bereits vordefiniert. Sie sind unter https://angular.io/api/common#pipes aufgelistet:

`AsyncPipe`	Unwraps a value from an asynchronous primitive.
`CurrencyPipe`	Transforms a number to a currency string, formatted according to locale rules that determine group sizing and separator, decimal-point character, and other locale-specific configurations.
`DatePipe`	Formats a date value according to locale rules.
`DecimalPipe`	Transforms a number into a string, formatted according to locale rules that determine group sizing and separator, decimal-point character, and other locale-specific configurations.
`I18nPluralPipe`	Maps a value to a string that pluralizes the value according to locale rules.
`I18nSelectPipe`	Generic selector that displays the string that matches the current value.
`JsonPipe`	Converts a value into its JSON-format representation. Useful for debugging.
`KeyValuePipe`	Transforms Object or Map into an array of key value pairs.
`LowerCasePipe`	Transforms text to all lower case.
`PercentPipe`	Transforms a number to a percentage string, formatted according to locale rules that determine group sizing and separator, decimal-point character, and other locale-specific configurations.
`SlicePipe`	Creates a new `Array` or `String` containing a subset (slice) of the elements.
`TitleCasePipe`	Transforms text to title case. Capitalizes the first letter of each word and transforms the rest of the word to lower case. Words are delimited by any whitespace character, such as a space, tab, or line-feed character.
`UpperCasePipe`	Transforms text to all upper case.

1.2.1 Ausgabepipes

Die einfachsten Filter sind jene, die die Ausgabe formatieren.

- currency
- number
- percent
- date
- lowercase / uppercase / titlecase
- json

Das Prinzip sollte anhand der folgenden Beispiele klar sein. Den Rest (also vor allem die zusätzlichen Parameter) in der jeweiligen Angular-Online-Doku nachschlagen (Link siehe oben).

```
fValue = 123.4567;
iValue = 1547452223346;
sValue = 'Hansi HUBER';
```

{{fValue}} {{iValue}} {{sValue}}	123.4567 1471792223346 Hansi HUBER
{{fValue | currency:'EUR':'symbol':'1.2-2'}} {{fValue | number:'1.1-3'}} {{sValue | lowercase}} {{sValue | uppercase}} {{sValue | titlecase}} {{iValue | date:'yyyy-MM-dd HH:mm:ss'}} {{iValue | date:'EEEE, dd. MMM' | uppercase}}	€123.46 123.457 hansi huber HANSI HUBER Hansi Huber 2013-12-29 14:27:03 SUNDAY, 21. AUG

Aufpassen muss man bei **currency**, denn da darf man z.B. nicht das **€**-Symbol direkt angeben, sondern muss das Kürzel laut ISO 4217 verwenden (http://www.iotafinance.com/de/ISO-4217-Waehrungs-Codes.html), also **EUR**. Als zweiten Parameter kann man **'symbol'** oder **'symbol-narrow'** angeben.
Die Parameter bei **number** heißen folgendes (entnommen aus obigem Link):

```
{minIntegerDigits}.{minFractionDigits}-{maxFractionDigits}
```

- `minIntegerDigits` is the minimum number of integer digits to use. Defaults to `1`.
- `minFractionDigits` is the minimum number of digits after fraction. Defaults to `0`.
- `maxFractionDigits` is the maximum number of digits after fraction. Defaults to `3`.

1.2.2 slice

slice ähnelt den Funktionen **Skip()**/**Take()** in LINQ bzw. der gleichnamigen Funktion für Javascript-Arrays:

```
<tr *ngFor="let person of persons | slice:3:5">
```

ID	Vorname	Nachname	Alter	Geschlecht	Staat
4	Betty	Torres	38	F	China
5	Evelyn	Coleman	76	F	Philippines

Für die beiden Parameter kann man wie vermutet auch Variable benutzen:

```
<tr *ngFor="let person of persons | slice:idxFrom:idxTo">
Von:<input [(ngModel)]="idxFrom" />
Bis:<input [(ngModel)]="idxTo" />
```

1.3 Eigene Pipes

Man kann auch gänzlich neue Pipes definieren, die man dann ähnlich den vorhin besprochenen verwendet. Auch diesen neuen Pipes kann man **Parameter** mitgeben, indem man sie durch ein **:** getrennt an die Pipe anhängt (siehe die oben erwähnte allgemeine Struktur).

Folgende Schritte sind notwendig:

1. Klasse erzeugen, die das **Interface `PipeTransform`** implementiert
2. Funktion **`transform`** programmieren – sie stellt die eigentliche Pipe dar
3. Klasse mit **`@Pipe`**-Dekorator versehen
4. Pipe im Modul als **Abhängigkeit** eintragen, und zwar bei **`declarations`**

1.3.1 ng generate

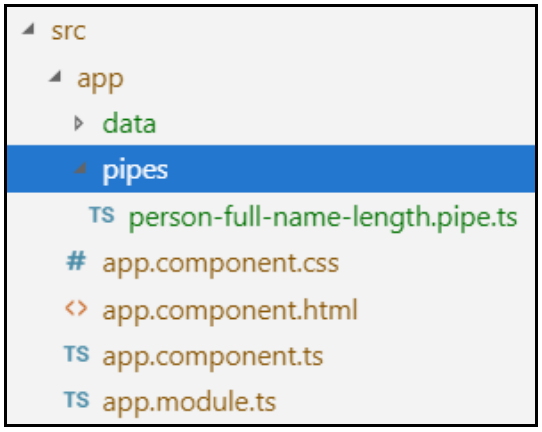
Auch hier wird man wieder von AngularCLI unterstützt. Der Befehl lautet **`ng generate pipe`** bzw. kurz **`ng g p`**:

```
PS C:\_PR\Angular\Angular8App> ng g p PersonFullNameLength
CREATE src/app/person-full-name-length.pipe.spec.ts (246 bytes)
CREATE src/app/person-full-name-length.pipe.ts (233 bytes)
UPDATE src/app/app.module.ts (547 bytes)
```

Bei mehreren Pipes bietet sich an, diese in einem eigenen Verzeichnis zusammenzufassen:

```
PS C:\_PR\Angular\Angular8App> ng g p ./pipes/PersonFullNameLength --skipTests=true
CREATE src/app/pipes/person-full-name-length.pipe.ts (233 bytes)
UPDATE src/app/app.module.ts (553 bytes)
```

Camel-case wird dabei automatisch auf kebab-case umgewandelt:



Die Datei sieht zu Beginn so aus:

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

@Pipe({
  name: 'personFullNameLength'
})
export class PersonFullNameLengthPipe implements PipeTransform {

  transform(value: unknown, ...args: unknown[]): unknown {
    return null;
  }

}
```

Eigene Pipes müssen (im Gegensatz zu den Core-Pipes) explizit der App bekanntgegeben werden. Dies geschieht im Dekorator des Moduls (also in **`app.module.ts`**) bei der Property **`declarations`**. Die Datei app.module.ts wurde beim Generieren aber schon automatisch aktualisiert:

```
import { PersonService } from './person.service';
import { PersonFullNameLengthPipe } from './pipes/person-full-name-length.pipe';

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

1.3.2 String-Pipe

Als erstes Beispiel soll eine Pipe entworfen werden, die einen String umwandelt. In diesem Fall ist der erste Parameter von **`transform`** ein String:

```
@Pipe({
  name: 'personGenderSign'
})
export class PersonGenderSignPipe implements PipeTransform {
  transform(gender: string): string {
    if (gender.toUpperCase() === 'F') return '♀'; //&#x2640;
    if (gender.toUpperCase() === 'M') return '♂'; //&#x2642;
    return gender;
  }
}
```

```
{{person.gender | personGenderSign}}
```

55	Mark	Fox	53	♂	*Russia*
65	Joan	Gray	28	♀	*Indonesia*
70	Robert	Lee	67	♂	*South Africa*
77	Joyce	Ross	24	♀	*Russia*

Auch hier wieder die Registrierung in app.module.ts nicht vergessen (falls man die Datei manuell erstellt hätte):

```
import { PersonService } from './person.service';
import { PersonFullNameLengthPipe } from './pipes/person-full-name-length.pipe';
import { PersonGenderSignPipe } from './pipes/person-gender-sign.pipe';

@NgModule({
  declarations: [AppComponent, PersonFullNameLengthPipe, PersonGenderSignPipe],
  imports: [BrowserModule, FormsModule, HttpClientModule],
  providers: [PersonService]
```

1.3.3 Collection-Pipe

Als zweites Beispiel soll eine Pipe **`personFullNameLength`** programmiert werden, die alle Personen filtert, deren Länge des Gesamtnamens unter einem angegebenen Wert liegt.
In der **`transform()`**-Funktion selbst hat man im ersten Parameter Zugriff auf die Collection, in den weiteren auf die Parameter, die bei der Verwendung der Pipe angegeben werden:

```
<tr *ngFor="let person of persons | personFullNameLength:10">
```

Die Pipe soll jetzt nur noch jene Einträge „durchlassen“, deren Namenslänge unter dem Wert liegt.

```
@Pipe({
  name: 'personFullNameLength'
})
export class PersonFullNameLengthPipe implements PipeTransform {
  transform(items: Person[], maxLen: number): Person[] {
    const filtered: Person[] = [];
    for (const item of items) {
      const fullName = `${item.lastname} ${item.firstname}`;
      const len = fullName.length;
      if (len < maxLen) filtered.push(item);
    }
    return filtered;
  }
}
```

Im Code könnte man natürlich auch die Javascript-Funktion filter() benutzen.

Die Übergabe des Parameterwerts als Component-Variable ist auch kein Problem:

```
Max Length:<input [(ngModel)]="maxLen" />
<table>
  <>...</>
  <tr *ngFor="let person of persons | personFullNameLength:maxLen">
```

1.3.4 Performance: pure Pipes

Fügt man in unserem ersten Beispiel die Eingangscollection (also persons) eine neue Person hinzu, etwa durch einen Buttonklick, wird die Anzeige nicht aktualisiert. Der Grund ist, dass Pipes per Default als pure Pipes verwendet werden. Dabei wird in der **change detection** nur geprüft, ob sich die Referenz (also die Adresse) des Objekts geändert hat. Beim Hinzufügen ist das aber nicht der Fall.
Um sozusagen ein „**deep watch**“, also auch eine Überprüfung der Properties des Objekts, durchzuführen, muss die Pipe explizit auf **`impure`** gesetzt werden:

```
@Pipe({
  name: 'personFullNameLength',
  pure: false
})
export class PersonFullNameLengthPipe implements PipeTransform {
```