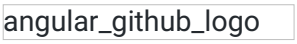
Angular Typescript Workshop



AngularJS Docs | TypeScript Docs | JavaScript Docs

Allgemeines

Typescript wurde aus JavaScript und C# entworfen

TypeScript

Typen in TypeScript

Die Basistypen von TS sind gleich wie die von JavaScript (Erweiterte Liste von TS-Typen)

- Bool (ein Boolesche Wert)
- Number (eine Zahl)
- String (eine Zeichenfolge)
- Array (eine Liste von Objekten)
- Any (also Undefinierter Typ, kann alles sein)

Variablentypen in TypeScript deklarieren

Eine Variable kann ohne Typ deklariert werden, sie übernimmt dann direkt den Typ des zugewiesenen Wertes
In folgendem Beispiel übernimmt Sie den Typen von '5' also 'number':

```
let myVar = 5
```

Mit Typ werden Variablen so definiert:

```
let isDone: boolean = false;
```

Je nachdem wie man eine Variable deklariert, ist sie in verschiedenen Scopes ersichtlich:

Globale Variable:

```
a: string = "this will be visible global"
```

Block-Scope gebundene Variable:

```
let a: string = "this will be visible in block scope"
```

Function-Scope gebundene Variable:

```
var a: string = "this will be visible in function scope"
```

Konstante (Im Gegensatz zu einer Variable muss eine Konstante jedoch bei ihrer Deklaration unmittelbar mit einem Wert initialisiert werden)

```
const a: string = "Das ist eine Konstante"
```

Operatoren

Folgende Operatoren existieren in TS sowie auch JS:

```
// arithmetische operatoren
let a = 10;
let b = 5;
let result: number;

result = a + b; // Summe 15
result = a - b; // Differenz 5
result = a * b; // Produkt: 50
result = a / b; // Quotient: 2
result = a % b; // Restwert: 0
a++; // a ist ab jetzt 11
b--; // b ist ab jetzt 4


// zuweisende operatoren
result = a; // Zuweisung: Linker Wert wird von rechtem überschrieben.
result += a; // ist wie result + a
result -= a; // ist wie result - a
result *= a; // ist wie result * a
result /= a; // ist wie result / a

// Strings
let string = 'abc' + 'def'; // ergibt abcdef
let stringAdditional = string + ' abc'; // ergibt abcdef abc

// Interpolation

let name = 'Marco';
let surname = 'Ruch';

string myString = `Ich heisse ${name} ${surname}`; // Resultat: Ich heisse Marco Ruch

// In den ${} Formatfeldern können auch Methoden mit Rückgabewert eingefügt werden
```

Klassen

Der Basic-Syntax für Klassen in TS (auch JS) ist wiefolgt:

```
export class MeineKlasse {
    // Rumpf der Klasse
    // hier werden Instanzvariablen, Eigenschaften
    // und Methoden definiert
}
```

Methoden

Beispiel in JavaScript/TypeScript:

```
export class MeineKlasse {

    public meineMethode(): void {
        // Methode 'addieren' in eigener Klasse aufrufen:
        console.log(this.addieren(1,2)); // 3
    }

    private addieren(num1: number, num2: number): number {
        return num1 + num2;
    }
}
```

Instanzvariablen und Eigenschaften

Beispiel in JavaScript/TypeScript:

```
export class MeineKlasse {

    private instanzvariable = "a";

    public get eigenschaft(): string {
        return this.instanzvariable;
    }

    public set eigenschaft(v: string) {
        this.instanzvariable = v;
    }
}
```

Klassen (Beispiel mit Konstruktor)

Beispiel in JavaScript/TypeScript:

```
export class Person {
    private _name: string;
    privaste _ahvnr: string;

    public get Name(): string { return this._name }
    public set Name(v: string) { this._name = v }

    public get AhvNr(): string { return this._ahvnr }

    constructor(name: string, ahvNr: string) {
        this._name = name;
        this._ahvnr = ahvNr;
    }
}


// Anderes File

import { Person } from './Person';

let p = new Person('Marco Ruch', '756.1234...');
console.log(p.Name); // gibt Marco Ruch aus
p.Name = 'Peter Meier'; // Name kann überschrieben werden
p.AhvNr = '756.4321...'; // AHV-Nr kann nicht überschrieben werden
```

Arrays und Objects

Beispiel in JavaScript/TypeScript:

```
// Arrays
let arr = [5,12,3,2];

// Wert an bestimmter Stelle ausgeben
console.log(arr[2]); // Ausgabe: 3

// Objects
let obj = {
    nachname: "Muster",
    vorname: "Peter", // , nach letzem member setzen, Trailing Commas sind häufig verwer
}

// Member ausgeben
console.log(obj.nachname); // folgendes wird ausgegeben: Muster
console.log(obj['vorname']); // folgendes wird ausgegeben: Peter
```

Entwicklungsumgebung

Es wird Visual Studio Code empfohlen
Git / Github verwenden

Angular Framework

Open Source JS-Framework
Angular JS hatte einige Schwachstellen die durch Angular behoben wurden.
September 2016 veröffentlicht durch Google.

Availability

Angular ist Cross Plattform tauglich - es läuft auf allen Browsern (also Computer, Tablet, Smartphones)
Das bedeutet: 1x Programmieren, überall verfügbar

PWA Grundsätze

- Offline verfügbar
- Installierbar

Architektur Node

NodeJS

- NPM
- Angular CLI

    - webpack
    - ngc (Angular Compiler) => Sorgt auch für Schnelligkeit der Applikation
        - TypeScript

Node

Plattform zur Ausführung von JavaScript, basiert auf "V8" der JavaScriptLaufzeitumgebung die ursprünglich für Google Chrome entwickelt wurde

npm

Package Manager für Node.js welcher unter anderem das Herunterladen von Software für Node.js, so wie auch die Angular CLI oder TypeScript

Angular CLI

Sammlung von Programmen welche die Entwicklung von AngularApplikationen massiv vereinfacht

ngc

Ahead-of-time (AoT) Compiler für Angular, welcher optimale Geschwindigkeit ermöglicht

TypeScript

TypeScript Compiler (tsc) welcher Sourcecode von TypeScript in ECMAScript wandelt

webpack

Optimierung von Webapplikationen in schlanke und effiziente Pakete

Angular Architektur

Angular ist ein Component basiertes JS-Framework

- View (Html)
- Code (Controller / JS)
- Stylesheet (Scss wird empfohlen)
    - Cleaner
    - Variablen
    - Logik
    - Spezfile

- Html Property-Binding zu Controller
- Eventbinding
- Alles durch Angular gesteuert (Properties / Events gebunden)

In Angular wird die Klasse (Controller Component) geladen und HTML wird durch sie geladen ("berechnet")
Oberste Component ist immer die App-Component (/src/app/app.component.ts)
Jede Angular-Applikation ist ein Baum von Components.
Komplette Applikationen können so in kleinere Einheiten zerlegt werden
Allgemeine Komponenten in einem 'Shared'-Ordner auf oberster Ebene ablegen.

Ordnerstruktur Beispiel für Komponenten

Komponenten gehören jeweils in einen eigenen Ordner.
Zusammengehörende Komponenten brauchen jeweils einen 'Eltern'-Ordner, insofern sie nur zusammen verwendet werden.
Eine Parent-Komponente ist direkt in dem Parent-Ordner und die in der Kompente enthaltenen Sub-Komponenten sind in jeweils eigenen Unterordnern abgelegt.

Hier sieht man zum Beispiel, dass die "owner-accounts"-Komponente in den "owner-details"-Komponentenordner untergeordnet ist, um Komponenten zu sammeln, welche in der Details-Komponente angezeigt werden.
Ordnerstruktur Beispiel

Bild von Code-Maze

Bild mit Erklärungen der einzelnen Struktur-Teilen

Styleguide

Wichtig Kleinschreibung da sonst auf Mac nicht kompilierbar
Beispiele:

- app.component.ts
- my-apps-homescreen.component.ts

Decorators

Beispiel von den Decorators

```
@Component({
  selector: 'app-bank-account',
  inputs: ['bankName', 'id: account-id'],
  template: `
    Bank Name: {{ bankName }}
    Account Id: {{ id }}
  `
})
```

Decorators stellen metadaten dar, welche der Komponente sagen, wie sie instanziiert wird und sich zu verhalten hat.

- templateUrl => für templateUrl
- styleUrls => als Array Stylesheets verlinken
- selector => Präfix wichtig, eindeutig setzen z.B. "CTS" (keine Namespaces, auch hilfreich für AutoImport)

Binding

Instanzvariablen (public) welche im Controller gespeichert sind, sollen in der View angezeigt werden
Property Bindings werden mit eckigen Klammern geschrieben
Textausgaben werden in doppelt geschweiften Klammern geschrieben (wie React {{ Obj.Prop }}) - à la 'innerHtml'

Property Binding

Es werden jeweils die Properties aus dem .ts-File ins Markdown abgefüllt.
Hier verwendet "titletext" aus dem .ts-File die öffentliche Variable 'titletext'
und 'Name' die ebenso öffentliche Variable 'Name' aus dem selben .ts-File.

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

@Component({
    selector: 'app-my-com',
    templateUrl: './my-com-component.html', // hier wieder trailing comma für nächsten E
})
export class MyComComponent {
    name = 'Marco';
    titletext = 'Name des Autors';
}
```

```
<h1>Begrüssung</h1>
<p [title]="titletext">  {{ Name }} </p>
```

Event Binding

Functions müssen innerhalb "this" verwenden. Hier verwendet "click" die aus dem .ts-File stammende öffentliche Funktion 'changeNameClick'.
Die Komponente wird nach Klicken der Schaltfläche neu gerendert, da sich eine Property mit Binding geändert hat.

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

@Component({
    selector: 'app-my-com',
    templateUrl: './my-com-component.html', // hier wieder trailing comma für nächsten E
})
export class MyComComponent {
    name = 'Marco';
    titletext = 'Name des Autors';

    changeNameClick(): void {
        this.name = 'Christian';
    }
}
```

```
<h1>Begrüssung</h1>
<p [title]="titletext">
    {{ Name }}
</p>
<button type="button" (click)="changeNameClick()">
    Name ändern
</button>
```

***ngIf**

Hier verwendet "showMessage" die aus dem .ts-File stammendeöffentliche Variable 'showMessage'

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

@Component({
    selector: 'app-condition',
    templateUrl: './condition.component.html', // hier wieder trailing comma für nächste
})
export class LoopComponent {
    showMessage = false;

    handleToggleClick(): void {
        showMessage = !showMessage;
    }
}
```

```
<div *ngIf="showMessage">Man sieht mich nur wenn showMessage = true ist</div>
<button type="button" (click)="handleToggleClick()">
    Toggle View
</button>
```

***ngFor**

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

@Component({
    selector: 'app-loop',
    templateUrl: './loop.component.html', // hier wieder trailing comma für nächsten Ent
})
export class LoopComponent {
    namenArray = ["Marco", "Kevin", "Dominik", "Sabrina"];
}
```

```
<div *ngFor="let name of namenArray"><{{name}}<div>
```

for in / for of

for of: Iteriert wie in foreach von C#

for in: Iteriert durch Object.keys(obj)

Properties von Parent zu Child-Component überliefern

parent-component.ts

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

@Component({
  selector: 'app-parent',
  template: `
    <app-child [childMessage]="parentMessage"></app-child>
  `,
  styleUrls: ['./parent.component.css']
})
export class ParentComponent{
  parentMessage = "message from parent"
  constructor() { }
}
```

child-component.ts

```
import { Component, Input } from '@angular/core';

@Component({
  selector: 'app-child',
  template: `
      Say {{ message }}
  `,
  styleUrls: ['./child.component.css']
})
export class ChildComponent {

  @Input() childMessage: string;

  constructor() { }

}
```

Weitere Möglichkeiten Daten auszutauschen

Angular Services

Injection

Services werden in Components injected
Services bzw. die Komponenten können sich dann untereinander austauschen
AppModule muss in Provider-Array alle Services enthalten.

Create Service File

```
ng generate service MemoryCalc
```

Angular Service erstellen

Die Variable memory bleibt während der Gesamten Laufzeit erhalten, sowie die ganze Instanz dieser Klasse

```
import { Injectable } from '@angular/core';

@Injectable({
    providedIn: 'root', //  EMpfehlung in welchem Modul der Service hinzugefügt werden s
)}

export class MemoryCalcService {
    private memory: number;

    /*
    * Addiert einen Wert zum vorherhigen Wert hinzu
    * @param sum Summand welcher zum Speicher hinzugefügt wird (So deklariert man Dokume
    */
    add(sum: number): number {
        this.memory += sum;
        return this.memory;
    }

    /*
    * Setzt gespeicherten Wert auf 0 zurück
    */
    reset(): void {
        thius.memory = 0;
    }
}
```

Angular Service Injecton

Beispiel für Injectable ```html @Injectable({ providedIn: 'root' })

Angular HttpClient

Beispiel der Verwendung des HttpClient-Services für Requests

Mehr Infos zu HttpClient

Beispiel eines Services

```
import { Injectable } from '@angular/core';
import { HttpClient } from '@angular/common/http';
import { Observable } from 'rxjs';
import { map } from 'rxjs/operators';
import { DateTime } from '../models/date-time';

@Injectable({
    providedIn: 'root', // hier wieder trailing Komma
})
export class DateTimeService {
    constructor(private httpClient: HttpClient) { }

    getDateTime(): Observable<Date> {
        return this.httpClient.get<DateTime>('http://date.jsontest.com/').pipe(
            map(x => new Date(`${x.date} ${x.time}`))
        );
    }
}
```

Verwenden eines Services

Hier veranschaulicht durch Verwendung des Konstruktors (Instanziierung des Services) und der Lifecycle-Methode ngOnInit:

```
  constructor(private dateTimeService: DateTimeService) { }

  ngOnInit(): void {
        this.dateTimeService.getDateTime().subscribe(date => this.datetime = date.toISOS
  }
```

Angular Lifecycle-Hooks

Mehr Lifecycle-Hooks!

Beispiel ngOnInit:

```
import { Component, OnInit } from '@angular/core';


@Component({
  selector: 'app-mycomponent',
})
export class MyComponent{

    constructor(private dateTimeService: DateTimeService) { }

    ngOnInit(): void {
        this.dateTimeService.getDateTime().subscribe(date => this.datetime = date.toISOS
    }
}
```

Shortcuts

Neue Komponente automatisch erstellen

```
ng generate component MyCom
```

Starten von App

```
ng serve --aot
```

Port ist standardmässig 4200
Erstellt Code der das HTML erstellt ahead of time

Veröffentlichen / Build

```
ng build --prod
```

unbedingt --prod verwenden, da sonst der dev build verwendet wird