ARS

Motivation

- Typsichere Entwicklung für das Web
- Potenzielle Probleme schon während der Entwicklung entdecken

ARS

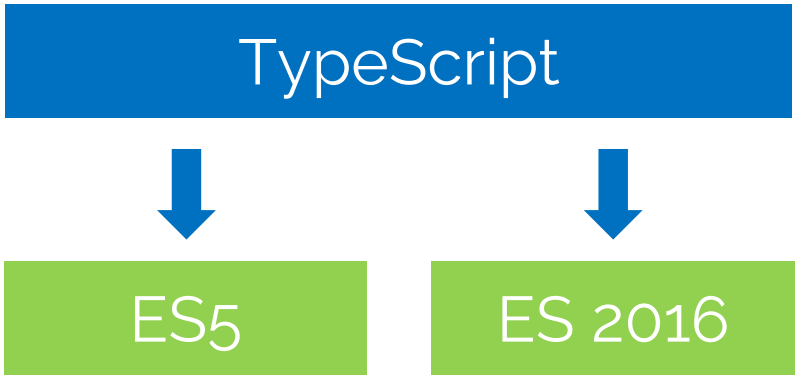
Lernziele

- Sie beschreiben die Bedeutung und den Nutzen von TypeScript.
- Sie erklären in eigenen Worten die wesentlichen Unterschiede zwischen TypeScript und JavaScript.
- Sie benennen wichtige TypeScript-spezifische Konstrukte und deren Bedeutung.

ARS

Was ist TypeScript?

- Durch Microsoft entwickelte Programmiersprache
- Erfinder Anders Hejlsberg (Turbo Pascal, Delphi, C#)
- Typisierte Obermenge von JavaScript
  - Jeder gültige JavaScript-Code ist gleichzeitig auch korrektes TypeScript
- Transpiliert zu JavaScript
- Ausführliche Dokumentation siehe ww.typescriptlang.org



ARS

Warum TypeScript?

Angular ist in TypeScript entwickelt!

ARS

Statische Typisierung

C#
```
int doubleNum(int i) { return 2 * i}
```
Java
```
public int doubleNum(int i) { return 2 * i}
```
Scala
```
def doubleNum(i: Int): Int = 2 * i
```
Swift
```
func doubleNum(i: Int) -> Int { return 2 * i }
```

Javascript

```
function doubleNum(i) { return 2 * i}
```

Vorteile der statischen Typisierung

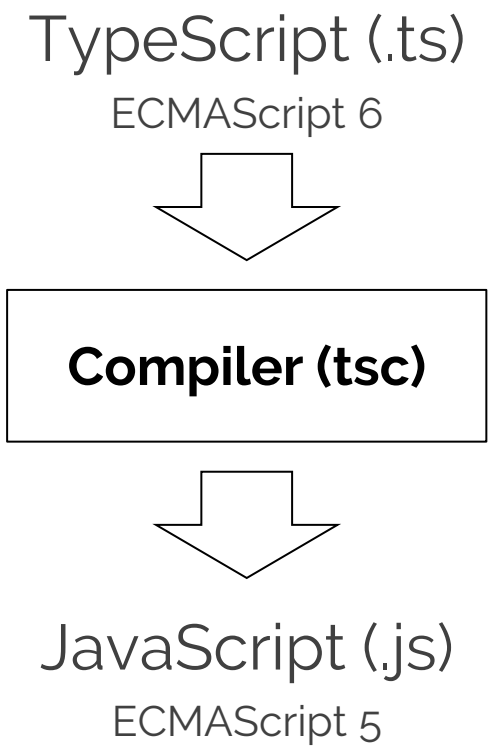
- Frühe Fehlererkennung schon während der Entwicklung
- Nachvollziehbarkeit und Stabilität
- Selbstdokumentierender Code
- Mehr Hilfe & Sicherheit beim Refactoring

ARS

Transpilierung

- Browser können TypeScript nicht ausführen
- Transpilierung
  - TypeScript-Code → JavaScript-Code
  - Übersetzung auf gleicher Abstraktionsebene
- Mögliche Transpilierungsziele
  - ES5
  - ES2016
- IDEs und Tools für automatische / gleichzeitige Transpilierung

  → `tsc code.ts --target ES5 --watch`
- APIs von JS-Bibliotheken unterstützen häufig TypeScript (Type Definition Files)



ARS

Primitive Typen

- Angabe von Typen bei Variablen
  - Mit Doppelpunkt nach Variablennamen
  - Nicht zwingend notwendig, da Typ-Inferenz bedingt korrekte Typen automatisch erkennen kann (vergleichbar mit C# var, oder Scala)
- String: Folge von Unicode-Zeichen

```
let text : string = 'Hallo Welt 42';
```

- Number: Ein Typ für alle Zahlen; kein Integer, Float oder Double

```
let num : number = 42;
```

- Boolean: Wahrheitswert (true | false)

```
let areWeHavingFunYet: boolean = true;
```

- Array: Sammlung von Elementen des gleichen Typs

```
const colors : string[] = ['red', 'blue', 'green'];
```

ARS

Primitive Typen

- Enum → Definition einer vorgegebenen Gruppe von Konstanten

```
enum TShirtSize {
      Small,
      Medium,
      Large
}
const mySize = TShirtSize.Large;
```

ARS

Primitive Typen

- Any
  - Obermenge aller Typen
  - Keine Typprüfung

```
let notSure: any = 4; // cool! Seems to be a number
notSure = "maybe a string instead";
notSure = false;      //okay, definitely a boolean now
```

ARS

Primitive Typen

- Tupel → Vergleichbar mit einem Array von Elementen, jedoch mit fester Anzahl und festen Typen

```
let tupel : [string, number] = ['Hallo', 42];
```

ARS

Funktionen

- Typ Annotationen für Parameter- und Rückgabetypen:

```
function doubleNum (num: number): number {
      return num * 2;
}
```

ARS

Funktionen

- Pflicht-, Default- und optionale Parameter:
  - Angabe von Default-Werten für Parametern, vergleichbar mit Zuweisung
  - Optionale Parameter über „?“-Operator

```
function showDetails (name: string, place: string = 'Munich'): void {
      console.log(`Name: ${name}, Place: ${place)
}

function showDetails (name: string, age?: number): void {
      console.log(`Name: ${name}, Age: ${age}`)
}
```

ARS

Funktionen

- Variable Anzahl von Parametern

```
function createList (...elements: string[]) {
        return elements.join(',');
}
createList('Munich', 'Berlin', 'Stuttgart');
```

ARS

Funktionen

- Überladen von Funktionen
  - Bei JavaScript nicht möglich, weil es dort nativ keine Typen gibt
  - Lediglich Definition einer alternativen Methoden-Signatur
  - Dient dazu Autovervollständigung zu unterstützen

```
function add (num1: number, num2: number): number;
function add (num1: string, num2: string): string;
function add (num1, num2): any{
        if (typeof num1 === "number") {
                return num1 + num2;
        }
        else if (typeof num1 === "string") {
                return `${num1} + ${num2}`
        }
}
```

ARS

Klassen

- Definition wie bei anderen Programmiersprachen auch
  - Typische Sichtbarkeits-Modifier: private, protected und public
  - Definition von Typen für Objektvariablen, Parametern und Funktionsergebnissen über „:“ (Doppelpunkt)
  - Besonderheiten:
    - Zugriff auf Objektvariablen und –methoden nur über „this.“ möglich
    - Übergabeparameter im Konstruktor mit Sichtbarkeits-Modifier → Wird automatisch als Objektvariable deklariert

ARS

Klassen

```
class Person {
  public surname: string;
  protected name: string;
  private age: number;

  constructor(surname, name, age, private gender: Gender) {
    this.surname = surname;
    this.name = name;
    this.age = age;
  }
```

…

ARS

Klassen

```
...
    public toString() {
        return this.getFullName() + ', Age: ' + this.getAge() +
        ', Gender: ' + this.gender;
    }
    protected getAge() {
        return this.age;
    }

    private getFullName(): string {
        return `${this.name} ${this.surname}`;
    }
}
```

ARS

Klassen – Vererbung und Interfaces

- Vererbung – Mehrfachvererbung nicht möglich!

```
class Animal {}

class Dog extends Animal {}
```

- Interface

```
interface Running {}

class Dog extends Animal implements Running {}
```

ARS

Klassen – Duck Typing

- TypeScript überprüft die Kompatibilität von Typen anhand der Struktur (Methoden, Objektvariablen)
- „Wenn es sich wie eine Ente verhält, und quakt wie eine Ente, dann ist es eine Ente“
  → Duck Typing
- **Vorteile**
  - Keine Transformationsmethoden zwischen verschiedenen Frameworks notwendig
  - Beispiel:
    Point = Point, so lange die Struktur gleich ist
- https://www.typescriptlang.org/docs/handbook/type-compatibility.html

ARS

Klassen – Duck Typing

```
class Dog {
  public name: string = 'Fido';
  constructor() { }
}

class Cat {
  public name: string = 'Mautzi';
  constructor() { }
}

const animal: Cat = new Dog();
```

Was passiert bei dieser Zuweisung?

ARS

Modulsystem

- TypeScript kommt mit einem Modulsystem (auf Basis von ES2016)
- Module entsprechen Dateien
- Dadurch kann gesteuert werden,
  was eine Datei nach außen zur Verfügung stellt und was nicht
- Vergleichbar mit Java „imports“

ARS

Modulsystem

AnimalUtilities.ts

```
function brush (animal) { … }
function soap (animal) { … }
function wash (animal) { … }

export function groom (animal) {
    soap (animal);
    wash (animal);
    brush (animal);
}
```

Main.ts

```
import { groom }
       from 'AnimalUtilities';

const dog = new Animal();

groom (dog);
```

ARS

Zusammenfassung

Um was ging es in diesem Modul?

- Programmiersprache TypeScript
  - Besonderheiten
  - Primitive Datentypen
  - Funktionen
  - Klassen
  - Transpilierung zu JavaScript

Wozu brauche ich das? Was will ich damit machen?

- Typsicher große Webanwendungen entwickeln
- Komplexe, dynamische Inhalte in Webanwendungwn umsetzen
- Breite Verfügbarkeit von JavaScript nutzen

ARS

Kontrollfragen

- Kann der Browser TypeScript direkt ausführen?
- Was passiert beim Übersetzungsvorgang von TypeScript zu JavaScript?
- Welche Besonderheiten sind beim Zugriff auf Objektvariablen und –methoden zu beachten?
- Was passiert, wenn Konstruktorparameter mit Sichtbarkeitsmodifiern versehen werden?